Reactive Probabilistic Programming

Guillaume Baudart

Louis Mandel

IBM Research

Eric Atkinson
Benjamin Sherman
Michael Carbin

MIT

Marc Pouzet

ENS Paris, INRIA/PSL

Uncertainty in Cyber-Physical Systems

Reactive systems

- Noisy environment, perceived through noisy sensors
- Interaction with other autonomous entities

Existing approaches

- Manually implement stochastic controllers
  Can be error prone
- Offline statistical tests
  Requires up-to-date offline data

Probabilistic Programming

- Make the underlying model explicit
- Generic inference: online learning
- Overt uncertainty: online monitoring
- Implement AI skills out of reach for classic controllers

Uncertainty in Cyber-Physical Systems

Reactive systems

- Noisy environment, perceived through noisy sensors
- Interaction with other autonomous entities

Existing approaches

- Manually implement stochastic controllers
  Can be error prone
- Offline statistical tests
  Requires up-to-date offline data

Probabilistic Programming

- Make the underlying model explicit
- Generic inference: online learning
- Overt uncertainty: online monitoring
- Implement AI skills out of reach for classic controllers

Uncertainty in Cyber-Physical Systems

Reactive systems

- Noisy environment, perceived through noisy sensors
- Interaction with other autonomous entities

Existing approaches

- Manually implement stochastic controllers
  Can be error prone
- Offline statistical tests
  Requires up-to-date offline data

Probabilistic Programming

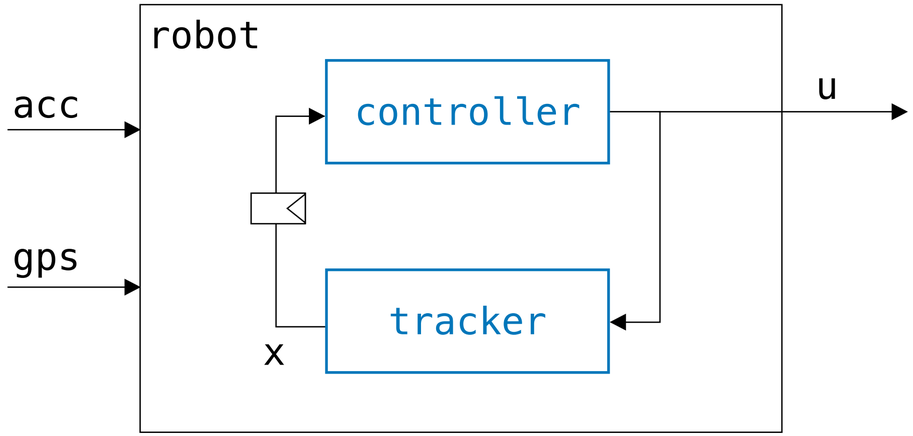
- Make the underlying model explicit
- Generic inference: online learning
- Overt uncertainty: online monitoring
- Implement AI skills out of reach for classic controllers

Reactive Systems

Synchronous data-flow languages and block diagrams

- Signal: stream of values
- System: stream processor

State: `x:` $(position \times velocity \times acceleration)$




```
let node robot (acc, gps) = u where
  rec u = controller (x0 → pre x)
  and x = tracker (u, acc, gps)
```

Reactive Probabilistic Systems

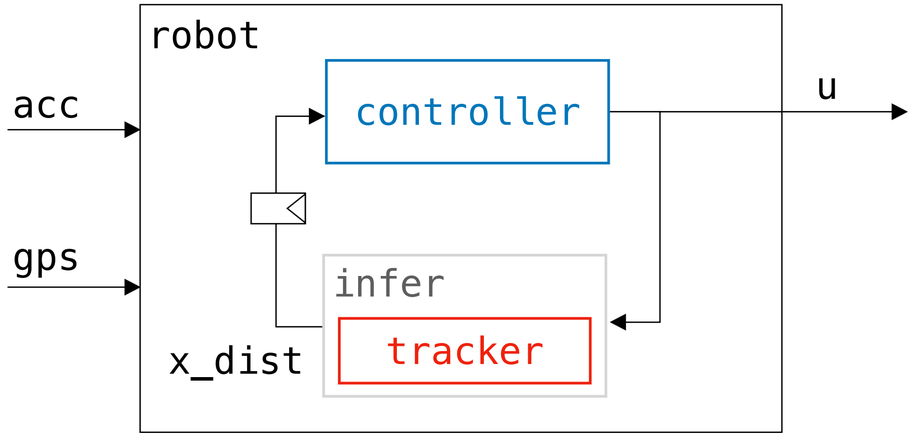
Synchronous data-flow languages and block diagrams

- Signal: stream of values
- System: stream processor

ProbZelus: add support to deal with uncertainty

- Extend Zelus with probabilistic constructs
- Parallel composition: deterministic/probabilistic
- Inference-in-the-loop
- Streaming inference

State: `x_dist:` $(position \times velocity \times acceleration)$ `dist`




```
let node robot (acc, gps) = u where
  rec u = controller (x0_dist → pre x_dist)
  and x_dist = infer tracker (u, acc, gps)
```

Outline

Hands-on introduction

- Programming with distributions and conditioning
- Inference
- Reactive probabilistic programming

Language design, typing, and semantics

- Parallel composition, control structures, and inference-in-the-loop
- Measure-based semantics

Compilation

Inference

- Sequential Monte Carlo methods
- Semi-symbolic inference: Delayed Sampling

Applications

https://github.com/IBM/probzelus

Hands-on!

Language, Typing, Semantics

Syntax

d ::= let node f x = e | let proba f x = e | d d
e ::= c | x | (e,e) | $op(e)$ | $f(e)$ | last x
| e where rec E
| present e -> e else e | reset e every e
| sample(e) | observe(e, e) | infer(e)
E ::= x = e | init x = c | E and E

```
let node integr (xo, x') = x where
  rec x = xo -> (pre x + x' * h)

let node integr (xo, x') = x where
  rec init first = true and init x = 0.
  and first = false
  and x = if last first then xo else last x + (x' * h)
```

We focus here on a schedule intermediate language

e where rec init x_1 = c_1 ... and init x_k = c_k
and y_1 = e_1 ... and y_n = e_n

Typing: deterministic vs. probabilistic

$$\frac{G \vdash^{\mathsf{D}} e : t}{G \vdash^{\mathsf{P}} e : t} \qquad \frac{typeOf(c) = t}{G \vdash^{\mathsf{D}} c : t} \qquad \frac{G(x) = t}{G \vdash^{\mathsf{D}} x : t} \qquad \frac{G \vdash^{k} e_1 : t_1 \quad G \vdash^{k} e_2 : t_2}{G \vdash^{k} (e_1,\ e_2) : t_1 \times t_2} \qquad \frac{typeOf(op) = t_1 \to^{\mathsf{D}} t_2 \quad G \vdash^{k} e : t_1}{G \vdash^{k} op(e) : t_2}$$

$$\frac{G(f) = t_1 \to^{k} t_2 \quad G \vdash^{\mathsf{D}} e : t_1}{G \vdash^{k} f(e) : t_2} \qquad \frac{G(x) = t}{G \vdash^{\mathsf{D}} \texttt{last}\ x : t} \qquad \frac{G \vdash^{k} E : G' \quad G + G' \vdash^{k} e : t}{G \vdash^{k} e\ \texttt{where rec}\ E : t}$$

$$\frac{G \vdash^{k} e : \texttt{bool} \quad G \vdash^{k} e_1 : t \quad G \vdash^{k} e_2 : t}{G \vdash^{k} \texttt{present}\ e\ \texttt{->}\ e_1\ \texttt{else}\ e_2 : t} \qquad \frac{G \vdash^{k} e_1 : t \quad G \vdash^{k} e_2 : \texttt{bool}}{G \vdash^{k} \texttt{reset}\ e_1\ \texttt{every}\ e_2 : t}$$

$$\frac{G \vdash^{\mathsf{D}} e : t\ \texttt{dist}}{G \vdash^{\mathsf{P}} \texttt{sample}(e) : t} \qquad \frac{G \vdash^{\mathsf{D}} e_1 : t\ \texttt{dist}^* \quad G \vdash^{\mathsf{D}} e_2 : t}{G \vdash^{\mathsf{P}} \texttt{observe}(e_1,\ e_2) : \texttt{unit}} \qquad \frac{G \vdash^{\mathsf{D}} e : \texttt{float}}{G \vdash^{\mathsf{P}} \texttt{factor}(e) : \texttt{unit}}$$

$$\frac{G \vdash^{\mathsf{P}} e : t}{G \vdash^{\mathsf{D}} \texttt{infer}(e) : t\ \texttt{dist}} \qquad \frac{G \vdash^{\mathsf{D}} e : T\ \texttt{dist}^*}{G \vdash^{\mathsf{D}} e : T\ \texttt{dist}}$$

$$\frac{G \vdash^{k} e : t}{G \vdash^{k} x\ \texttt{=}\ e : [t/x]} \qquad \frac{G \vdash^{k} e : t}{G \vdash^{k} \texttt{init}\ x\ \texttt{=}\ e : [t/x]} \qquad \frac{G + G_1 + G_2 \vdash^{k} E_1 : G_1 \quad G + G_1 + G_2 \vdash^{k} E_2 : G_2}{G \vdash^{k} E_1\ \texttt{and}\ E_2 : G_1 + G_2}$$

$$\frac{G + [t_1/x] \vdash^{\mathsf{D}} e : t_2}{G \vdash^{\mathsf{D}} \texttt{let node}\ f\ x\ \texttt{=}\ e : G + [t_1 \to^{\mathsf{D}} t_2/f]} \qquad \frac{G + [t_1/x] \vdash^{\mathsf{P}} e : t_2}{G \vdash^{\mathsf{D}} \texttt{let proba}\ f\ x\ \texttt{=}\ e : G + [t_1 \to^{\mathsf{P}} t_2/f]} \qquad \frac{G \vdash^{\mathsf{D}} d_1 : G_1 \quad G_1 \vdash^{\mathsf{D}} d_2 : G_2}{G \vdash^{\mathsf{D}} d_1\ \ d_2 : G_2}$$

Typing: deterministic vs. probabilistic

$$\frac{G \vdash^{\mathsf{D}} e : t}{G \vdash^{\mathsf{P}} e : t} \qquad \frac{typeOf(c) = t}{G \vdash^{\mathsf{D}} c : t} \qquad \frac{G(x) = t}{G \vdash^{\mathsf{D}} x : t} \qquad \frac{G \vdash^{k} e_1 : t_1 \quad G \vdash^{k} e_2 : t_2}{G \vdash^{k} (e_1,\ e_2) : t_1 \times t_2} \qquad \frac{typeOf(op) = t_1 \to^{\mathsf{D}} t_2 \quad G \vdash^{k} e : t_1}{G \vdash^{k} op(e) : t_2}$$

$$\frac{G(f) = t_1 \to^{k} t_2 \quad G \vdash^{\mathsf{D}} e : t_1}{G \vdash^{k} f(e) : t_2} \qquad \frac{G(x) = t}{G \vdash^{\mathsf{D}} \texttt{last}\ x : t} \qquad \frac{G \vdash^{k} E : G' \quad G + G' \vdash^{k} e : t}{G \vdash^{k} e\ \texttt{where rec}\ E : t}$$

$$\frac{G \vdash^{\mathsf{D}} e : T\ \mathsf{dist}}{G \vdash^{\mathsf{P}} \texttt{sample}(e) : T} \qquad \frac{G \vdash^{\mathsf{D}} e_1 : T\ \mathsf{dist}^* \quad G \vdash^{\mathsf{D}} e_2 : T}{G \vdash^{\mathsf{P}} \texttt{observe}(e_1,\ e_2) : \mathsf{unit}}$$

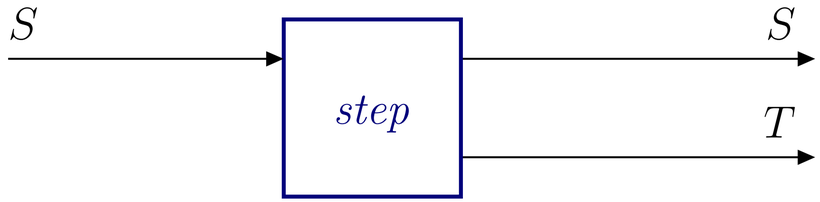
$$\frac{G \vdash^{\mathsf{D}} e : T}{G \vdash^{\mathsf{P}} e : T} \qquad \frac{G \vdash^{\mathsf{P}} e : T}{G \vdash^{\mathsf{D}} \texttt{infer}(e) : T\ \mathsf{dist}} \qquad \frac{G \vdash^{\mathsf{D}} e : T\ \mathsf{dist}^*}{G \vdash^{\mathsf{D}} e : T\ \mathsf{dist}}$$

$$\frac{G + [t_1/x] \vdash^{\mathsf{D}} e : t_2}{G \vdash^{\mathsf{D}} \texttt{let node}\ f\ x\ =\ e : G + [t_1 \to^{\mathsf{D}} t_2/f]} \qquad \frac{G + [t_1/x] \vdash^{\mathsf{P}} e : t_2}{G \vdash^{\mathsf{D}} \texttt{let proba}\ f\ x\ =\ e : G + [t_1 \to^{\mathsf{P}} t_2/f]} \qquad \frac{G \vdash^{\mathsf{D}} d_1 : G_1 \quad G_1 \vdash^{\mathsf{D}} d_2 : G_2}{G \vdash^{\mathsf{D}} d_1\ \ d_2 : G_2}$$

Semantics of Deterministic Streams

Initial state, transition function

$CoStream(T, S) = S \times (S \to S \times T)$

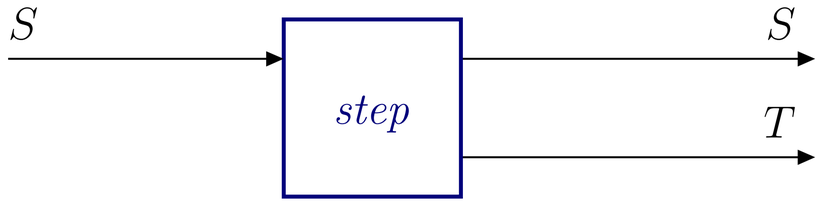


Caspi & Pouzet. 1998

Semantics of Deterministic Streams

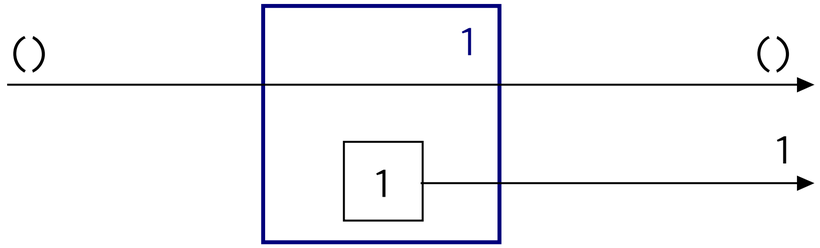
Initial state, transition function

$$CoStream(T, S) = S \times (S \to S \times T)$$



Constant 1: $\texttt{unit} \to (\texttt{unit} \times \texttt{int})$

- Initial state: the value of type unit
- Step function: return the constant 1

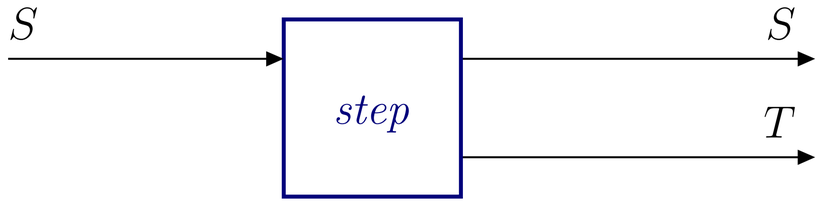


Caspi & Pouzet. 1998

Semantics of Deterministic Streams

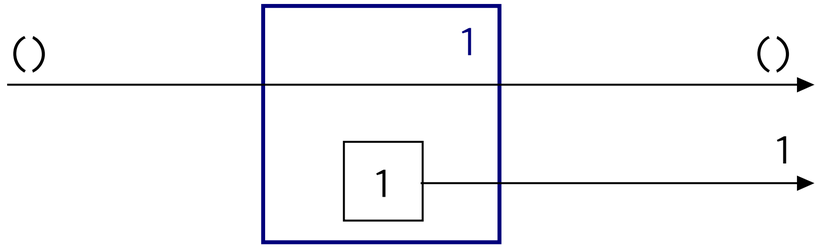
Initial state, transition function

$CoStream(T, S) = S \times (S \to S \times T)$



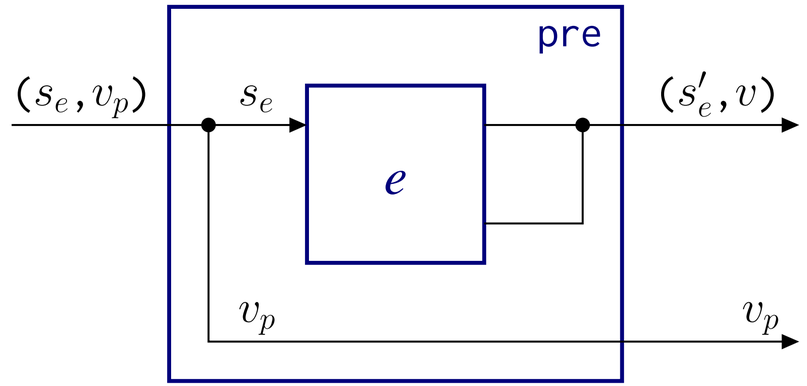
Constant 1: `unit` $\to$ (`unit` $\times$ `int`)

- Initial state: the value of type unit
- Step function: return the constant 1



Unit delay `pre` e: $(S \times T) \times (S \times T \to (S \times T) \times T)$

- Initial state: the initial state of e and default value
- Step function: the result of e is stored in the state and returned at the next iteration

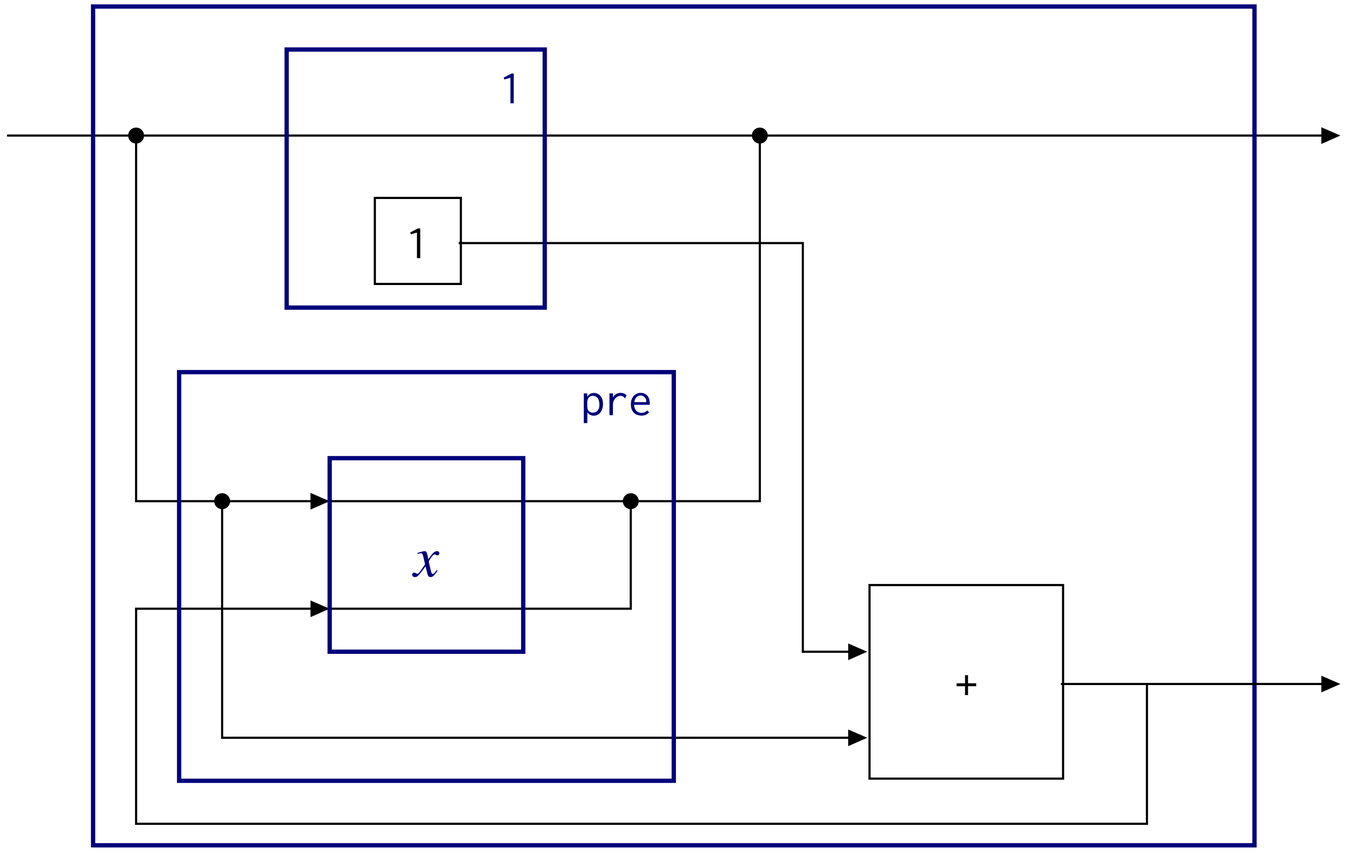


Caspi & Pouzet. 1998

Example

```
rec x = 1 + pre x
```

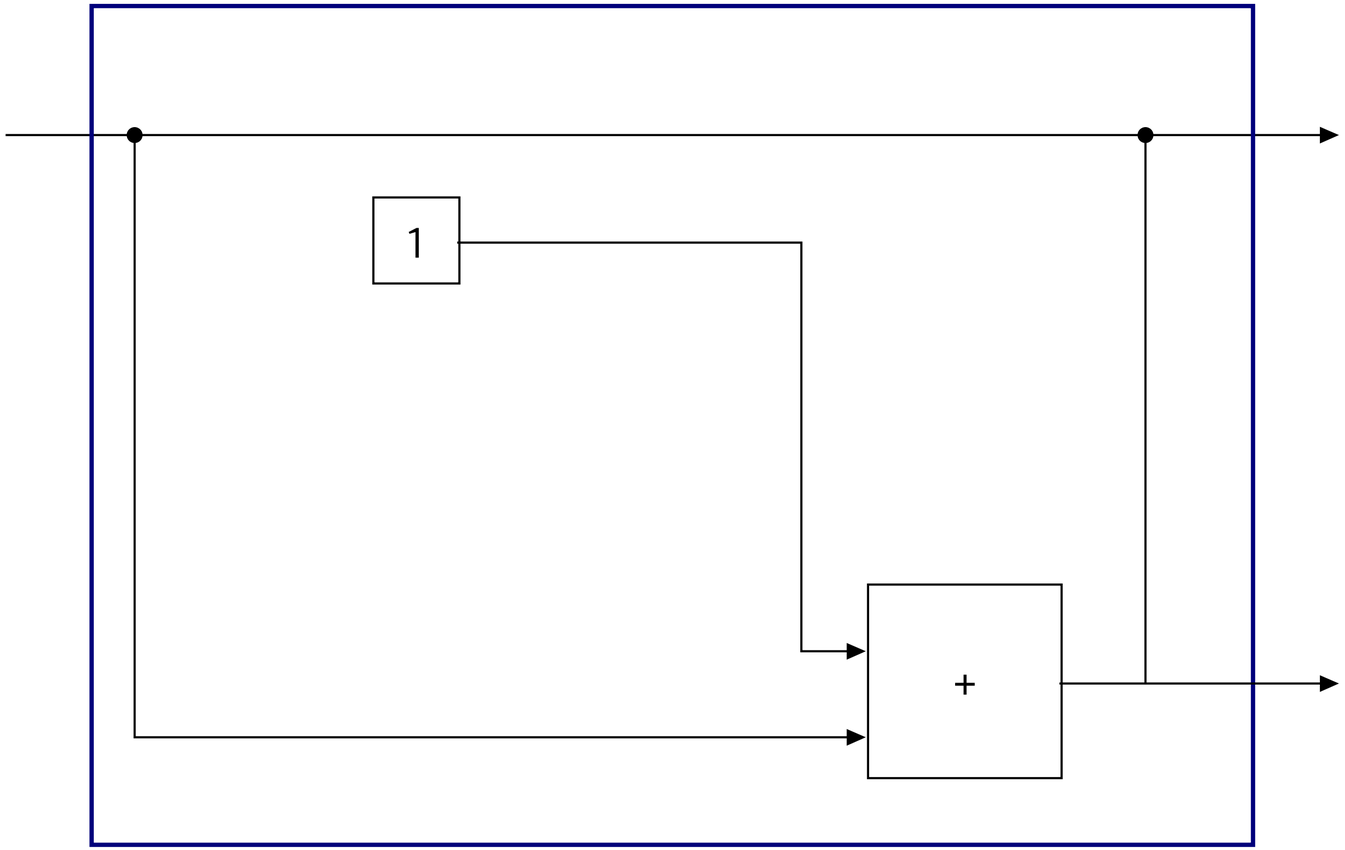
- Initial state: `(), 0`
- Output: `1, 2, 3, 4, ....`



Example

```
rec x = 1 + pre x
```

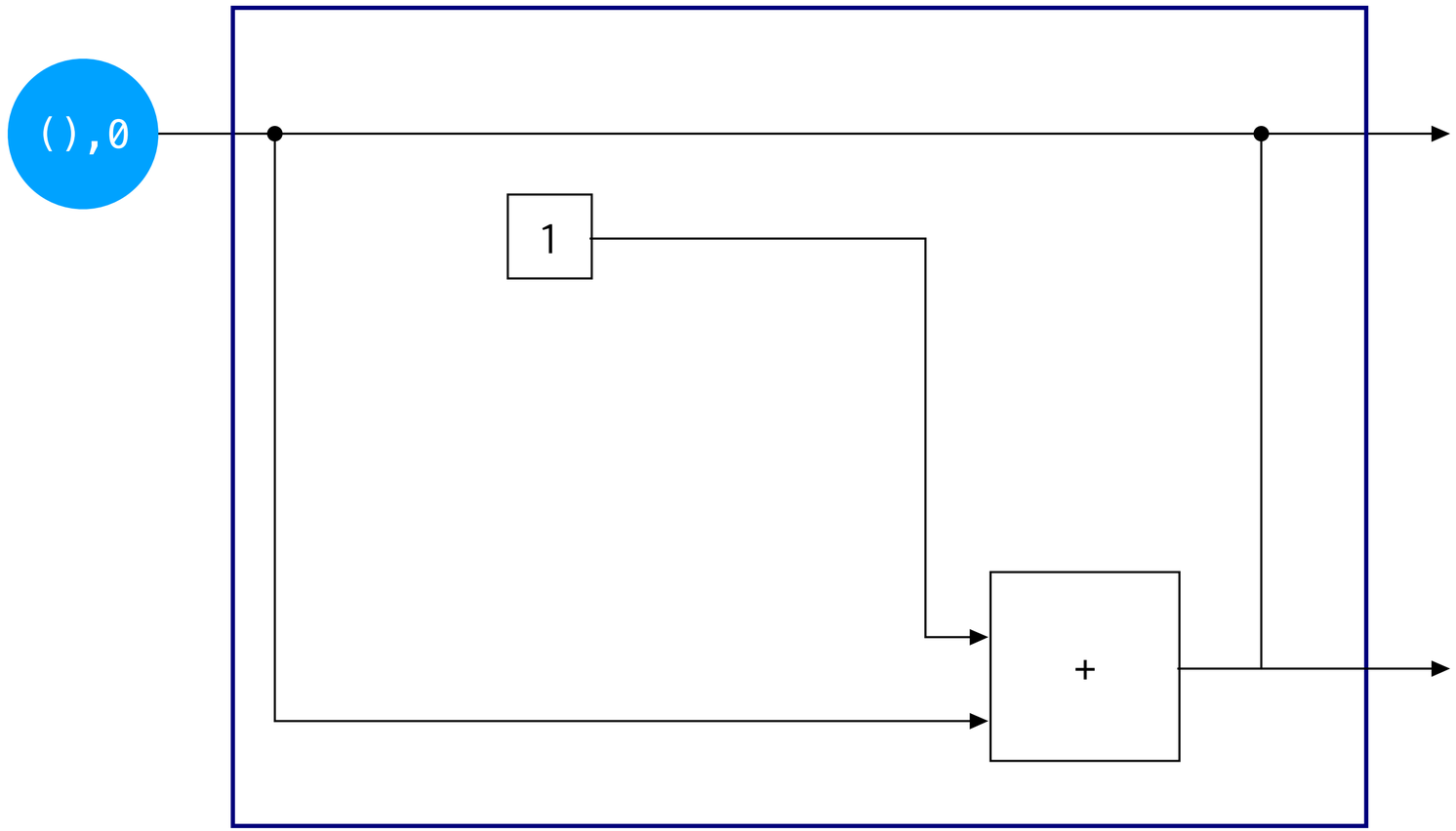
- Initial state: `(), 0`
- Output: `1, 2, 3, 4, ....`



Example

```
rec x = 1 + pre x
```

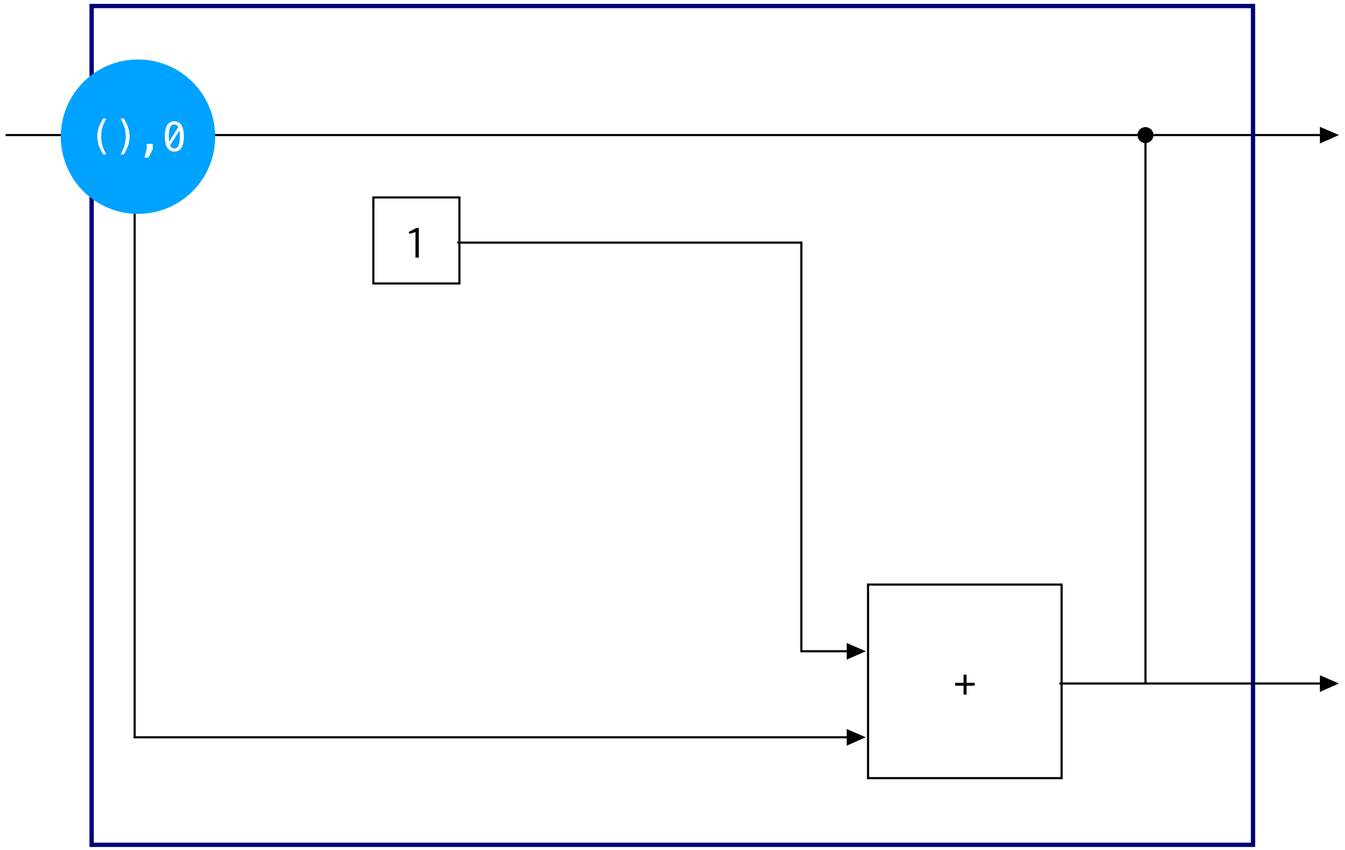
- Initial state: `(), 0`
- Output: `1, 2, 3, 4, ....`



Example

```
rec x = 1 + pre x
```

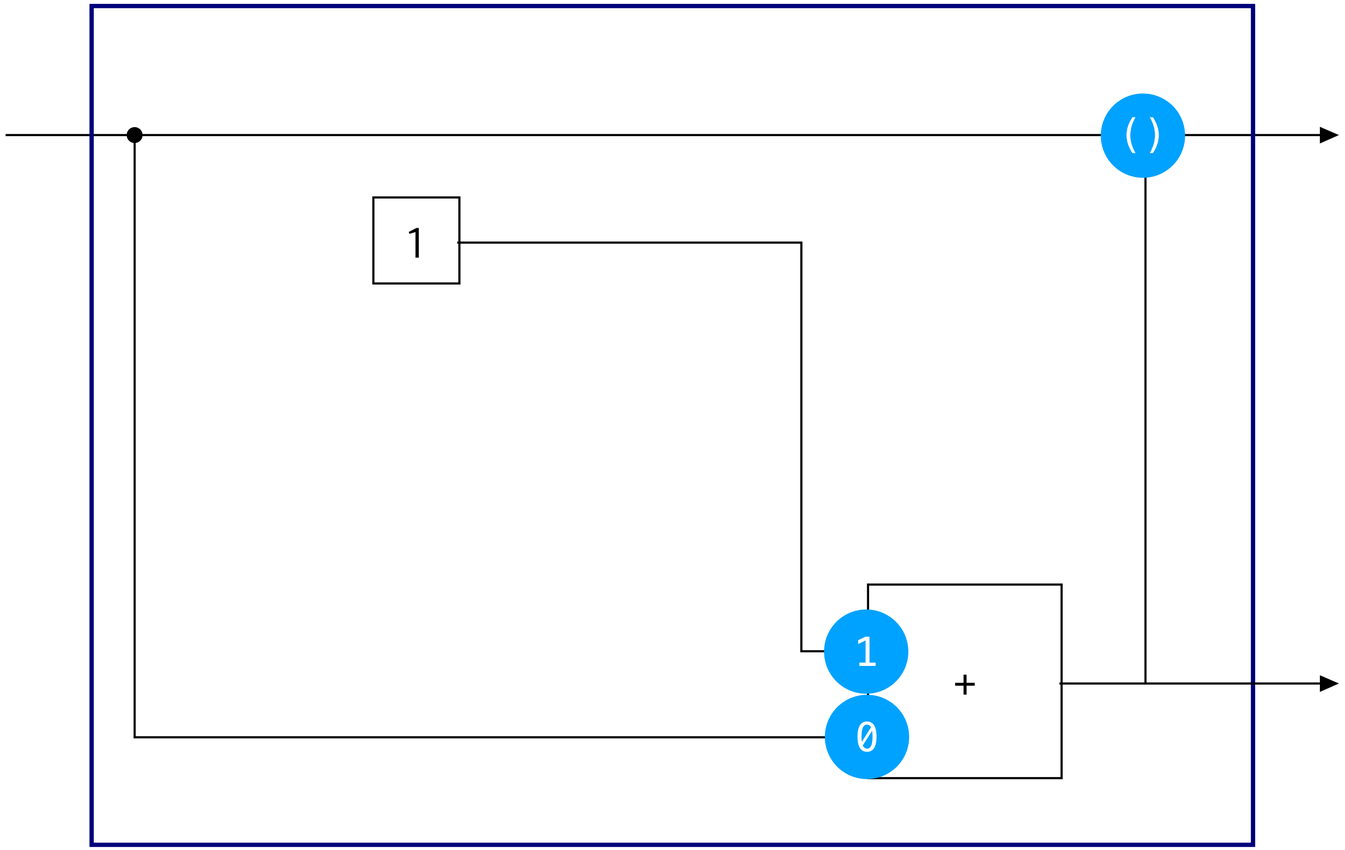
- Initial state: `(), 0`
- Output: `1, 2, 3, 4, ....`



Example

```
rec x = 1 + pre x
```

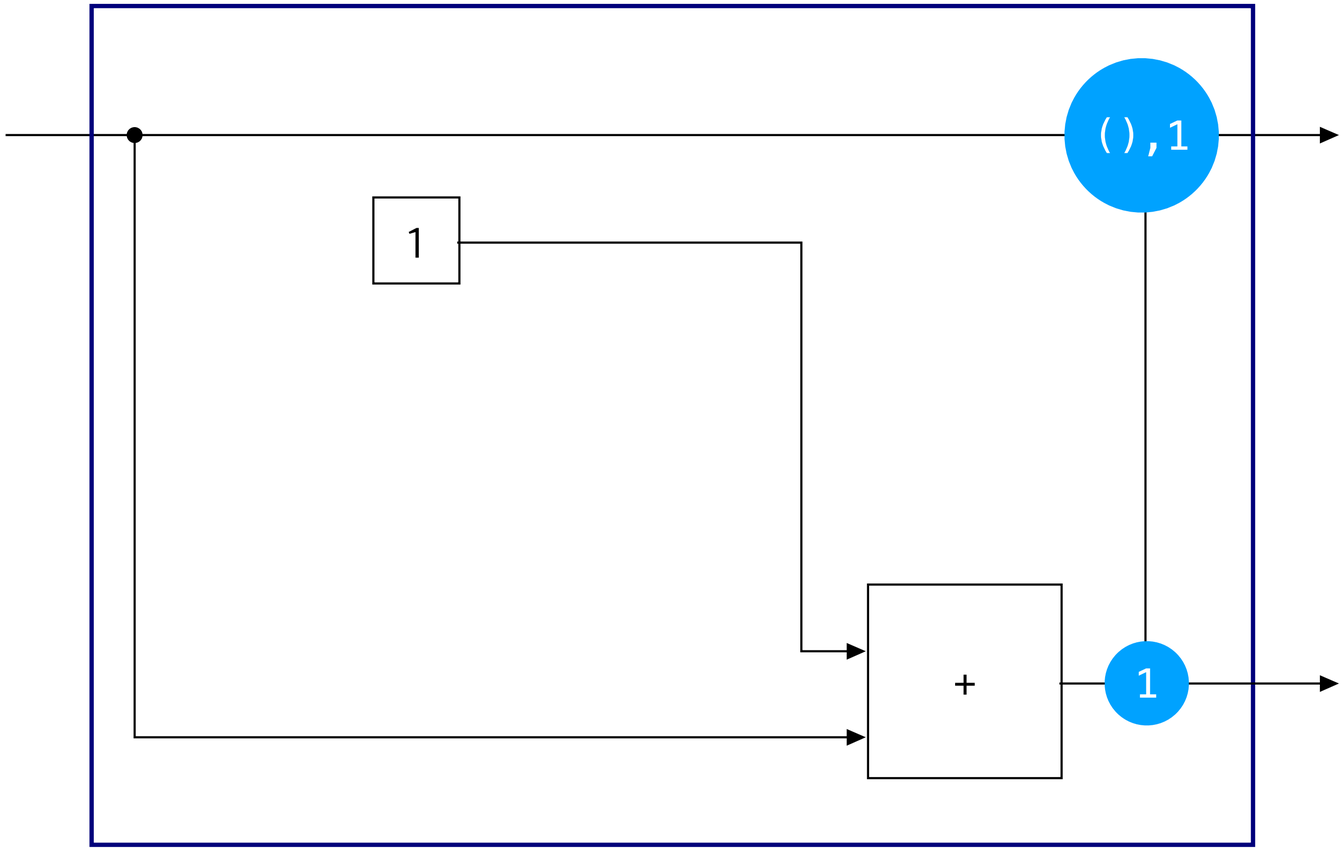
- Initial state: `(), 0`
- Output: `1, 2, 3, 4, ....`



Example

```
rec x = 1 + pre x
```

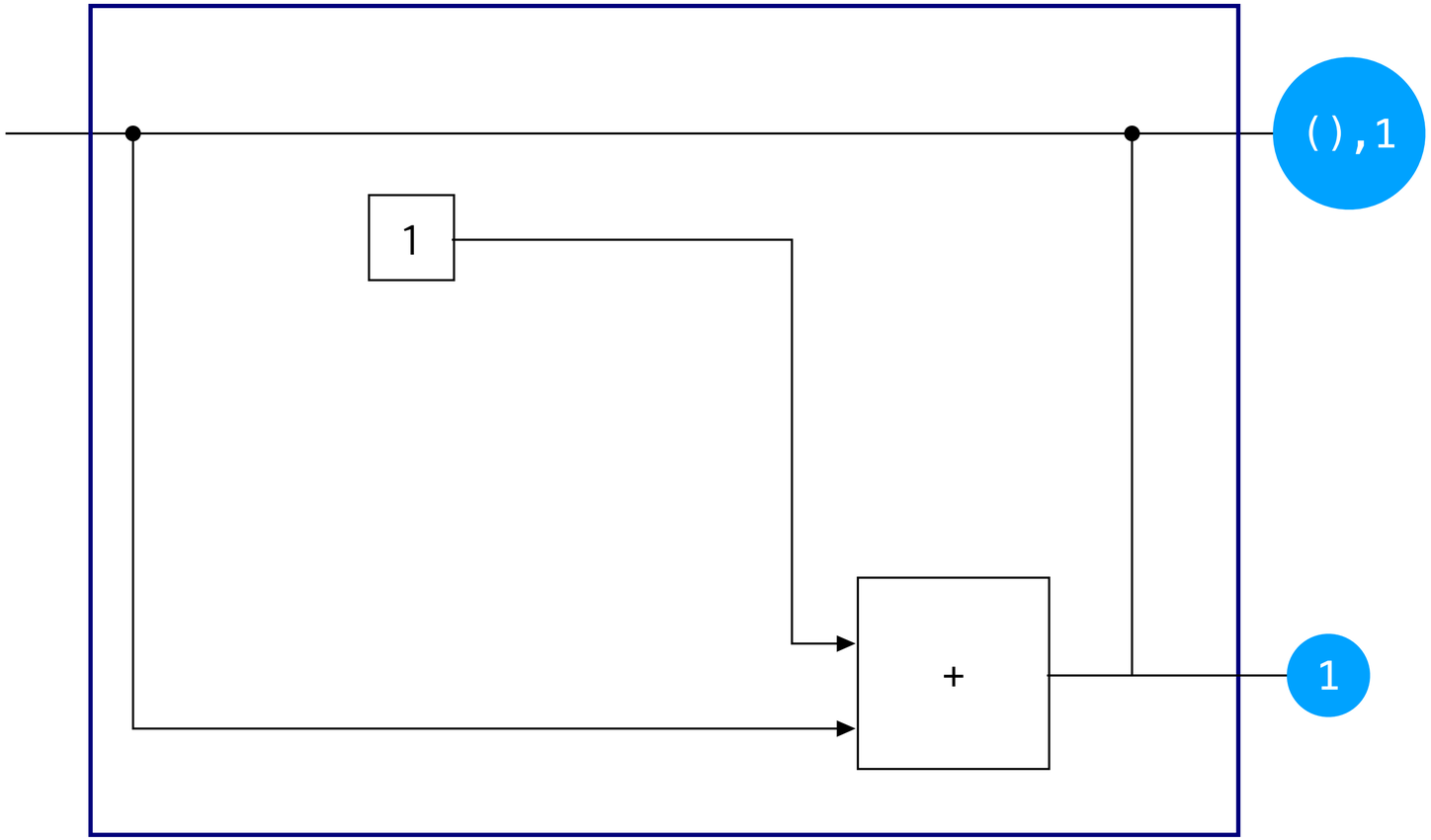
- Initial state: `(), 0`
- Output: `1, 2, 3, 4, ....`



Example

```
rec x = 1 + pre x
```

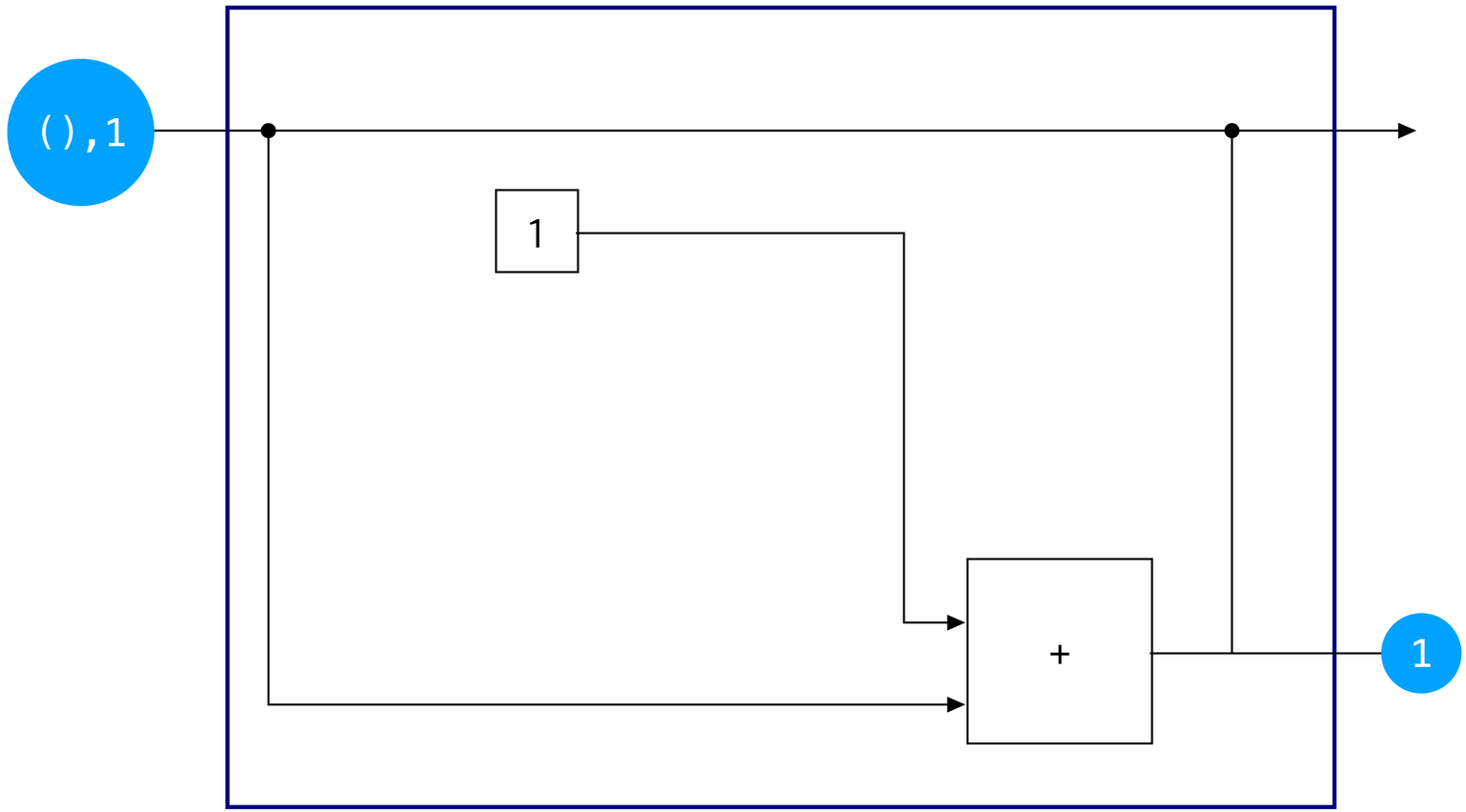
- Initial state: `(), 0`
- Output: `1, 2, 3, 4, ....`



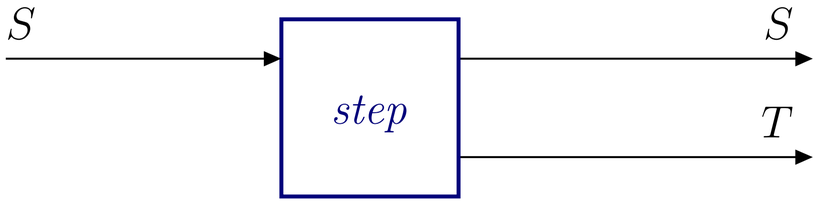
Example

```
rec x = 1 + pre x
```

- Initial state: `(), 0`
- Output: `1, 2, 3, 4, ....`



Semantics of Deterministic Streams



$$[\![x]\!]^i_\gamma = ()$$
$$[\![x]\!]^s_\gamma = \lambda s.\ (\gamma(x), s)$$

$$[\![\texttt{present}\ e\ \texttt{->}\ e_1\ \texttt{else}\ e_2]\!]^i_\gamma = ([\![e]\!]^i_\gamma, [\![e_1]\!]^i_\gamma, [\![e_2]\!]^i_\gamma)$$
$$[\![\texttt{present}\ e\ \texttt{->}\ e_1\ \texttt{else}\ e_2]\!]^s_\gamma =$$
$$\lambda(s, s_1, s_2).\ \mathit{let}\ v, s' = [\![e]\!]^s_\gamma(s)\ \mathit{in}$$
$$\mathit{if}\ v\ \mathit{then}\ \mathit{let}\ v_1, s_1' = [\![e_1]\!]^s_\gamma(s_1)\ \mathit{in}\ (v_1, (s', s_1', s_2))$$
$$\mathit{else}\ \ \mathit{let}\ v_2, s_2' = [\![e_2]\!]^s_\gamma(s_2)\ \mathit{in}\ (v_2, (s', s_1, s_2'))$$

$$\left[\!\!\left[\begin{array}{l} e\ \texttt{where rec init}\ x_1\ \texttt{=}\ c_1\ \texttt{...}\ \texttt{and init}\ x_k\ \texttt{=}\ c_k \\ \qquad\qquad \texttt{and}\ y_1\ \texttt{=}\ e_1\ \texttt{...}\ \texttt{and}\ y_n\ \texttt{=}\ e_n \end{array}\right]\!\!\right]^i_\gamma =$$
$$((c_1, \ldots, c_k),\ ([\![e_1]\!]^i_\gamma, \ldots, [\![e_n]\!]^i_\gamma),\ [\![e]\!]^i_\gamma)$$

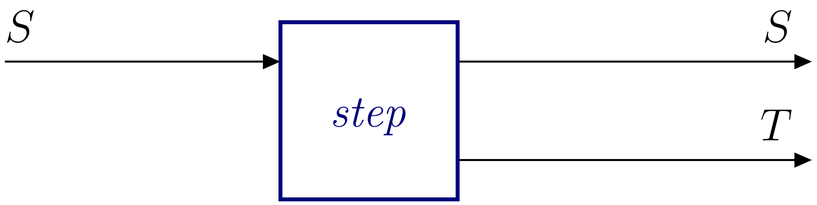
$$\left[\!\!\left[\begin{array}{l} e\ \texttt{where rec init}\ x_1\ \texttt{=}\ c_1\ \texttt{...}\ \texttt{and init}\ x_k\ \texttt{=}\ c_k \\ \qquad\qquad \texttt{and}\ y_1\ \texttt{=}\ e_1\ \texttt{...}\ \texttt{and}\ y_n\ \texttt{=}\ e_n \end{array}\right]\!\!\right]^s_\gamma =$$
$$\lambda((m_1, \ldots, m_k), (s_1, \ldots, s_n), s).$$
$$\mathit{let}\ \gamma_1 = \gamma[m_1/x_1\texttt{\_last}]\ \mathit{in}\ \ldots\ \mathit{let}\ \gamma_k = \gamma_{k-1}[m_k/x_k\texttt{\_last}]\ \mathit{in}$$
$$\mathit{let}\ v_1, s_1' = [\![e_1]\!]^s_{\gamma_k}(s_1)\ \mathit{in}\ \mathit{let}\ \gamma_1' = \gamma_k[v_1/y_1]\ \mathit{in}\ \ldots$$
$$\mathit{let}\ v_n, s_n' = [\![e_n]\!]^s_{\gamma'_{n-1}}(s_n)\ \mathit{in}\ \mathit{let}\ \gamma_n' = \gamma'_{n-1}[v_n/y_n]\ \mathit{in}$$
$$\mathit{let}\ v, s' = [\![e]\!]^s_{\gamma'_n}(s)\ \mathit{in}$$
$$v, ((\gamma_n'[x_1], \ldots, \gamma_n'[x_k]), (s_1', \ldots, s_n'), s')$$

Deterministic vs. Probabilistic

Deterministic Streams

Transition function returns a pair of state and value

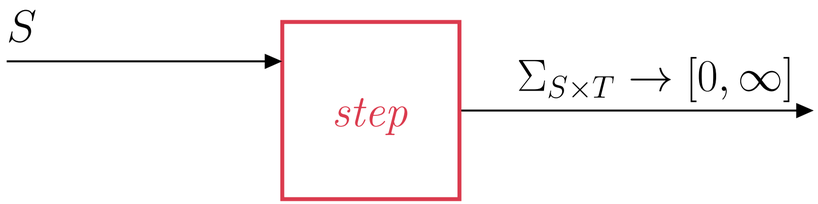
$$CoStream(T, S) = S \times (S \to S \times T)$$



Probabilistic Streams

Transition function returns a **measure** over (state, value)

$$CoPStream(T, S) = S \times (S \to \Sigma_{S \times T} \to [0, \infty])$$



Deterministic vs. Probabilistic

Measures

Maps a set of possible outcomes to a positive score: $\mu : \Sigma_X \to [0,\infty]$

Probability distributions are normalized measures, i.e., $\mu(\top) = 1$

Bernoulli distribution (discrete)

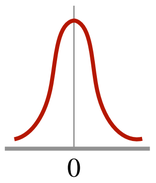
$\mathscr{B}(0.3)(\{1\}) = 0.3$

$\mathscr{B}(0.3)(\{0,1\}) = 1$

Normal distribution (continuous)

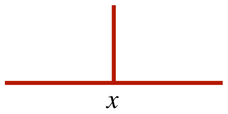
$\mathcal{N}(0,1)([0,1]) \approx 0.34$

$\mathcal{N}(0,1)((-\infty,0]) = 0.5$



Dirac delta measure

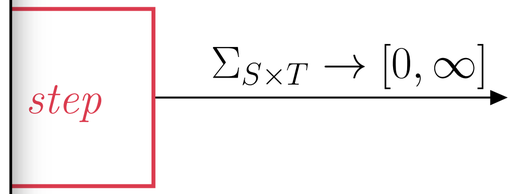
$$\delta_x(U) = \begin{cases} 1 & \text{if } x \in U \\ 0 & \text{otherwise} \end{cases}$$



ams

s a **measure** over (state, value)

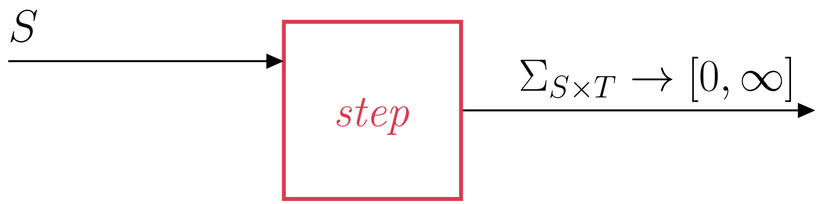
$(S \to \Sigma_{S\times T} \to [0,\infty])$



Probabilistic Semantics

Transition function returns a *measure*

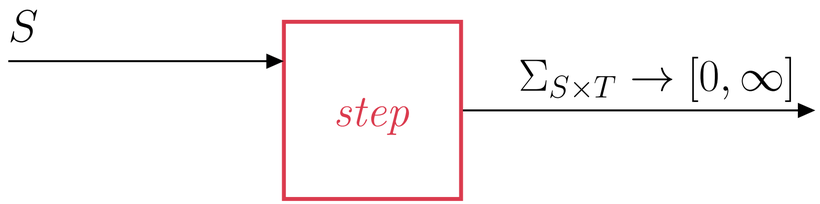
$$CoPStream(T, S) = S \times (S \to \Sigma_{S \times T} \to [0,\infty])$$



Probabilistic Semantics

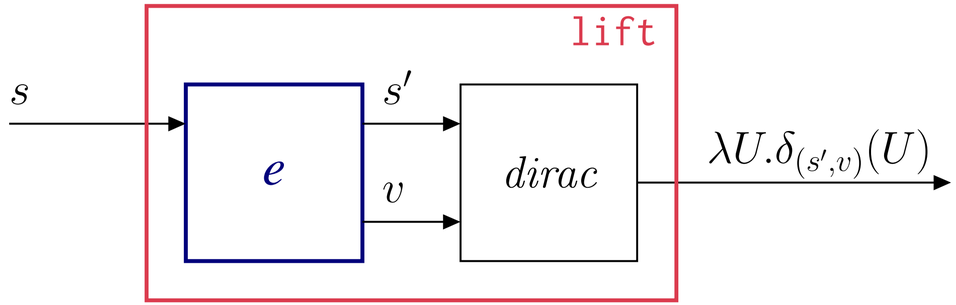
Transition function returns a *measure*

$CoPStream(T, S) = S \times (S \to \Sigma_{S \times T} \to [0,\infty])$



`lift` turns a deterministic expression into a probabilistic one

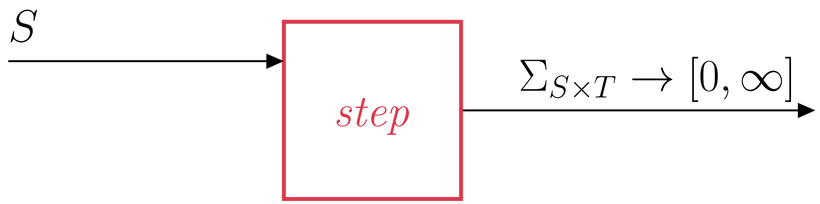
`lift:` $S \times (S \to S \times T) \to S \times (S \to \Sigma_{S \times T} \to [0,\infty])$



Probabilistic Semantics

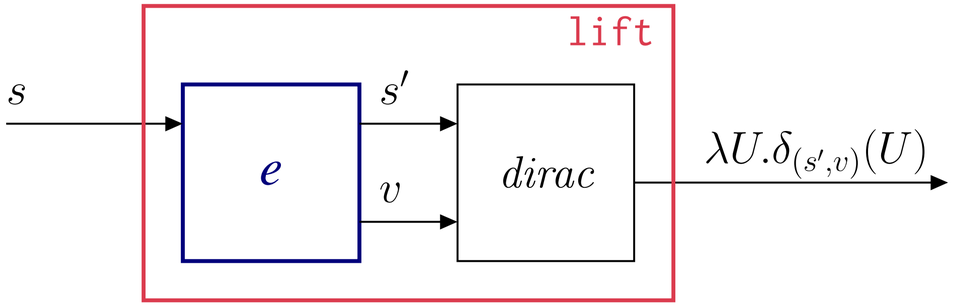
Transition function returns a *measure*

$CoPStream(T, S) = S \times (S \to \Sigma_{S \times T} \to [0,\infty])$



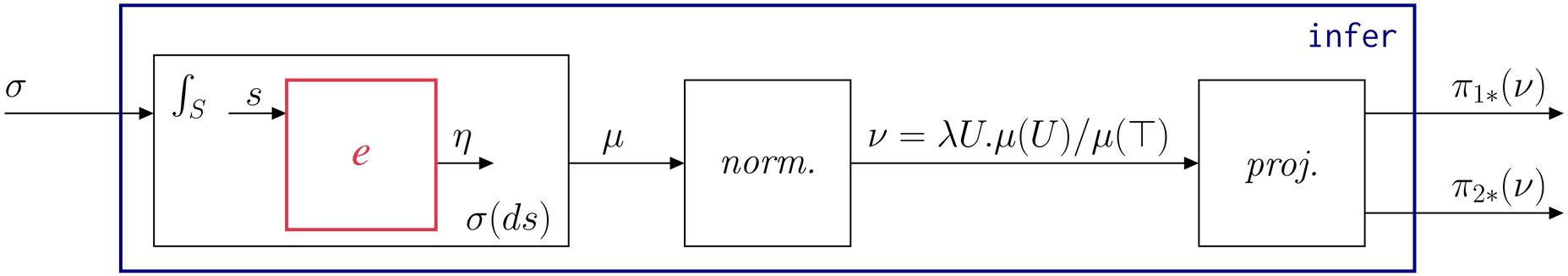
`lift` turns a deterministic expression into a probabilistic one

`lift:` $S \times (S \to S \times T) \to S \times (S \to \Sigma_{S \times T} \to [0,\infty])$

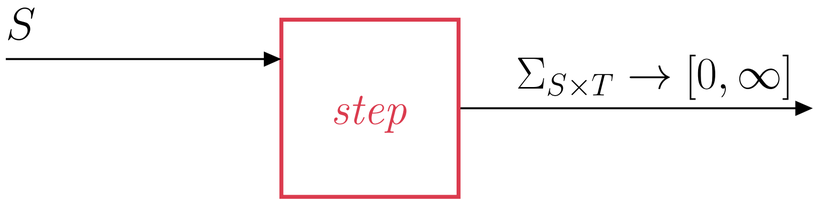


`infer` turns probabilistic expressions to a pair of distributions

`infer:` $S \times (S \to \Sigma_{S \times T} \to [0,\infty]) \to S$ `dist` $\times (S$ `dist` $\to S$ `dist` $\times T$ `dist`$)$



Semantics of Probabilistic Streams



$$
\begin{array}{lcll}
\{\!|e|\!\}^i_\gamma & = & [\![e]\!]^i_\gamma & \text{if } kindOf(e) = \mathtt{D} \\
\{\!|e|\!\}^s_\gamma & = & \lambda s.\ \lambda U.\ \delta_{[\![e]\!]^s_\gamma(s)}(U) & \text{if } kindOf(e) = \mathtt{D}
\end{array}
$$

$$
\begin{array}{lcl}
\{\!|\mathtt{sample}(e)|\!\}^i_\gamma & = & [\![e]\!]^i_\gamma \\
\{\!|\mathtt{sample}(e)|\!\}^s_\gamma & = & \lambda s.\ \lambda U.\ let\ \mu, s' = [\![e]\!]^s_\gamma(s)\ in\ \int_T \mu(dv)\ \delta_{v,s'}(U)
\end{array}
$$

$$
\begin{array}{lcl}
\{\!|\mathtt{observe}(e_1,\ e_2)|\!\}^i_\gamma & = & ([\![e_1]\!]^i_\gamma, [\![e_2]\!]^i_\gamma) \\
\{\!|\mathtt{observe}(e_1,\ e_2)|\!\}^s_\gamma & = & \\
\quad \lambda(s_1, s_2).\ \lambda U. & & \\
\quad\quad let\ \mu, s_1' = [\![e_1]\!]^s_\gamma(s_1)\ in & & \\
\quad\quad let\ v, s_2' = [\![e_2]\!]^s_\gamma(s_2)\ in\ \mu_{\text{pdf}}(v) * \delta_{(),(s_1',s_2')}(U) &&
\end{array}
$$

$$
\left\{\!\!\left|
\begin{array}{l}
e \ \mathtt{where\ rec\ init}\ x_1 = c_1\ \ldots\ \mathtt{and\ init}\ x_k = c_k \\
\qquad\qquad \mathtt{and}\ y_1 = e_1\ \ldots\ \mathtt{and}\ y_n = e_n
\end{array}
\right|\!\!\right\}^s_\gamma =
$$

$$
\begin{array}{l}
\lambda((m_1, \ldots, m_k), (s_1, \ldots, s_n), s).\ \lambda U. \\
\quad let\ \gamma_1 = \gamma[m_1/x_1\_\mathtt{last}]\ in\ \ldots let\ \gamma_k = \gamma_{k-1}[m_k/x_k\_\mathtt{last}]\ in \\
\quad let\ \mu_1 = \{\!|e_1|\!\}^s_{\gamma_k}(s_1)\ in \\
\quad \int \mu_1(dv_1, ds_1') let\ \gamma_1' = \gamma_k[v_1/y_1]\ in\ \ldots \\
\quad\quad let\ \mu_n = [\![e_n]\!]^s_{\gamma_{n-1}'}(s_n)\ in \\
\quad\quad \int \mu_n(dv_n, ds_n') let\ \gamma_n' = \gamma_{n-1}'[v_n/y_n]\ in \\
\quad\quad\quad let\ \mu = \{\!|e|\!\}^s_{\gamma_n'}(s)\ in \\
\quad\quad\quad \int \mu(dv, ds')\ \delta_{v,((\gamma_n'[x_1],\ldots,\gamma_n'[x_k]),(s_1',\ldots,s_n'),s')}(U)
\end{array}
$$

Compilation

Target

μF a first-order functional probabilistic language

$$
\begin{aligned}
d &::= \texttt{let}\ f\ \texttt{=}\ e \mid d\ d \\
e &::= c \mid x \mid (e, e) \mid op(e) \mid e(e) \\
&\quad \mid \texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e \mid \texttt{let}\ p\ \texttt{=}\ e\ \texttt{in}\ e \mid \texttt{fun}\ p\ \texttt{->}\ e \\
&\quad \mid \texttt{sample}(e) \mid \texttt{observe}(e,\ e) \mid \texttt{infer}((\texttt{fun}\ x\ \texttt{->}\ e),\ e) \\
p &::= x \mid (p,\ p)
\end{aligned}
$$

Semantics

$$
\begin{aligned}
\{\!|\texttt{let}\ f\ \texttt{=}\ e|\!\}_\gamma &= \gamma[\{\!|e|\!\}_\gamma / f] \\
\{\!|d_1\ d_2|\!\}_\gamma &= \mathit{let}\ \gamma_1 = \{\!|d_1|\!\}_\gamma\ \mathit{in}\ \{\!|d_2|\!\}_{\gamma_1} \\
\{\!|e|\!\}_\gamma &= \lambda U.\ \delta_{[\![e]\!]_\gamma}(U)\ \mathit{if}\ \mathit{kindOf}(e) = \mathtt{D} \\
\{\!|e_1(e_2)|\!\}_\gamma &= \lambda U.\ ([\![e_1]\!]_\gamma([\![e_2]\!]_\gamma))(U) \\
\{\!|\texttt{let}\ p\ \texttt{=}\ e_1\ \texttt{in}\ e_2|\!\}_\gamma &= \lambda U.\ \textstyle\int_T \{\!|e_1|\!\}_\gamma(du)\{\!|e_2|\!\}_{\gamma+[u/p]}(U) \\
\{\!|\texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2|\!\}_\gamma &= \\
&\quad \lambda U.\ \mathit{if}\ [\![e]\!]_\gamma\ \mathit{then}\ \{\!|e_1|\!\}_\gamma(U)\ \mathit{else}\ \{\!|e_2|\!\}_\gamma(U) \\
\{\!|\texttt{fun}\ p\ \texttt{->}\ e|\!\}_\gamma &= \lambda v.\ \{\!|e|\!\}_{[v/p]} \\
\{\!|\texttt{sample}(e)|\!\}_\gamma &= \lambda U.\ [\![e]\!]_\gamma(U) \\
\{\!|\texttt{observe}(e_1,\ e_2)|\!\}_\gamma &= \\
&\quad \lambda U.\ \mathit{let}\ \mu = [\![e_1]\!]_\gamma\ \mathit{in}\ \mu_{\mathrm{pdf}}([\![e_2]\!]_\gamma) * \delta_{()}(U) \\
\{\!|\texttt{factor}(e)|\!\}_\gamma &= \lambda U.\ \exp([\![e]\!]_\gamma) * \delta_{()}(U) \\
[\![\texttt{infer}(\texttt{fun}\ x\ \texttt{->}\ e_1,\ e_2)]\!]_\gamma &= \\
&\quad \mathit{let}\ \sigma = [\![e_2]\!]_\gamma\ \mathit{in} \\
&\quad \mathit{let}\ \mu = \lambda U.\ \textstyle\int_S \sigma(ds)\{\!|e|\!\}^s_{\gamma+[s/x]}(U)\ \mathit{in} \\
&\quad \mathit{let}\ \nu = \lambda U.\ \mu(U)/\mu(\top)\ \mathit{in} \\
&\quad (\pi_{1*}(\nu), \pi_{2*}(\nu))
\end{aligned}
$$

Compilation: Allocation function

$\mathcal{A}(c)$ = ()

$\mathcal{A}(x)$ = ()

$\mathcal{A}(\texttt{last } x)$ = ()

$\mathcal{A}((e_1, e_2)) = (\mathcal{A}(e_1), \mathcal{A}(e_2))$

$\mathcal{A}(e \texttt{ where}$
  $\texttt{rec init } x_1 = c_1 \ \ldots$
  $\texttt{and init } x_k = c_k$
  $\texttt{and } y_1 = e_1 \ \ldots$
  $\texttt{and } y_n = e_n) =$
    $((c_1, \ldots, c_k),$
     $(\mathcal{A}(e_1), \ldots, \mathcal{A}(e_n)),$
     $\mathcal{A}(e))$

$\mathcal{A}(\texttt{present } e \texttt{ -> } e_1 \texttt{ else } e_2) = (\mathcal{A}(e), \mathcal{A}(e_1), \mathcal{A}(e_2))$

$\mathcal{A}(\texttt{reset } e_1 \texttt{ every } e2) =$
  $(\mathcal{A}(e_1), \mathcal{A}(e_1), \mathcal{A}(e_2))$

$\mathcal{A}(op(e)) = \mathcal{A}(e)$

$\mathcal{A}(f(e)) = (f\_\texttt{init}, \mathcal{A}(e))$

$\mathcal{A}(\texttt{sample}(e)) = \mathcal{A}(e)$

$\mathcal{A}(\texttt{factor}(e)) = \mathcal{A}(e)$

$\mathcal{A}(\texttt{observe}(e_1, e_2)) =$
  $(\mathcal{A}(e_1), \mathcal{A}(e_2))$

$\mathcal{A}(\texttt{infer}(e)) = (\mathcal{A}(e))$

Compilation: Transition Function

```
C(let node f x = e) =
  let f_init = A(e)
  let f_step =
    fun (s,x) -> C(e)(s)

C(d1 d2) = C(d1) C(d2)
C(c) = fun s -> (c, s)
C(x) = fun s -> (x, s)
C(last x) = fun s -> (x_last, s)

C((e1, e2)) = fun (s1,s2) ->
  let v1,s1' = C(e1)(s1) in
  let v2,s2' = C(e2)(s2) in
  ((v1,v2), (s1',s2'))

C(op(e)) = fun s ->
  let v,s' = C(e)(s) in
  (op(v), s')

C(f(e)) = fun (s1,s2) ->
  let v1,s1' = C(e)(s1) in
  let v2,s2' = f_step(s2,v) in
  (v2, (s1',s2'))
```

```
C(e where
  rec init x1 = c1 ... and init xk = ck
  and y1 = e1 ... and yn = en) =
fun ((m1,...,mk),(s1, ...,sn),s) ->
  let x1_last = m1 in ...
  let xk_last = mk in
  let y1, s1' = C(e1)(s1) in
  let yn, sn' = C(en)(sn) in
  let v,s' = C(e)(s) in
  (v, (s1', ..., sn'), s')

C(present e -> e1 else e2) =
fun (s,s1,s2) ->
  let v, s' = C(e)(s) in
  if v then let v1,s1' = C(e1)(s1) in
    (v1, (s',s1',s2))
  else let v2,s2' = C(e2)(s2) in
    (v2, (s',s1,s2'))

C(reset e1 every e2) =
fun (s0,s1,s2) ->
  let v2,s2' = C(e2)(s2) in
  let s = if v2 then s0 else s1 in
  let v1,s1' = C(e1)(s) in
  (v1, (s0,s1',s2'))
```

```
C(sample(e)) = fun s ->
  let mu,s' = C(e)(s) in
  let v = sample(mu) in (v, s')

C(observe(e1, e2)) = fun (s1,s2) ->
  let v1,s1' = C(e1)(s1) in
  let v2,s2' = C(e2)(s2) in
  let _ = observe(v1,v2) in
  ((), (s1',s2'))

C(factor(e)) = fun s ->
  let v,s' = C(e)(s) in
  let _ = factor(v) in ((), s')

C(infer(e)) = fun sigma ->
  let mu,sigma' = infer(C(e), sigma) in
  (mu, sigma')

C(let proba f x = e) =
  let f_init = A(e)
  let f_step = fun (s,x) -> C(e)(s)
```

Semantics equivalence

Theorem. For all ProbZelus expression e, for all state s and environment γ:

- if $kindOf(e) = \texttt{D}$ then $[\![e]\!]^s_\gamma(s) = [\![C(e)]\!]_\gamma(s)$, and,
- if $kindOf(e) = \texttt{P}$ then $\{\!|e|\!\}^s_\gamma(s) = \{\!|C(e)|\!\}_\gamma(s)$.

Proof. By induction on the structure of expressions.

$\{\!|C(\texttt{sample}(e))|\!\}_\gamma(s)$

$$= \left\{\!\!\left| \begin{array}{l} \texttt{fun s -> let mu, s' = } C(e)\texttt{(s) in} \\ \qquad\qquad \texttt{let v = sample(mu) in (v, s')} \end{array} \right|\!\!\right\}^{(s)}_{\gamma}$$

$$= \lambda U.\ \int \delta_{[\![C(e)]\!]_\gamma(s)}(d\mu, ds')\ \{\!|\texttt{let v = sample(mu) in (v, s')}|\!\}_{\gamma[\mu/\texttt{mu},\ s'/\texttt{s'}]}(U)$$

$$= \lambda U.\ let\ (\mu, s) = [\![C(e)]\!]_\gamma(s)\ in\ \{\!|\texttt{let v = sample(mu) in (v, s')}|\!\}_{\gamma[\mu/\texttt{mu},\ s'/\texttt{s'}]}(U)$$

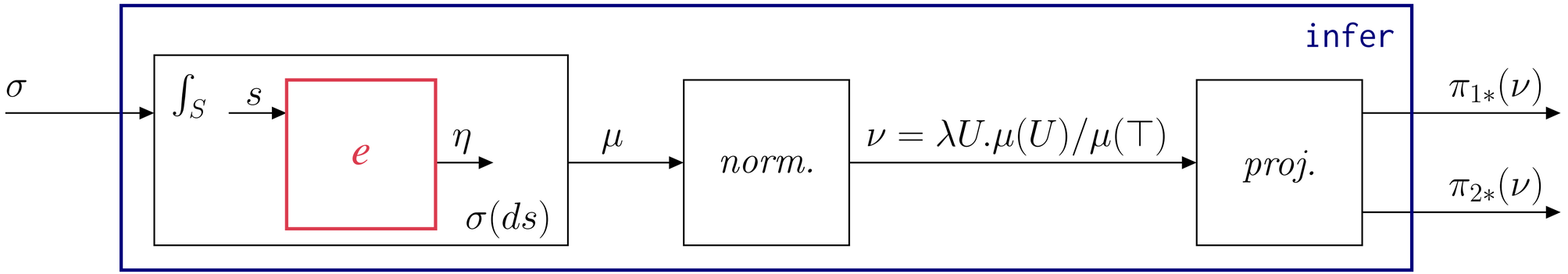
$$= \lambda U.\ let\ (\mu, s') = [\![e]\!]^s_\gamma(s)\ in\ \int \mu(dv)\delta_{v,s'}(U)$$

$$= \{\!|\texttt{sample}(e)|\!\}^s_\gamma(s)$$

□

Inference

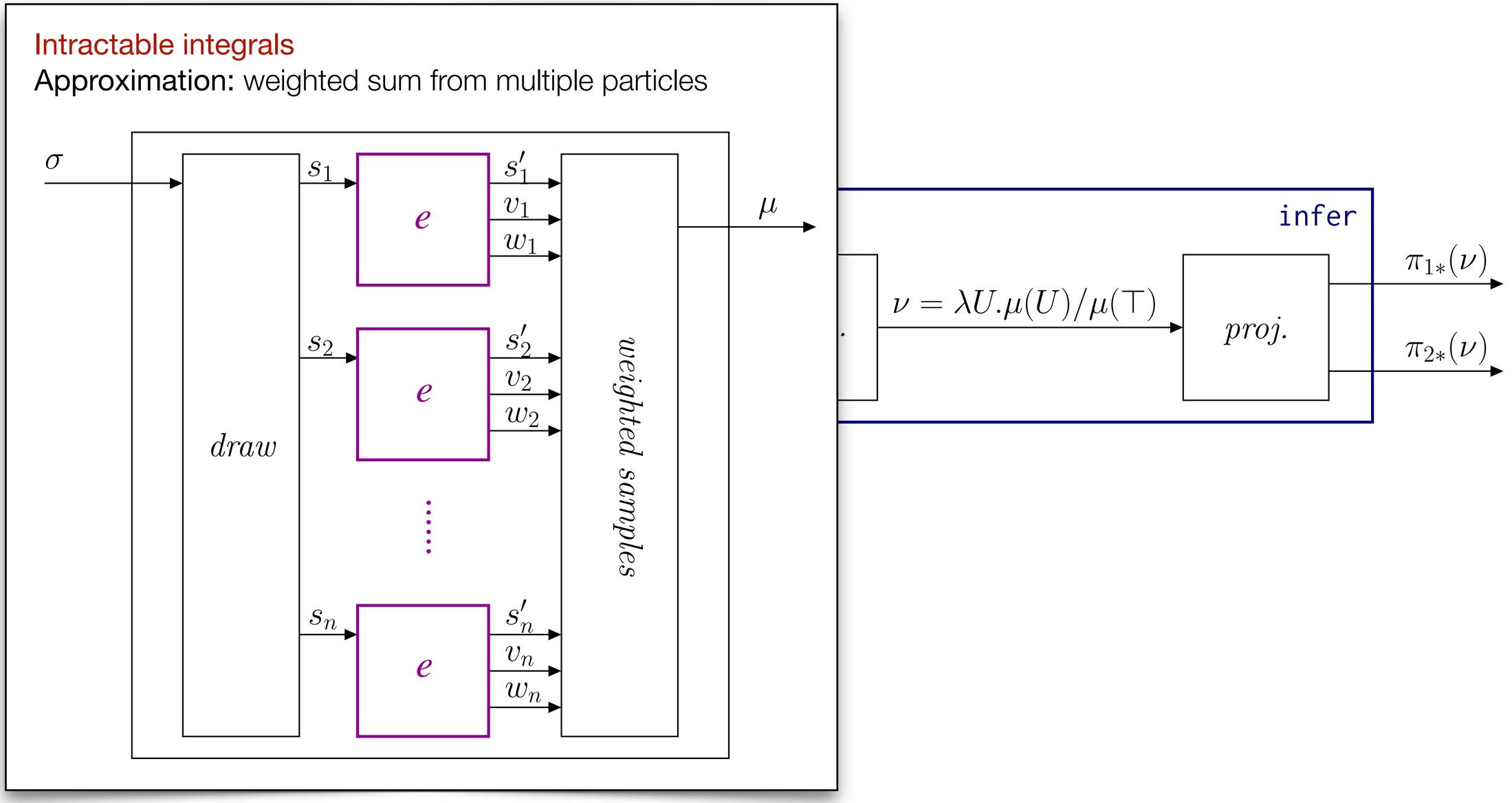
Particle Filtering



Particle Filtering

Intractable integrals

Approximation: weighted sum from multiple particles



Particle Filtering

```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

Particle Filtering

```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

$t = 0$

Particle Filtering

```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

$t = 0$

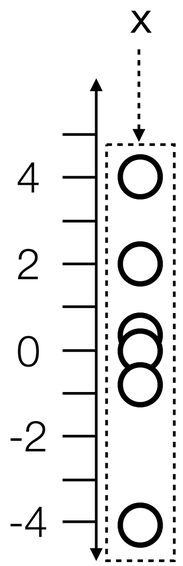
```
sample (gaussian (0, 10))
```

Particle Filtering

```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10)) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

$t = 0$

```
sample (gaussian (0, 10))
```

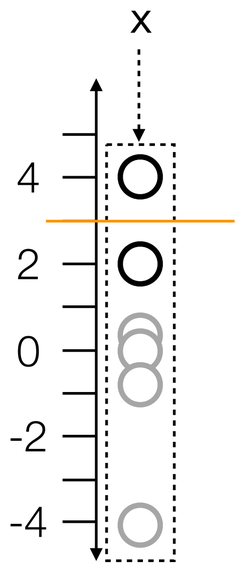


Particle Filtering

```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

$t = 0$

```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```

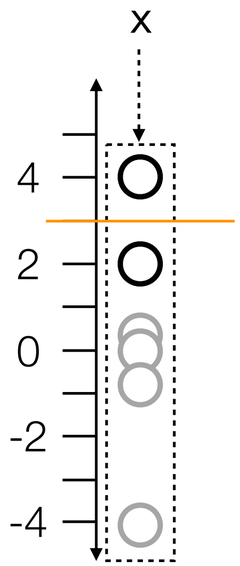


Particle Filtering

```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

$t = 0$

```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```



$t = 1$

Particle Filtering

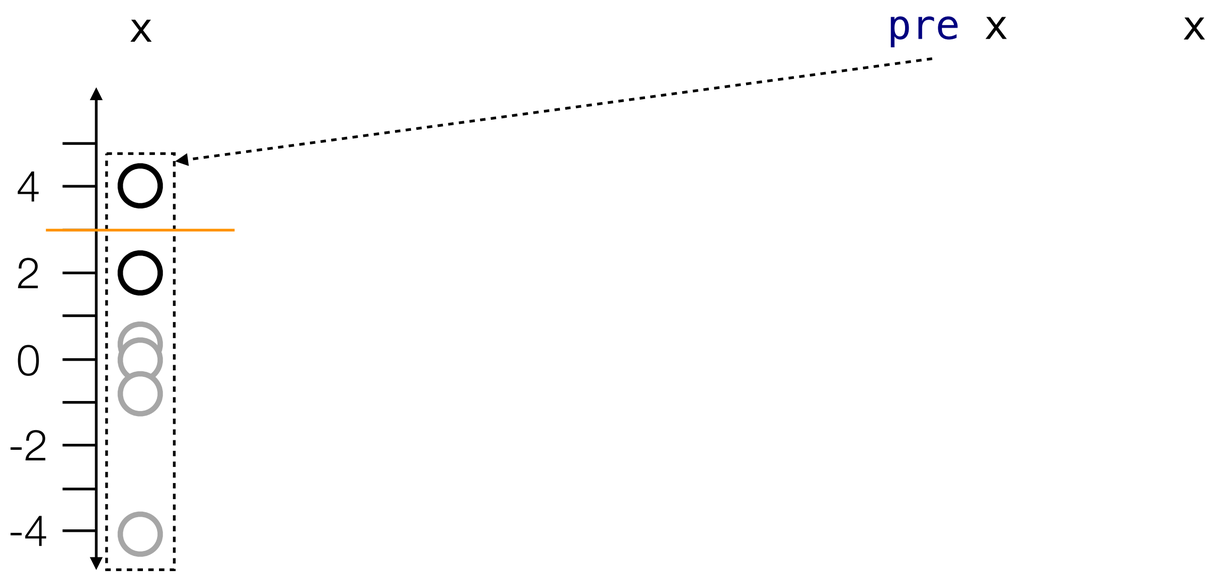
```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

$t = 0$

```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```

$t = 1$

```
sample (gaussian (pre x, 1))
```



Particle Filtering

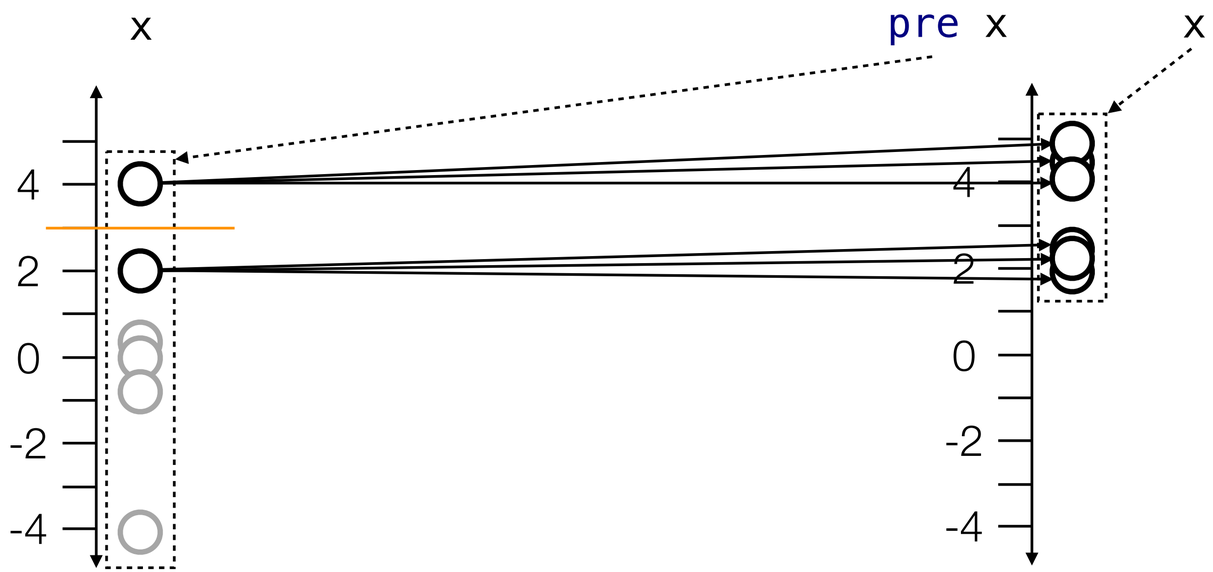
```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

$t = 0$

```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```

$t = 1$

```
sample (gaussian (pre x, 1))
```



Particle Filtering

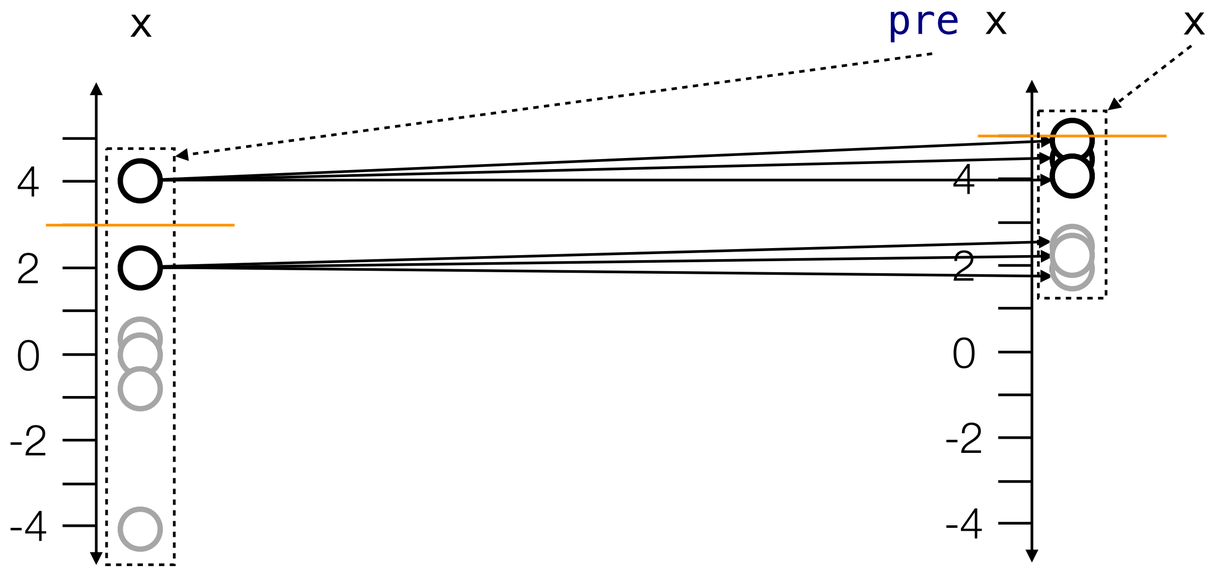
```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

$t = 0$

```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```

$t = 1$

```
sample (gaussian (pre x, 1))
observe (gaussian (x, 1), 5)
```



Particle Filtering

```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

$t = 0$

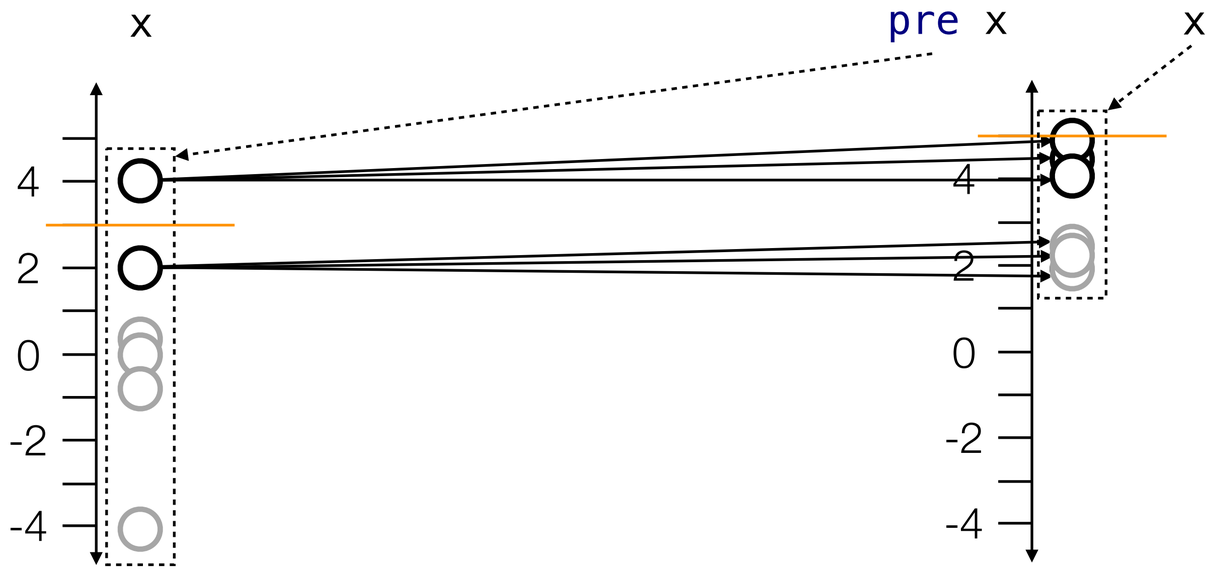
```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```

$t = 1$

```
sample (gaussian (pre x, 1))
observe (gaussian (x, 1), 5)
```

$t = 2$

```
sample (gaussian (pre x, 1))
observe (gaussian (x, 1), …)
```



...

Particles Filtering Semantics

$$[\![\texttt{let } f \texttt{ = } e]\!]_\gamma = \gamma[\{\!|e|\!\}_{\gamma,1}/f] \quad \textit{if } kindOf(e) = \texttt{P}$$

$$\{\!|e|\!\}_{\gamma,w} = ([\![e]\!]_\gamma, w) \quad \textit{if } kindOf(e) = \texttt{D}$$

$$\{\!|e_1(e_2)|\!\}_{\gamma,w} = \textit{let } v_2 = [\![e_2]\!]_\gamma \textit{ in } [\![e_1]\!]_\gamma(v_2, w)$$

$$\{\!|\texttt{if } e \texttt{ then } e_1 \texttt{ else } e_2|\!\}_{\gamma,w} = \textit{if } [\![e]\!]_\gamma \textit{ then } \{\!|e_1|\!\}_{\gamma,w} \textit{ else } \{\!|e_2|\!\}_{\gamma,w}$$

$$\{\!|\texttt{let } p \texttt{ = } e_1 \texttt{ in } e_2|\!\}_{\gamma,w} = \textit{let } v_1, w_1 = \{\!|e_1|\!\}_{\gamma,w} \textit{ in } \{\!|e_2|\!\}_{\gamma[v_1/p],w_1}$$

$$\{\!|\texttt{fun } p \texttt{ -> } e|\!\}_{\gamma,w} = \textit{let } f = \lambda(v, w'). \{\!|e|\!\}_{[v/p],w'} \textit{ in } (f, w)$$

$$\{\!|\texttt{sample(}e\texttt{)}|\!\}_{\gamma,w} = (\mathrm{draw}([\![e]\!]_\gamma), w)$$

$$\{\!|\texttt{factor(}e\texttt{)}|\!\}_{\gamma,w} = ((), w * \exp([\![e]\!]_\gamma))$$

$$\{\!|\texttt{observe(}e_1\texttt{,}e_2\texttt{)}|\!\}_{\gamma,w} = \textit{let } \mu = [\![e_1]\!]_\gamma \textit{ in } ((), w * \mu_{\mathrm{pdf}}([\![e_2]\!]_\gamma))$$

$$[\![\texttt{infer(fun } s \texttt{ -> } e\texttt{, } \sigma\texttt{)}]\!]_\gamma = \textit{let } \mu = \lambda U. \sum_{i=1}^{N} \textit{let } s_i = \mathrm{draw}([\![\sigma]\!]_\gamma) \textit{ in } \textit{let } (v_i, s_i'), w_i = \{\!|\texttt{fun } s \texttt{ -> } e|\!\}_{\gamma,1}(s_i) \textit{ in } \overline{w_i} * \delta_{v_i,s_i'}(U) \textit{ in } (\pi_{1*}(\mu), \pi_{2*}(\mu))$$

$$\overline{w_i} = w_i / \sum_{i=1}^{N} w_i$$

Delayed Sampling

Particles Filters can impractical

- Require lot of computing power
- Poor approximation

Exact inference is often impossible

Semi-Symbolic inference

- Perform as much exact computation as possible
- Fall back to a Particle Filter when symbolic computation fails

Delayed Sampling

```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```


Murray et al. 2018

Delayed Sampling

```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

$t = 0$

Murray et al. 2018

Delayed Sampling

```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

$t = 0$

```
sample (gaussian (0, 10))
```

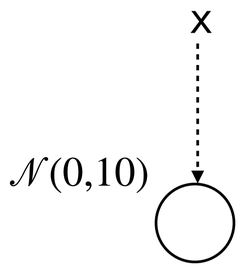

Murray et al. 2018

Delayed Sampling

```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

$t = 0$

```
sample (gaussian (0, 10))
```



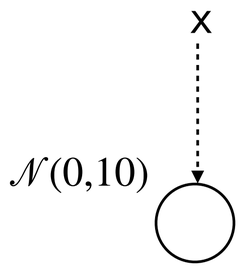
Murray et al. 2018

Delayed Sampling

```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

$t = 0$

```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```



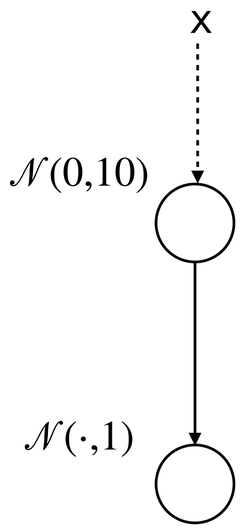
Murray et al. 2018

Delayed Sampling

```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

$t = 0$

```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```



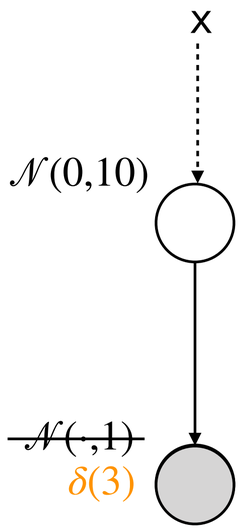
Murray et al. 2018

Delayed Sampling

```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

$t = 0$

```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```



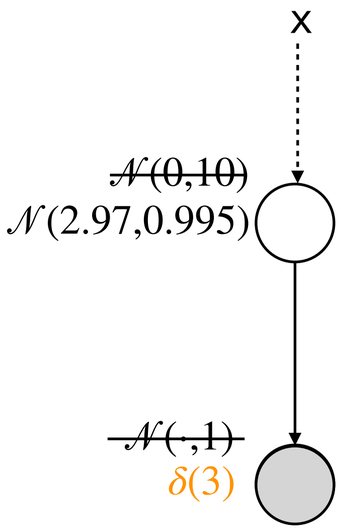
Murray et al. 2018

Delayed Sampling

```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

$t = 0$

```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```



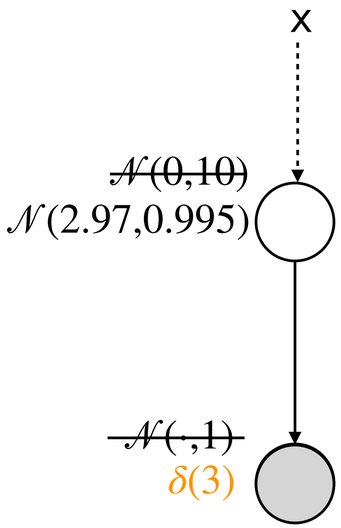
Murray et al. 2018

Delayed Sampling

```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

$t = 0$

```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```



$t = 1$

```
sample (gaussian (pre x, 1))
```


Murray et al. 2018

Delayed Sampling

```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

$t = 0$

```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```

x

$\mathcal{N}(0,10)$

$\mathcal{N}(2.97,0.995)$

$\mathcal{N}(\cdot,1)$

$\delta(3)$

$t = 1$

```
sample (gaussian (pre x, 1))
```

pre x

x

Murray et al. 2018

Delayed Sampling

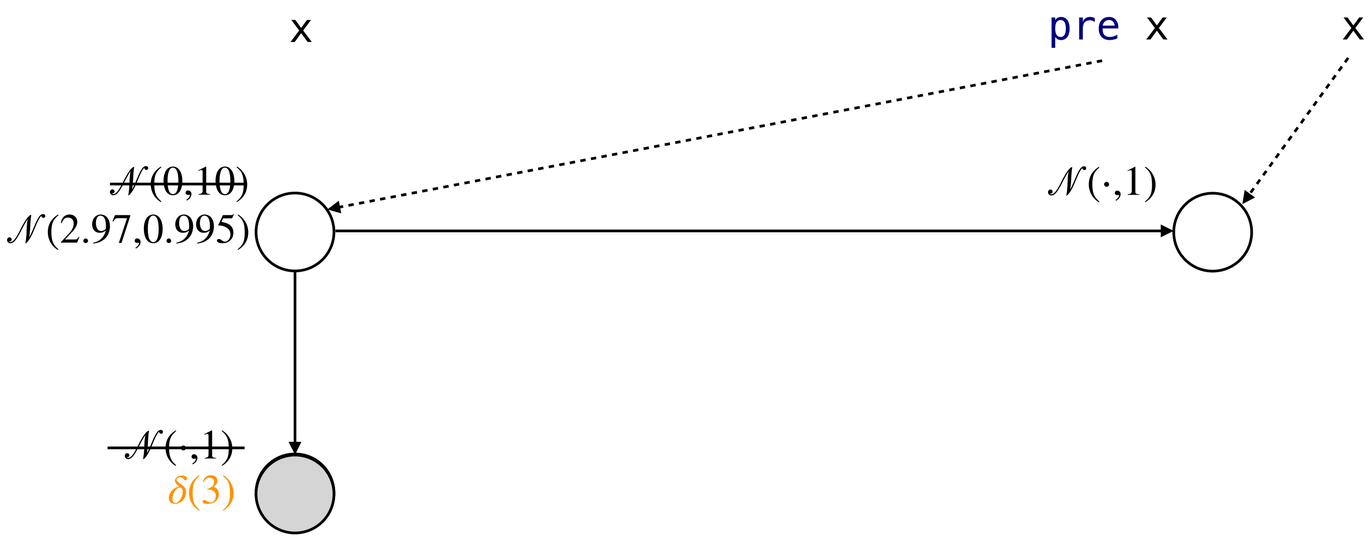
```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

$t = 0$

```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```

$t = 1$

```
sample (gaussian (pre x, 1))
```



Murray et al. 2018

Delayed Sampling

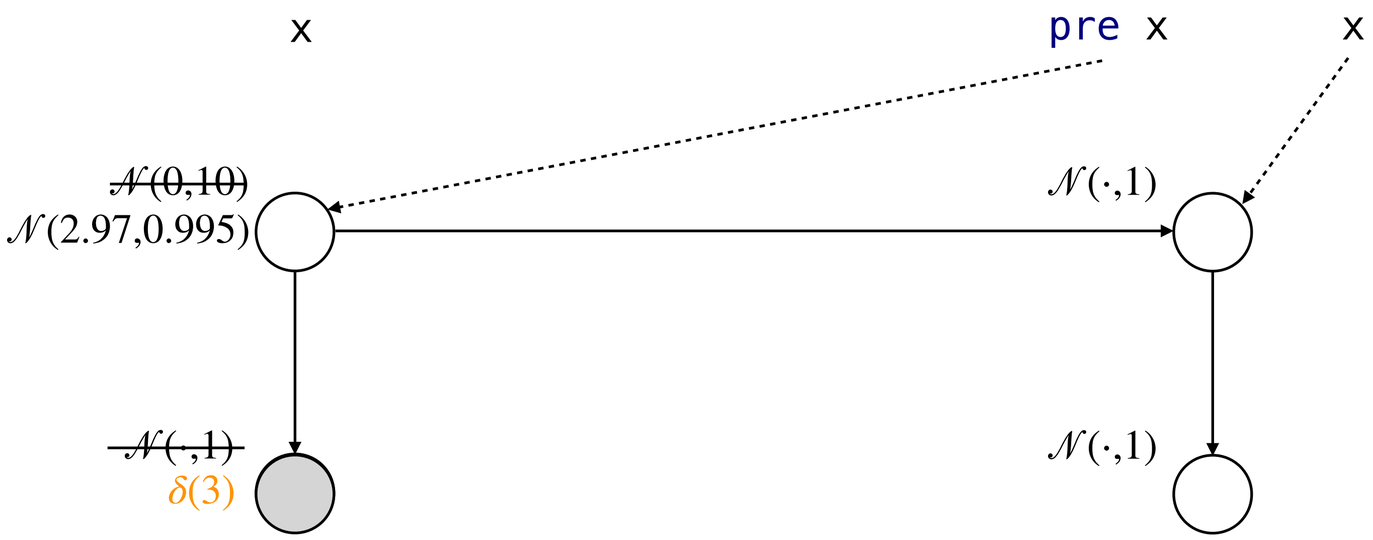
```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

$t = 0$

```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```

$t = 1$

```
sample (gaussian (pre x, 1))
observe (gaussian (x, 1), 5)
```



Murray et al. 2018

Delayed Sampling

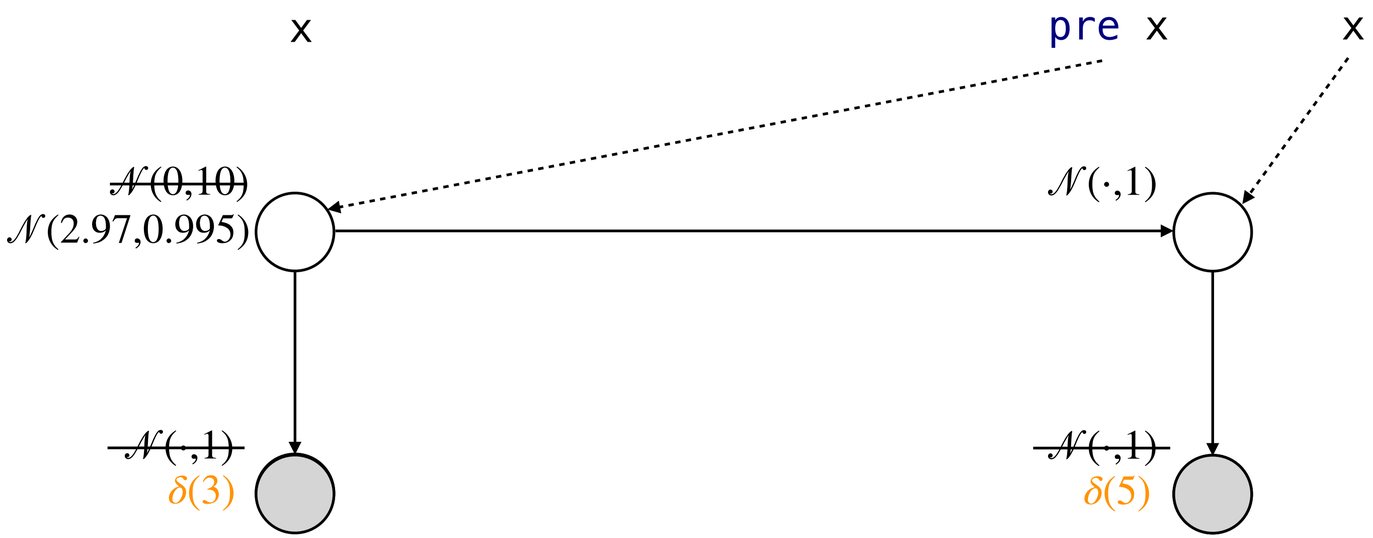
```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

$t = 0$

```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```

$t = 1$

```
sample (gaussian (pre x, 1))
observe (gaussian (x, 1), 5)
```



Murray et al. 2018

Delayed Sampling

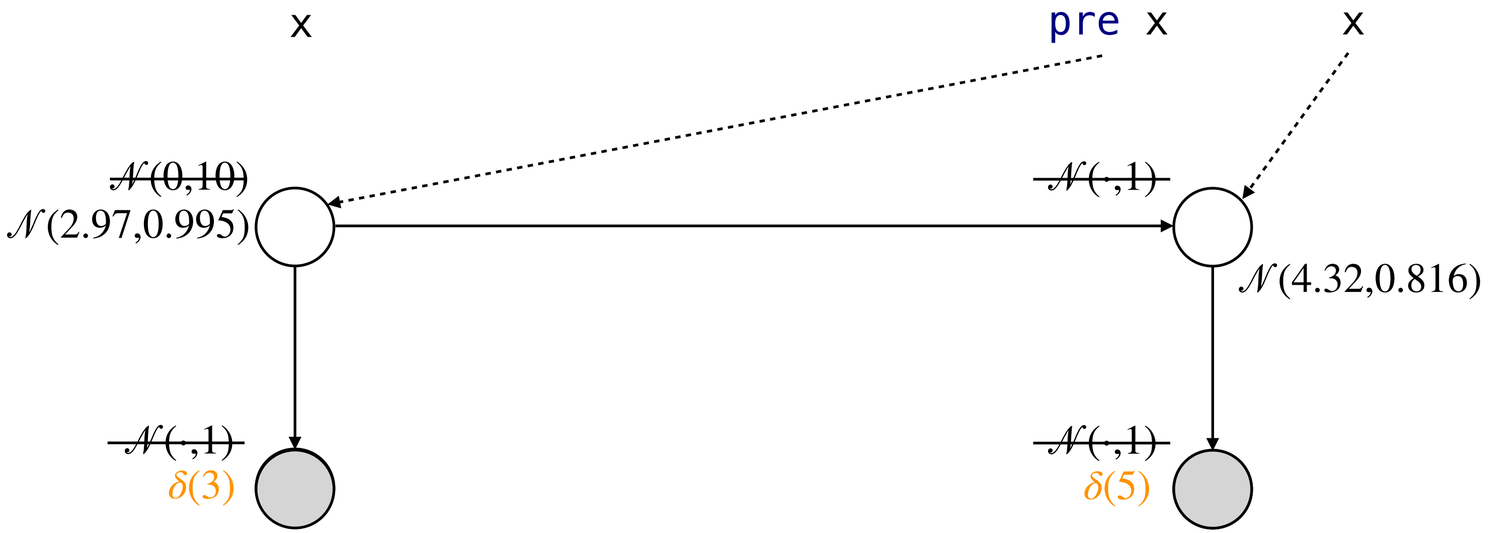
```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

$t = 0$

```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```

$t = 1$

```
sample (gaussian (pre x, 1))
observe (gaussian (x, 1), 5)
```



Murray et al. 2018

Delayed Sampling

```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

$t = 0$

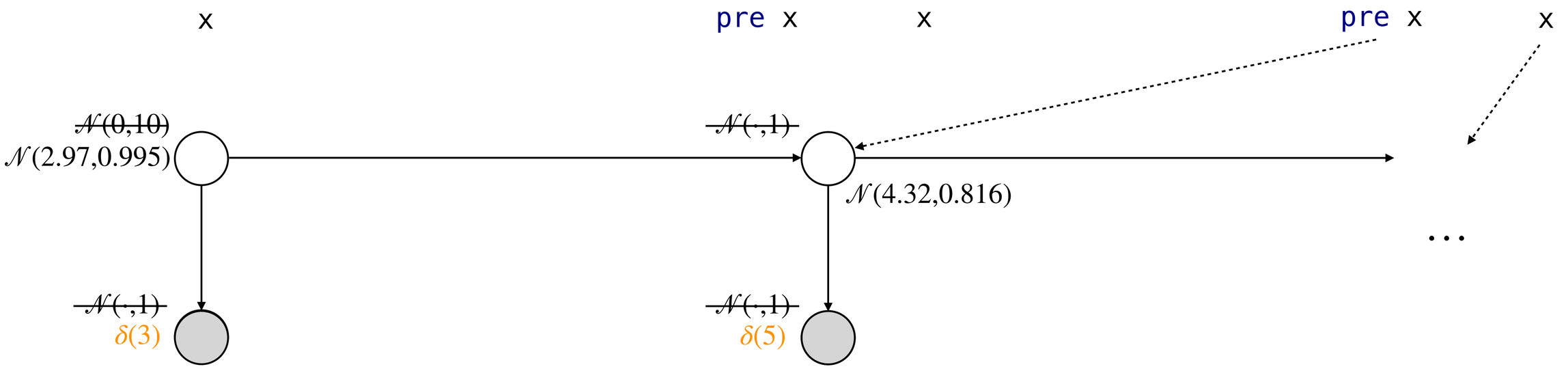
```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```

$t = 1$

```
sample (gaussian (pre x, 1))
observe (gaussian (x, 1), 5)
```

$t = 2$

```
sample (gaussian (pre x, 1))
observe (gaussian (x, 1), …)
```



Murray et al. 2018

Delayed Sampling

```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

$t = 0$

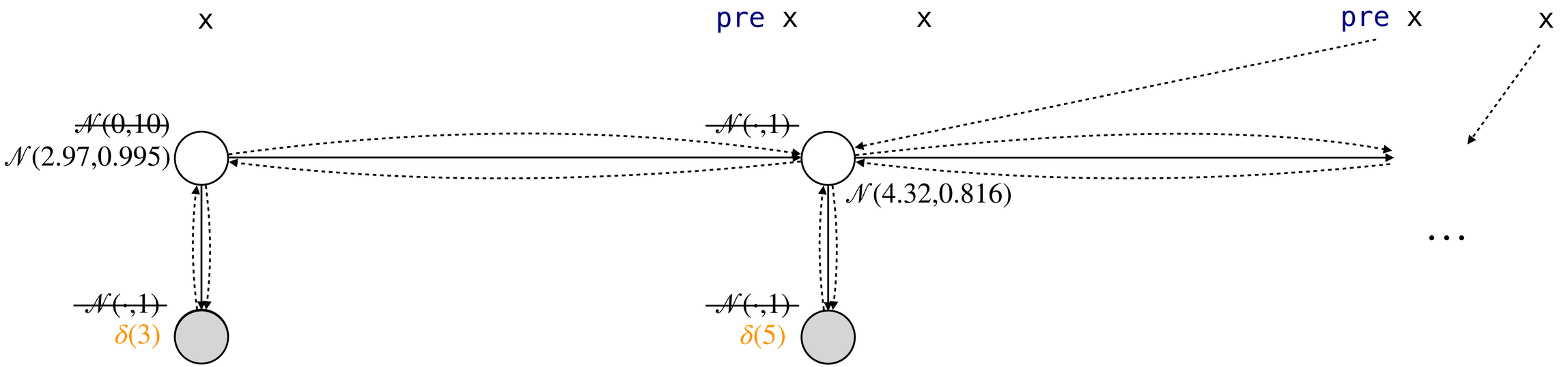
```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```

$t = 1$

```
sample (gaussian (pre x, 1))
observe (gaussian (x, 1), 5)
```

$t = 2$

```
sample (gaussian (pre x, 1))
observe (gaussian (x, 1), …)
```



Murray et al. 2018

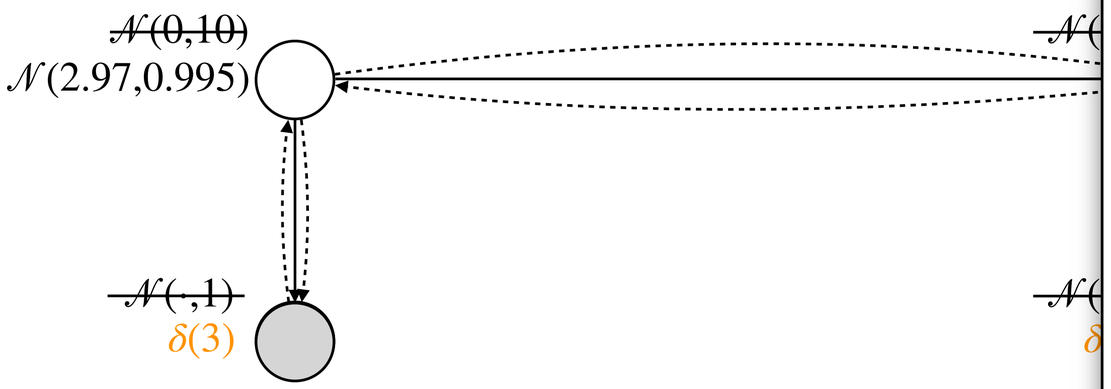
Delayed Sampling

```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

$t = 0$

```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```

x

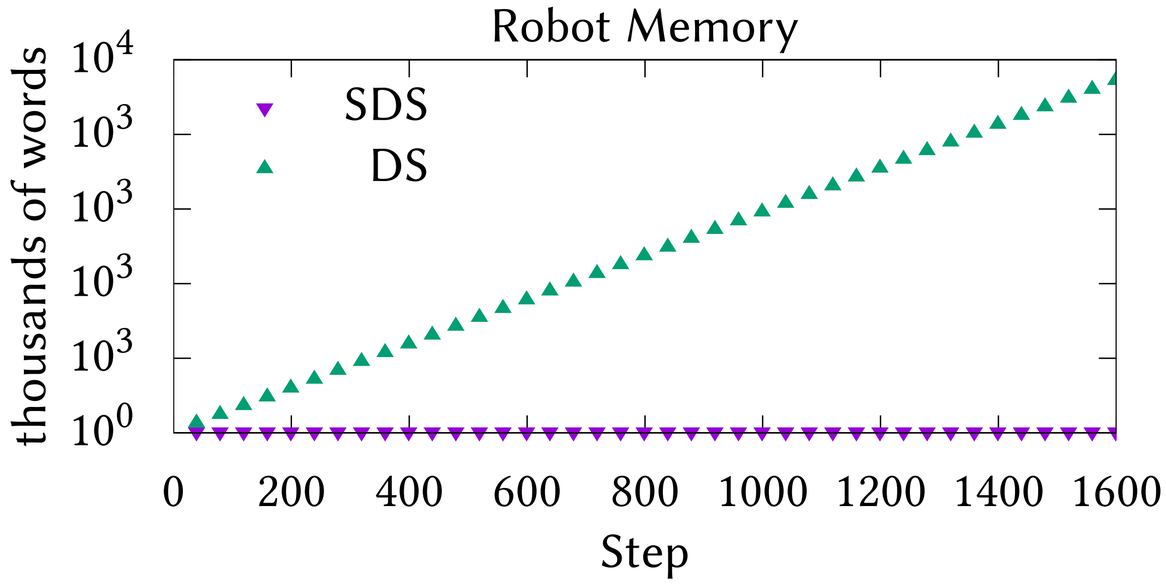


$t = 1$

```
sample (gau
observe (ga
pr
```

$t = 2$

Unbounded resources



Murray et al. 2018

Streaming Delayed Sampling

```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

$t = 0$

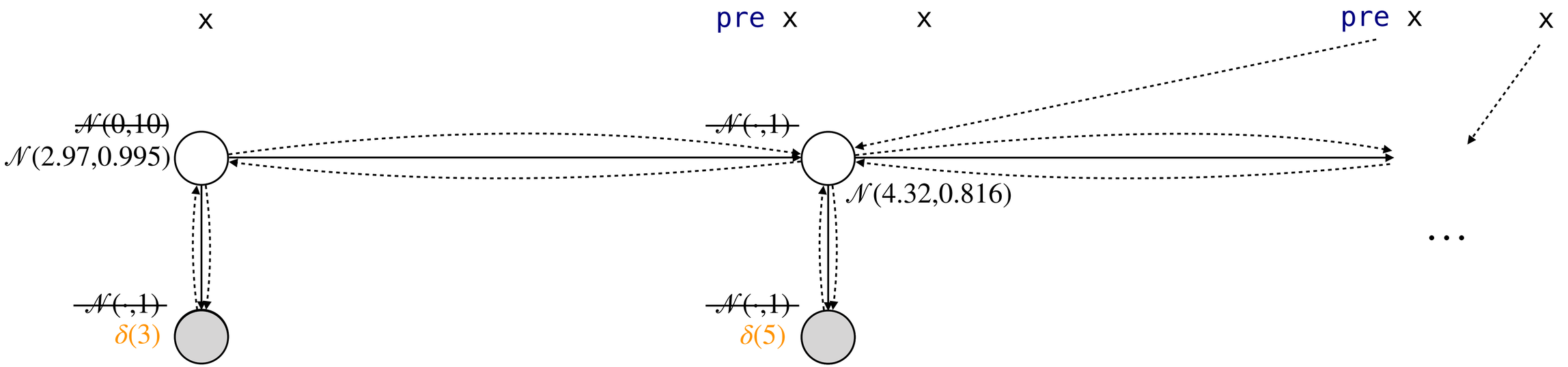
```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```

$t = 1$

```
sample (gaussian (pre x, 1))
observe (gaussian (x, 1), 5)
```

$t = 2$

```
sample (gaussian (pre x, 1))
observe (gaussian (x, 1), …)
```



Streaming Delayed Sampling

```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

$t = 0$

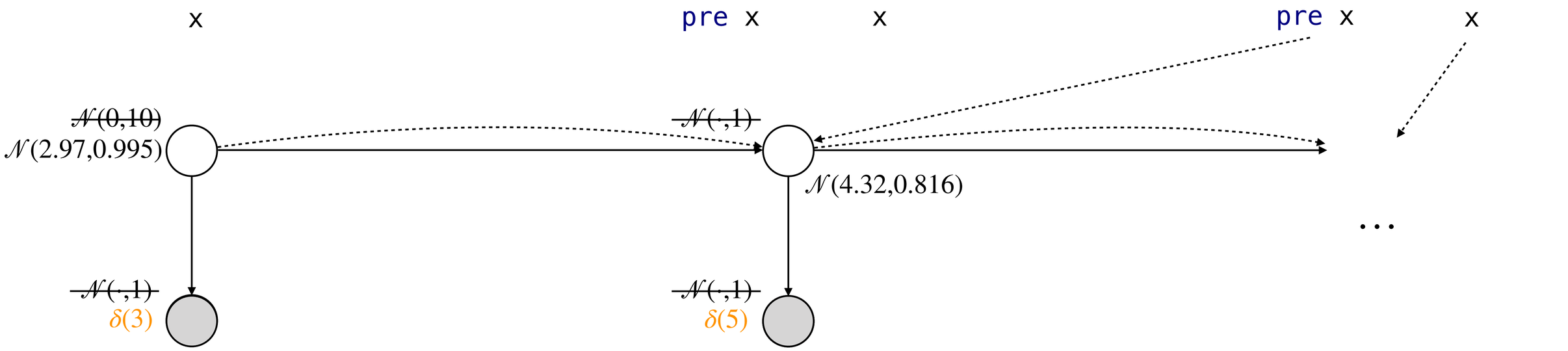
```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```

$t = 1$

```
sample (gaussian (pre x, 1))
observe (gaussian (x, 1), 5)
```

$t = 2$

```
sample (gaussian (pre x, 1))
observe (gaussian (x, 1), …)
```



Streaming Delayed Sampling

```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

$t = 0$

```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```

x

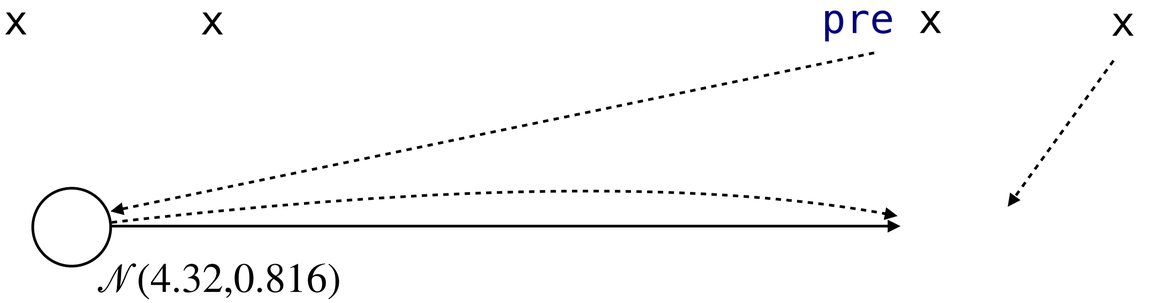
$t = 1$

```
sample (gaussian (pre x, 1))
observe (gaussian (x, 1), 5)
```

pre x x

$t = 2$

```
sample (gaussian (pre x, 1))
observe (gaussian (x, 1), …)
```



Delayed Sampling Semantics

$$\{\!|op(e)|\!\}_{\gamma,g,w} =$$
$$\quad let\ (e', g_e, w_e) = \{\!|e|\!\}_{\gamma,g,w}\ in\ (\mathsf{app}(op, e'), g_e, w_e)$$

$$\{\!|\texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2|\!\}_{\gamma,g,w} =$$
$$\quad let\ e', g_e, w_e = \{\!|e|\!\}_{\gamma,g,w}\ in$$
$$\quad let\ v, g_v = value(e', g_e)\ in$$
$$\quad if\ v\ then\ \{\!|e_1|\!\}_{\gamma,g_v,w_e}\ else\ \{\!|e_2|\!\}_{\gamma,g_v,w_e}$$

$$\{\!|\texttt{sample}(e)|\!\}_{\gamma,g,w} =$$
$$\quad let\ \mu, g_e, w' = \{\!|e|\!\}_{\gamma,g,w}\ in$$
$$\quad let\ X, g' = assume(\mu, g_e)\ in\ (X, g', w')$$

$$\{\!|\texttt{observe}(e_1, e_2)|\!\}_{\gamma,g,w} =$$
$$\quad let\ \mu, g_1, w_1 = \{\!|e_1|\!\}_{\gamma,g,w}\ in\ let\ X, g_x = assume(\mu, g_1)\ in$$
$$\quad let\ e_2', g_2, w_2 = \{\!|e_2|\!\}_{\gamma,g_x,w_1}\ in\ let\ v, g_v = value(e_2', g_2)\ in$$
$$\quad let\ g' = observe(X, v, g_v)\ in\ ((), g', w_2 * \mu_{\mathrm{pdf}}(v))$$

$$[\![\texttt{infer}(\texttt{fun}\ s\ \texttt{->}\ e,\ \sigma)]\!]_\gamma =$$
$$\quad let\ \mu = \lambda U.\ \sum_{i=1}^{N}\ let\ s_i, g_i = \mathrm{draw}([\![\sigma]\!]_\gamma)\ in$$
$$\qquad let\ (e_i, s_i'), w_i, g_i' = \{\!|\texttt{fun}\ s\ \texttt{->}\ e|\!\}_{\gamma,1,g_i}(s_i)\ in$$
$$\qquad let\ d_i = distribution(e_i, g_i')\ in$$
$$\qquad \overline{w_i} * d_i(\pi_1(U)) * \delta_{s_i', g_i'}(\pi_2(U))$$
$$\quad in\ (\pi_{1*}(\mu), \pi_{2*}(\mu))$$

$$\overline{w_i} = w_i / \sum_{i=1}^{N} w_i$$

Evaluation

Language features

- Moving parameters
- Fixed parameters
- Inference-in-the-loop

Algorithms comparison

- PF Particle Filtering
- SDS Streaming Delayed Sampling

Evaluation

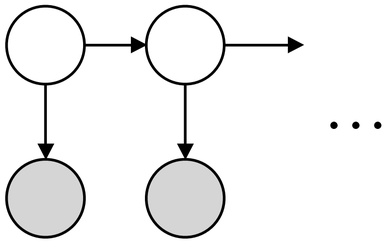
Language features

- Moving parameters
- Fixed parameters
- Inference-in-the-loop

Algorithms comparison

- PF Particle Filtering
- SDS Streaming Delayed Sampling

Moving parameters



Evaluation

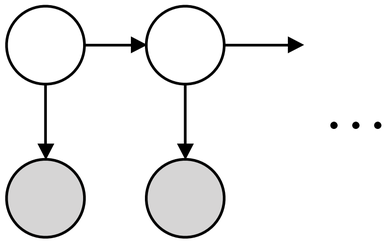
Language features

- Moving parameters
- Fixed parameters
- Inference-in-the-loop

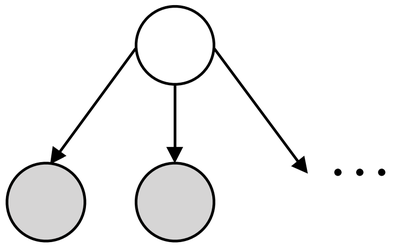
Algorithms comparison

- PF Particle Filtering
- SDS Streaming Delayed Sampling

Moving parameters



Fixed parameters



Evaluation

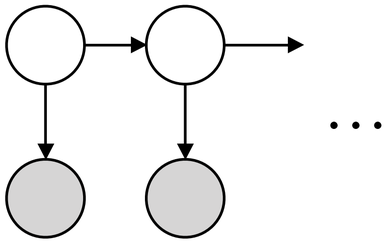
Language features

- Moving parameters
- Fixed parameters
- Inference-in-the-loop

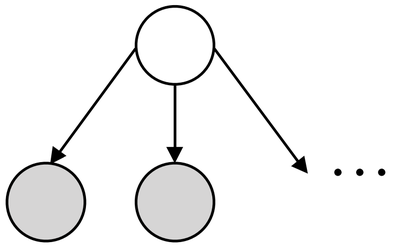
Algorithms comparison

- PF Particle Filtering
- SDS Streaming Delayed Sampling

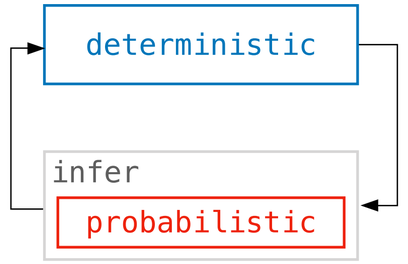
Moving parameters



Fixed parameters



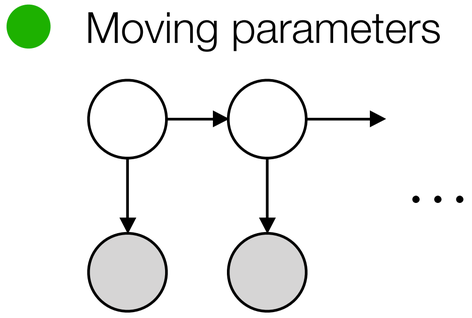
Inference-in-the-loop

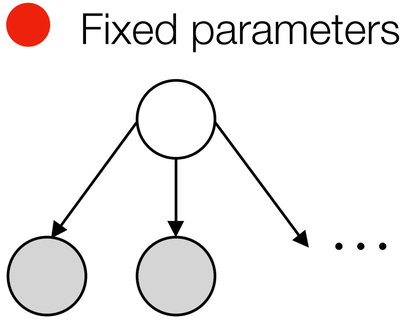


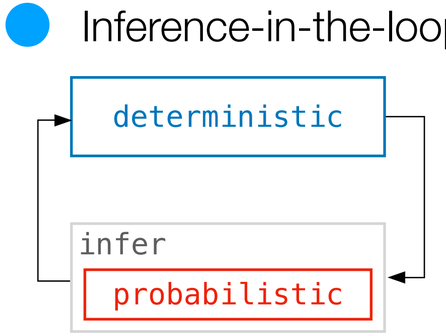
Evaluation

Language features

- Moving parameters
- Fixed parameters
- Inference-in-the-loop

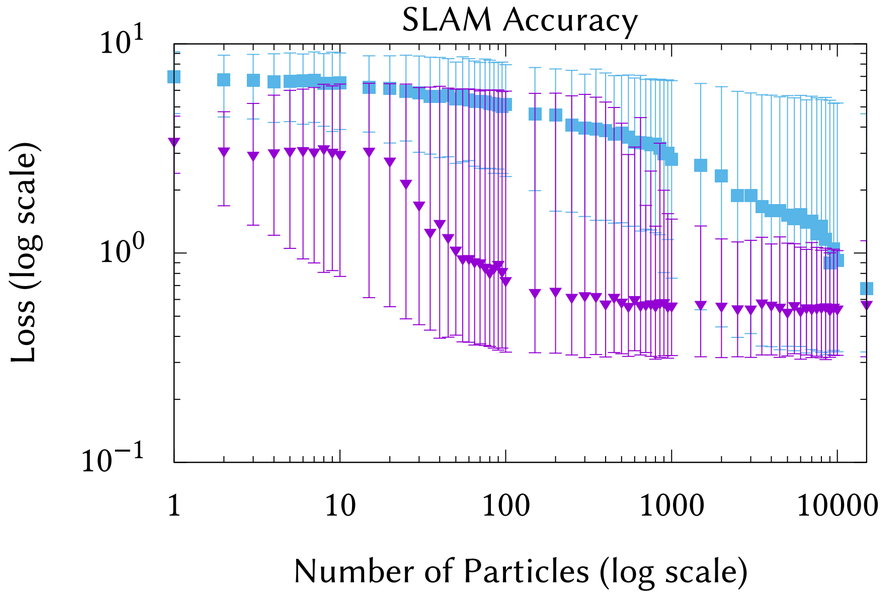


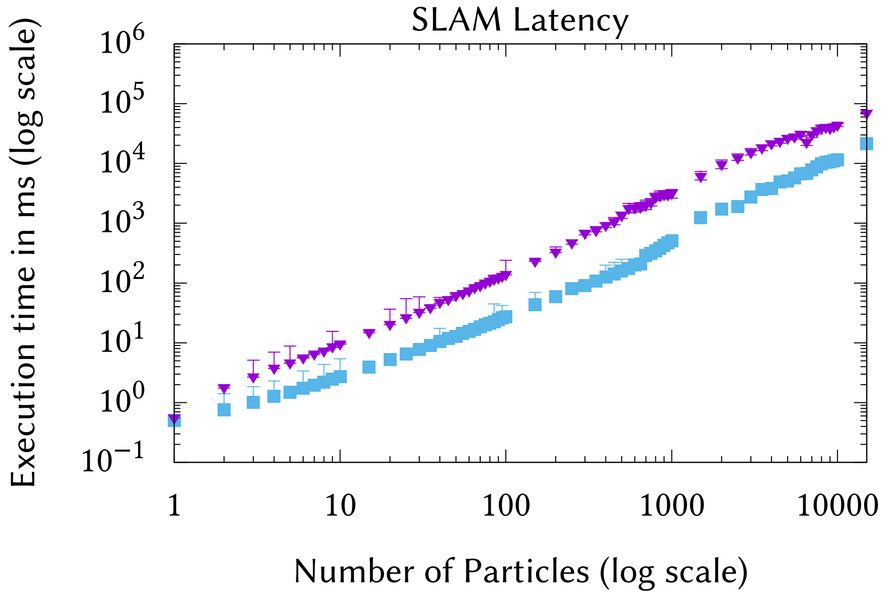




Algorithms comparison

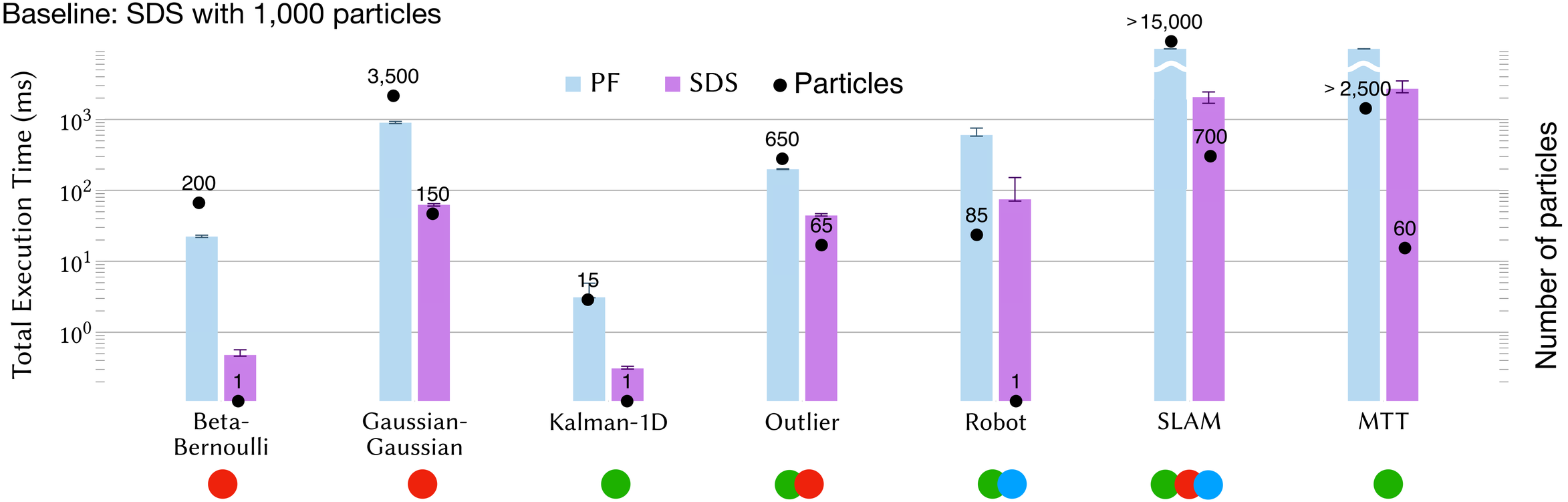
- PF Particle Filtering
- SDS Streaming Delayed Sampling





Benchmarks

Baseline: SDS with 1,000 particles



Moving parameters
Fixed parameters
Inference-in-the-loop

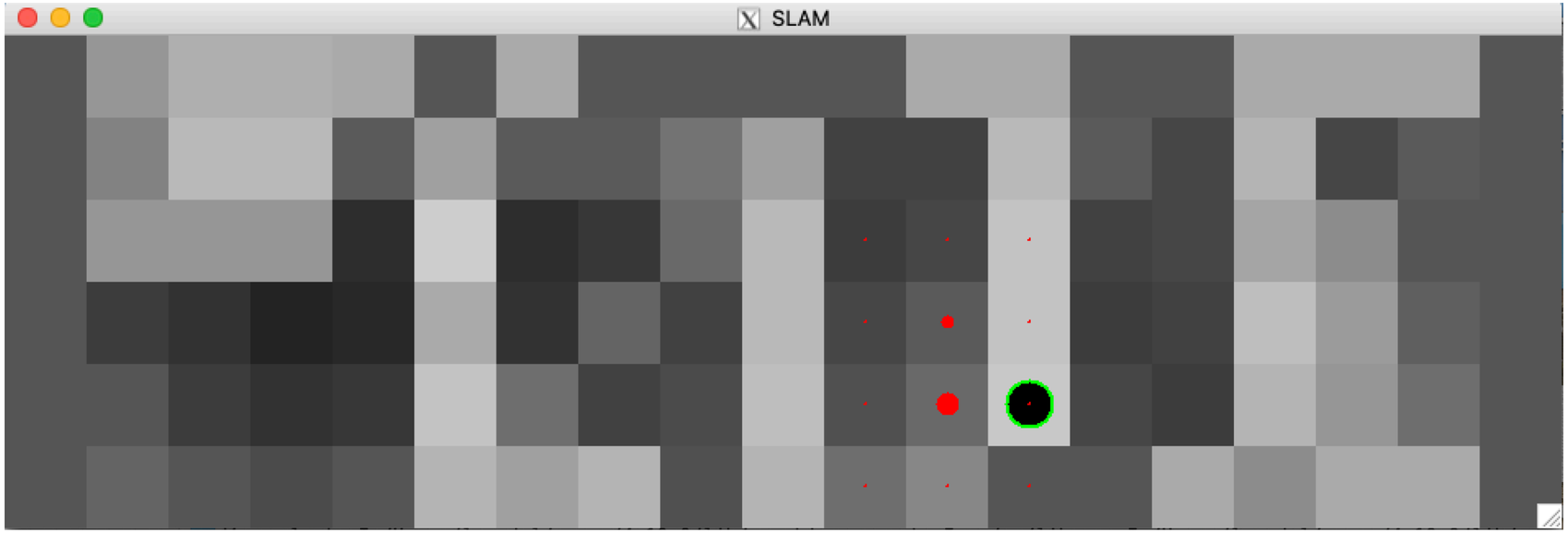
Conclusions

- SDS is always faster to match accuracy
- Reduction in particle count outweighs symbolic overhead
- SDS can be exact (1 particle)
- PF is impractical for advanced examples

Applications, Examples

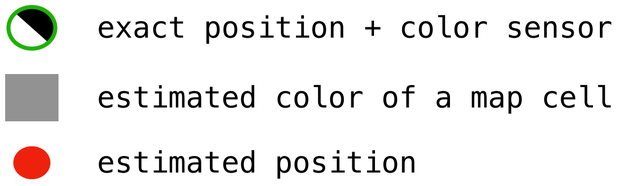
Simultaneous **L**ocalization **A**nd **M**apping

- Environment: slippery wheels and noisy color sensor
- System: infer current position and map, output command (left/right/up/down)



At each step:

- Move to the next position
- Observe the color of the ground
- Use inferred position to compute next command



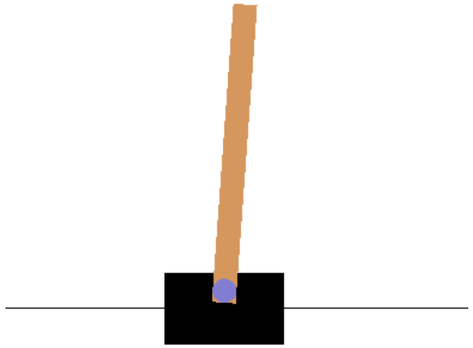
Cartpole

Learn the p, i, d coefficients that minimize the angle

- Randomly initialize the weights
- Compute action with the pid controller
- Use a simple model to simulate the effect of actions
- Favor actions that result in the smallest pole angle

Or… Learn from examples

- Randomly initialize the weights
- Compute action with the pid controller
- Favor actions similar to the example



Summary

How to implement a simple probabilistic language

- Importance sampling
- Extend a synchronous language with probabilistic constructs
- Particles Filtering

ProbZelus

- Language design
- Typing deterministic / probabilistic
- Co-iteration measure-based semantics
- Compilation

Inference

- Formalization of Particle Filtering
- Semi-symbolic inference with Delayed Sampling

Applications

Short Bibliography

Guillaume Baudart, Louis Mandel, Eric Atkinson, Benjamin Sherman, Marc Pouzet, Michael Carbin:
Reactive probabilistic programming. PLDI 2020

Guillaume Baudart, Louis Mandel, Eric Atkinson, Benjamin Sherman, Marc Pouzet, Michael Carbin:
Programmation d'Applications Réactives Probabilistes. JFLA 2020

Noah D. Goodman and Andreas Stuhlmüller.
The Design and Implementation of Probabilistic Programming Languages. http://dippl.org 2014

Lawrence M. Murray, Daniel Lundén, Jan Kudlicka, David Broman, Thomas B. Schön:
Delayed Sampling and Automatic Rao-Blackwellization of Probabilistic Programs. AISTATS 2018

Paul Caspi and Marc Pouzet.
A Co-iterative Characterization of Synchronous Stream Functions. CMCS 1998

https://github.com/IBM/probzelus