Sequential Monte Carlo in String Diagrams

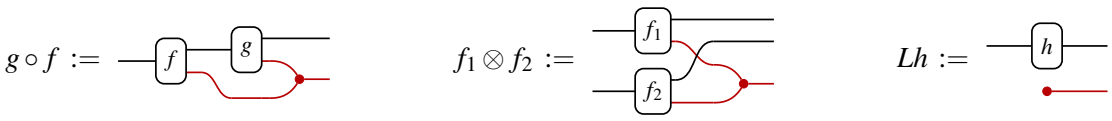
Marius Furter
University of Zurich
Zurich, Switzerland
marius.furter@math.uzh.ch

Sequential Monte Carlo (SMC) algorithms [2], also called particle filters, were developed for approximate inference in state-space models. They maintain a population of weighted samples, called particles, that are periodically resampled to prevent weight degeneracy. More broadly, SMC applies whenever a target distribution can be decomposed into a sequence of intermediate steps [4]. Traditionally, these algorithms are formalized using discrete-time Feynman-Kac models [3] which consist of a sequence of Markov kernels $M_t : X_{t-1} \rightsquigarrow X_t$ and potential functions $G_t : X_{t-1} \times X_t \to \mathbb{R}_{\geq 0}$.

In this talk[1], we interpret SMC categorically: any string diagram of s-finite kernels gives rise to a weighted particle sampling scheme. This extends the string-diagrammatic account of exact filtering given in [6]. We establish correctness by deriving a law of large numbers from compositional stability properties. Finally, we describe how this framework has guided the design of probabilistic programming languages (PPLs) in Julia [8] and PyTorch [9] and enables intuitive reasoning about programs.

Weighted kernels A kernel $f : X \rightsquigarrow Y$ between measurable spaces is a function $f : \Sigma_Y \times X \to \overline{\mathbb{R}}_{\geq 0}$ such that $B \mapsto f(B \mid x)$ is a measure, and $x \mapsto f(B \mid x)$ is measurable. A kernel is *s-finite* [7, 10] if it can be written as a countable sum of finite kernels: $f(B \mid x) = \sum f_i(B \mid x)$, where $\sup_x f_i(B \mid x) < \infty$. Measurable spaces and s-finite kernels assemble into a cd-category[2] sfKern [10, 1], enabling string-diagrammatic reasoning. When all kernels in a diagram are *Markov kernels* ($f(Y \mid x) = 1$), one can approximate the composite by ***ancestral sampling***: topologically order the kernels, then iteratively draw $y_i \sim f_i(- \mid x_i)$.

To extend this sampling approach to non-normalized kernels we introduce ***weights***. Any commutative monoid object $W \in \mathsf{Ob}(\mathsf{sfKern})$ induces a symmetric monoidal monad $(- \otimes W) : \mathsf{sfKern} \to \mathsf{sfKern}$. This makes the Kleisli category $\mathsf{wKern} := \mathsf{Kl}_{(-\otimes W)}$ a cd-category [5]. We call arrows of wKern ***weighted kernels***. Drawing weights in red, and monoid multiplication as merging, weighted kernels compose by:



Any s-finite kernel $h : X \rightsquigarrow Y$ lifts to the weighted kernel $Lh : X \rightsquigarrow Y \otimes W$ returning unit weight e. To go in the opposite direction, we introduce a ***supply of integrators***: a monoidal transformation $\int : (- \otimes W) \Rightarrow \mathsf{Id}$ where every component $\int_Y : Y \otimes W \to Y$ is an algebra for $(- \otimes W)$. These properties are sufficient for $f \mapsto f \fatsemi \int$ to define a symmetric strict monoidal functor $\int_W : \mathsf{wKern} \to \mathsf{sfKern}$.

We instantiate these results with $W := (\overline{\mathbb{R}}_{\geq 0}, \cdot, 1)$ and supply of integrators defined by $\int_Y (B \mid y, w) := w \, [\![y \in B]\!]$, where $[\![\phi]\!]$ is the indicator of ϕ. In this case, $\int_W$ maps a weighted kernel $k : X \rightsquigarrow Y \otimes W$ to its ***weighted expectation*** $(\int_W k)(B \mid x) = \int_W w \, k(B, dw \mid x)$. Conversely, by the Radon-Nikodym theorem

[1] This work is part of my upcoming Ph.D. thesis. A draft containing the relevant material can be found here.

[2] A cd-category is a symmetric monoidal category where every object has a commutative comonoid structure compatible with the tensor product.

Submitted to:
ACT 2026

for s-finite kernels [11], any s-finite kernel $f\colon X \rightsquigarrow Y$ between standard Borel spaces can be lifted to a weighted *Markov kernel* $k\colon X \rightsquigarrow Y \otimes W$ with $\int_W k = f$ by reweighting any Markov kernel $p\colon X \rightsquigarrow Y$ that dominates f in an appropriate sense. This yields a categorical account of ***sequential importance sampling*** (SIS): Given a diagram of s-finite kernels f_i between standard Borel spaces, lift them to weighted Markov kernels $k_i \in \mathsf{wKern}$ with $\int_W k_i = f_i$. Apply ancestral sampling to the composite weighted kernel, then push the result down to sfKern via $\int_W$ by taking weighted averages.

Particle algorithms SIS becomes unstable as the number of steps increases, with only a few samples keeping relatively high weights. This problem can be mitigated by resampling: we maintain a population of ***particles***, and periodically discard low-weight and duplicate high-weight particles in a consistent manner. The simplest method draws new particle indices independently according to their weights.

We define a ***particle transformer*** $\varphi\colon (X,m) \to (Y,n)$ to be a kernel $\varphi\colon (X \otimes W)^m \rightsquigarrow (Y \otimes W)^n$ that operates on arrays of weighted samples. Let $\mathsf{emp}_Y\colon (Y \otimes W)^n \rightsquigarrow Y$ be the kernel computing the empirical distribution $(y_{1:n}, w_{1:n}) \mapsto \frac{1}{n}\sum_i w_i\, \delta_{y_i}$. We call φ *mixable* if $\mathsf{emp}_Y \circ \varphi = \bar{\varphi} \circ \mathsf{emp}_X$ for a unique *mean kernel* $\bar{\varphi}\colon X \to Y$. The mean $\bar{\varphi}$ returns the expected empirical distribution of φ applied to $(x,1)_{j=1}^m$.

Mixable particle transformers form a category Part under kernel composition. There is a functor $P\colon \mathsf{Part} \to \mathsf{sfKern}$ sending $\varphi \mapsto \bar{\varphi}$. Any weighted Markov kernel $k\colon X \to Y \otimes W$ lifts functorially to a particle transformer $k^n\colon (X,n) \to (Y,n)$ with $P(k^n) = \int_W k$. We may therefore evaluate an s-finite kernel $f = \int_W k$ by computing the expected empirical distribution of the outputs of k^n. The same works for any sequential decomposition of k. Classic resampling schemes define particle transformers r with $P(r) = \mathsf{id}$, so functoriality implies that resampling steps may be inserted freely.

In practice, we approximate $\int_W k$ using a single sample $(y_{1:n}, w_{1:n}) \sim k^n$. This works since empirical averages $\frac{1}{n}\sum_i w_i\, t(y_i)$ converge to expectations $\mathbb{E}_{\int_W k}[t(y)]$ as $n \to \infty$. We prove this using a compositional stability condition, controlling how much particle transformers increase the variance $\mathrm{Var}\left[\frac{1}{n}\sum_i w_i\, t(y_i)\right]$ and mean second moment $\mathbb{E}\left[\frac{1}{n}\sum_i w_i^2\right]$ of particles. Classic resampling schemes are stable, as is k^n when k has bounded first and second weight moments.

Programming with weighted kernels The concepts above translate into a PPL whose basic components are ***weighted samplers*** `k` that draw $(y,w) \sim k(- \mid x)$ for $k\colon X \rightsquigarrow Y \otimes W$. If available, `k` records a *density*[3] $l_k\colon X \times Y \to \overline{\mathbb{R}}_{\geq 0}$ for $\int_W k$ w.r.t. base measure ν_k. We can use unweighted samplers `p` by assigning unit weights. More generally, to sample from a density l_t w.r.t. ν_p, we can sample from `p` and assign weight l_t / l_p. Finally, to observe a value y_0 from `p` we only return the likelihood weight $l_p(x, y_0)$.

These ideas have been implemented in Julia [8] and PyTorch [9] libraries that provide a simple model building syntax while handling vectorization and weight management in the background. We conclude the talk by showing how one can easily reason about counterintuitive programs using string diagrams.

```julia
@model function linear_regression(data)
  a ~ Normal(0, 5) #sample
  b ~ Normal(0, 5)
  for (x, y) in data
    y => Normal(a + b * x, 0.1) #observe
  end end
```

```python
@model
def linear_regression(data):
  a = sample("a", Normal(0, 5))
  b = sample("b", Normal(0, 5))
  for x, y in data:
    observe(y, Normal(a + b * x, 0.1))
```

Figure 1: Bayesian linear regression in WeightedSampling.jl [8] (left) and WeightedSampling.torch [9] (right).

[3] A kernel f has density $l_f\colon X \times Y \to \overline{\mathbb{R}}_{\geq 0}$ with respect to ν if $f(B \mid x) = \int_B l_f(x,y)\nu(dy)$.

References

[1] Kenta Cho & Bart Jacobs (2019): *Disintegration and Bayesian inversion via string diagrams*. *Mathematical Structures in Computer Science* 29(7), pp. 938–971, doi:10.1017/S0960129518000488. arXiv:arXiv:1709.00322.

[2] Nicolas Chopin & Omiros Papaspiliopoulos (2020): *An Introduction to Sequential Monte Carlo*. Springer International Publishing, doi:10.1007/978-3-030-47845-2.

[3] Pierre Del Moral (2004): *Feynman-Kac Formulae*. Springer New York, doi:10.1007/978-1-4684-9393-1.

[4] Pierre Del Moral, Arnaud Doucet & Ajay Jasra (2006): *Sequential Monte Carlo Samplers*. *Journal of the Royal Statistical Society Series B: Statistical Methodology* 68(3), pp. 411–436, doi:10.1111/j.1467-9868.2006.00553.x.

[5] Tobias Fritz (2020): *A synthetic approach to Markov kernels, conditional independence and theorems on sufficient statistics*. *Advances in Mathematics* 370, p. 107239, doi:https://doi.org/10.1016/j.aim.2020.107239. arXiv:arXiv:1908.07021.

[6] Tobias Fritz, Andreas Klingler, Drew McNeely, Areeb Shah Mohammed & Yuwen Wang (2025): *Hidden Markov Models and the Bayes Filter in Categorical Probability*. *IEEE Transactions on Information Theory* 71(9), pp. 7052–7075, doi:10.1109/tit.2025.3584695. arXiv:arXiv:2401.14669.

[7] Olav Kallenberg (2017): *Random Measures, Theory and Applications*. Springer International Publishing, doi:10.1007/978-3-319-41598-7.

[8] Marius Furter (2026): *WeightedSampling.jl*. Available at `https://github.com/MariusFurter/WeightedSampling.jl`.

[9] Marius Furter (2026): *WeightedSampling.torch*. Available at `https://github.com/MariusFurter/WeightedSampling.torch`.

[10] Sam Staton (2017): *Commutative Semantics for Probabilistic Programming*, pp. 855–879. doi:10.1007/978-3-662-54434-1_32.

[11] Matthijs Vákár & Luke Ong (2018): *On S-Finite Measures and Kernels*. doi:10.48550/arXiv.1810.01837.