Reactive Probabilistic Programming*

Guillaume Baudart — *Inria*

Christine Tasson — *ISAE-Supaero*

L. Mandel — *IBM Research*

E. Atkinson
B. Sherman
C. Yuan
M. Carbin — *MIT*

M. Pouzet — *ENS*

**De la suite dans les IID* (Trad. Guatto)

Uncertainty in embedded systems

Uncertainty in embedded systems

Synchronous languages

- High-level specification language
- Generate correct-by-construction embedded code
- Industrial tool: ANSYS Scade

Challenges

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

Uncertainty in embedded systems

Synchronous languages

- High-level specification language
- Generate correct-by-construction embedded code
- Industrial tool: ANSYS Scade

Challenges

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

Uncertainty in embedded systems

Synchronous languages

- High-level specification language
- Generate correct-by-construction embedded code
- Industrial tool: ANSYS Scade

Challenges

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

Uncertainty in embedded systems

Synchronous languages

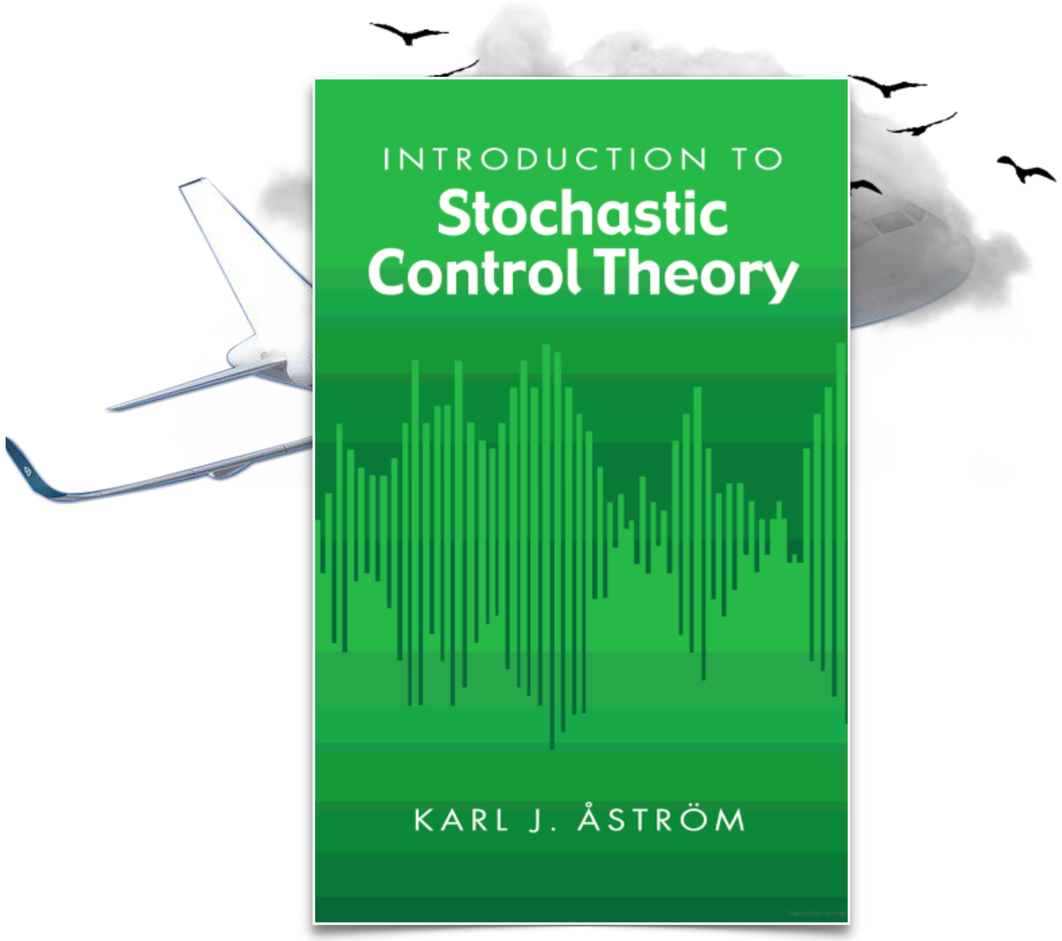
- High-level specification language
- Generate correct-by-construction embedded code
- Industrial tool: ANSYS Scade

Challenges

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

Existing approaches

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



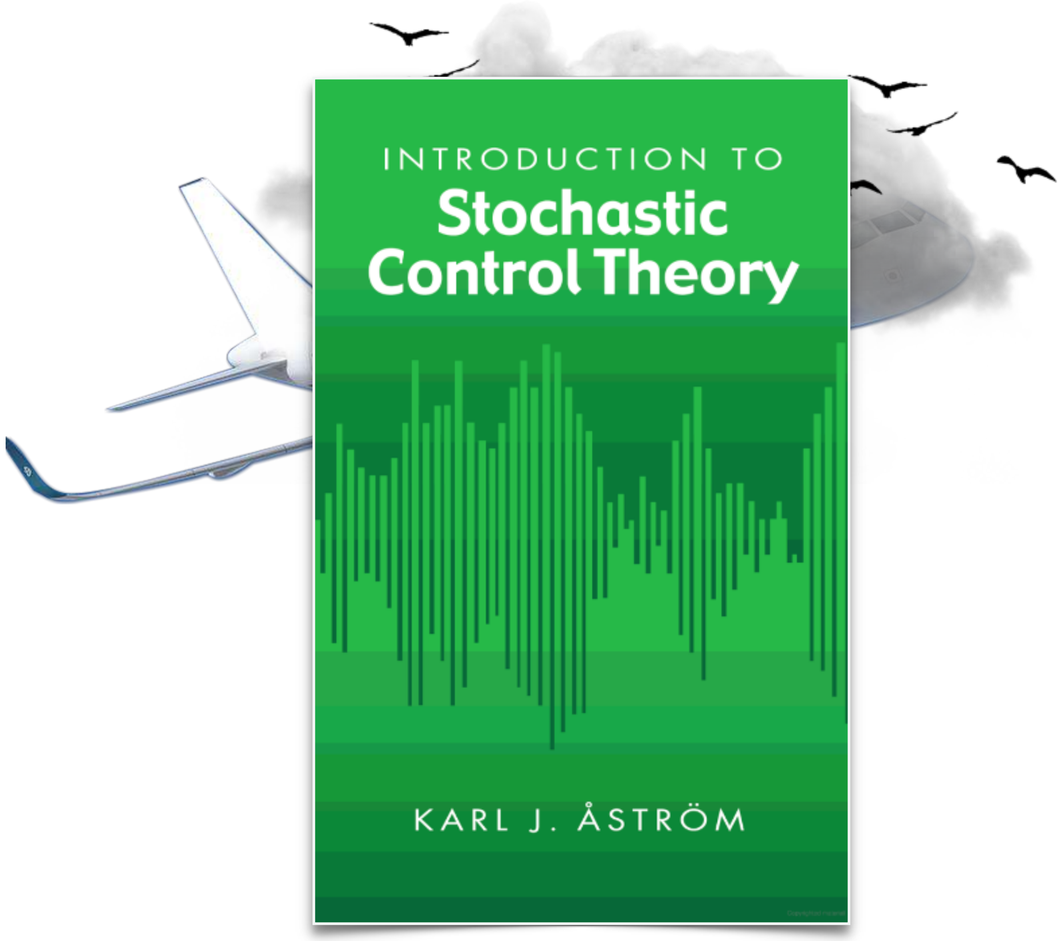
Uncertainty in embedded systems

Synchronous languages
- High-level specification language
- Generate correct-by-construction embedded code
- Industrial tool: ANSYS Scade

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

Existing approaches
- Manually implement stochastic controller: *Can be error prone*
- Offline statistical tests: *Requires up-to-date offline data*

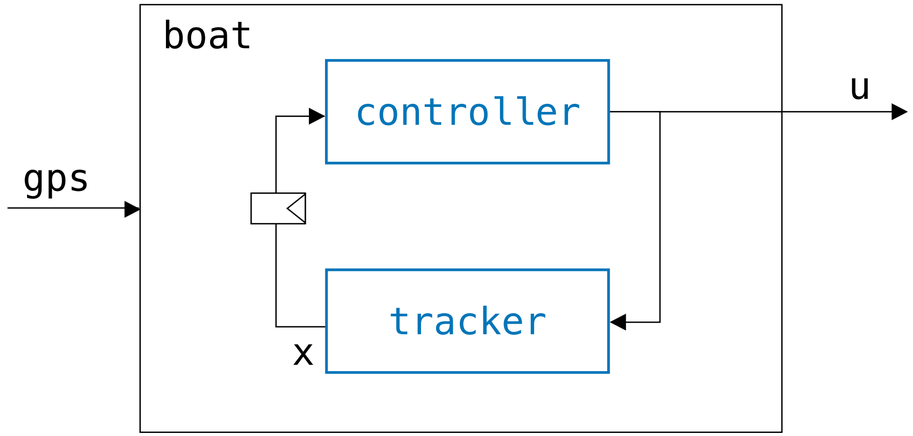
Reactive Probabilistic Programming
- Synchronous languages with probabilistic constructs
- Make the probabilistic model explicit
- Automatically learn posterior distributions from observations



Reactive systems

Synchronous data-flow languages and block diagrams

- Signal: stream of values
- System: stream processor



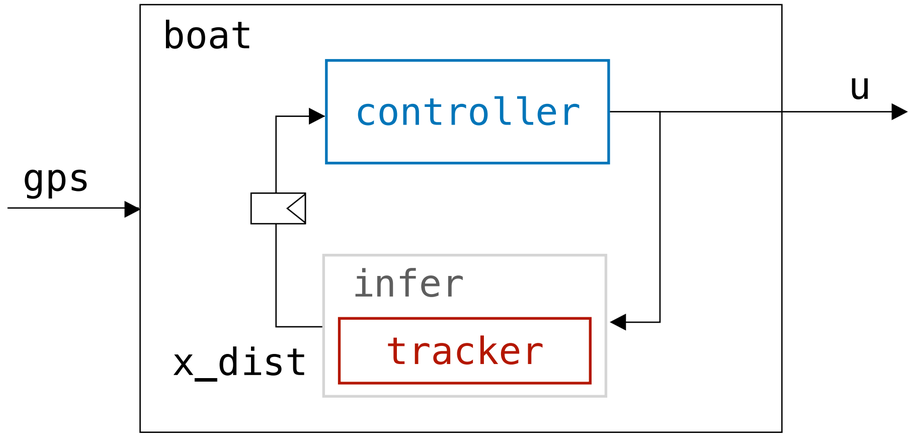
Reactive probabilistic systems

Synchronous data-flow languages and block diagrams

- Signal: stream of values
- System: stream processor

ProbZelus: add support to deal with uncertainty

- Extend a synchronous language
- Parallel composition: deterministic/probabilistic
- Inference-in-the-loop
- Streaming inference



Part I. Programming in ProbZelus

Synchronous programming

Reactive Probabilistic Programming

Lustre → Lucid Synchrone → Zelus → ProbZelus

Dataflow synchronous programming

- Set of stream equations
- Discrete logical time steps
- At each step, compute the current value given inputs and previous values

Stream operations

- Constant are lifted to stream: `1 = 1, 1, 1, ....`
- Temporal operators: `→, pre, fby`
- Control structures: `reset/every, present, automaton`

Bourke, Pouzet 2013

Lustre → Lucid Synchrone → Zelus → ProbZelus

Dataflow synchronous programming

- Set of stream equations
- Discrete logical time steps
- At each step, compute the current value given inputs and previous values

```
node nat v = cpt where
  rec cpt = v → pre cpt + 1
```

$$cpt_n = if\ (n = 0)\ then\ v_0\ else\ cpt_{n-1} + 1$$

Bourke, Pouzet 2013

Lustre → Lucid Synchrone → Zelus → ProbZelus

Dataflow synchronous programming

- Set of stream equations
- Discrete logical time steps
- At each step, compute the current value given inputs and previous values

```
node nat v = cpt where
  rec cpt = v → pre cpt + 1
```

$$cpt_n = if\ (n = 0)\ then\ v_0\ else\ cpt_{n-1} + 1$$

$t = 0$

Bourke, Pouzet 2013

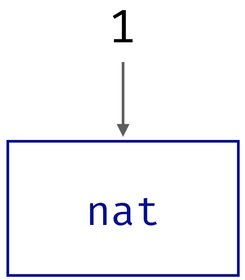
Lustre → Lucid Synchrone → Zelus → ProbZelus

Dataflow synchronous programming

- Set of stream equations
- Discrete logical time steps
- At each step, compute the current value given inputs and previous values

```
node nat v = cpt where
  rec cpt = v → pre cpt + 1
```

$$cpt_n = if\ (n = 0)\ then\ v_0\ else\ cpt_{n-1} + 1$$



$t = 0$

Bourke, Pouzet 2013

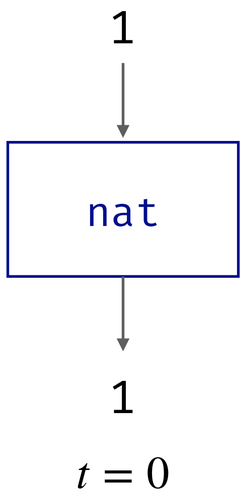
Lustre → Lucid Synchrone → Zelus → ProbZelus

Dataflow synchronous programming

- Set of stream equations
- Discrete logical time steps
- At each step, compute the current value given inputs and previous values

```
node nat v = cpt where
  rec cpt = v → pre cpt + 1
```

$$cpt_n = if\ (n = 0)\ then\ v_0\ else\ cpt_{n-1} + 1$$



Bourke, Pouzet 2013

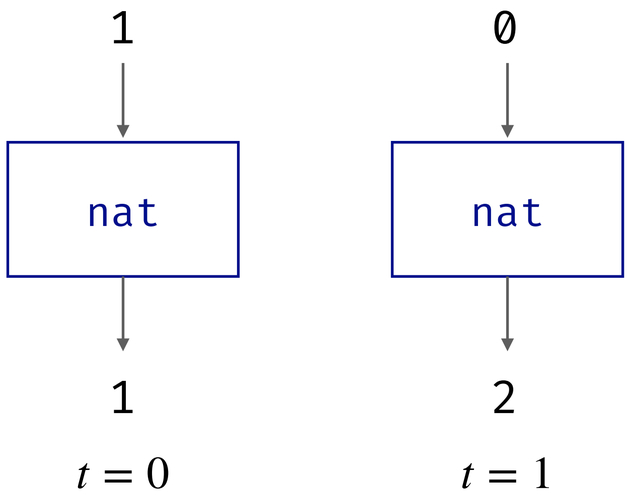
Lustre → Lucid Synchrone → Zelus → ProbZelus

Dataflow synchronous programming

- Set of stream equations
- Discrete logical time steps
- At each step, compute the current value given inputs and previous values

```
node nat v = cpt where
  rec cpt = v → pre cpt + 1
```

$$cpt_n = if\ (n = 0)\ then\ v_0\ else\ cpt_{n-1} + 1$$



Bourke, Pouzet 2013

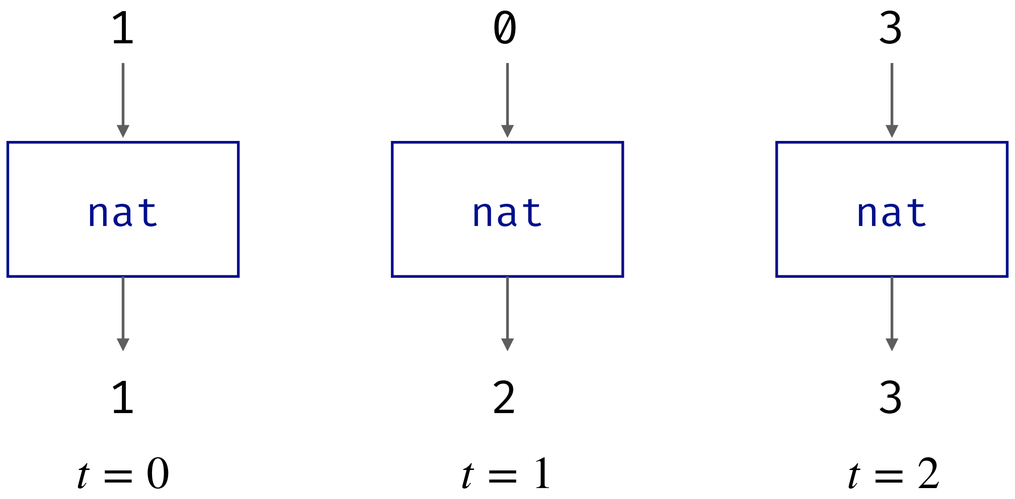
Lustre → Lucid Synchrone → Zelus → ProbZelus

Dataflow synchronous programming

- Set of stream equations
- Discrete logical time steps
- At each step, compute the current value given inputs and previous values

```
node nat v = cpt where
  rec cpt = v → pre cpt + 1
```

$$cpt_n = if\ (n = 0)\ then\ v_0\ else\ cpt_{n-1} + 1$$



Bourke, Pouzet 2013

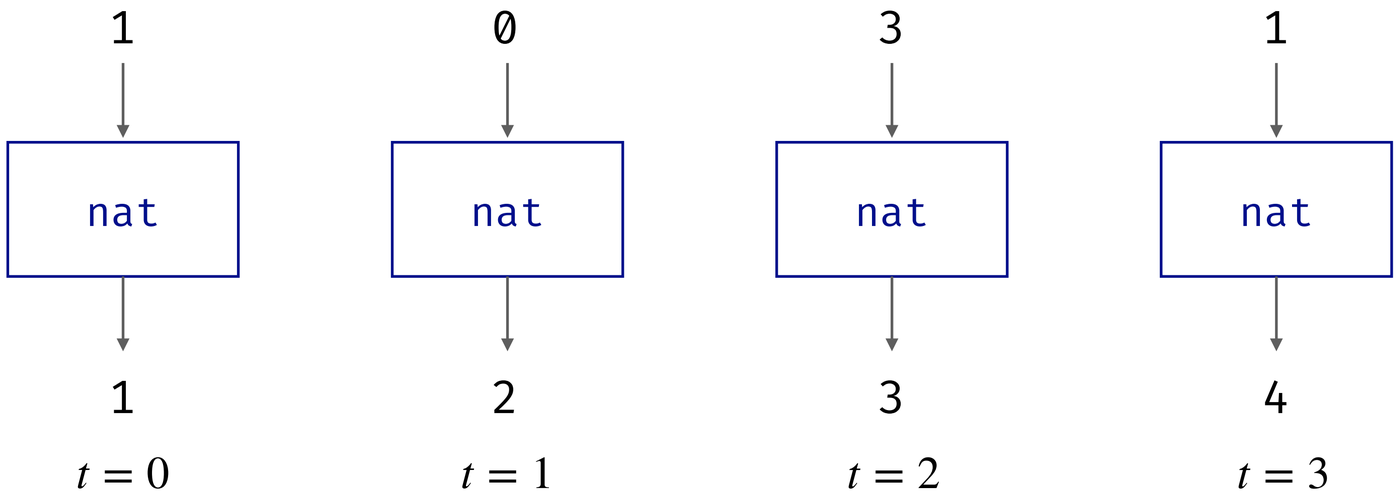
Lustre → Lucid Synchrone → Zelus → ProbZelus

Dataflow synchronous programming

- Set of stream equations
- Discrete logical time steps
- At each step, compute the current value given inputs and previous values

```
node nat v = cpt where
  rec cpt = v → pre cpt + 1
```

$$cpt_n = if\ (n = 0)\ then\ v_0\ else\ cpt_{n-1} + 1$$



Bourke, Pouzet 2013

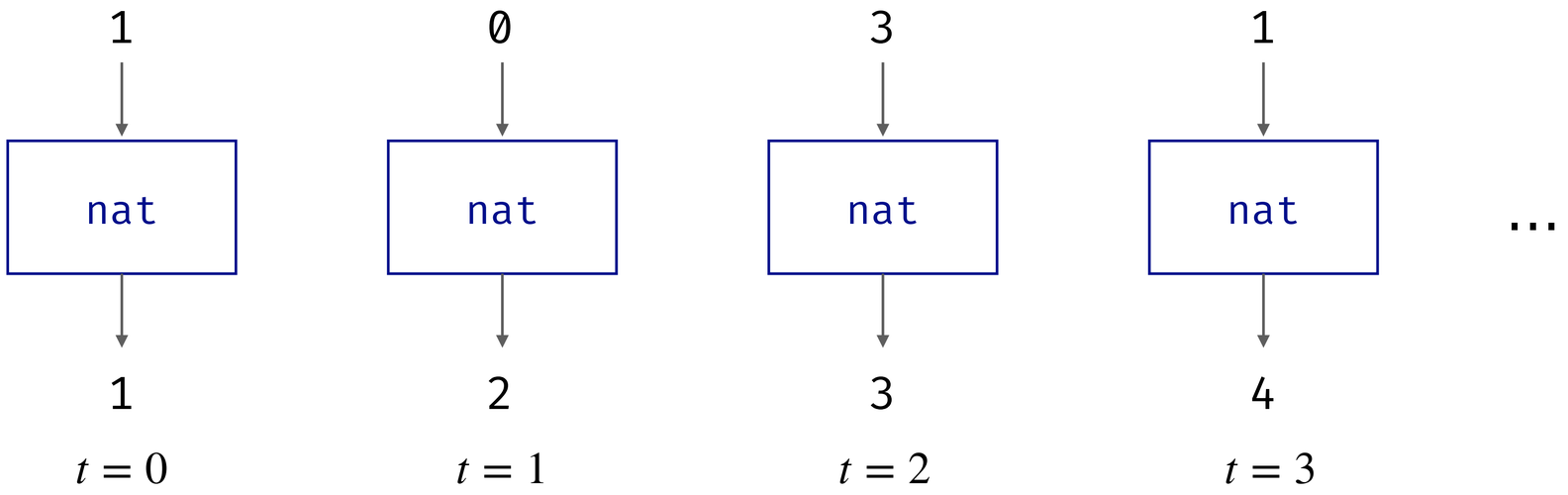
Lustre → Lucid Synchrone → Zelus → ProbZelus

Dataflow synchronous programming

- Set of stream equations
- Discrete logical time steps
- At each step, compute the current value given inputs and previous values

```
node nat v = cpt where
  rec cpt = v → pre cpt + 1
```

$$cpt_n = if\ (n = 0)\ then\ v_0\ else\ cpt_{n-1} + 1$$



Bourke, Pouzet 2013

Demo

Probabilistic programming

Reactive Probabilistic Programming

Probabilistic programming languages

Probabilistic programming languages

General purpose programming languages extended with probabilistic constructs

- `x = sample(d)`: introduce a random variable `x` of distribution `d`
- `observe(d, y)`: condition on the fact that `y` was sampled from `d`
- `infer m y`: compute posterior distribution of `m` given `y`

Probabilistic programming languages

General purpose programming languages extended with probabilistic constructs

- `x = sample(d)`: introduce a random variable `x` of distribution `d`
- `observe(d, y)`: condition on the fact that `y` was sampled from `d`
- `infer m y`: compute posterior distribution of `m` given `y`

Multiple examples:

- Church, Anglican (lisp, clojure), 2008
- WebPPL (javascript), 2014
- Pyro/NumPyro (python), 2017/2019
- Gen (julia), 2018
- ProbZelus (Zelus), 2019
- ...

Probabilistic programming languages

General purpose programming languages extended with probabilistic constructs

- `x = sample(d)`: introduce a random variable `x` of distribution `d`
- `observe(d, y)`: condition on the fact that `y` was sampled from `d`
- `infer m y`: compute posterior distribution of `m` given `y`

Multiple examples:

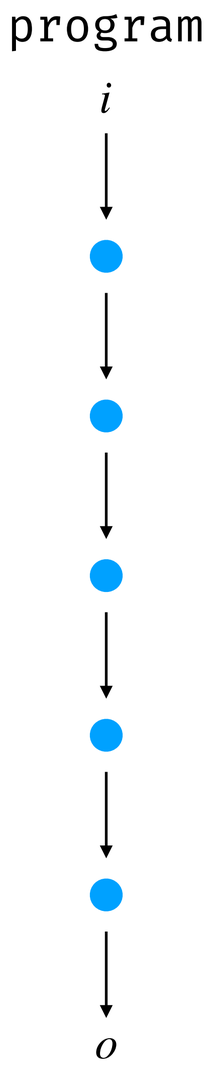
- Church, Anglican (lisp, clojure), 2008
- WebPPL (javascript), 2014
- Pyro/NumPyro (python), 2017/2019
- Gen (julia), 2018
- ProbZelus (Zelus), 2019
- ...

More and more, incorporating new ideas:

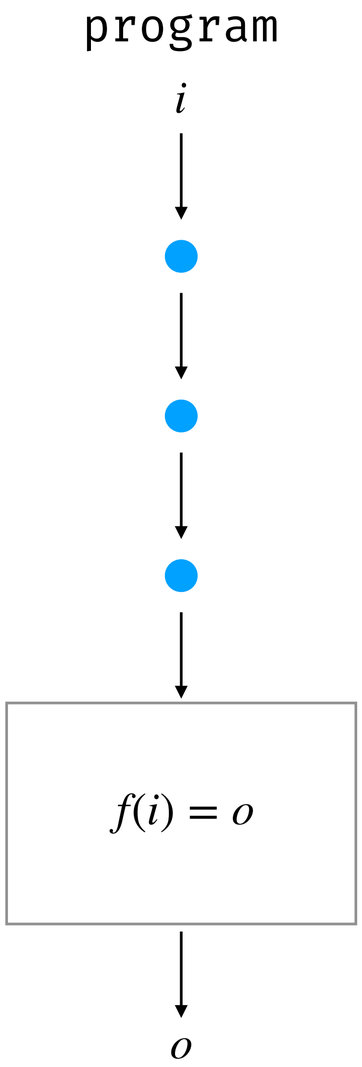
- New inference techniques, e.g., stochastic variational inference (SVI)
- Interaction with neural nets (deep probabilistic programming)

$\texttt{infer} : (\alpha \to \beta) \to \alpha \to \beta\ \texttt{dist}$

infer : $(\alpha \to \beta) \to \alpha \to \beta$ dist



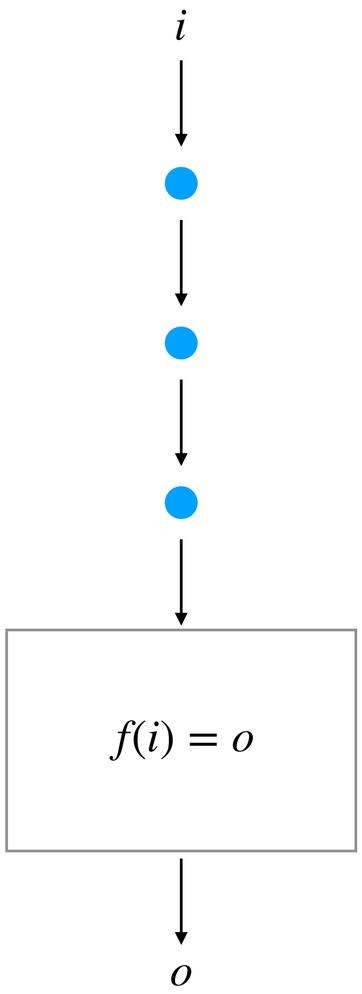
infer : $(\alpha \to \beta) \to \alpha \to \beta$ dist



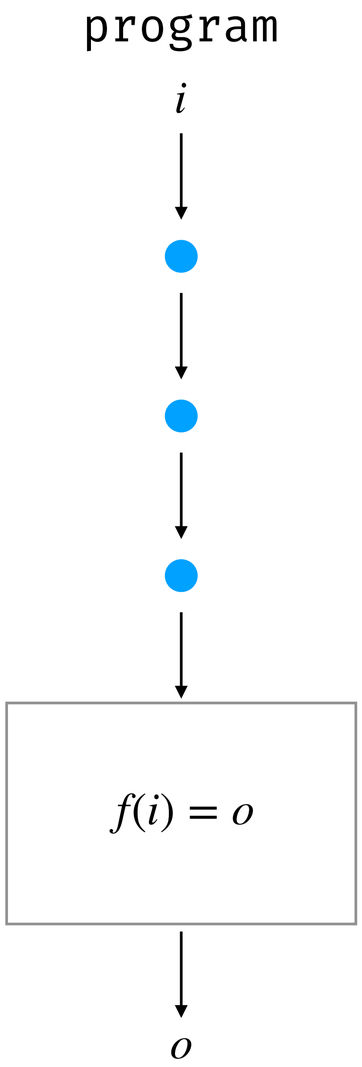
infer : $(\alpha \to \beta) \to \alpha \to \beta$ dist

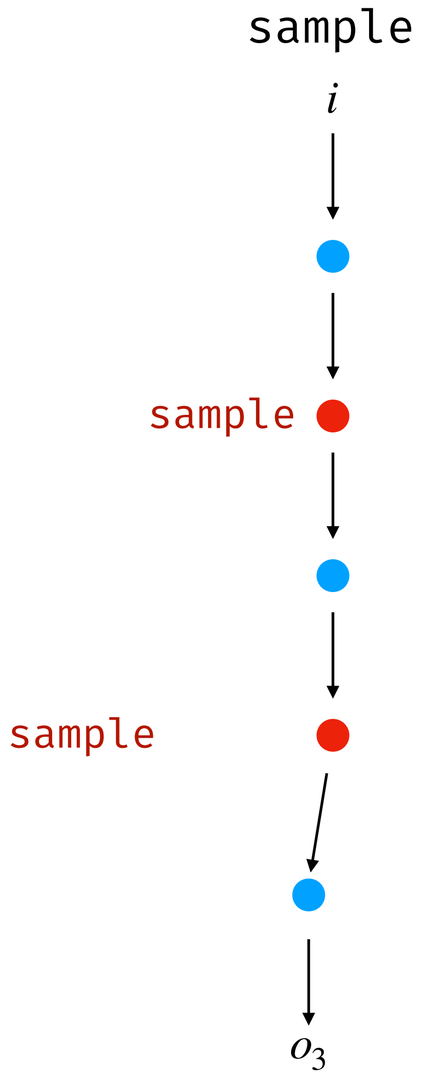
program

sample

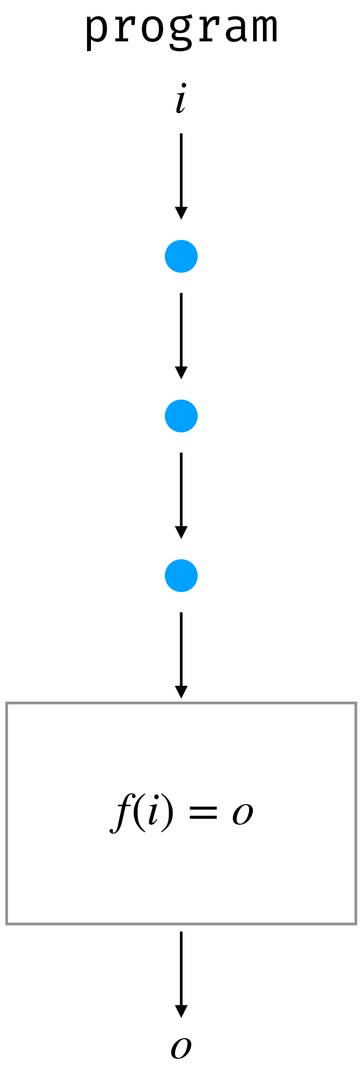


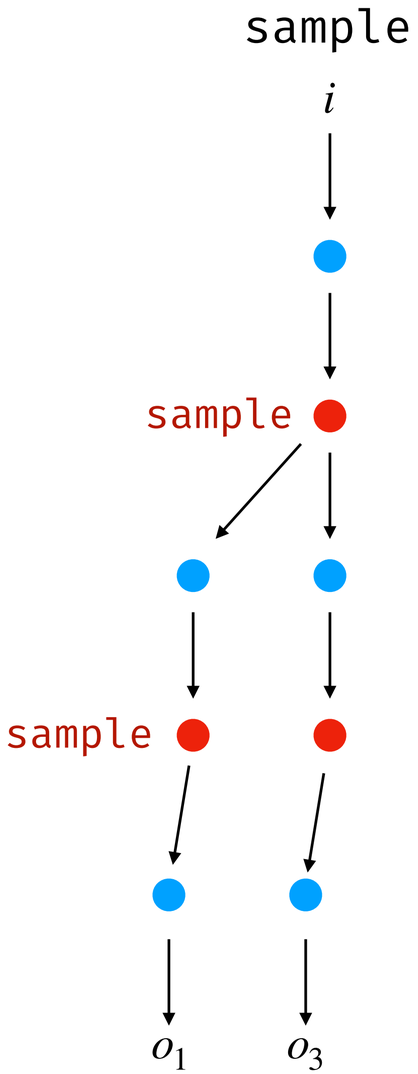
infer : $(\alpha \to \beta) \to \alpha \to \beta$ dist



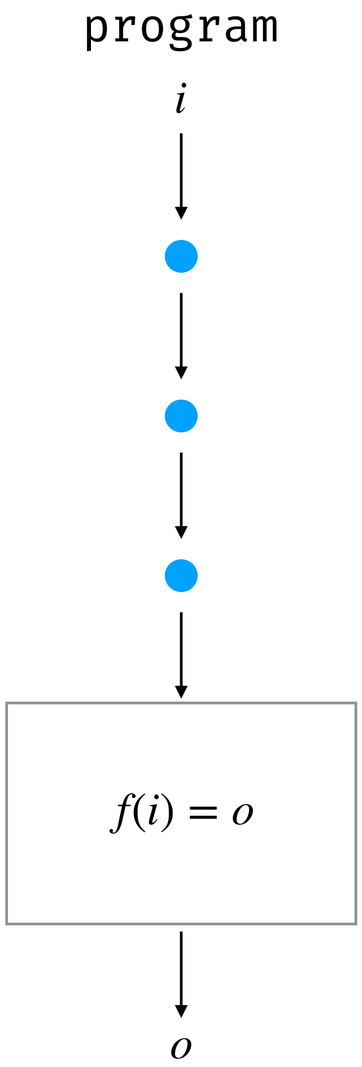


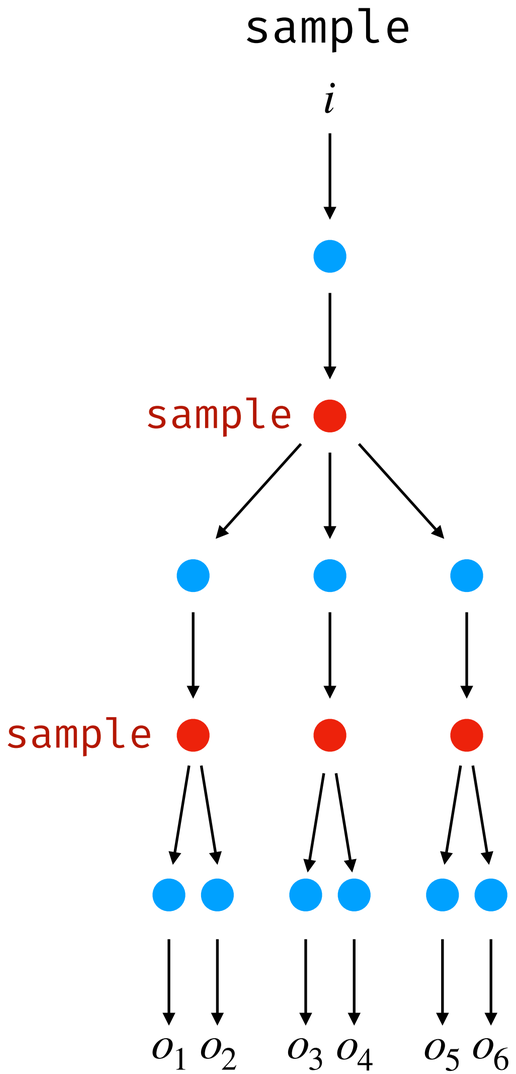
infer : $(\alpha \to \beta) \to \alpha \to \beta$ dist



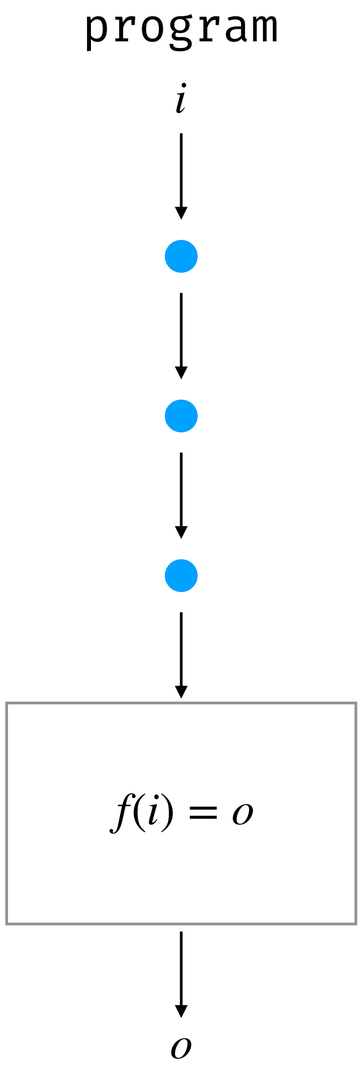


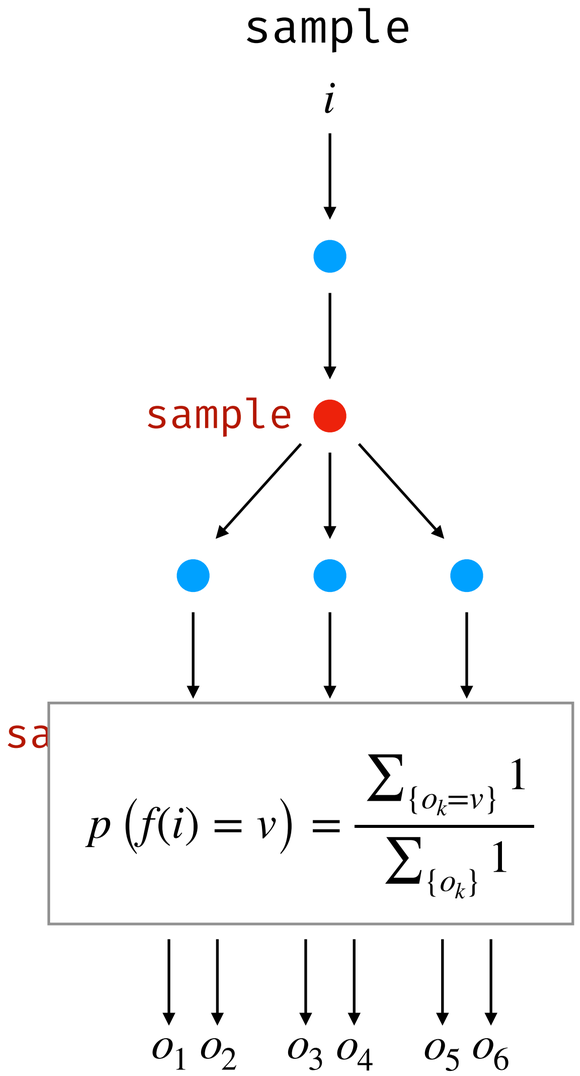
infer : $(\alpha \to \beta) \to \alpha \to \beta$ dist





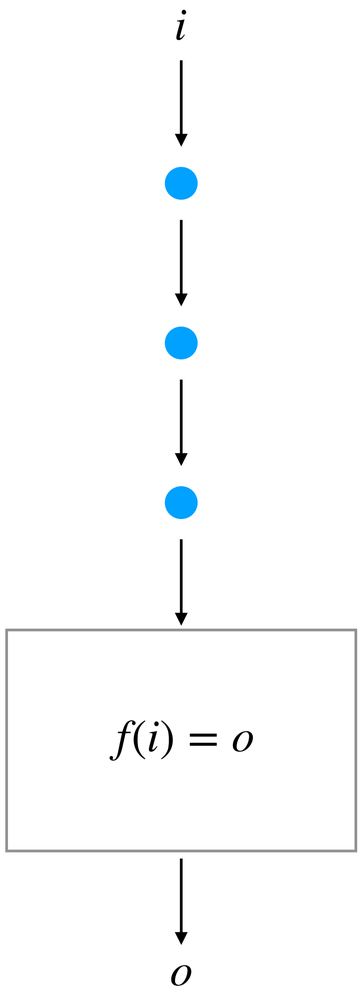
infer : $(\alpha \to \beta) \to \alpha \to \beta$ dist



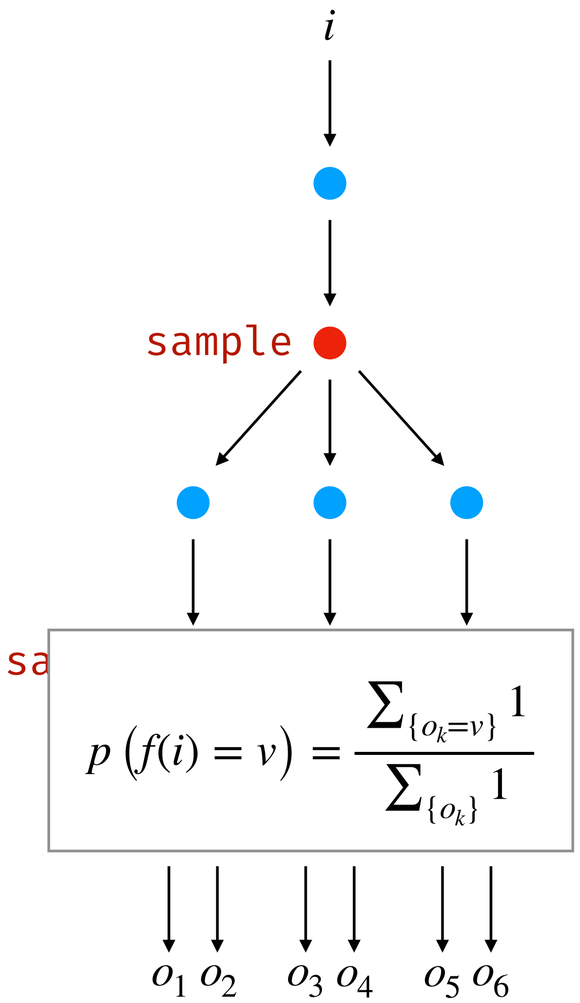


infer : $(\alpha \to \beta) \to \alpha \to \beta$ dist

program

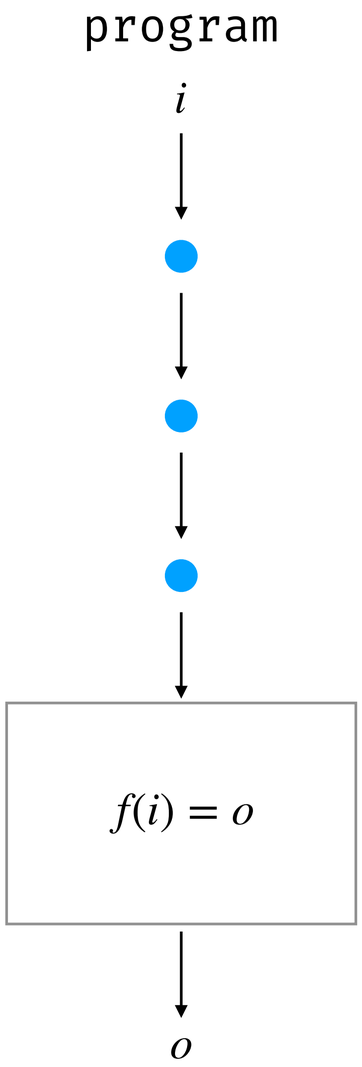


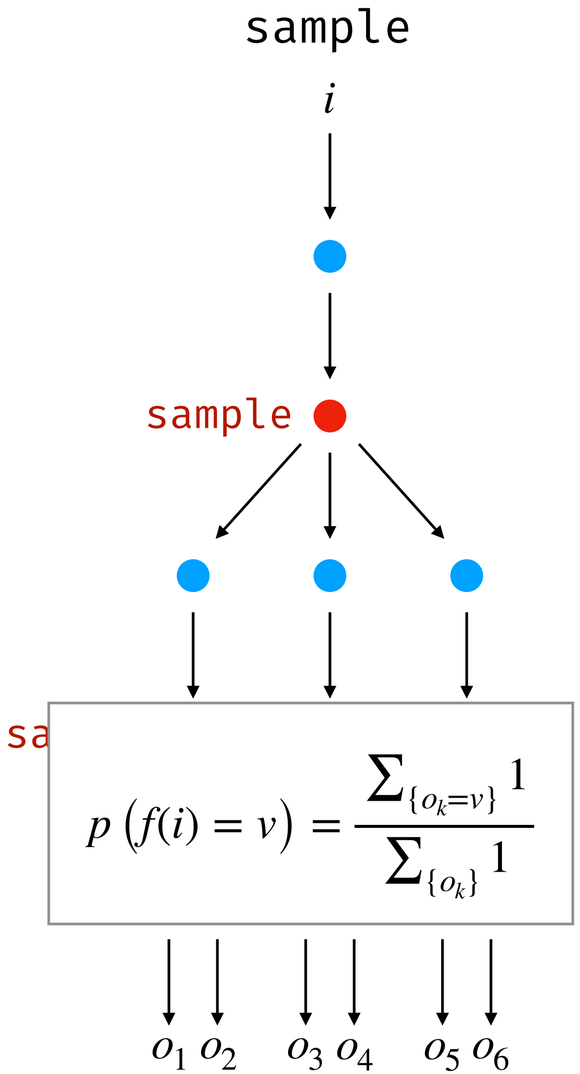
sample

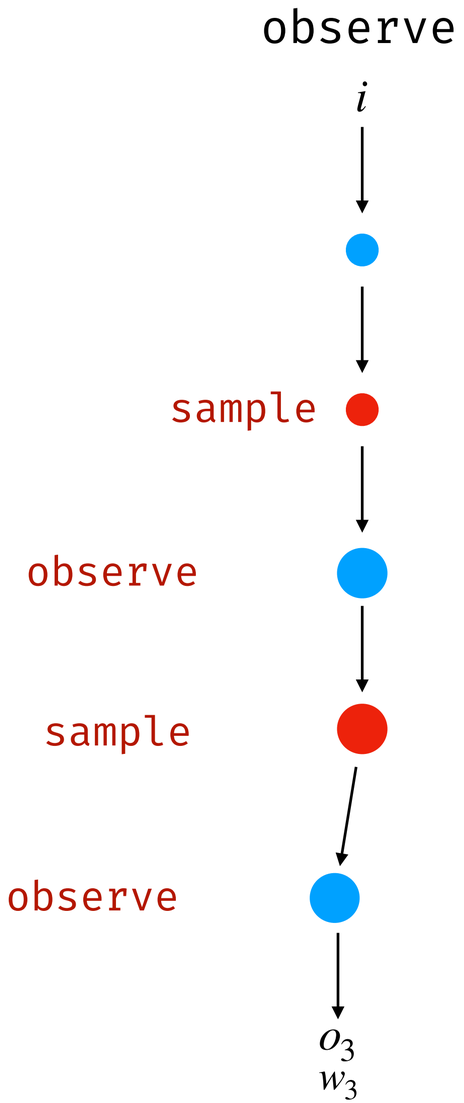


observe

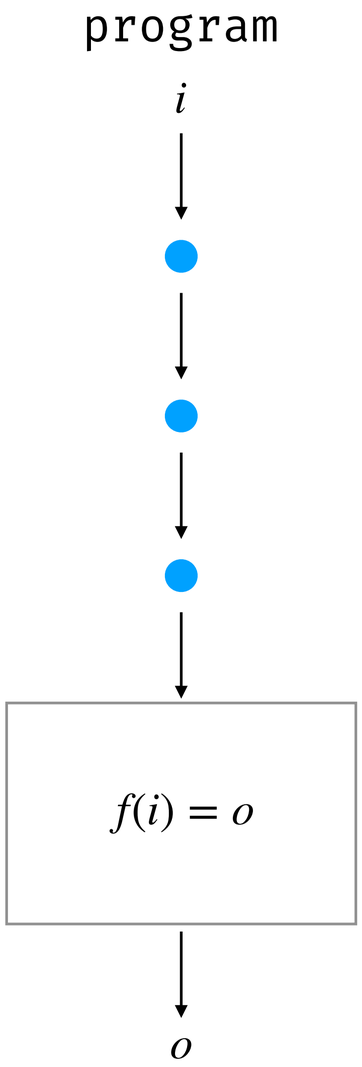
infer : $(\alpha \to \beta) \to \alpha \to \beta$ dist

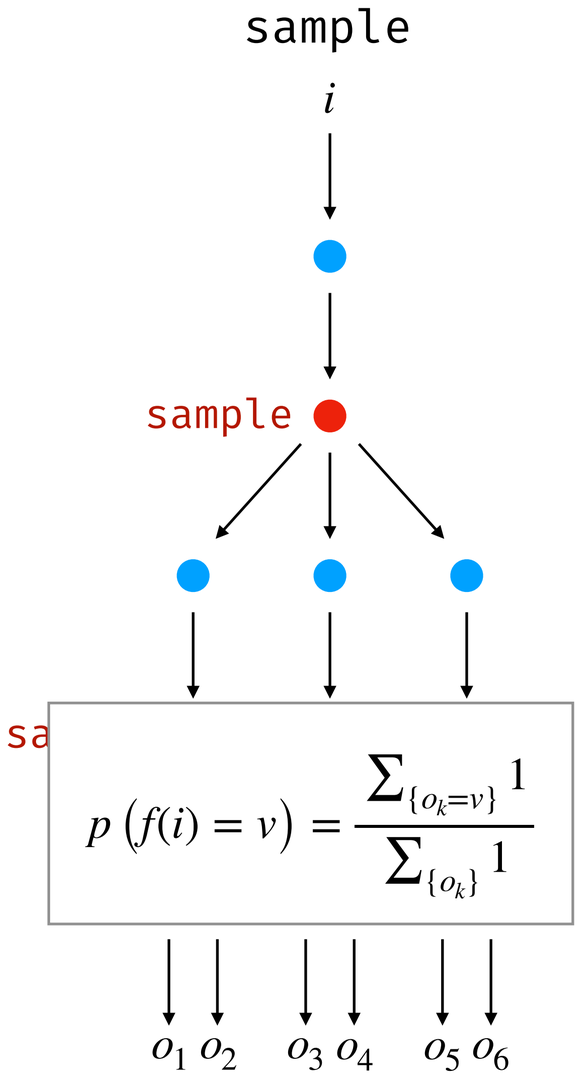


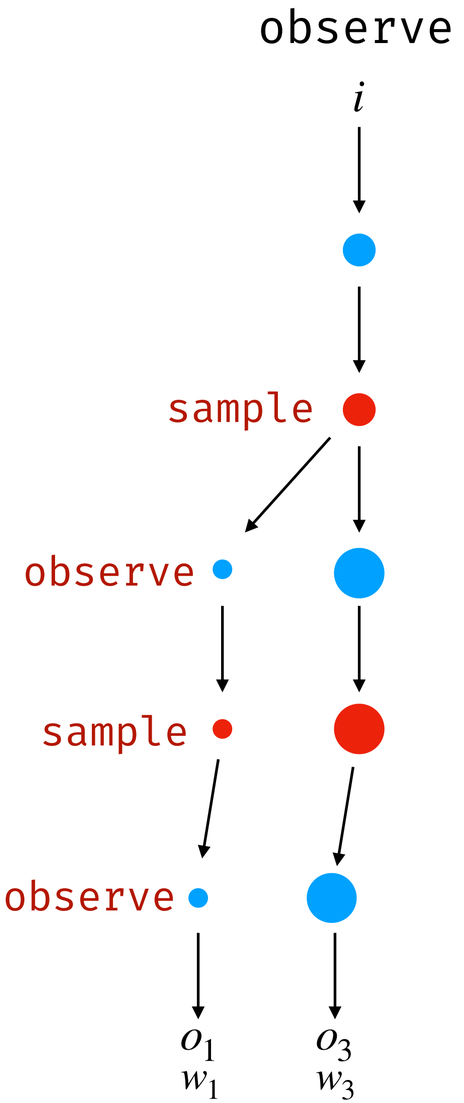




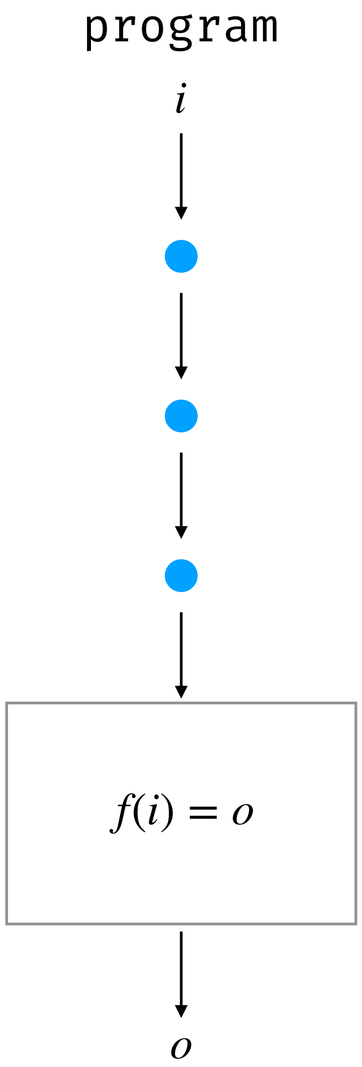
infer : $(\alpha \to \beta) \to \alpha \to \beta$ dist

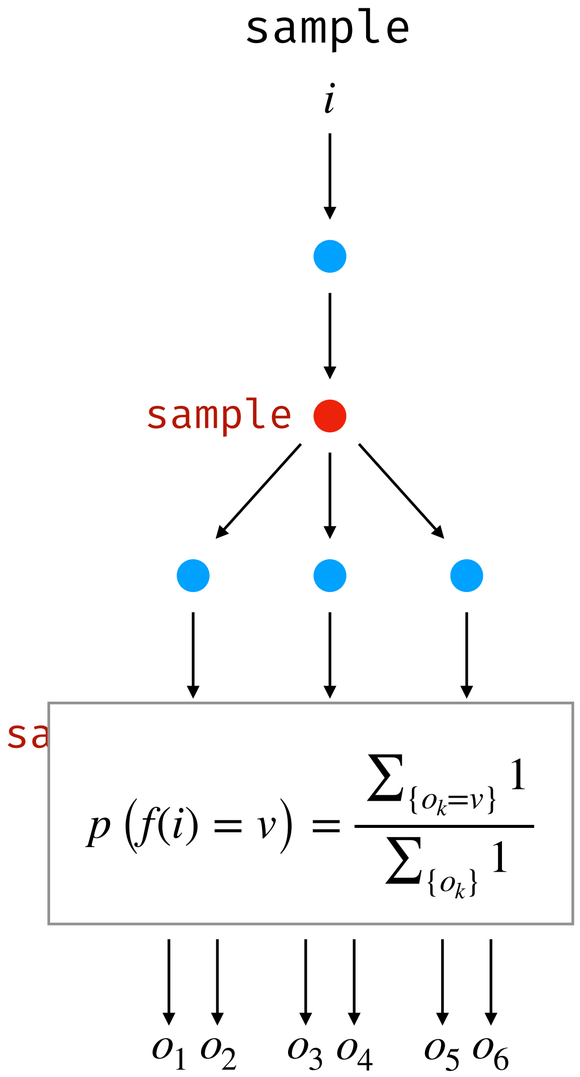


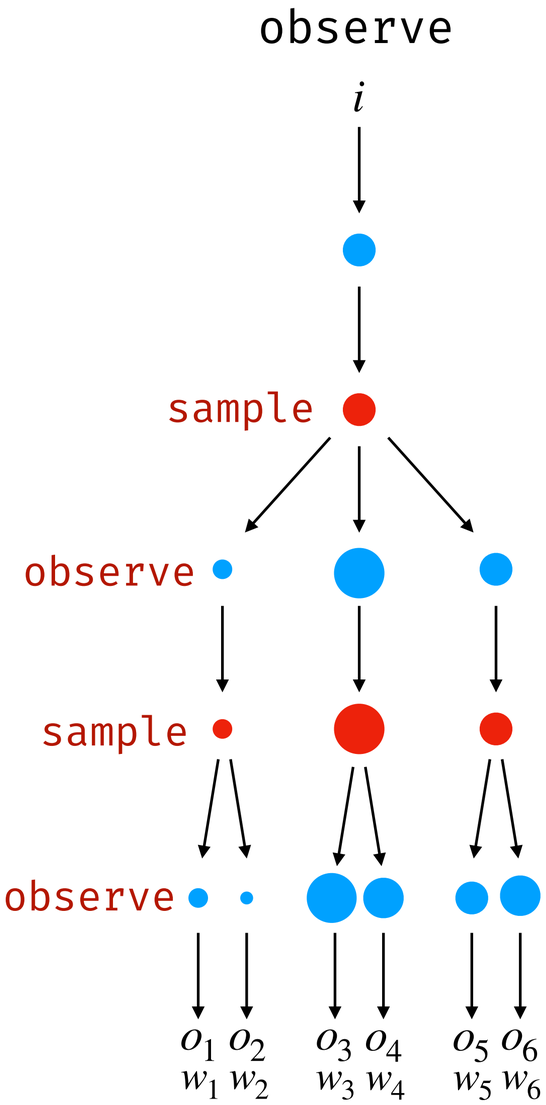




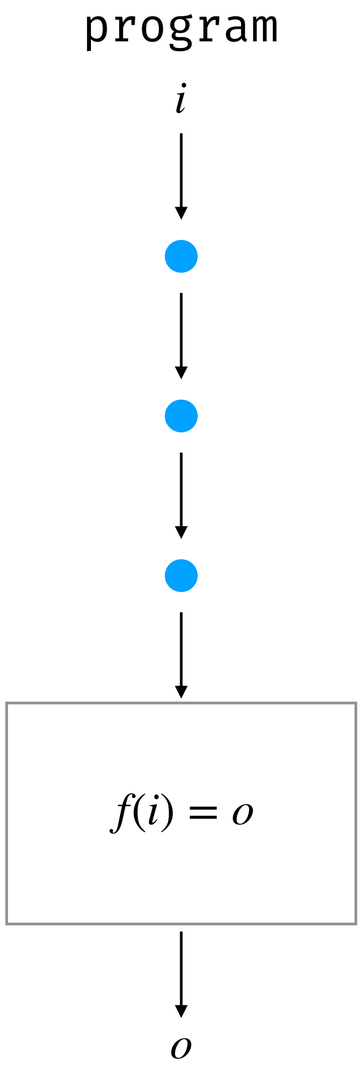
infer : $(\alpha \to \beta) \to \alpha \to \beta$ dist

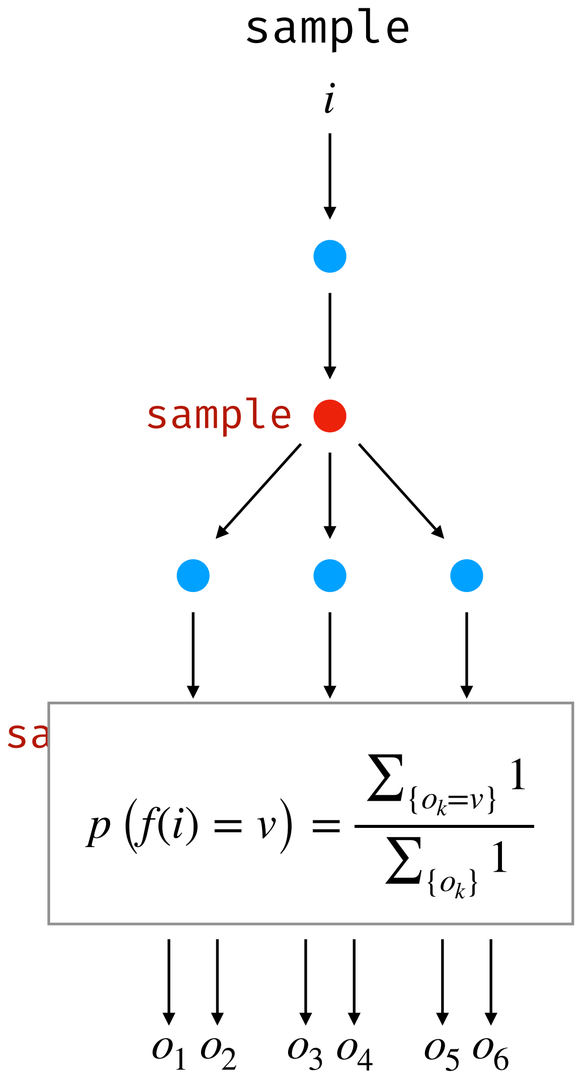


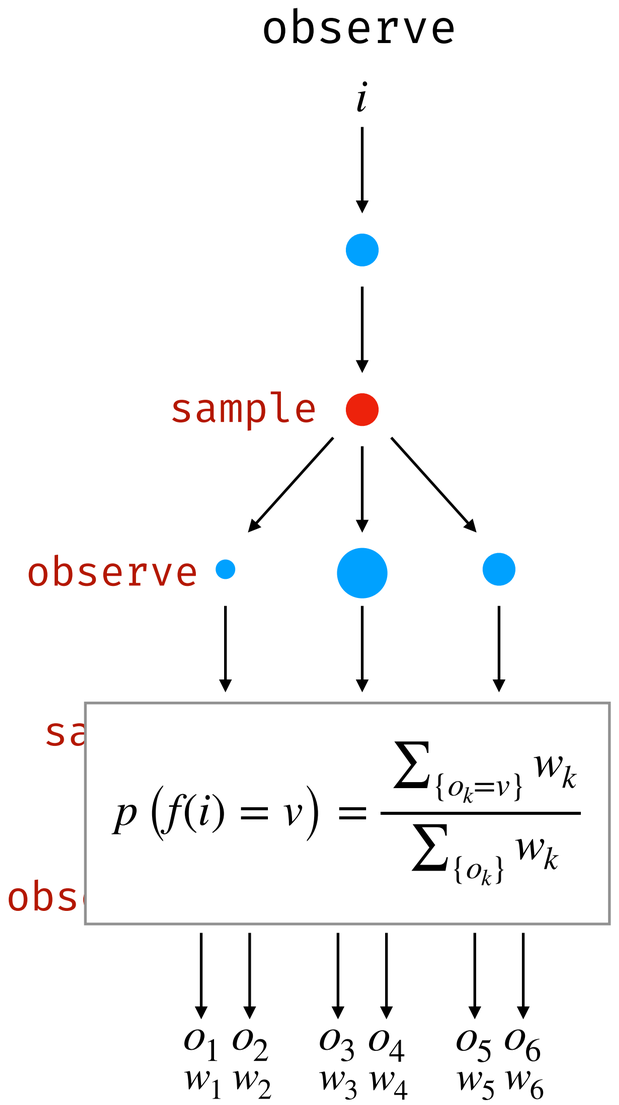




infer : $(\alpha \to \beta) \to \alpha \to \beta$ dist







Bayesian reasoning

$$p(x \mid y_1, \ldots, y_n) = \frac{p(x)\, p(y_1, \ldots, y_n \mid x)}{p(y_1, \ldots, y_n)} \qquad \text{(Bayes' theorem)}$$

$$\propto p(x)\, p(y_1, \ldots, y_n \mid x) \qquad \text{(Data are constants)}$$

Thomas Bayes (1701-1761)

https://en.wikipedia.org/wiki/Thomas_Bayes

Bayesian reasoning

Bayesian Inference: learn parameters from data

- Latent parameter x
- Observed data $y_1, \ldots, y_n$

$$p(x \mid y_1, \ldots, y_n) = \frac{p(x)\, p(y_1, \ldots, y_n \mid x)}{p(y_1, \ldots, y_n)} \qquad \text{(Bayes' theorem)}$$

$$\propto p(x)\, p(y_1, \ldots, y_n \mid x) \qquad \text{(Data are constants)}$$

Thomas Bayes (1701-1761)

https://en.wikipedia.org/wiki/Thomas_Bayes

Bayesian reasoning

Bayesian Inference: learn parameters from data

- Latent parameter x
- Observed data $y_1, \ldots, y_n$

$$p(x \mid y_1, \ldots, y_n) = \frac{p(x)\, p(y_1, \ldots, y_n \mid x)}{p(y_1, \ldots, y_n)} \quad \text{(Bayes' theorem)}$$

posterior

$$\propto p(x)\, p(y_1, \ldots, y_n \mid x) \quad \text{(Data are constants)}$$

prior *likelihood*

Thomas Bayes (1701-1761)

Probabilistic constructs

- `x = sample(d)`: introduce a random variable `x` of distribution `d`
- `observe(d, y)`: condition on the fact that `y` was sampled from `d`
- `infer m y`: compute posterior distribution of `m` given `y`

https://en.wikipedia.org/wiki/Thomas_Bayes

Bayesian reasoning

Bayesian Inference: learn parameters from data

- Latent parameter x
- Observed data $y_1, \ldots, y_n$

$$p(x \mid y_1, \ldots, y_n) = \frac{p(x)\, p(y_1, \ldots, y_n \mid x)}{p(y_1, \ldots, y_n)} \quad \text{(Bayes' theorem)}$$

posterior

$$\propto p(x)\, p(y_1, \ldots, y_n \mid x) \quad \text{(Data are constants)}$$

prior *likelihood*

```
let model (y1, ..., yn) =
  let x = sample prior in
  let () = observe (likelihood x) (y1, ..., yn) in
  x

let posterior = infer model (y1, ..., yn)
```

Thomas Bayes (1701-1761)

https://en.wikipedia.org/wiki/Thomas_Bayes

Probabilistic programming

Probabilistic constructs

- `x = sample(d)`: introduce a random variable `x` of distribution `d`
- `observe(d, y)`: condition on the fact that `y` was sampled from `d`
- `infer m y`: compute posterior distribution of `m` given `y`

More general than classic Bayesian Reasoning

```
let rec weird () =
  let b = sample (bernoulli ~p:0.5) in
  let mu = if (b = 1) then 0.5 else 1.0 in
  let theta = sample (gaussian ~mu ~sigma:1.0) in
  if theta > 0. then
    observe (gaussian ~mu ~sigma:0.5) theta;
    theta
  else weird ()

let weird_dist = infer weird ()
```

Thomas Bayes (1701-1761)

Probabilistic programming

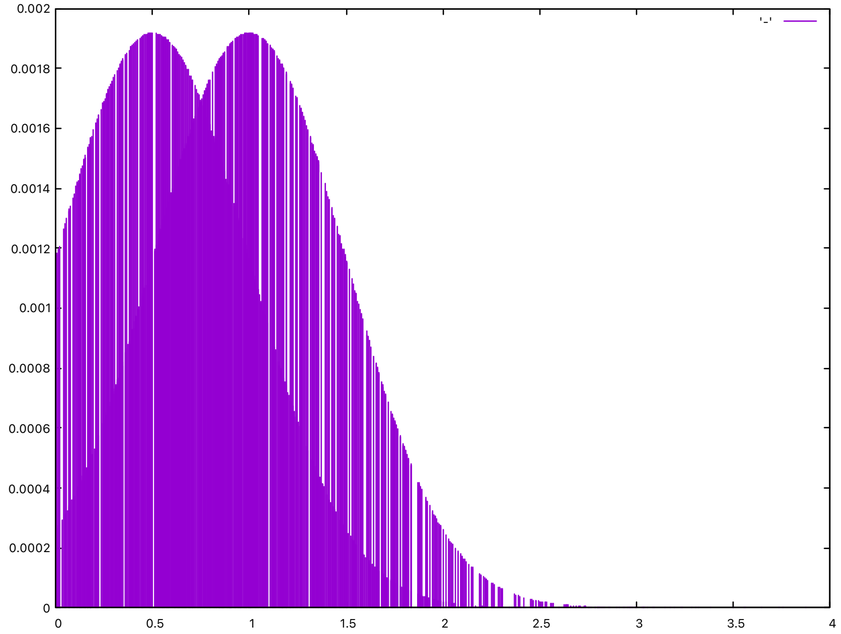
Probabilistic constructs

- `x = sample(d)`: introduce a random variable `x` of distribution `d`
- `observe(d, y)`: condition on the fact that `y` was sampled from `d`
- `infer m y`: compute posterior distribution of `m` given `y`

More general than classic Bayesian Reasoning

```
let rec weird () =
  let b = sample (bernoulli ~p:0.5) in
  let mu = if (b = 1) then 0.5 else 1.0 in
  let theta = sample (gaussian ~mu ~sigma:1.0) in
  if theta > 0. then
    observe (gaussian ~mu ~sigma:0.5) theta;
    theta
  else weird ()

let weird_dist = infer weird ()
```



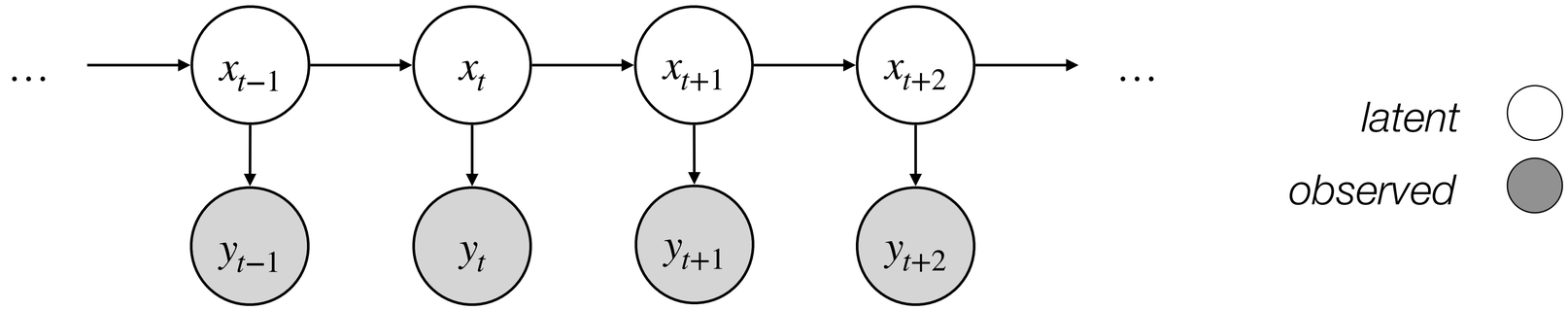
ProbZelus

Reactive Probabilistic Programming

Reactive probabilistic programming

Probabilistic constructs

- `x = sample(d)`: introduce a random variable `x` of distribution `d`
- `observe(d, y)`: condition on the fact that `y` was sampled from `d`
- `infer m y`: compute posterior distribution of `m` given `y`

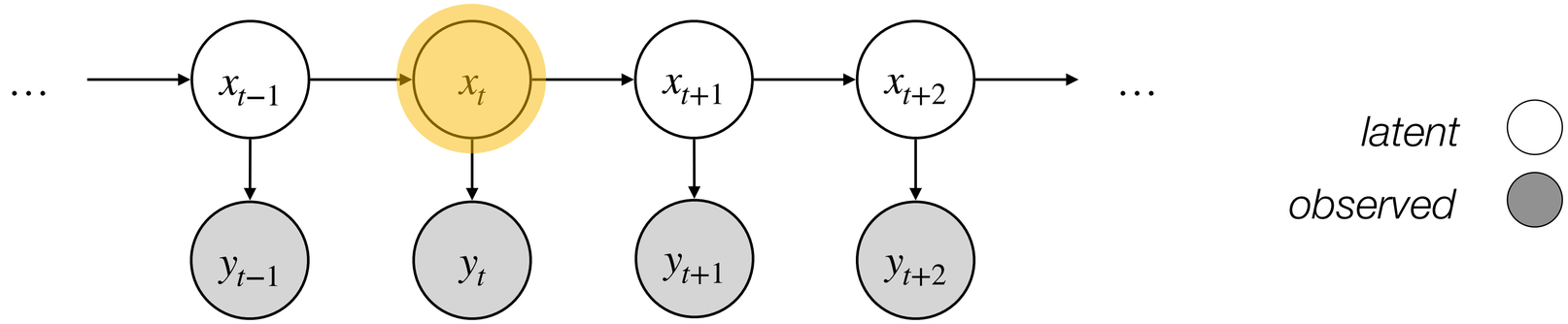



```
let proba tracker (y) = x where
    rec x = x0 → sample(mv_gaussian(f *@ (pre x), q))
    and () = observe(mv_gaussian(h *@ x, r), y)
```

Reactive probabilistic programming

Probabilistic constructs

- `x = sample(d)`: introduce a random variable `x` of distribution `d`
- `observe(d, y)`: condition on the fact that `y` was sampled from `d`
- `infer m y`: compute posterior distribution of `m` given `y`

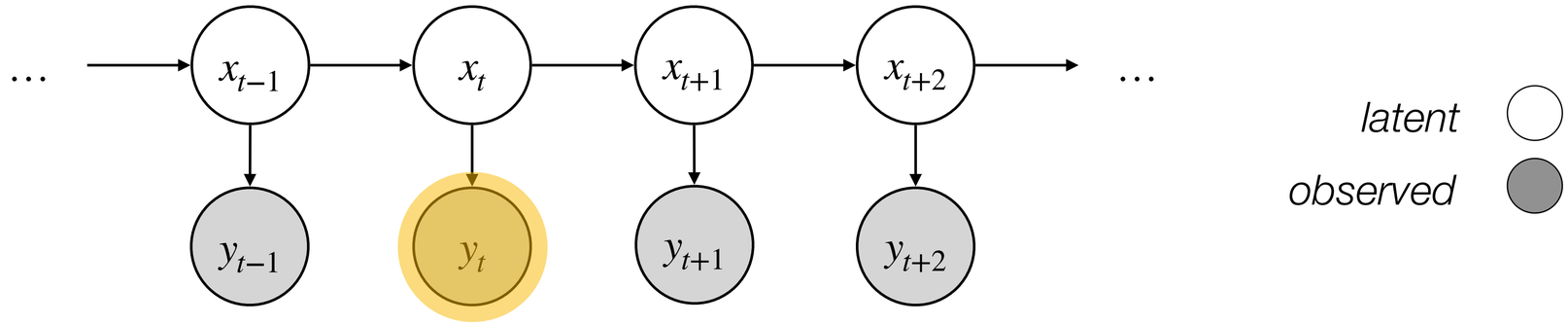



```
let proba tracker (y) = x where
  rec x = x0 → sample(mv_gaussian(f *@ (pre x), q))
  and () = observe(mv_gaussian(h *@ x, r), y)
```

Reactive probabilistic programming

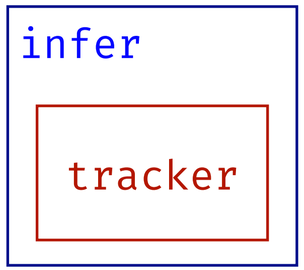
Probabilistic constructs

- `x = sample(d)`: introduce a random variable `x` of distribution `d`
- `observe(d, y)`: condition on the fact that `y` was sampled from `d`
- `infer m y`: compute posterior distribution of `m` given `y`



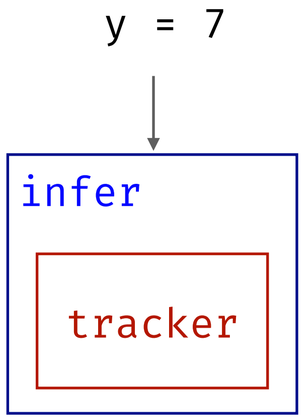

```
let proba tracker (y) = x where
    rec x = x0 → sample(mv_gaussian(f *@ (pre x), q))
  and () = observe(mv_gaussian(h *@ x, r), y)
```

Reactive probabilistic programming



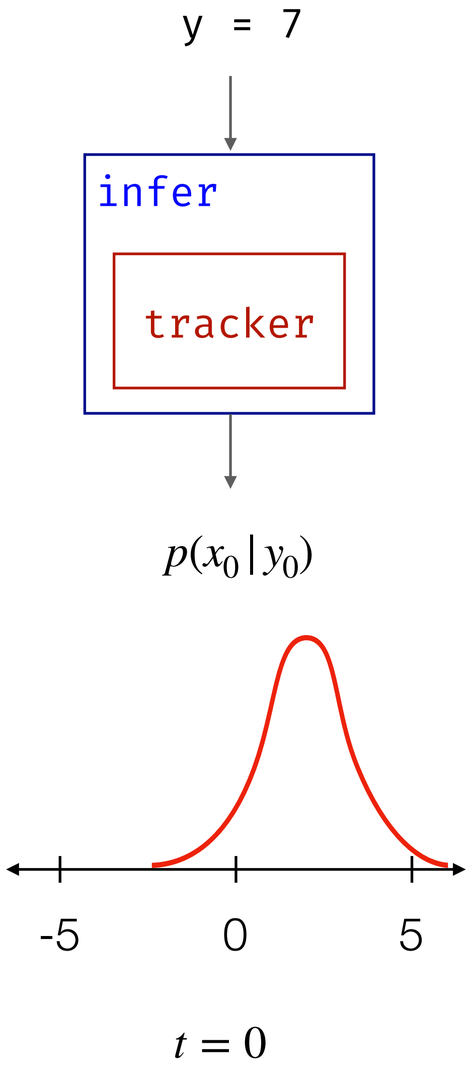
$t = 0$

Reactive probabilistic programming

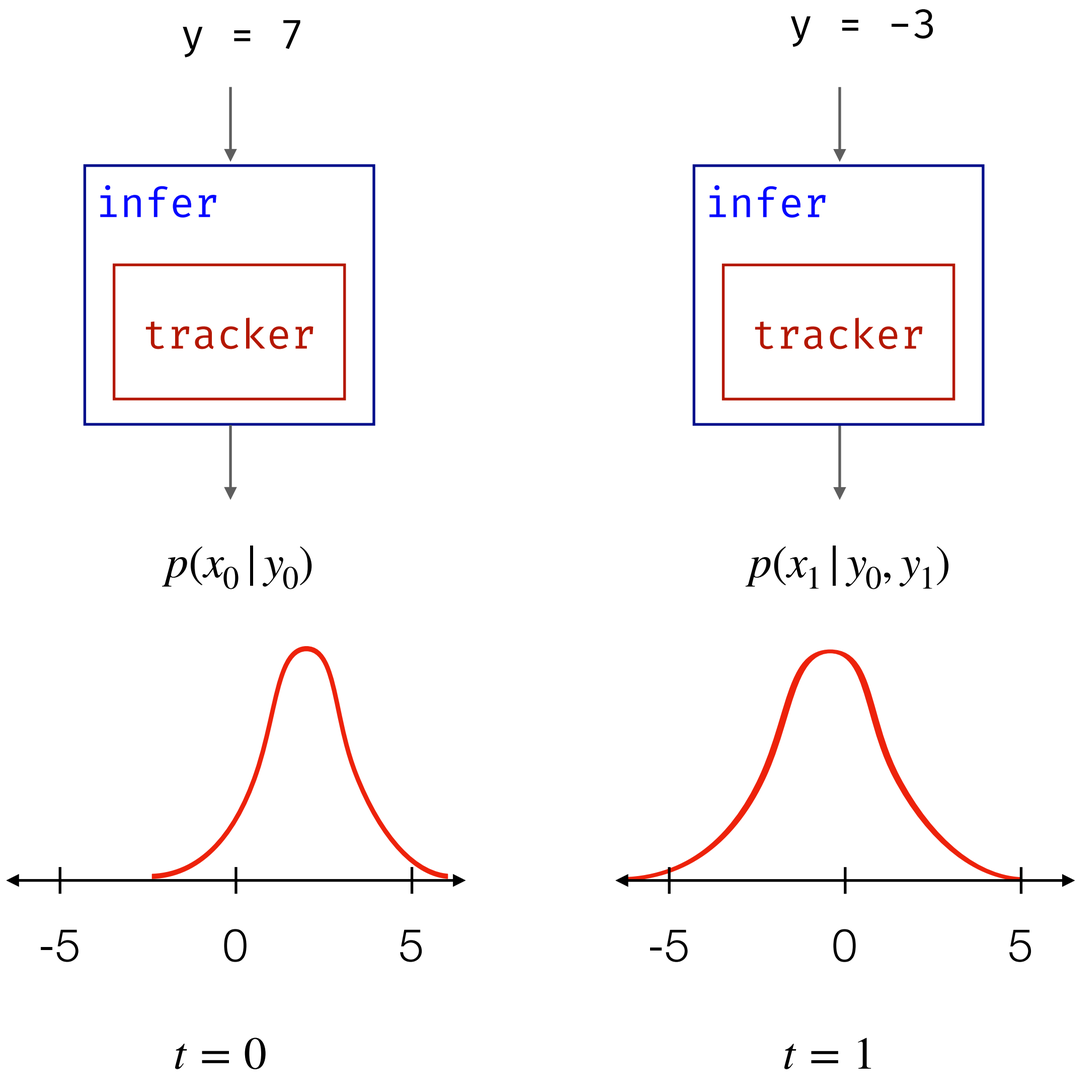


$t = 0$

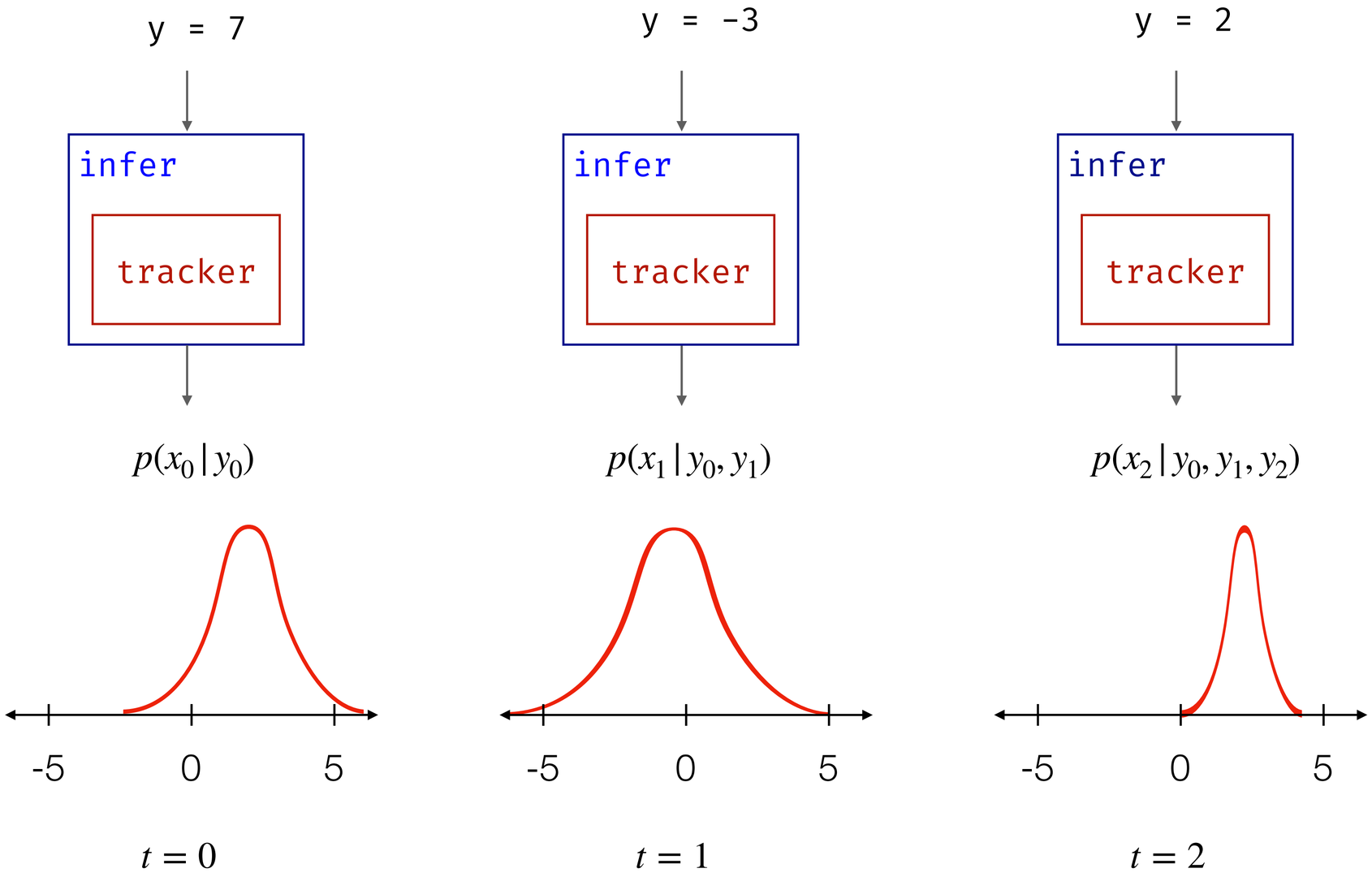
Reactive probabilistic programming



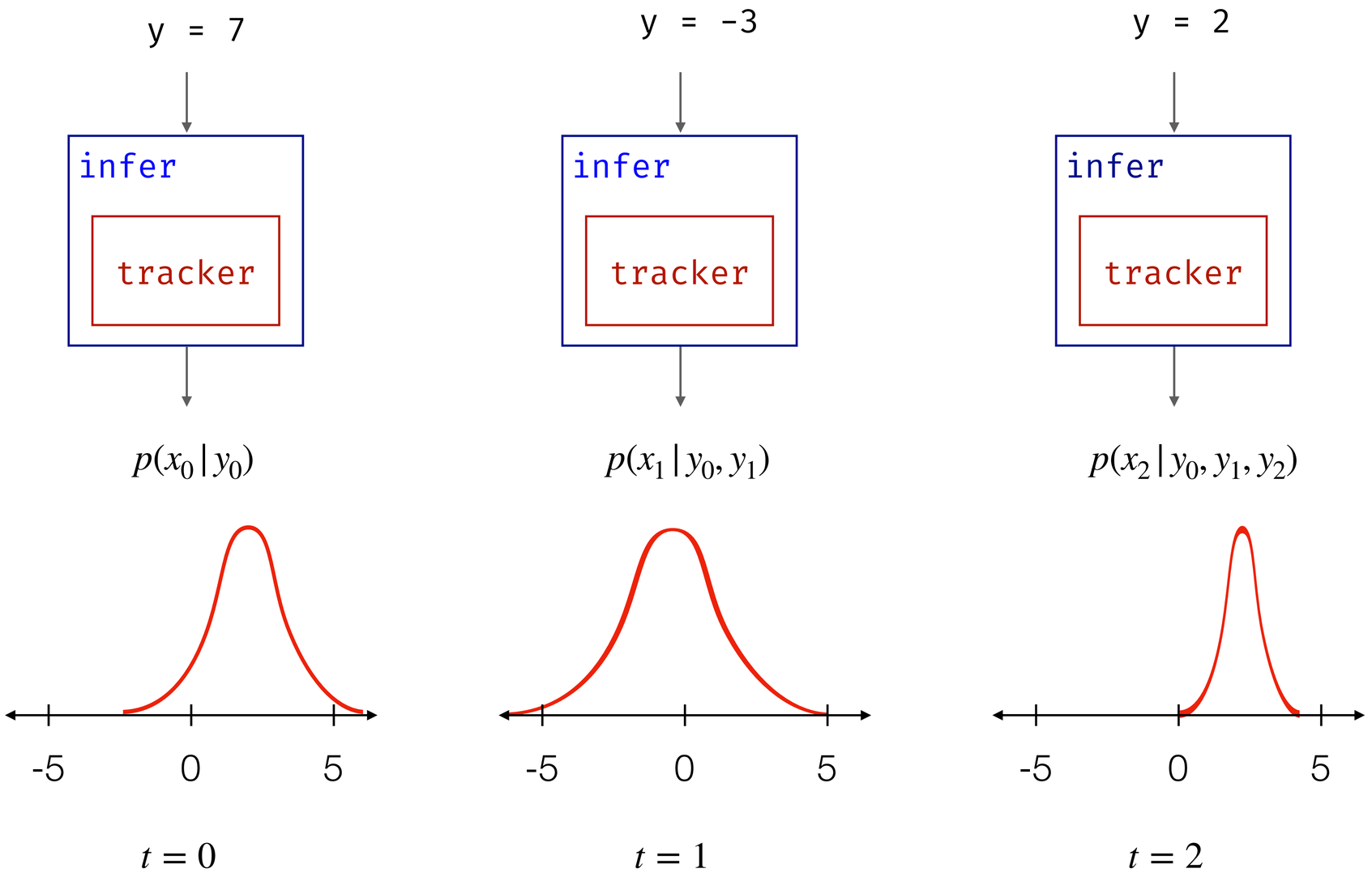
Reactive probabilistic programming



Reactive probabilistic programming



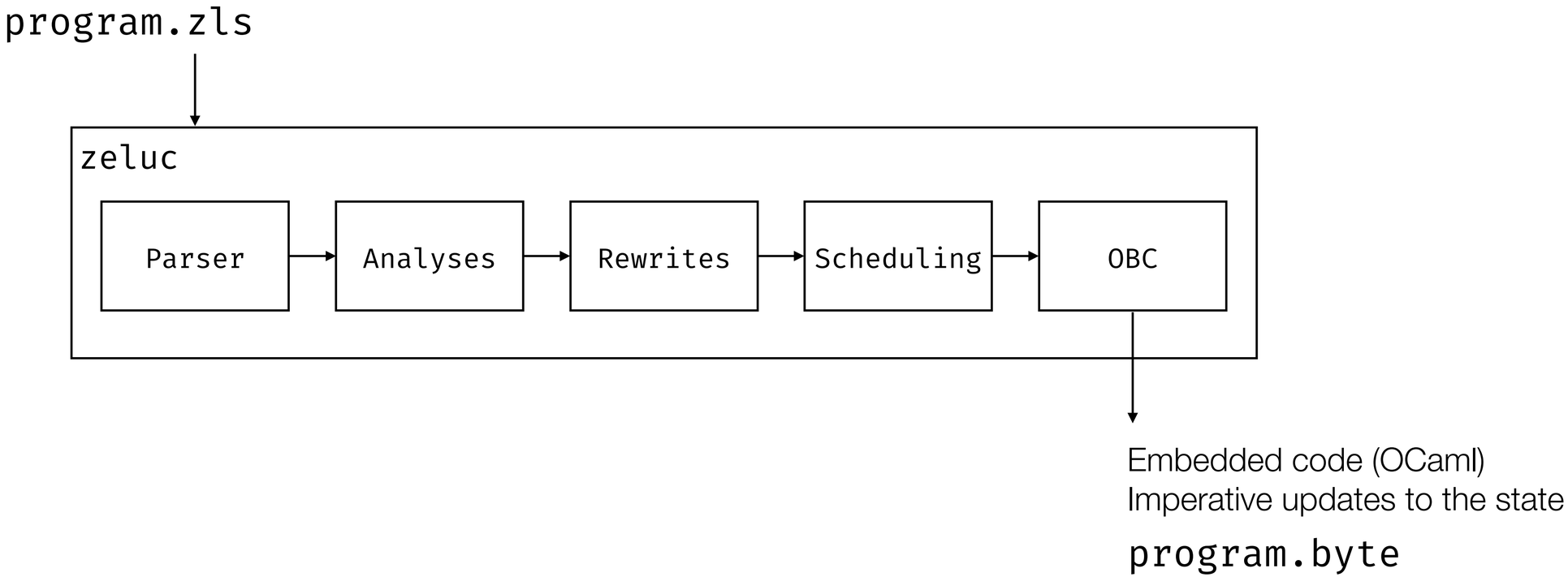
Reactive probabilistic programming



Demo

Part II. Programming (mini) ProbZelus

The Zelus compiler



Bourke, Pouzet 2013

Generated code

```
(* a synchronous stream function with type 'a -D→ 'b *)
(* is represented by an OCaml value of type ('a, 'b) node *)
type ('a, 'b) node =
    Node:
      { alloc : unit → 's; (* allocate the state *)
        step : 's → 'a → 'b; (* compute a step *)
        reset : 's → unit; (* reset/initialize the state *)
      } → ('a, 'b) cnode

(*
  let m = alloc () in
  reset m;
  while true do
    let o = step m i in …
  done
*)
```


https://github.com/INRIA/zelus/blob/main/lib/std/ztypes.ml

Mini ProbZelus runtime

```
type 'a distribution
type prob (* instantiated by infer *)

val sample : prob * 'a distribution -AD→ 'a
val observe : prob * 'a distribution * 'a -AD→ unit

(* if m : 'a → 'b, and  y : 'a, infer m y : 'b distribution *)
val infer : ((prob * 'a) -D→ 'b) -S→ 'a -D→ 'b distribution
```

Mini ProbZelus runtime

```
type 'a distribution
type prob (* instantiated by infer *)

val sample : prob * 'a distribution -AD→ 'a
val observe : prob * 'a distribution * 'a -AD→ unit

(* if m : 'a → 'b, and  y : 'a, infer m y : 'b distribution *)
val infer : ((prob * 'a) -D→ 'b) -S→ 'a -D→ 'b distribution
```

```
let proba tracker (y) = x where
  rec x = x0 → sample(mv_gaussian(  ...
  and () = observe(mv_gaussian(  ...

infer tracker y
```

Mini ProbZelus runtime

```
type 'a distribution
type prob (* instantiated by infer *)

val sample : prob * 'a distribution -AD-> 'a
val observe : prob * 'a distribution * 'a -AD-> unit

(* if m : 'a -> 'b, and  y : 'a, infer m y : 'b distribution *)
val infer : ((prob * 'a) -D-> 'b) -S-> 'a -D-> 'b distribution
```

```
let proba tracker (y) = x where
  rec x = x0 -> sample(mv_gaussian(  ...
  and () = observe(mv_gaussian(  ...

infer tracker y
```

```
let node tracker (prob, y) = x where
  rec x = x0 -> sample(prob, mv_gaussian(  ...
  and () = observe(prob, mv_gaussian(  ...

infer tracker y
```

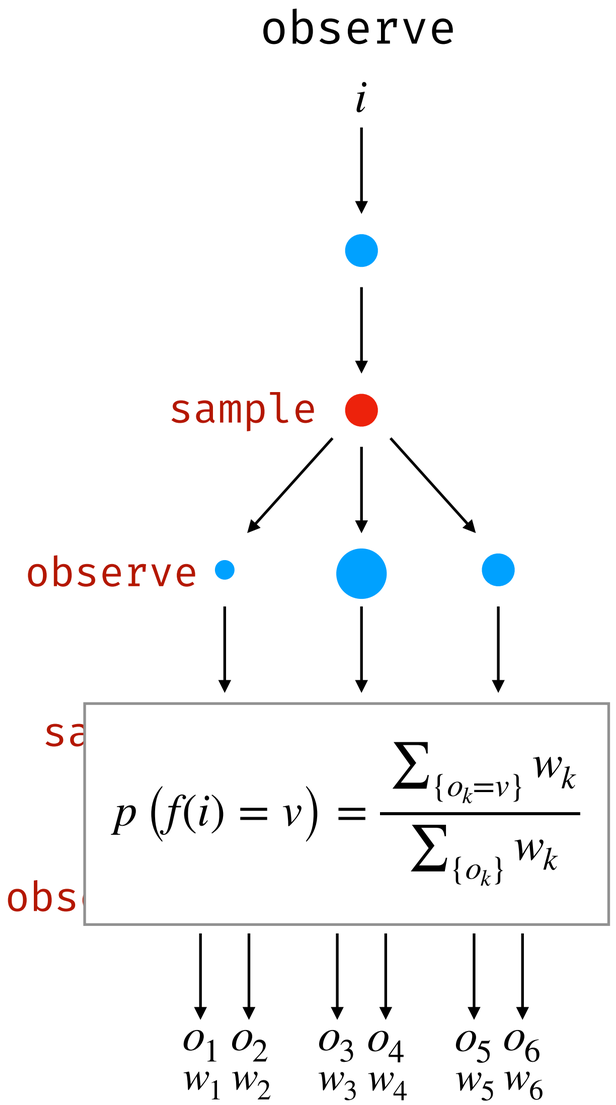
Streaming inference

Reactive Probabilistic Programming

Importance sampling

Approximate inference algorithm

- Run a set of n independent executions
- `sample(d)`: draw a sample from a distribution
- `observe(d, x)`: add `logpdf d x` to the current score
- `infer`: gather (values, scores) to approximate the distribution



Importance sampling

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

Importance sampling

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

Importance sampling

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10)) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

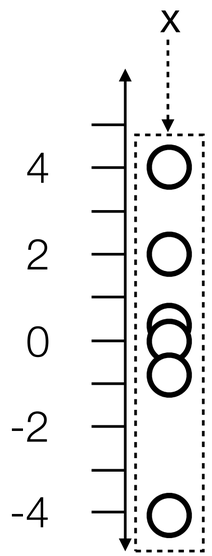
```
sample (gaussian (0, 10))
```

Importance sampling

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10)) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

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

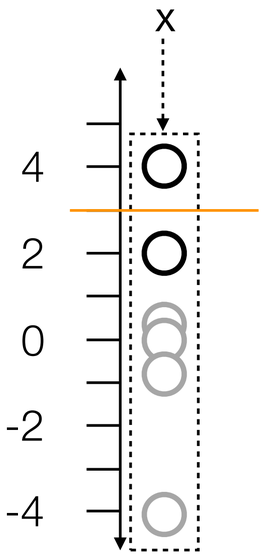


Importance sampling

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

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

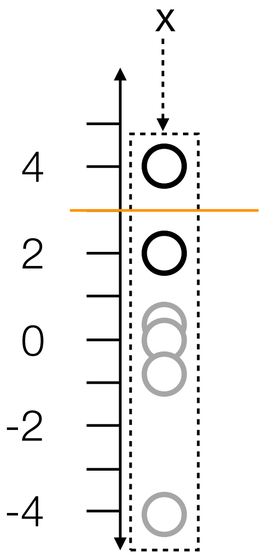


Importance sampling

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

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



$t = 1$

Importance sampling

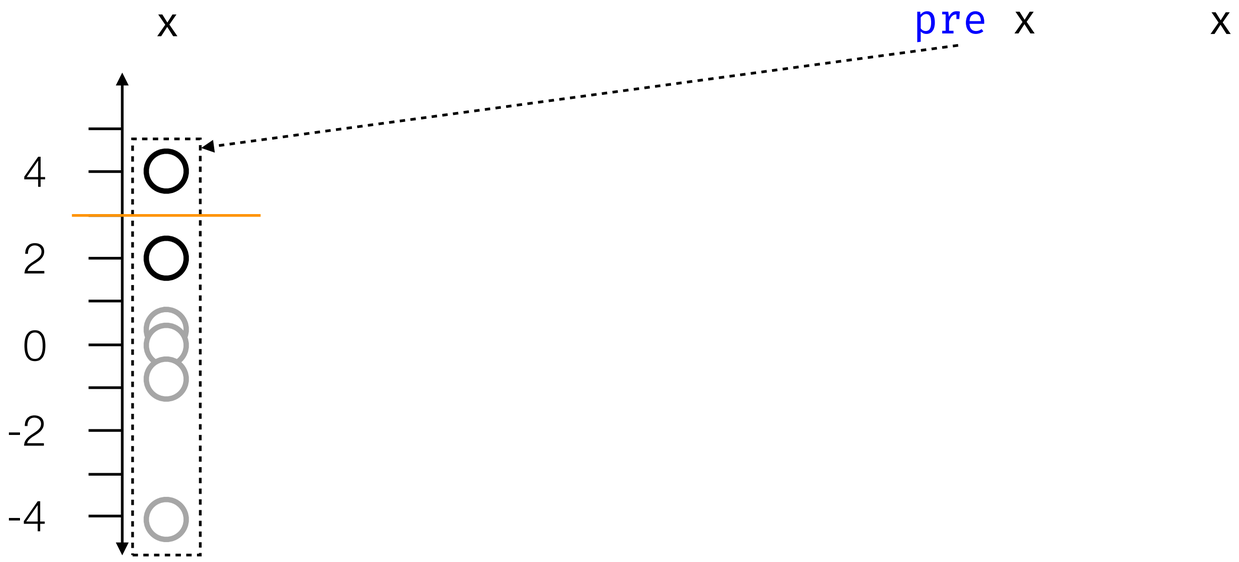
```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

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

$t = 1$

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



Importance sampling

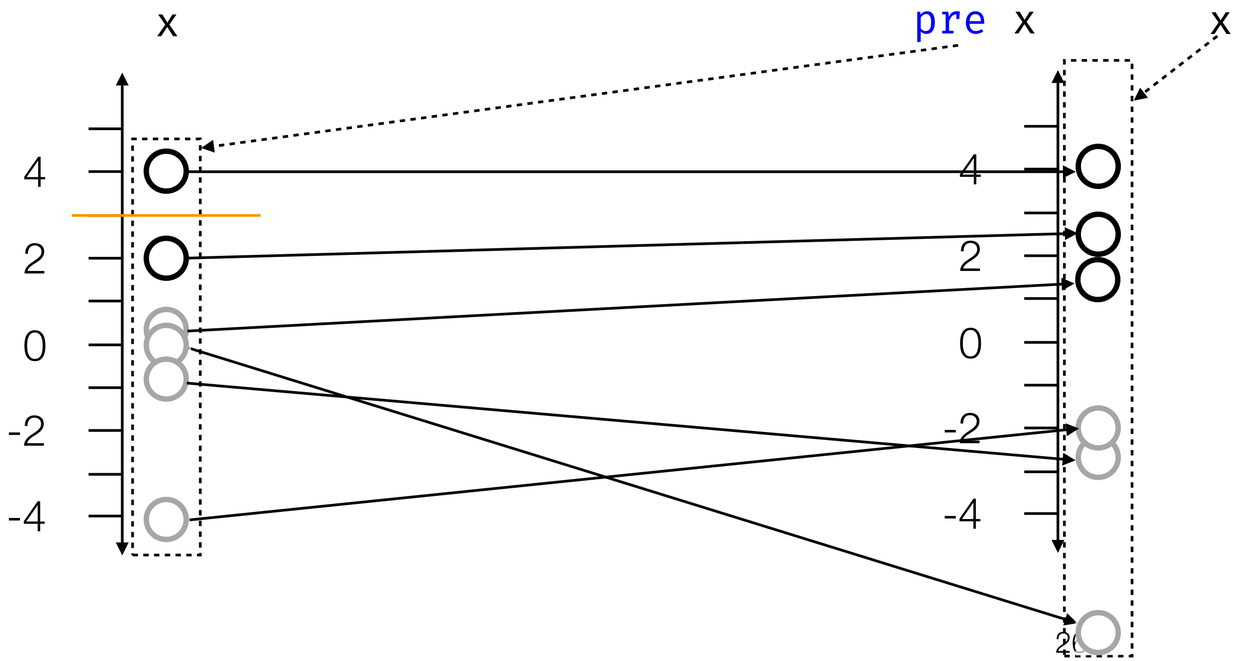
```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

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

$t = 1$

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



Importance sampling

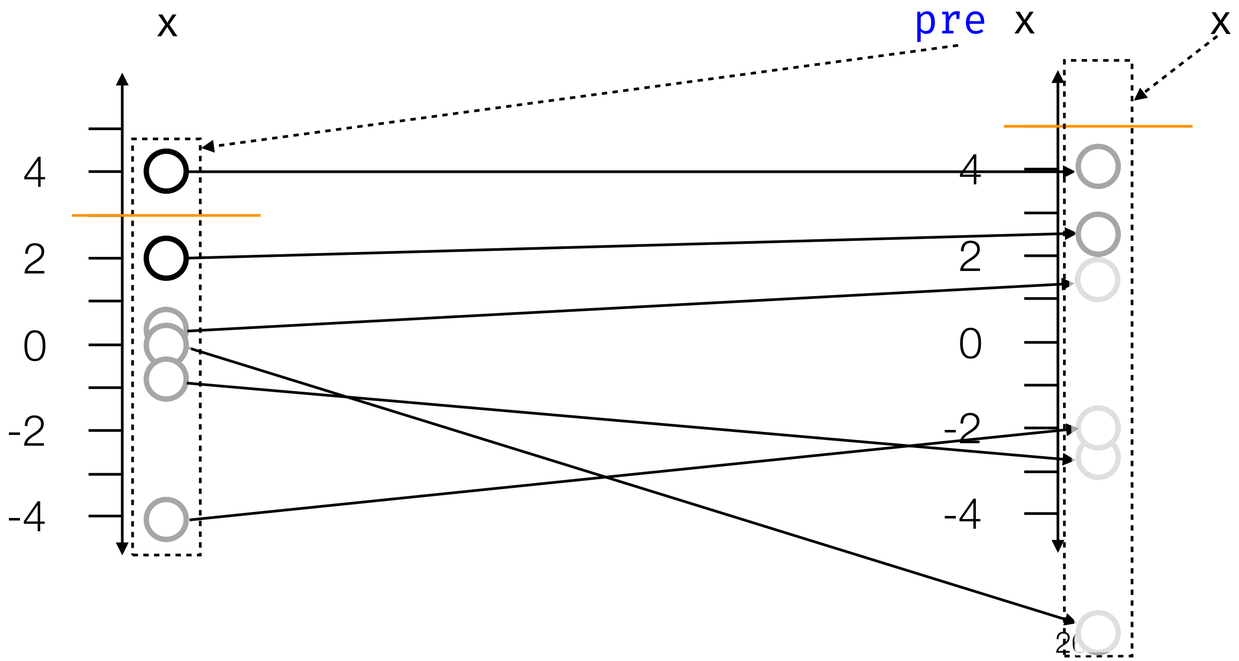
```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

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

$t = 1$

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



Importance sampling

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

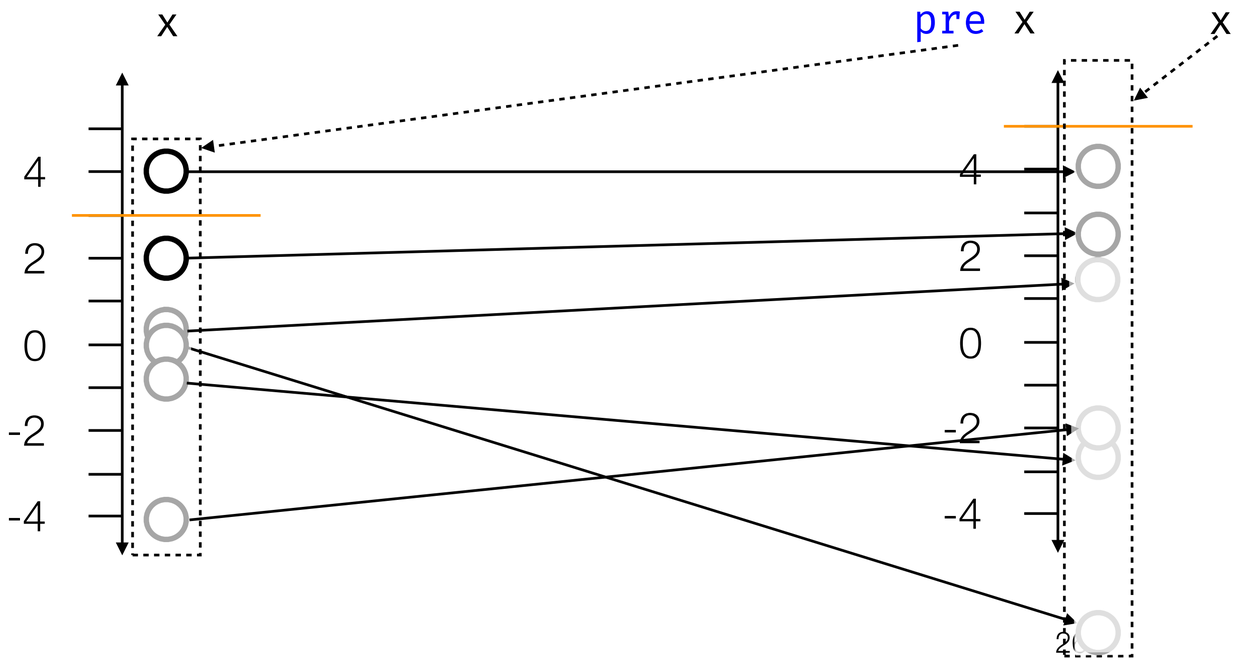
```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```

$t = 1$

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

$t = 2$

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



…

Importance sampling

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

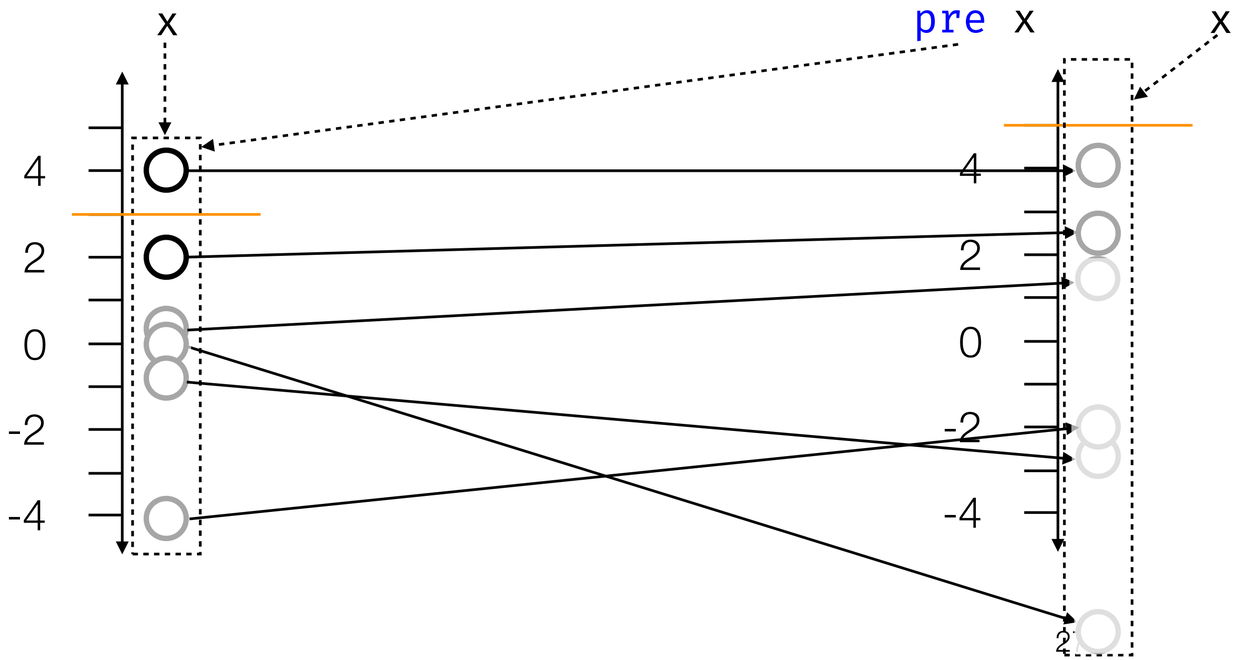
```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```

$t = 1$

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

$t = 2$

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



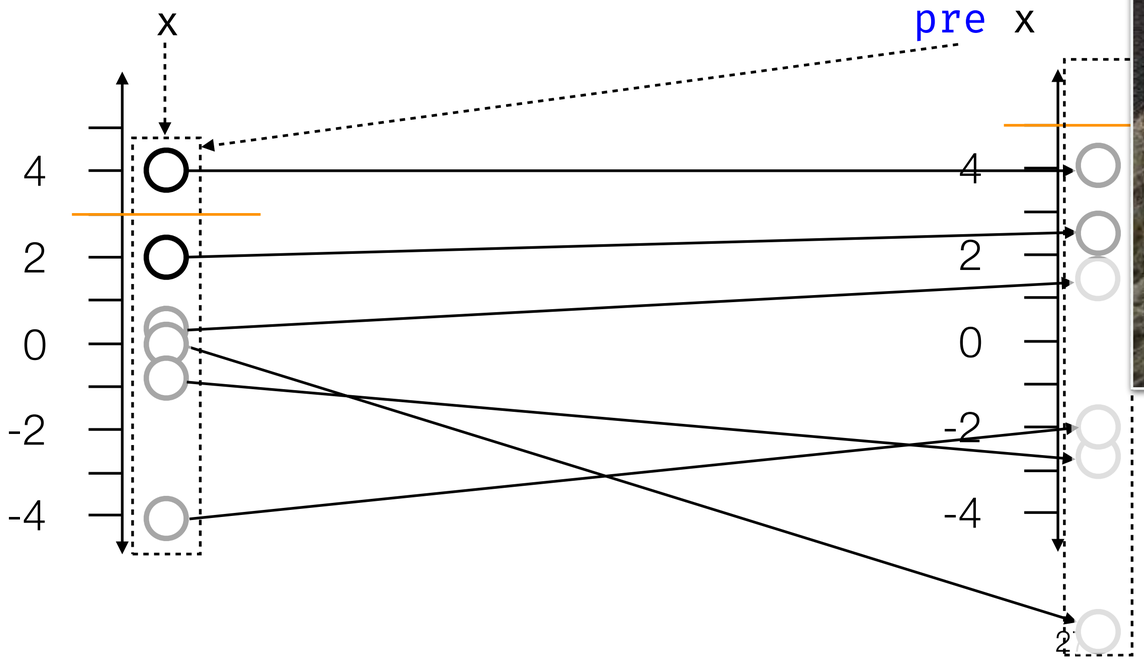
Sven Sachsalber, *Looking for a needle in the Haystack*, 2014

Importance sampling

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

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



$t = 1$

```
sample (gaussian
observe (gaussian
```


$t = 2$

1))

Sven Sachsalber, *Looking for a needle in the Haystack*, 2014

Particle filter

Approximate inference algorithm : importance sampling, but...

- Add a resampling step at each `observe`
- Compute the score of the particles to compute a distribution
- Re-sample a new set of particles from this distribution

How can we duplicate a particle during execution?

- Continuation Passing Style (CPS)?
- Clone the memory state?

Particle filter

Approximate inference algorithm : importance sampling, but...

- Add a resampling step at each `observe`
- Compute the score of the particles to compute a distribution
- Re-sample a new set of particles from this distribution

How can we duplicate a particle during execution?

- Continuation Passing Style (CPS)?
- Clone the memory state?

```
(* the same with a method copy *)
type ('a, 'b) cnode =
    Cnode:
      { alloc : unit → 's; (* allocate the state *)
        copy : 's → 's → unit; (* copy the source into the destination *)
        step : 's → 'a → 'b; (* compute a step *)
        reset : 's → unit; (* reset/initialize the state *)
      } → ('a, 'b) cnode
```


https://github.com/INRIA/zelus/blob/main/lib/std/ztypes.ml

Particle filter

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

Particle filter

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

Particle filter

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10)) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$$t = 0$$

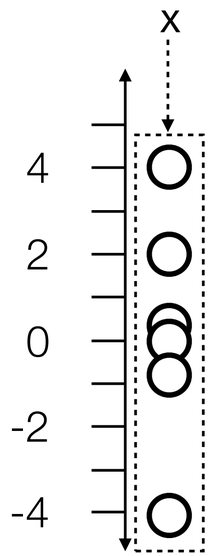
```
sample (gaussian (0, 10))
```

Particle filter

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10)) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

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

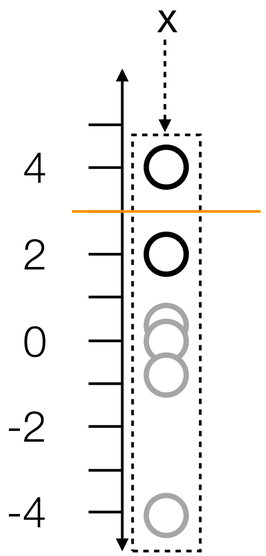


Particle filter

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

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

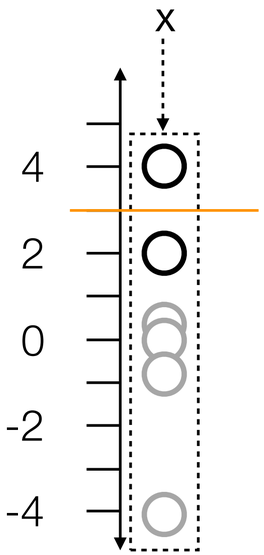


Particle filter

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

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



$t = 1$

Particle filter

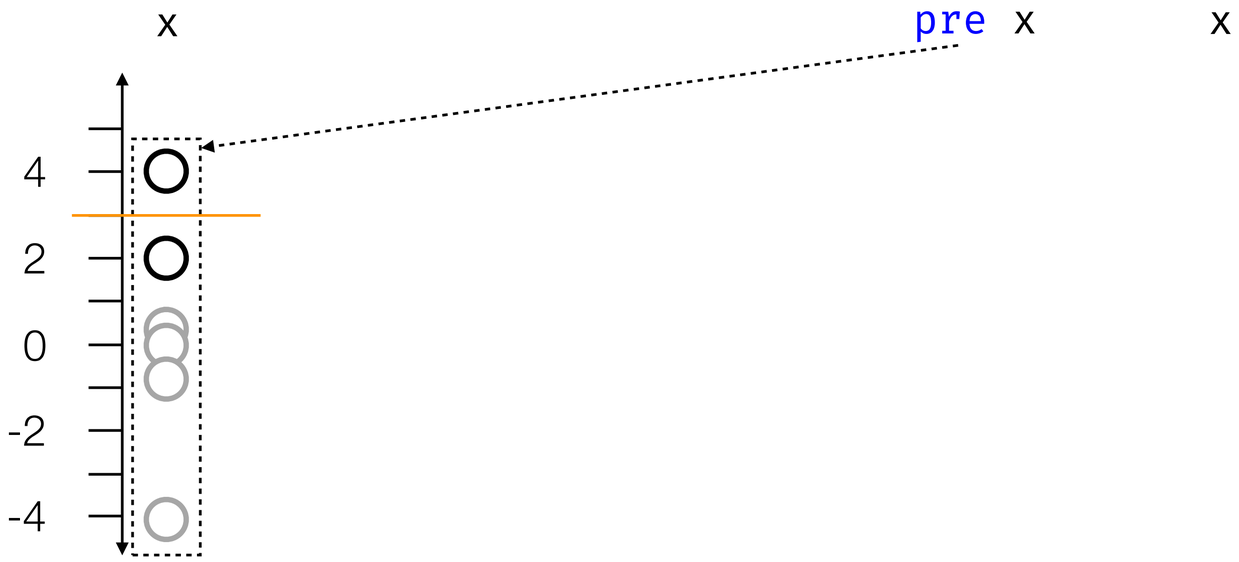
```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

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

$t = 1$

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



Particle filter

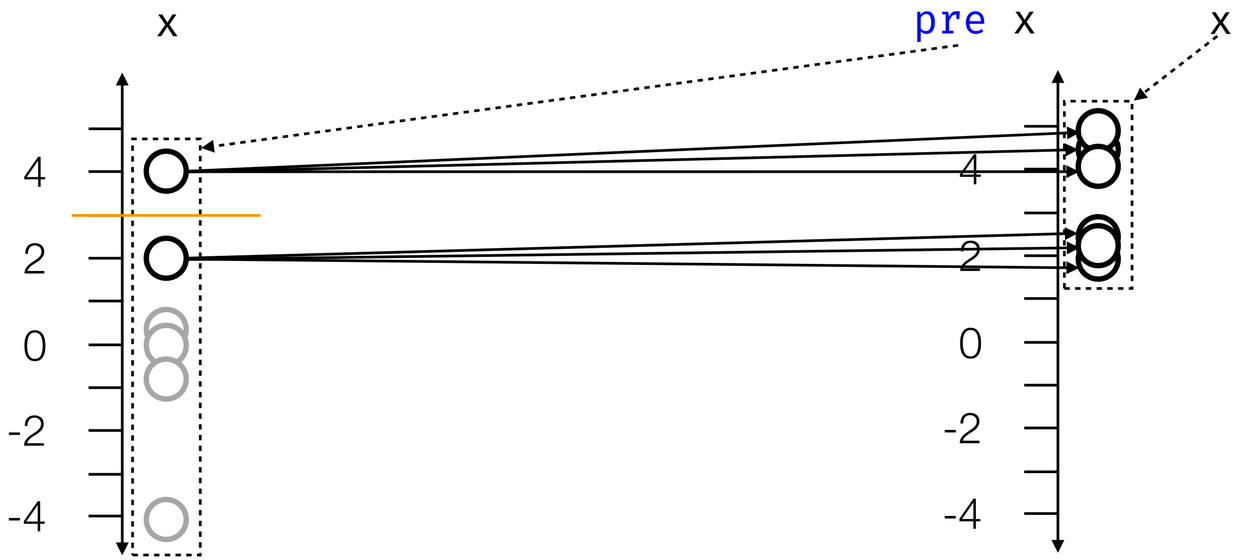
```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

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

$t = 1$

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



Particle filter

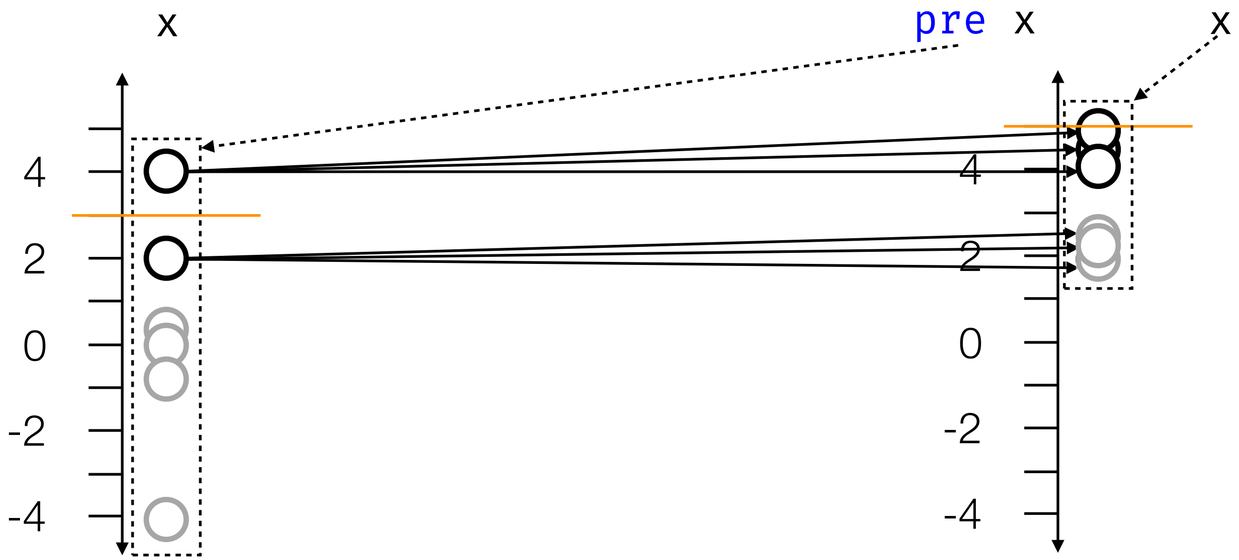
```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

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

$t = 1$

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



Particle filter

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

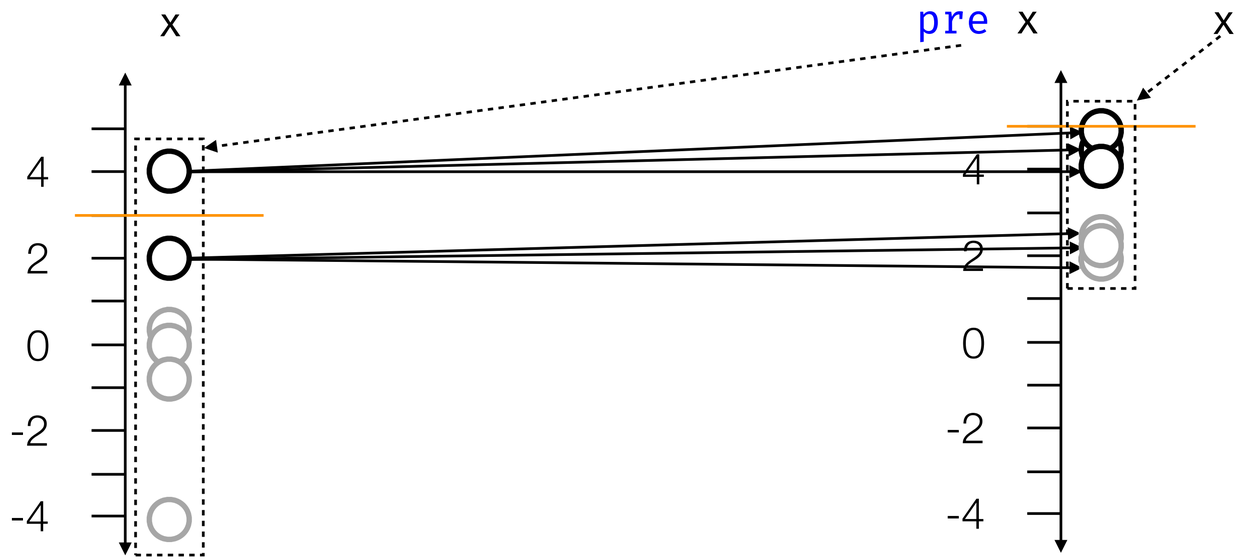
```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```

$t = 1$

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

$t = 2$

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



...

Demo

Delayed sampling

Simple Particles Filters can be impractical

- Require lot of computing power
- Poor approximation

Exact inference is often possible

Semi-Symbolic inference

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

Main idea

- Keep track of conjugacy relationships
- Incorporate observations analytically
- Sample only when necessary

Murray et al. 2018

Delayed sampling

Simple Particles Filters can be impractical

- Require lot of computing power
- Poor approximation

Exact inference is often possible

Semi-Symbolic inference

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

Main idea

- Keep track of conjugacy relationships
- Incorporate observations analytically
- Sample only when necessary

Example: Conjugate Gaussians

$$x \sim \mathcal{N}(\mu_0, \sigma_0)$$
$$y \sim \mathcal{N}(x, \sigma)$$

$$x \,|\, (y = v) \sim \mathcal{N}(\mu_1, \sigma_1)$$

$$\mu_1 = \left(\frac{1}{\sigma_0^2} + \frac{1}{\sigma^2}\right)^{-1} \left(\frac{\mu_0}{\sigma_0^2} + \frac{v}{\sigma^2}\right)$$

$$\sigma_1 = \left(\frac{1}{\sigma_0^2} + \frac{1}{\sigma^2}\right)^{-2}$$

Murray et al. 2018

Delayed sampling

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```


Murray et al. 2018

Delayed sampling

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

Murray et al. 2018

Delayed sampling

```
let proba tracker (y) = x where
  rec x   = sample (gaussian (0, 10)) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

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

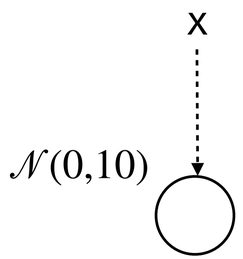

Murray et al. 2018

Delayed sampling

```
let proba tracker (y) = x where
  rec x   = sample (gaussian (0, 10)) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

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



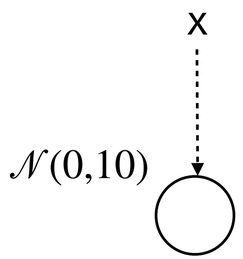
Murray et al. 2018

Delayed sampling

```
let proba tracker (y) = x where
  rec x   = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

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



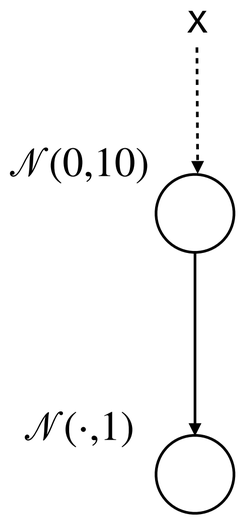
Murray et al. 2018

Delayed sampling

```
let proba tracker (y) = x where
  rec x   = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

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



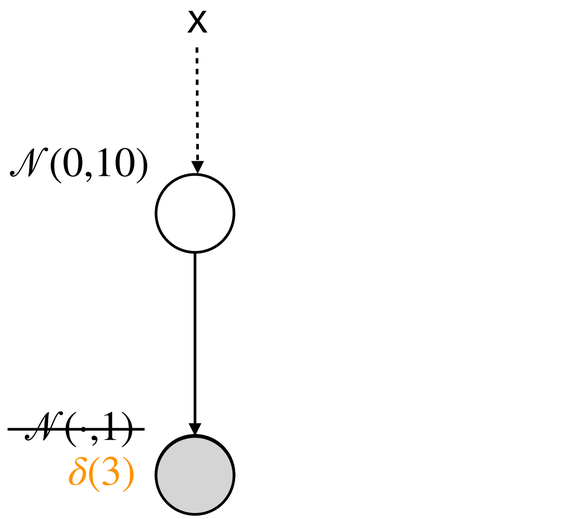
Murray et al. 2018

Delayed sampling

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

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



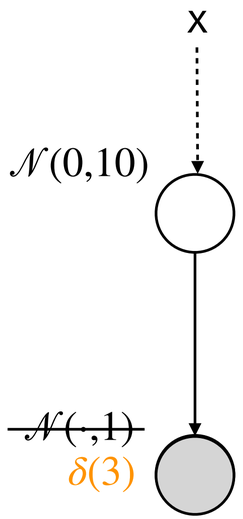
Murray et al. 2018

Delayed sampling

```
let proba tracker (y) = x where
  rec x   = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

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



Example: 2 Gaussians

$x \sim \mathcal{N}(\mu_0, \sigma_0)$
$y \sim \mathcal{N}(x, \sigma)$

$x \,|\, (y = v) \sim \mathcal{N}(\mu_1, \sigma_1)$

$$\mu_1 = \left(\frac{1}{\sigma_0^2} + \frac{1}{\sigma^2}\right)^{-1} \left(\frac{\mu_0}{\sigma_0^2} + \frac{v}{\sigma^2}\right)$$

$$\sigma_1 = \left(\frac{1}{\sigma_0^2} + \frac{1}{\sigma^2}\right)^{-2}$$

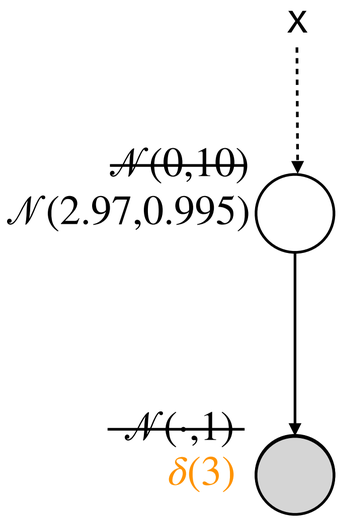
Murray et al. 2018

Delayed sampling

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

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



Example: 2 Gaussians

$x \sim \mathcal{N}(\mu_0, \sigma_0)$
$y \sim \mathcal{N}(x, \sigma)$

$x \,|\, (y = v) \sim \mathcal{N}(\mu_1, \sigma_1)$

$$\mu_1 = \left(\frac{1}{\sigma_0^2} + \frac{1}{\sigma^2}\right)^{-1} \left(\frac{\mu_0}{\sigma_0^2} + \frac{v}{\sigma^2}\right)$$

$$\sigma_1 = \left(\frac{1}{\sigma_0^2} + \frac{1}{\sigma^2}\right)^{-2}$$

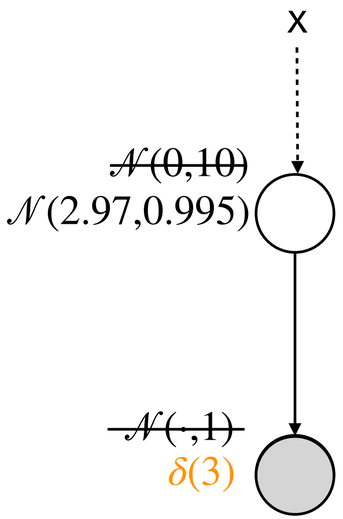
Murray et al. 2018

Delayed sampling

```
let proba tracker (y) = x where
  rec x   = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

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



$t = 1$

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


Murray et al. 2018

Delayed sampling

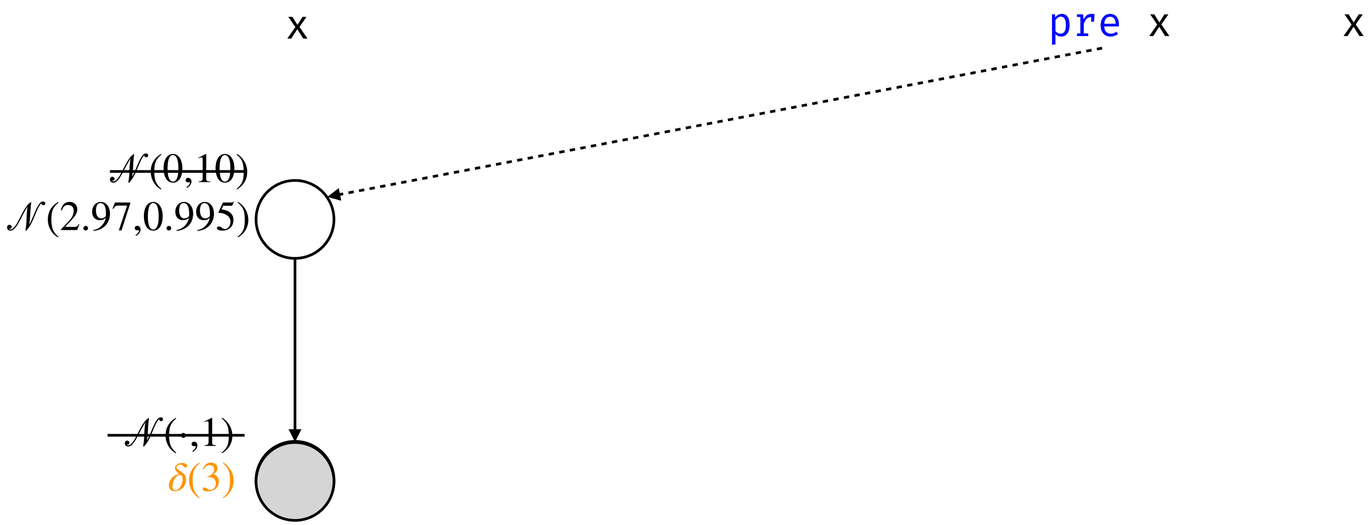
```
let proba tracker (y) = x where
  rec x   = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

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

$t = 1$

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



Murray et al. 2018

Delayed sampling

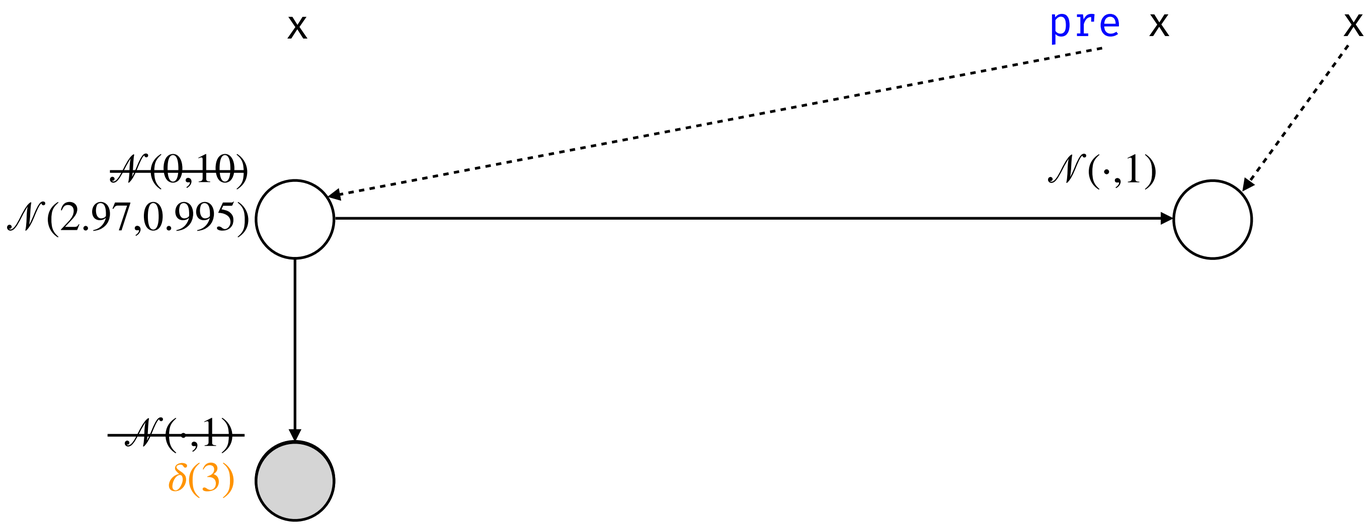
```
let proba tracker (y) = x where
  rec x   = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

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

$t = 1$

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



Murray et al. 2018

Delayed sampling

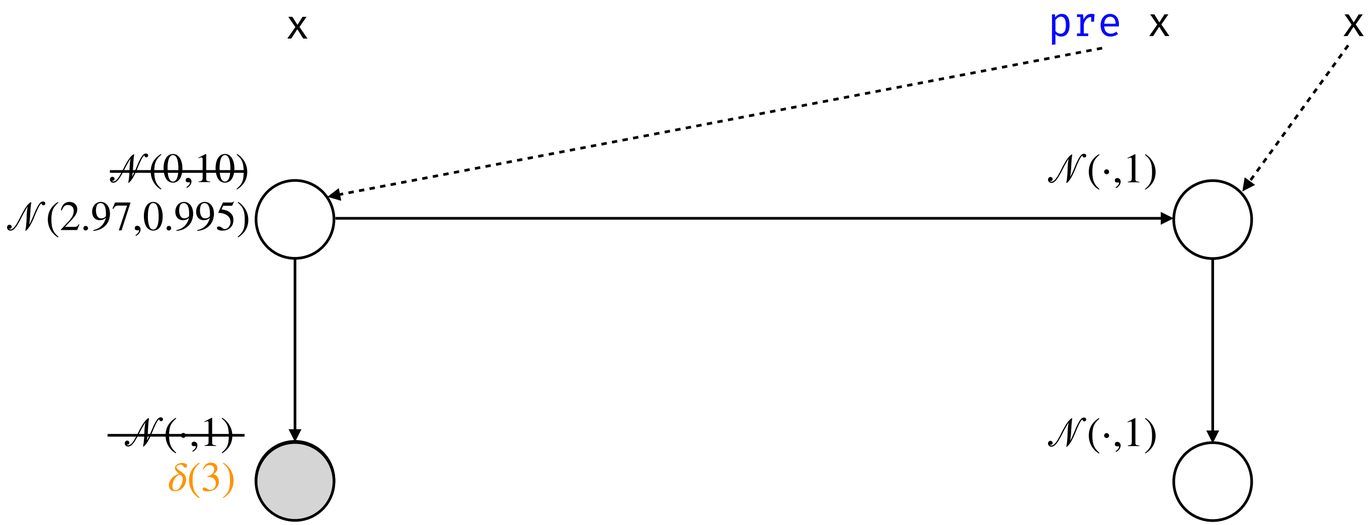
```
let proba tracker (y) = x where
  rec x   = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

```
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

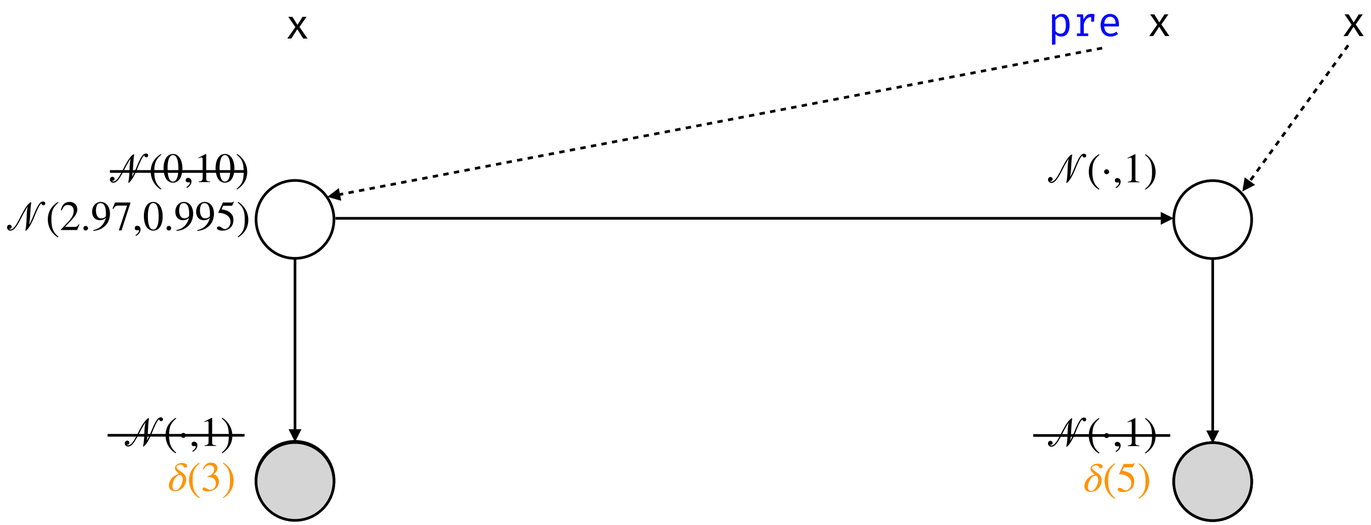
```
let proba tracker (y) = x where
  rec x   = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

```
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

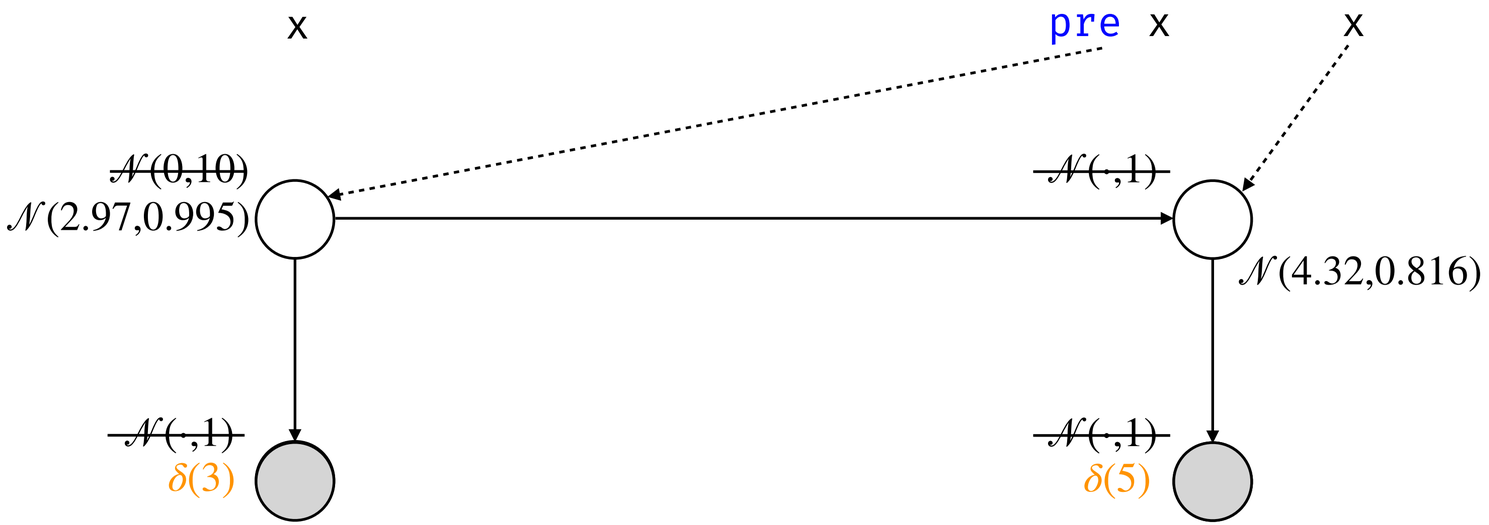
```
let proba tracker (y) = x where
  rec x   = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

```
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 (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

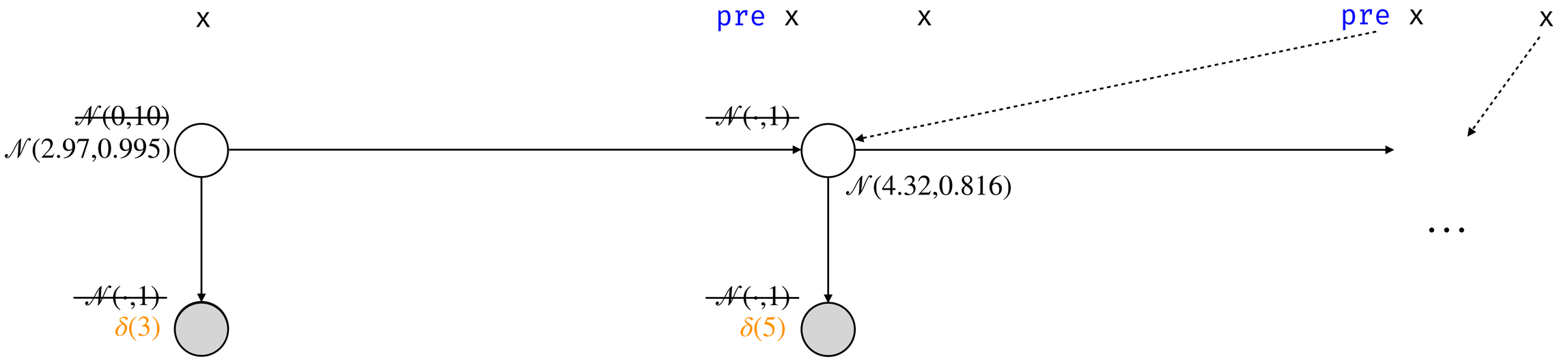
```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```

$t = 1$

```
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 (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

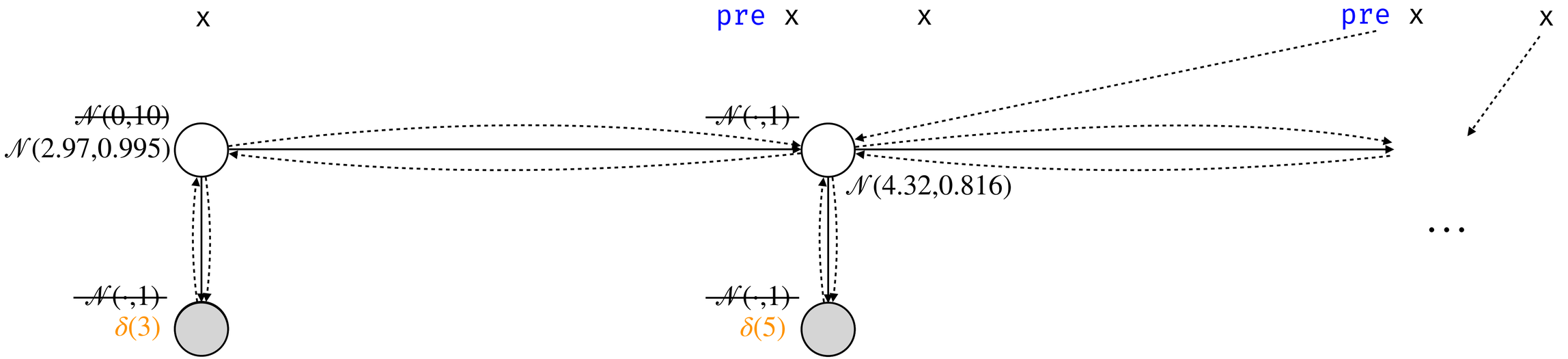
```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```

$t = 1$

```
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

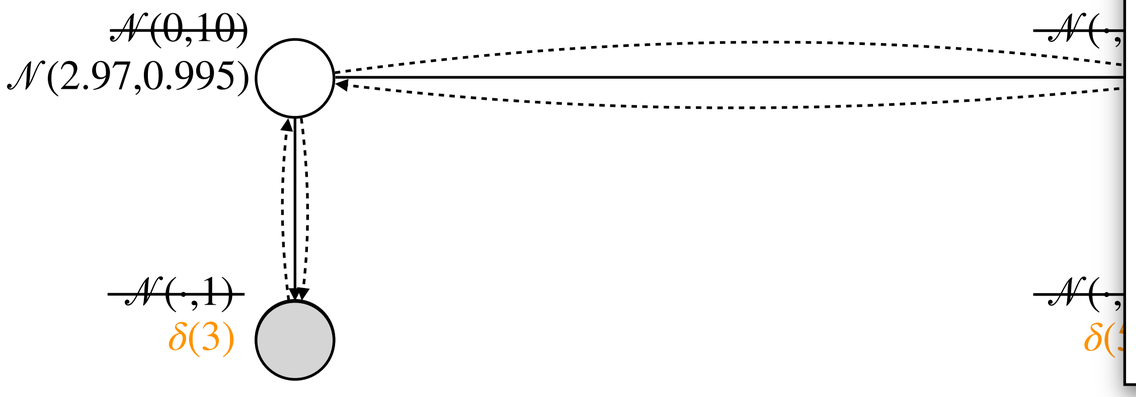
Delayed sampling

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

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

x




```
sample (gaus
observe (gau
```

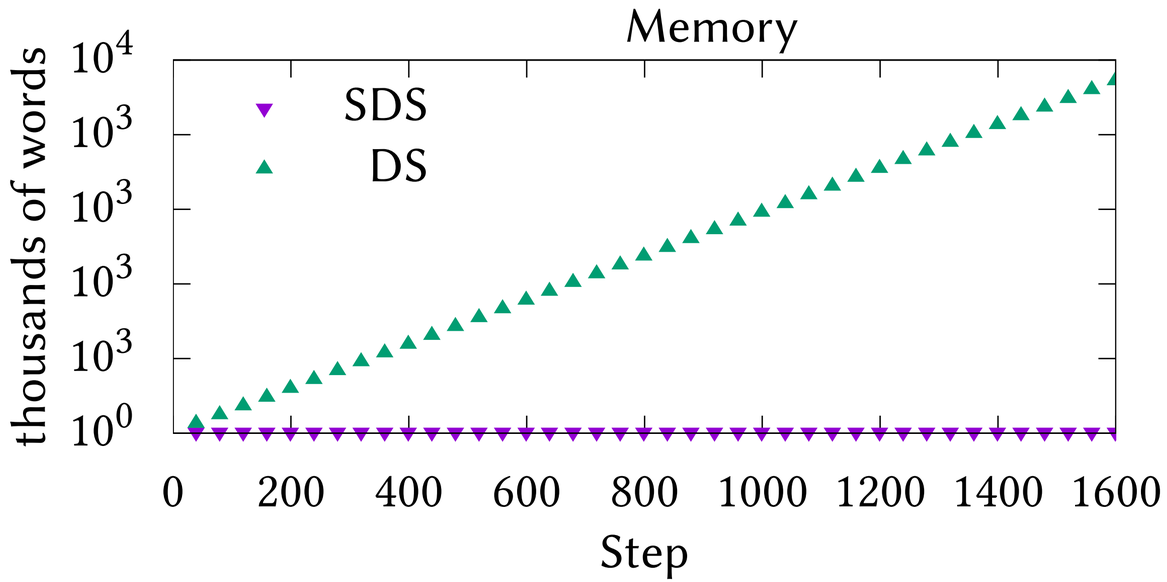
pre

$\mathcal{N}(\cdot,$

$\mathcal{N}(\cdot,$

$\delta($

Unbounded resources



Murray et al. 2018

Delayed sampling

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

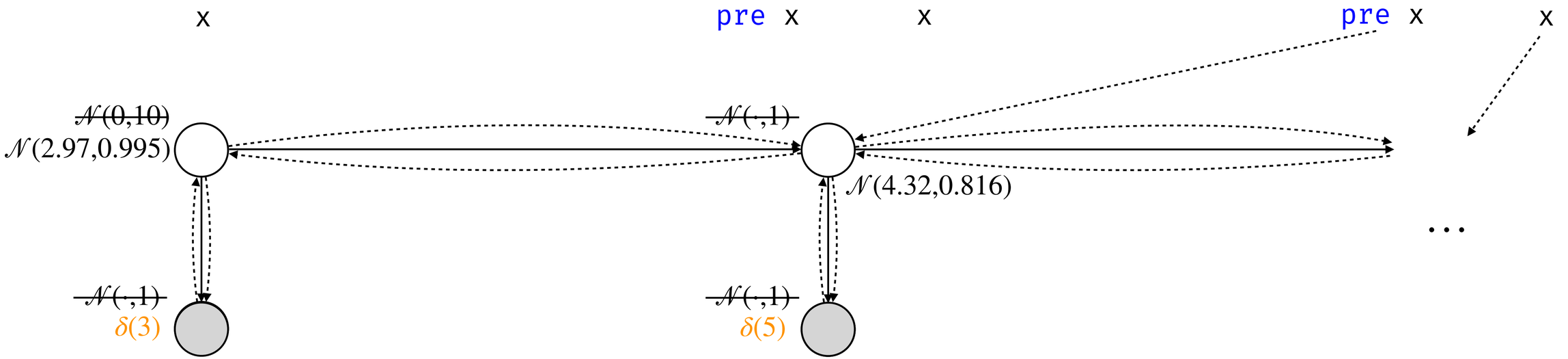
```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```

$t = 1$

```
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

Streaming Delayed sampling

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

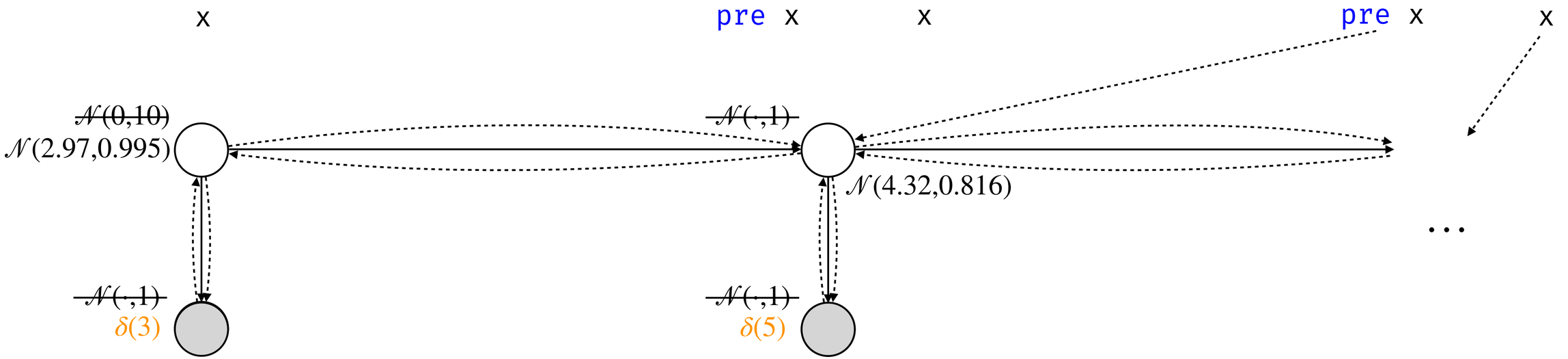
```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```

$t = 1$

```
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 (y) = x where
  rec x   = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

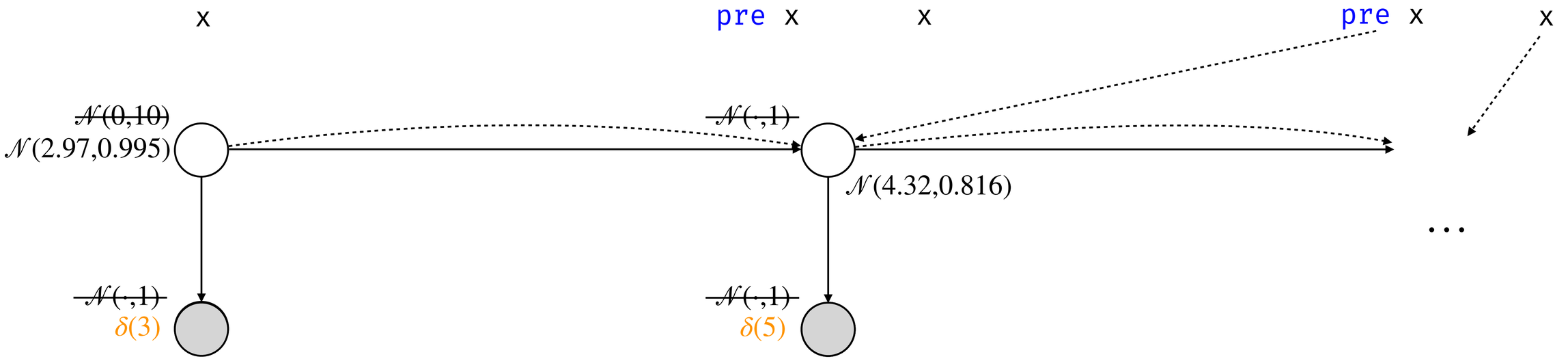
```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```

$t = 1$

```
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 (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

$t = 0$

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

x

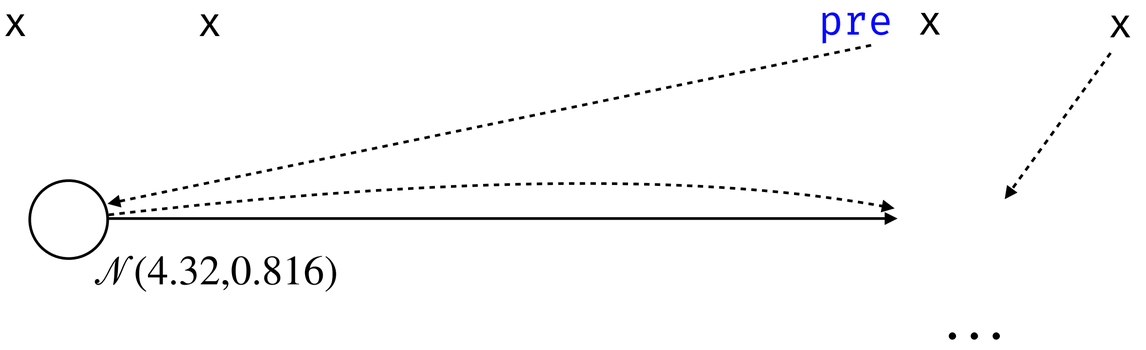
$t = 1$

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

pre x x

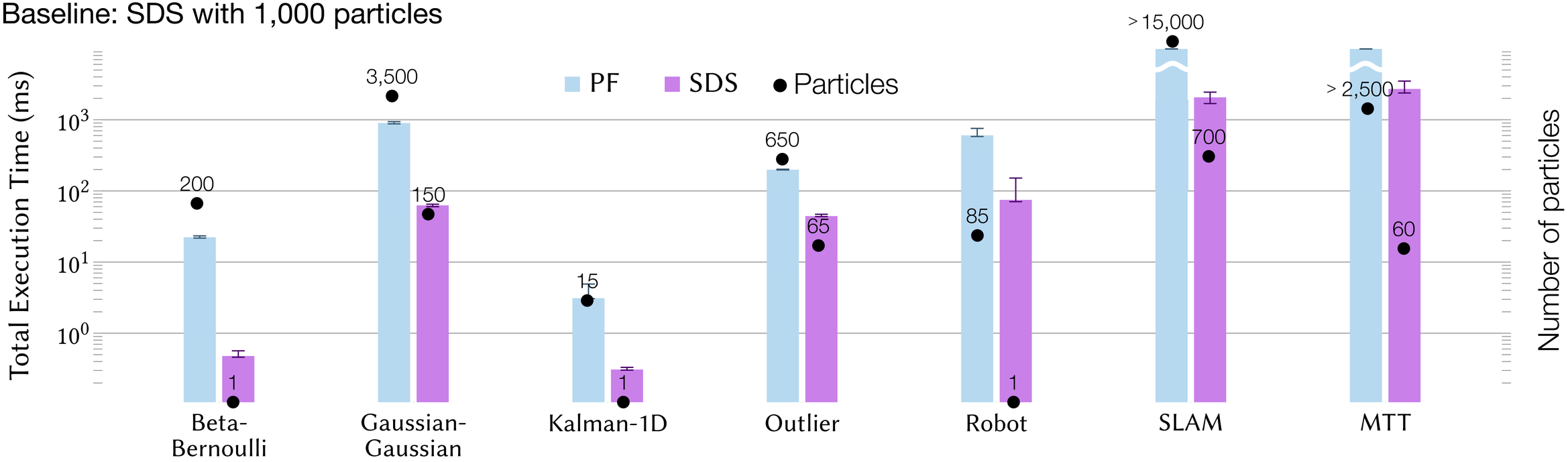
$t = 2$

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



Benchmarks

Baseline: SDS with 1,000 particles



Conclusions

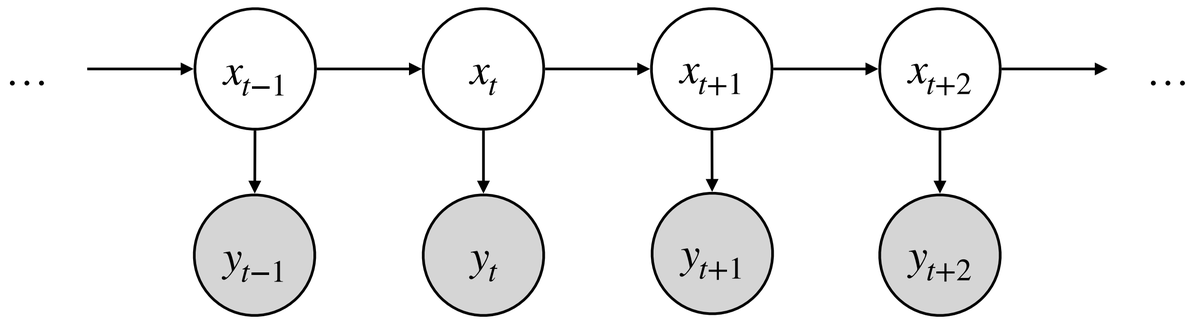
- SDS is always faster to match accuracy
- Reduction in particle count outweighs symbolic overhead
- SDS can be exact (1 particle)
- PF is impractical for advanced examples

Static analysis

Reactive Probabilistic Programming

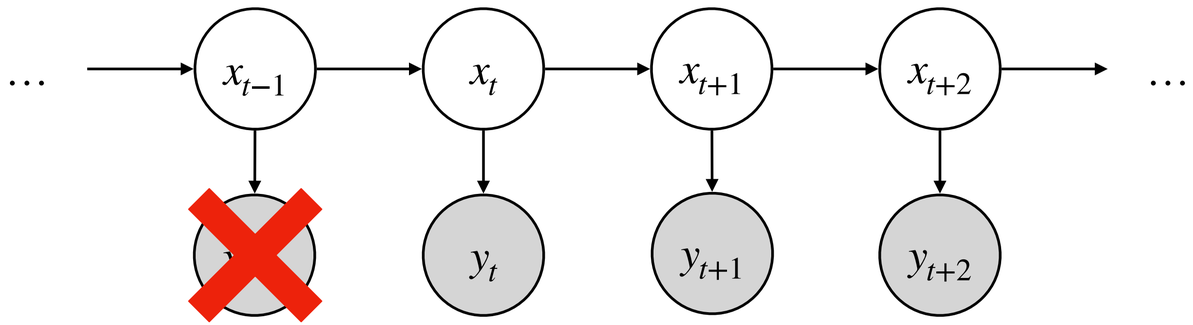
Bounded memory delayed sampling?

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```



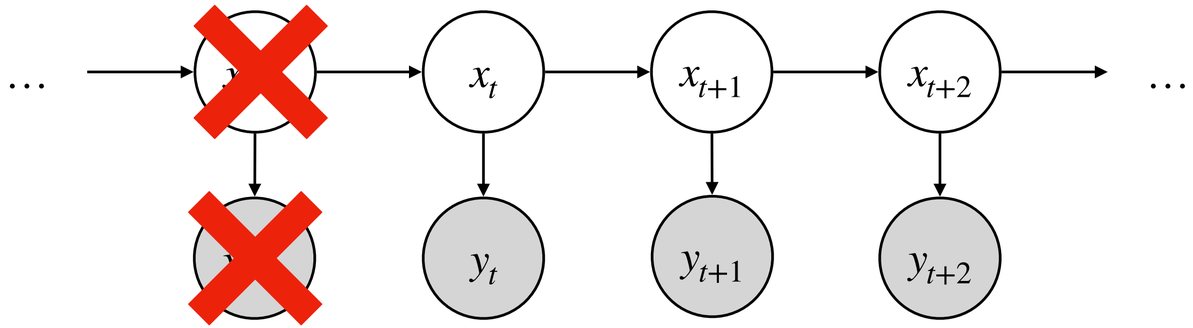
Bounded memory delayed sampling?

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```



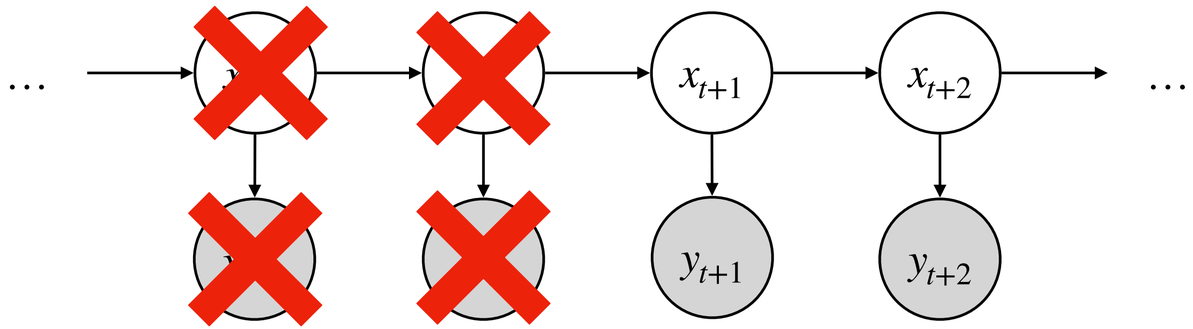
Bounded memory delayed sampling?

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```



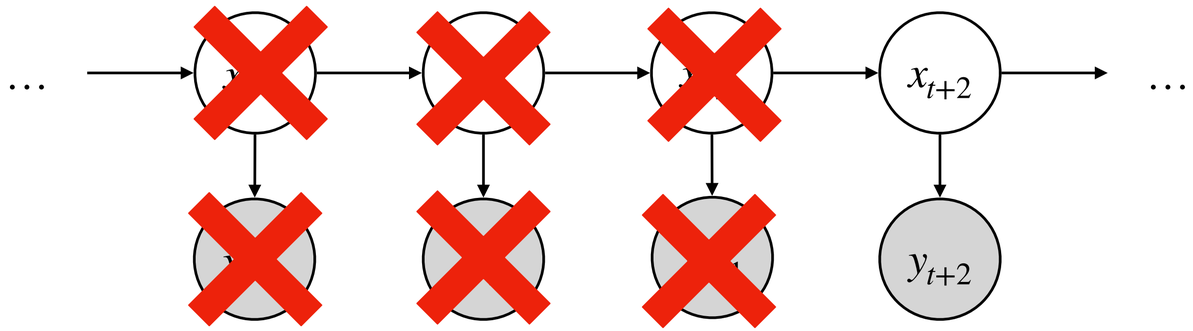
Bounded memory delayed sampling?

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```



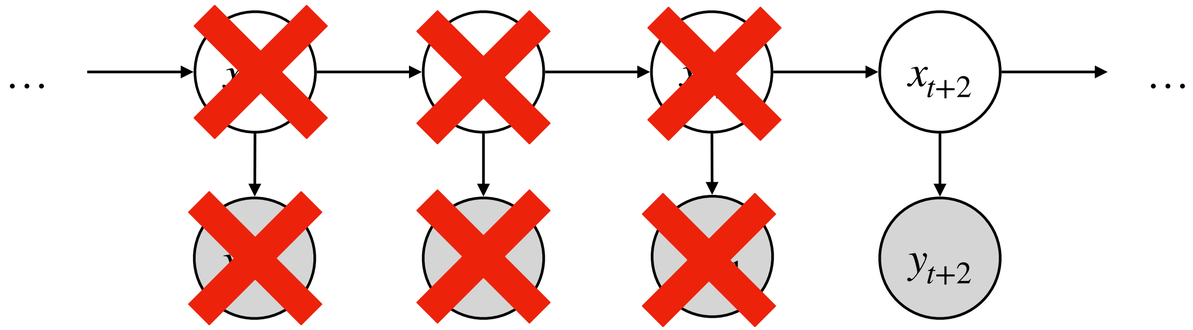
Bounded memory delayed sampling?

```
let proba tracker (y) = x where
  rec x   = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```



Bounded memory delayed sampling?

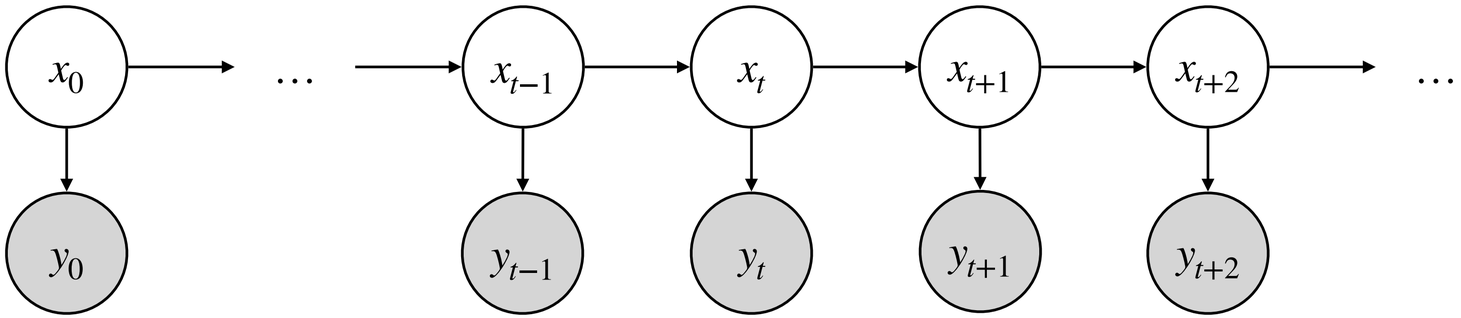
```
let proba tracker (y) = x where
  rec x   = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```



Yes!

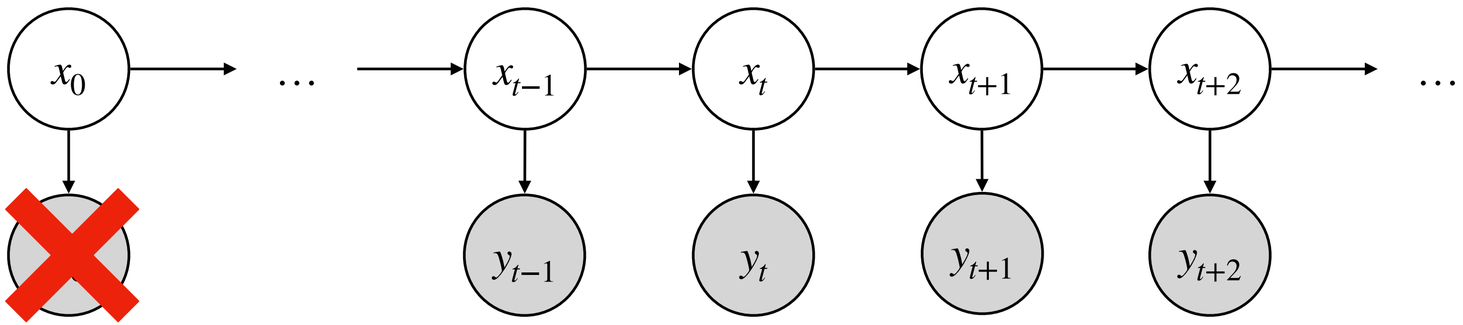
Bounded memory delayed sampling?

```
let proba tracker (y) = x, x0 where
  rec init x0 = sample (gaussian (0, 10))
  and x  = x0 → sample (gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```



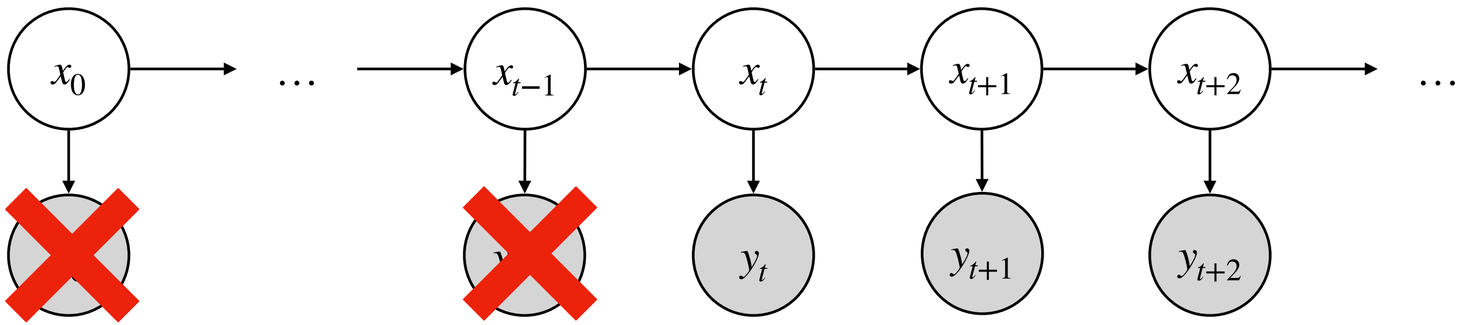
Bounded memory delayed sampling?

```
let proba tracker (y) = x, x0 where
  rec init x0 = sample (gaussian (0, 10))
  and x  = x0 → sample (gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```



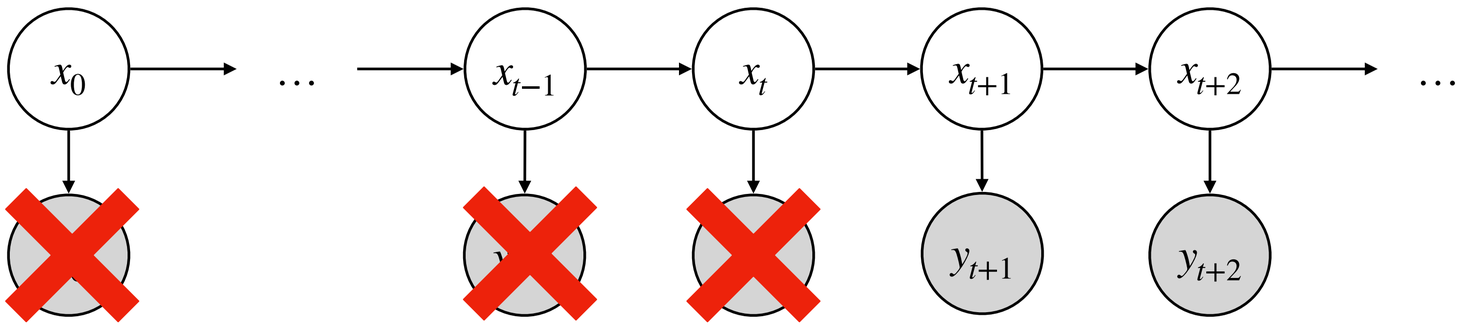
Bounded memory delayed sampling?

```
let proba tracker (y) = x, x0 where
  rec init x0 = sample (gaussian (0, 10))
  and x  = x0 → sample (gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```



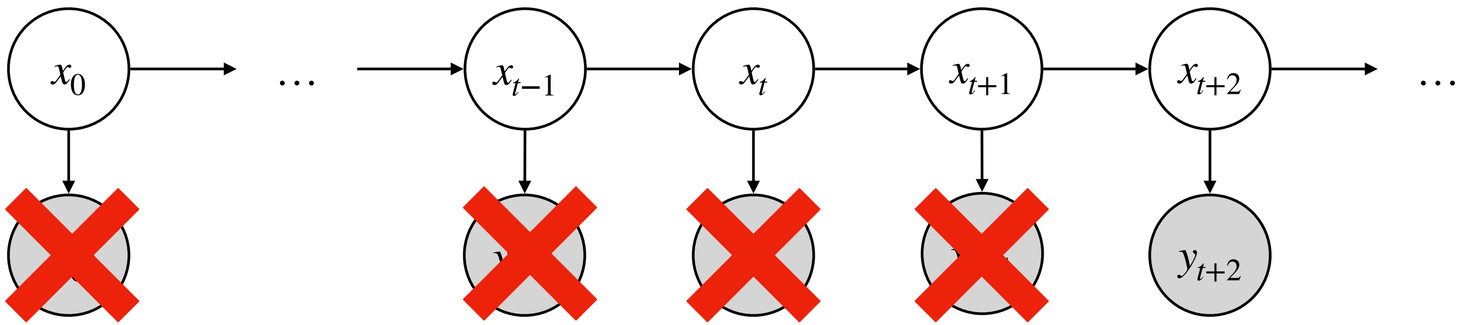
Bounded memory delayed sampling?

```
let proba tracker (y) = x, x0 where
  rec init x0 = sample (gaussian (0, 10))
  and x  = x0 → sample (gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```



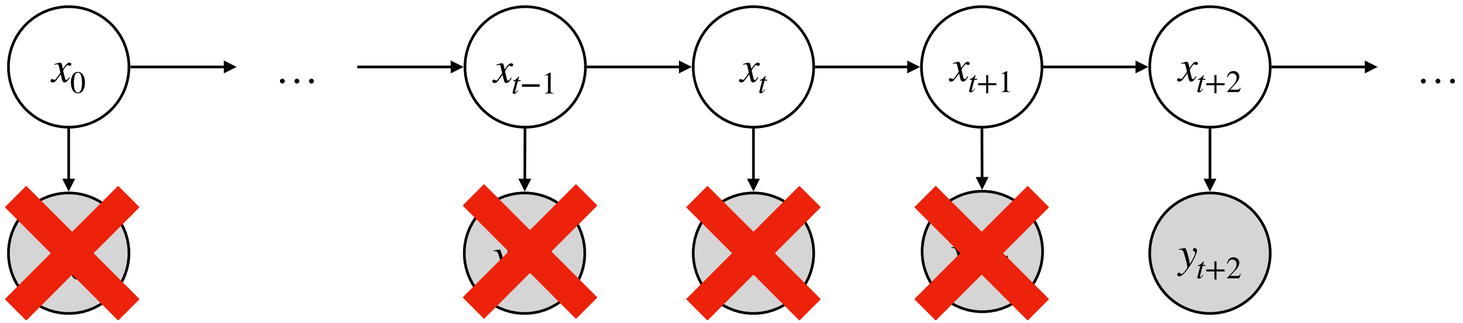
Bounded memory delayed sampling?

```
let proba tracker (y) = x, x0 where
  rec init x0 = sample (gaussian (0, 10))
  and x  = x0 → sample (gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```



Bounded memory delayed sampling?

```
let proba tracker (y) = x, x0 where
  rec init x0 = sample (gaussian (0, 10))
  and x  = x0 → sample (gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

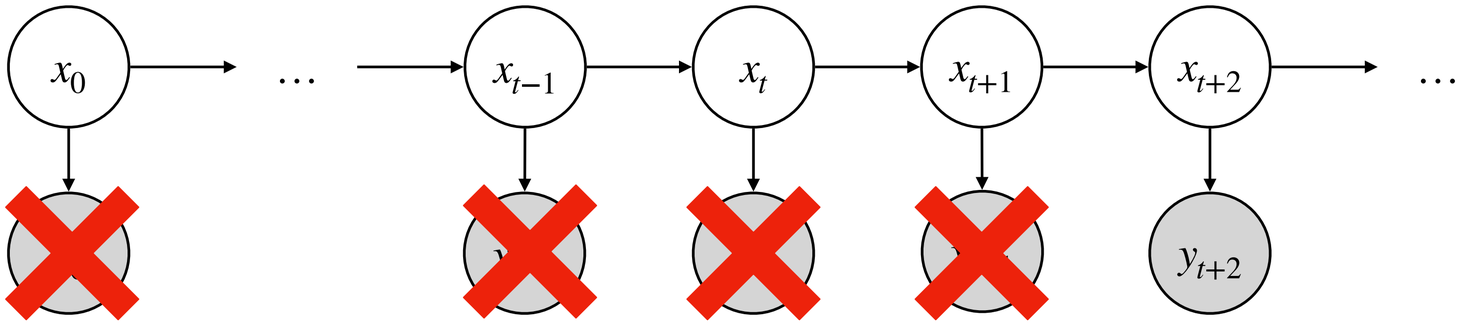


No!

Bounded memory delayed sampling?

```
let proba tracker (y) = x, x0 where
  rec init x0 = sample (gaussian (0, 10))
  and x  = x0 → sample (gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

Can we determine if a given program will run in bounded memory?



No!

Trace: abstract execution

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

trace	state	time
$x_0 \leftarrow \bot$ ::	x = x_0	$t = 0$
$y_0 \leftarrow x_0$ ::		
observe y_0 ::		
$x_1 \leftarrow x_0$ ::	x = x_1, pre x = x_0	$t = 1$
$y_1 \leftarrow x_1$ ::		
observe y_1 ::		
$x_2 \leftarrow x_1$ ::	x = x_2, pre x = x_1	$t = 2$
$y_2 \leftarrow x_2$ ::		
…		

Trace: abstract execution

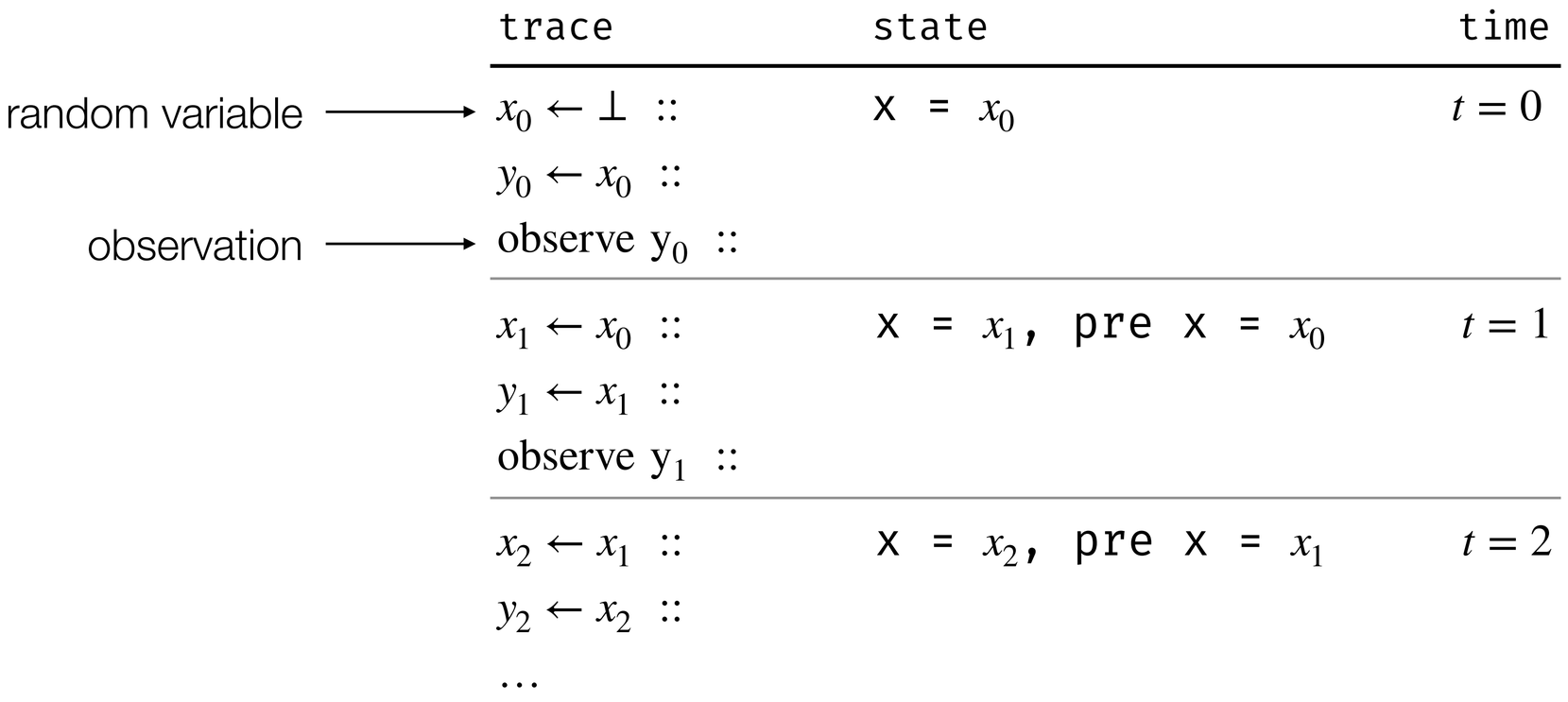
```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

random variable ⟶

trace	state	time
$x_0 \leftarrow \bot$ ::	x = x_0	$t = 0$
$y_0 \leftarrow x_0$ ::		
observe y_0 ::		
$x_1 \leftarrow x_0$ ::	x = x_1, pre x = x_0	$t = 1$
$y_1 \leftarrow x_1$ ::		
observe y_1 ::		
$x_2 \leftarrow x_1$ ::	x = x_2, pre x = x_1	$t = 2$
$y_2 \leftarrow x_2$ ::		
…		

Trace: abstract execution

```
let proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```



Static analysis for delayed sampling

Semantic properties

m-consumed property
Chains of variables before an observe are bounded

unseparated paths property
Chains of variables referenced in the state are bounded

Theorem: *The program satisfies these two properties iff it executes in bounded memory*

Static analysis for delayed sampling

Semantic properties

m-consumed property
Chains of variables before an observe are bounded

unseparated paths property
Chains of variables referenced in the state are bounded

Theorem: *The program satisfies these two properties iff it executes in bounded memory*

Static analysis

Track variables introduced but not used yet

Track maximal path between pairs of variable in the state

Theorem: Any *program that passes the analysis executes in bounded memory*

m-consumed property

```
proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

trace	state	time
$x_0 \leftarrow \bot$ ::	x = x_0	$t = 0$
$y_0 \leftarrow x_0$ ::		
observe y_0 ::		
$x_1 \leftarrow x_0$ ::	x = x_1, pre x = x_0	$t = 1$
$y_1 \leftarrow x_1$ ::		
observe y_1 ::		
$x_2 \leftarrow x_1$ ::	x = x_2, pre x = x_1	$t = 2$
$y_2 \leftarrow x_2$ ::		
…		

m-consumed property

```
proba tracker (y) = x where
  rec x   = sample (gaussian (0, 10) → gauss
  and () = observe (gaussian (x, 1), y)
```

Chains of variables before an observe are bounded

trace	state	time
$x_0 \leftarrow \bot$ ::	x = x_0	$t = 0$
$y_0 \leftarrow x_0$ ::		
observe y_0 ::		
$x_1 \leftarrow x_0$ ::	x = x_1, pre x = x_0	$t = 1$
$y_1 \leftarrow x_1$ ::		
observe y_1 ::		
$x_2 \leftarrow x_1$ ::	x = x_2, pre x = x_1	$t = 2$
$y_2 \leftarrow x_2$ ::		
…		

m-consumed property

```
proba tracker (y) = x where
  rec x   = sample (gaussian (0, 10) → gauss
  and () = observe (gaussian (x, 1), y)
```

Chains of variables before an observe are bounded

y_0 is 0-consumed ⟶

trace	state	time
$x_0 \leftarrow \bot$ ::	x = x_0	$t = 0$
$y_0 \leftarrow x_0$ ::		
observe y_0 ::		
$x_1 \leftarrow x_0$ ::	x = x_1, pre x = x_0	$t = 1$
$y_1 \leftarrow x_1$ ::		
observe y_1 ::		
$x_2 \leftarrow x_1$ ::	x = x_2, pre x = x_1	$t = 2$
$y_2 \leftarrow x_2$ ::		
…		

m-consumed property

```
proba tracker (y) = x where
  rec x   = sample (gaussian (0, 10) → gauss
  and () = observe (gaussian (x, 1), y)
```

Chains of variables before an observe are bounded

x_0 is 1-consumed ⟶

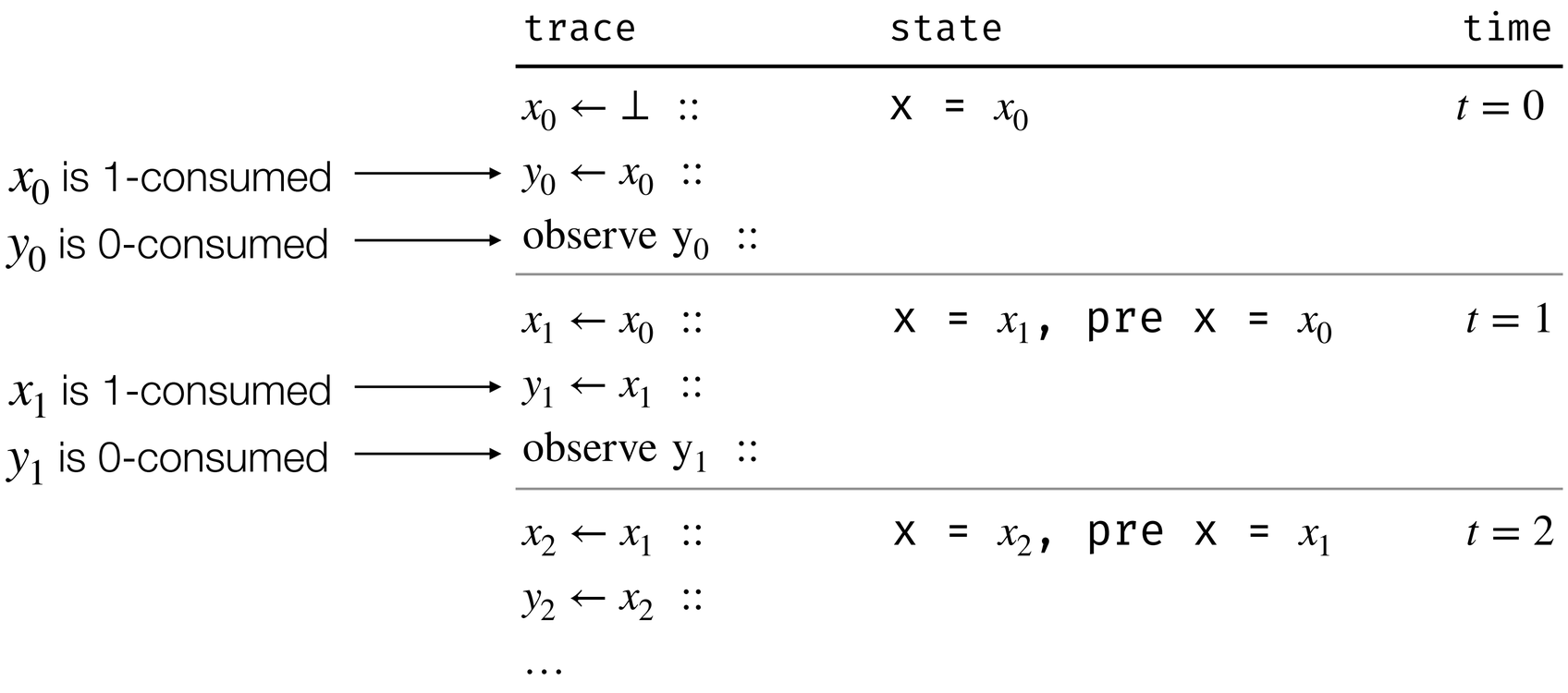
y_0 is 0-consumed ⟶

trace	state	time
$x_0 \leftarrow \bot$ ::	x = x_0	$t = 0$
$y_0 \leftarrow x_0$ ::		
observe y_0 ::		
$x_1 \leftarrow x_0$ ::	x = x_1, pre x = x_0	$t = 1$
$y_1 \leftarrow x_1$ ::		
observe y_1 ::		
$x_2 \leftarrow x_1$ ::	x = x_2, pre x = x_1	$t = 2$
$y_2 \leftarrow x_2$ ::		
…		

m-consumed property

```
proba tracker (y) = x where
  rec x   = sample (gaussian (0, 10) → gauss
  and () = observe (gaussian (x, 1), y)
```

Chains of variables before an observe are bounded



m-consumed property

```
proba tracker (y) = x where
  rec x   = sample (gaussian (0, 10) → gauss
  and () = observe (gaussian (x, 1), y)
```

Chains of variables before an observe are bounded

	trace	state	time
	$x_0 \leftarrow \bot$ ::	x = x_0	$t = 0$
x_0 is 1-consumed →	$y_0 \leftarrow x_0$ ::		
y_0 is 0-consumed →	observe y_0 ::		
	$x_1 \leftarrow x_0$ ::	x = x_1, pre x = x_0	$t = 1$
x_1 is 1-consumed →	$y_1 \leftarrow x_1$ ::		
y_1 is 0-consumed →	observe y_1 ::		
	$x_2 \leftarrow x_1$ ::	x = x_2, pre x = x_1	$t = 2$
	$y_2 \leftarrow x_2$ ::		
	…		

Yes!

Unseparated paths property

```
proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), y)
```

trace	state	time
$x_0 \leftarrow \bot$ ::	x = x_0	$t = 0$
$y_0 \leftarrow x_0$ ::		
observe y_0 ::		
$x_1 \leftarrow x_0$ ::	x = x_1, pre x = x_0	$t = 1$
$y_1 \leftarrow x_1$ ::		
observe y_1 ::		
$x_2 \leftarrow x_1$ ::	x = x_2, pre x = x_1	$t = 2$
$y_2 \leftarrow x_2$ ::		
…		

Unseparated paths property

```
proba tracker (y) = x where
  rec x  = sample (gaussian (0, 10) → gauss
  and () = observe (gaussian (x, 1), y)
```

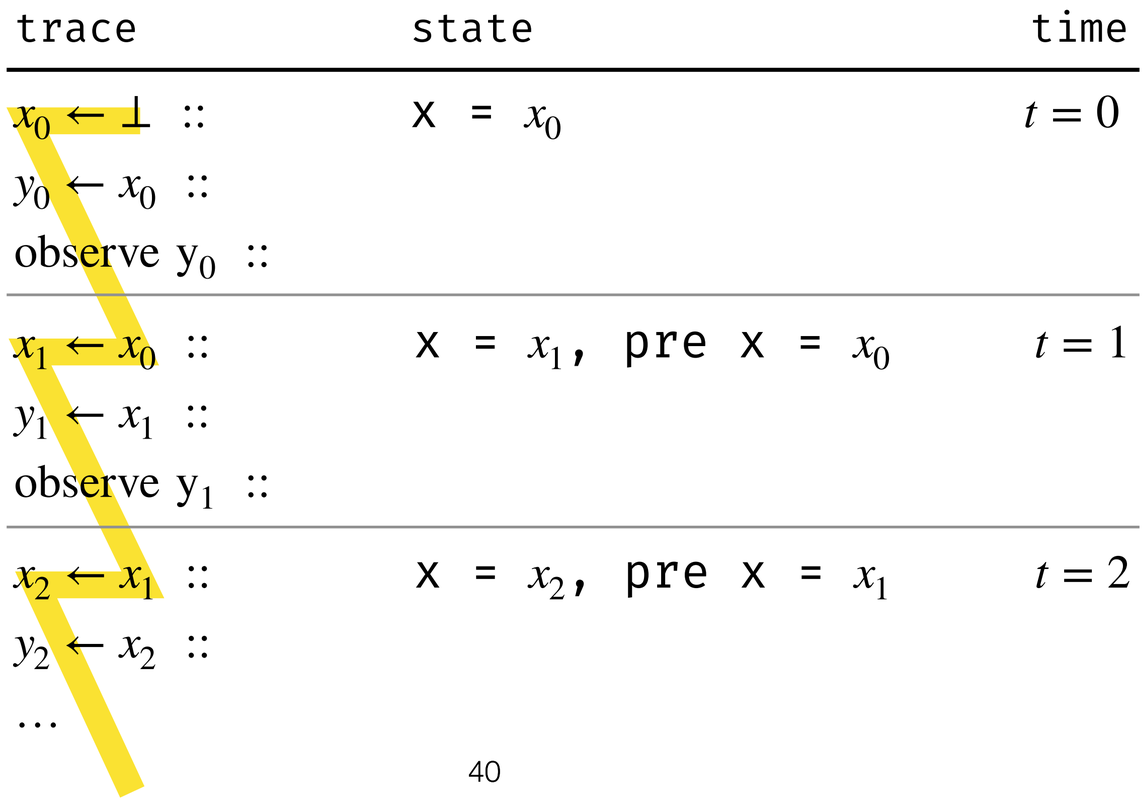
Chains of variables referenced in the state are bounded

trace	state	time
$x_0 \leftarrow \bot$ ::	x = x_0	$t = 0$
$y_0 \leftarrow x_0$ ::		
observe y_0 ::		
$x_1 \leftarrow x_0$ ::	x = x_1, pre x = x_0	$t = 1$
$y_1 \leftarrow x_1$ ::		
observe y_1 ::		
$x_2 \leftarrow x_1$ ::	x = x_2, pre x = x_1	$t = 2$
$y_2 \leftarrow x_2$ ::		
…		

Unseparated paths property

```
proba tracker (y) = x where
  rec x   = sample (gaussian (0, 10) → gauss
  and () = observe (gaussian (x, 1), y)
```

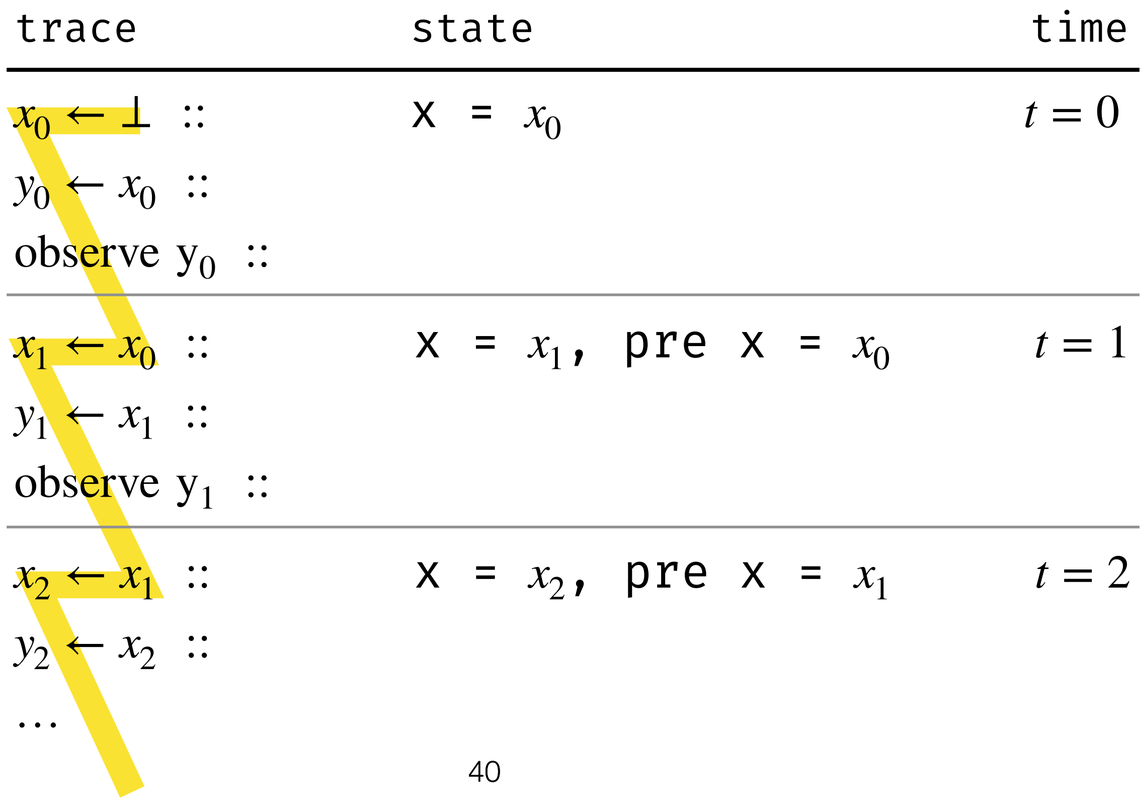
Chains of variables referenced in the state are bounded



Unseparated paths property

```
proba tracker (y) = x where
  rec x   = sample (gaussian (0, 10) → gauss
  and () = observe (gaussian (x, 1), y)
```

Chains of variables referenced in the state are bounded



Yes!

Unseparated paths property

```
proba tracker (y) = x where
  rec init x0 = sample (gaussian (0, 10))
  and x  = x0 → sample (gaussian (pre x, 1)
  and () = observe (gaussian (x, 1), y)
```

Chains of variables referenced in the state are bounded

trace	state	time
$x_0 \leftarrow \bot$ ::	x = x_0	$t = 0$
$y_0 \leftarrow x_0$ ::	x0 = x_0	
observe y_0 ::		
$x_1 \leftarrow x_0$ ::	x = x_1, pre x = x_0	$t = 1$
$y_1 \leftarrow x_1$ ::	x0 = x_0	
observe y_1 ::		
$x_2 \leftarrow x_1$ ::	x = x_2, pre x = x_1	$t = 2$
$y_2 \leftarrow x_2$ ::	x0 = x_0	
…		

Unseparated paths property

```
proba tracker (y) = x where
  rec init x0 = sample (gaussian (0, 10))
  and x  = x0 → sample (gaussian (pre x, 1)
  and () = observe (gaussian (x, 1), y)
```

Chains of variables referenced in the state are bounded

trace	state	time
$x_0 \leftarrow \bot$ ::	x = x_0	$t = 0$
$y_0 \leftarrow x_0$ ::	x0 = x_0	
observe y_0 ::		
$x_1 \leftarrow x_0$ ::	x = x_1, pre x = x_0	$t = 1$
$y_1 \leftarrow x_1$ ::	x0 = x_0	
observe y_1 ::		
$x_2 \leftarrow x_1$ ::	x = x_2, pre x = x_1	$t = 2$
$y_2 \leftarrow x_2$ ::	x0 = x_0	
…		

Unseparated paths property

```
proba tracker (y) = x where
  rec init x0 = sample (gaussian (0, 10))
  and x  = x0 → sample (gaussian (pre x, 1)
  and () = observe (gaussian (x, 1), y)
```

Chains of variables referenced in the state are bounded

trace	state	time
$x_0 \leftarrow \bot$ ::	x = x_0	$t = 0$
$y_0 \leftarrow x_0$ ::	x0 = x_0	
observe y_0 ::		
$x_1 \leftarrow x_0$ ::	x = x_1, pre x = x_0	$t = 1$
$y_1 \leftarrow x_1$ ::	x0 = x_0	
observe y_1 ::		
$x_2 \leftarrow x_1$ ::	x = x_2, pre x = x_1	$t = 2$
$y_2 \leftarrow x_2$ ::	x0 = x_0	
…		

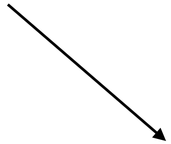
No!

Evaluation

	m-consumed		unsep. paths		bounded mem.	
	output	actual	output	actual	output	actual
Kalman	✓	✓	✓	✓	✓	✓
Kalman Hold-First	✓	✓	✗	✗	✗	✗
Gaussian Random Walk	✗	✗	✓	✓	✗	✗
Robot	✓	✓	✓	✓	✓	✓
Coin	✓	✓	✓	✓	✓	✓
Gaussian-Gaussian	✓	✓	✓	✓	✓	✓
Outlier	✗	✗	✓	✓	✗	✗
MTT	✗	✗	✓	✓	✗	✗
SLAM	✗	✓	✓	✓	✗	✓

Evaluation

memory is probabilistically bounded



	m-consumed		unsep. paths		bounded mem.	
	output	actual	output	actual	output	actual
Kalman	✓	✓	✓	✓	✓	✓
Kalman Hold-First	✓	✓	✗	✗	✗	✗
Gaussian Random Walk	✗	✗	✓	✓	✗	✗
Robot	✓	✓	✓	✓	✓	✓
Coin	✓	✓	✓	✓	✓	✓
Gaussian-Gaussian	✓	✓	✓	✓	✓	✓
Outlier	✗	✗	✓	✓	✗	✗
MTT	✗	✗	✓	✓	✗	✗
SLAM	✗	✓	✓	✓	✗	✓

Evaluation

memory is probabilistically bounded

memory is always bounded

	m-consumed		unsep. paths		bounded mem.	
	output	actual	output	actual	output	actual
Kalman	✓	✓	✓	✓	✓	✓
Kalman Hold-First	✓	✓	✗	✗	✗	✗
Gaussian Random Walk	✗	✗	✓	✓	✗	✗
Robot	✓	✓	✓	✓	✓	✓
Coin	✓	✓	✓	✓	✓	✓
Gaussian-Gaussian	✓	✓	✓	✓	✓	✓
Outlier	✗	✗	✓	✓	✗	✗
MTT	✗	✗	✓	✓	✗	✗
SLAM	✗	✓	✓	✓	✗	✓

Take away

ProbZelus: a probabilistic synchronous languages

- Bayesian inference on streams
- Inference in the loop
- Make the underlying probabilistic model explicit

Inference with bounded resources

- Monte Carlo approximations: Importance sampling, particle filter
- Semi-symbolic inference on streaming models

Static analysis

- Can delayed sampling run in bounded memory?

More tomorrow!

- What is the semantics of these programs?
- How can we reason about program equivalence?

Take away

https://github.com/IBM/probzelus

ProbZelus: a probabilistic synchronous languages

- Bayesian inference on streams
- Inference in the loop
- Make the underlying probabilistic model explicit

Inference with bounded resources

- Monte Carlo approximations: Importance sampling, particle filter
- Semi-symbolic inference on streaming models

Static analysis

- Can delayed sampling run in bounded memory?

More tomorrow!

- What is the semantics of these programs?
- How can we reason about program equivalence?