Programming Reactive Probabilistic Applications

Guillaume Baudart, Louis Mandel
IBM Research

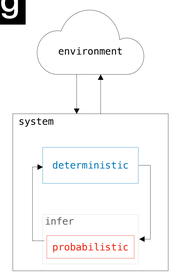
Eric Atkinson, Benjamin Sherman, Michael Carbin
MIT

Marc Pouzet
ENS Paris, INRIA/PSL

Reactive Probabilistic Programming

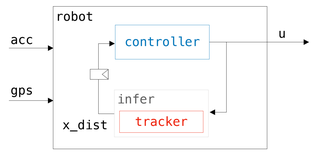
Synchronous data-flow languages and block diagrams
- State of the art model for embedded systems
- E.g., Matlab/Simulink, SCADE, Lustre, Zelus
- Signal: stream of values; System: stream processor

ProbZelus: add support to deal with uncertainty
- Extend Zelus with probabilistic constructs
- Parallel composition: deterministic/probabilistic
- Inference-in-the-loop
- Inference on streams



Example: Robot

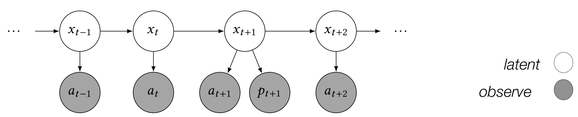
- Goal: control a robot's position using a noisy accelerometer and intermittent GPS
- Feedback between deterministic `controller` and probabilistic `tracker`




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

Probabilistic Programming

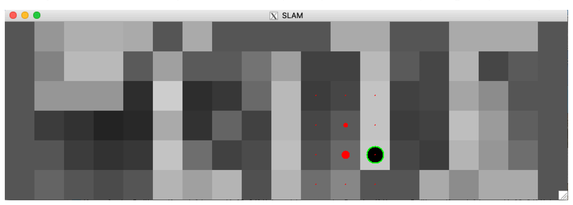
- `x = sample(d)`: introduce a random variable `x` of distribution `d`
- `observe(d, y)`: condition on the fact that `y` was sampled from `d`
- `infer m obs`: compute the distribution of output of the model `m` with respect to `obs`




```
let proba tracker (u, acc, gps) = x where
  rec  x = sample (mv_gaussian ((x0 -> pre x), noise)
  and () = observe (gaussian (get_acc x, 1.0), acc)
  and present gps (pos) ->  do () = observe (gaussian (get_pos x, 0.01), pos) done
```

Example: SLAM

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)

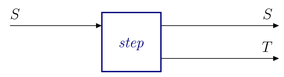


At each step:
- Move to the next position
- Observe the color of the ground
- Use inferred position to compute next command

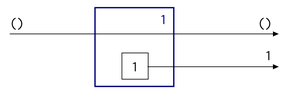
Semantics

Deterministic Streams

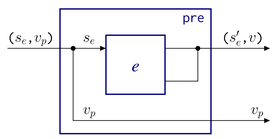
Initial state, transition function
$CoStream(T, S) = S \times (S \to S \times T)$



Constant `1` : `unit` → (`unit` × `int`)
- Initial state: the value of type `unit`
- Step function: return the constant `1`

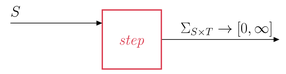


Unit delay `pre` e: $(S \times T) \times (S \times T \to (S \times T) \times T)$
- Initial state: the initial state of e and default value
- Step function: the result of e is stored in the state and returned at the next iteration

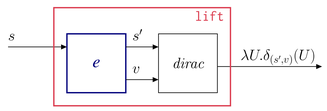


Probabilistic Streams

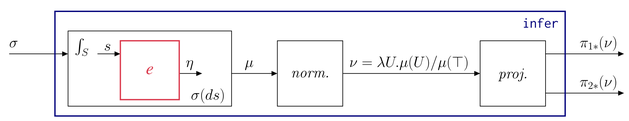
Transition function returns a *measure*
$CoPStream(T, S) = S \times (S \to \Sigma_{S \times T} \to [0,\infty])$



`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`$)$



Inference

Example: a 1-D version of the robot `tracker`

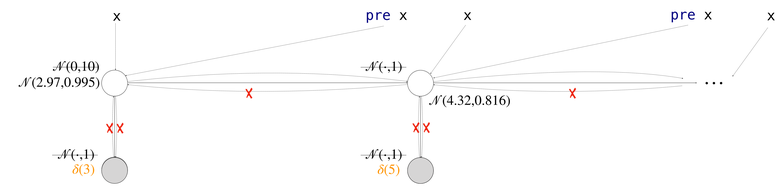
```
let proba tracker (obs) = x where
  rec x  = sample (gaussian (0, 10) -> gaussian (pre x, 1))
  and () = observe (gaussian (x, 1), obs)
```

Particle Filtering

$t = 0$
```
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), …)
```

Delayed Sampling

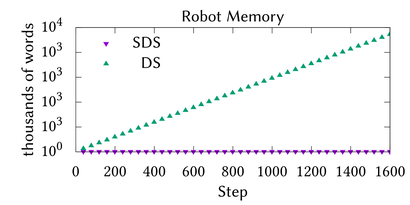
$t = 0$
```
sample (gaussian (0, 10))
observe (gaussian (x, 1), 3)
```
$t = 1$
```
sample (gaussian (pre x, 1))
observe (gaussian (x, 1), 5)
```
$t = 2$
```
sample (gaussian (pre x, 1))
observe (gaussian (x, 1), …)
```



Streaming Delayed Sampling

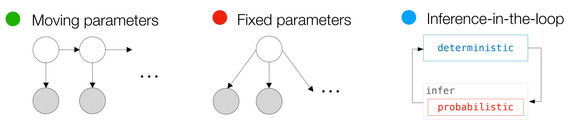
- Use delayed sampling as above
- Remove unnecessary pointers (shown by X)
- Now, only reachable nodes can affect future distributions

SDS: Inference with Bounded Resources

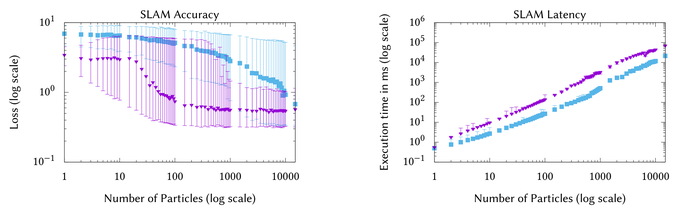


Evaluation

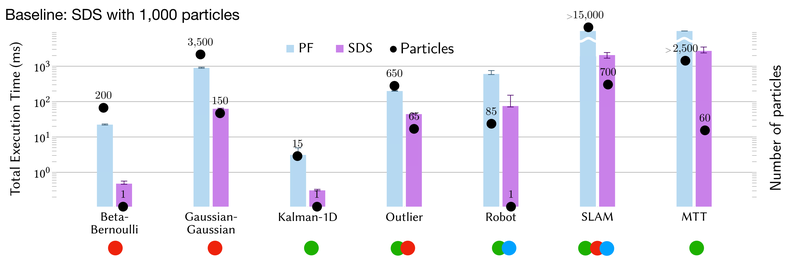
Language features:



Algorithm comparison
- PF Particle Filtering
- SDS Streaming Delayed Sampling



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