Feynman-Kac models for universal probabilistic programming

Michele Boreale and Luisa Collodi

Università degli Studi di Firenze, Italy
Dipartimento di Statistica, Informatica, Applicazioni "G. Parenti"
{michele.boreale,luisa.collodi}@unifi.it

Abstract. We study provably correct and efficient implementations of Sequential Monte Carlo (SMC) inference algorithms in the context of formal operational semantics of Probabilistic Programs (PPs). We focus on *universal* PPs featuring sampling from arbitrary measures and conditioning/reweighting in unbounded loops. We first equip Probabilistic Program Graphs (PPGs), an automata-theoretic description format of PPs, with an expectation-based semantics over infinite execution traces, which also incorporates trace weights. We then prove a finite approximation theorem that provides bounds to this semantics based on expectations taken over finite, fixed-length traces. This enables us to frame our semantics within a Feynman-Kac (FK) model, and ensures the consistency of the Particle Filtering (PF) algorithm, an instance of SMC, with respect to our semantics. Building on these results, we introduce VPF, a vectorized version of the PF algorithm tailored to PPGs and our semantics. Experiments conducted with a proof-of-concept implementation of VPF show very promising results compared to state-of-the-art PP inference tools.

Keywords: Probabilistic programming, operational semantics, measure theory, Feynman-Kac models, Sequential Monte Carlo, SIMD parallelism.

1 Introduction

Probabilistic Programming Languages (PPLs) [26, 7] offer a systematic approach to define arbitrarily complicated probabilistic models. One is typically interested in performing *inference* on these models, given observed data; for example, finding the posterior distribution of the program's variables conditioned on the observed data.

In terms of formal semantics of PPLs, the denotational approach introduced by Kozen [32] offers a solid mathematical foundation. However, when it comes to practical algorithms for PPL-based inference, the landscape appears somewhat fragmented. On one hand, *symbolic* and *static analysis* techniques, see e.g. [43, 23, 38, 8, 10, 42], yield results with correctness guarantees firmly grounded in the semantics of PPLs but often struggle with scalability. On the other hand, practical languages and inference algorithms predominantly leverage Monte Carlo (MC) *sampling* techniques (MCMC, SMC), which are more scalable but often lack a clear connection to formal semantics [25, 17, 12]. Notable exceptions to this situation include works such as [40, 51, 34, 13, 33], which are discussed in the related work section further below.

Establishing the consistency of an inference algorithm with respect to a PPL's formal semantics is not merely a theoretical pursuit. In the context of *universal* [24] PPLs integrating unbounded loops and conditioning with MC sampling, which requires truncating computations at a finite time, presents significant challenges [10]. Additionally, the interplay between continuous and discrete distributions in these PPLs can lead to complications, potentially causing existing sampling-based algorithms to yield incorrect results [51].

In the present work, we establish a precise connection between *Probabilistic Program Graphs (PPGs)*, a general automata-theoretic description format of PPs, and *Feynman-Kac (FK)* models, a formalism for state-based probabilistic processes and observations defined over a finite time horizon [19, Ch.5]. This connection enables us to prove the consistency for PPGs of the *Particle Filtering (PF)* inference algorithm, one of the incarnations of Sequential Monte Carlo approach [19, Ch.10]. Departing from the traditional denotational approach, we adopt a decisively operational perspective.

In more detail, in a PPG (Section 3), computation (essentially, sampling) progresses in successive stages specified by the direct edges of a graph (transitions), with nodes serving as *checkpoints* between stages for *conditioning* on observed data or more generally updating computation weights. The operational semantics of PPGs is formalized in terms of Markov kernels and score functions. Building on this, we introduce a measure-theoretic, infinite-trace semantics (Section 4, with the necessary measure theory reviewed in Section 2). A finite approximation theorem then allows us to relate this trace semantics precisely to a finite-time horizon FK model (Section 5). PF is known to be *consistent* for FK models asymptotically: as the number N of simulated instances (*particles*) tends to infinity, the distribution of these particles converges to the measure defined by the FK model [19, Ch.11]. Therefore, consistency of PF for PPGs will automatically follow.

Our approach yields additional insights. First, the finite approximation theorem holds for a class of *prefix-closed functions* defined on infinite traces: these are the functions where the output only depends on a finite initial segment of the input argument. They can be viewed as a generalization of the prefix-closed sets involved in the definition of Safety properties in model checking [5]. The finite approximation theorem implies that the expectation of a prefix-closed function, defined on the probability space of infinite traces, can be approximated by the expectation of functions defined over truncated traces, with respect to a measure defined on a suitable FK model. As expectation in a FK model can be effectively estimated, via PF or other algorithms, our finite approximation result lays a sound basis for the statistical model checking of PPs. Second, the automata-theoretic operational semantics of PPGs translates into a *vectorized* implementation of PF, leveraging the fine-grained, SIMD parallelism exisisting at the level of particles. Specifically, the transition function and the score functions are applied simultaneously to the entire vector of N simulated particles at each step. This is practically significant, as modern CPUs and programming languages offer extensive support for vectorization, that may lead to dramatic speedups. We demonstrate this with a prototype vectorized implementation of a PPG-based PF algorithm using TensorFlow [1], called VPF. Experiments comparing VPF with state-of-the-art PPLs on challenging examples from the literature show very promising results (Section 6). Concluding remarks are provided in the final section (Section 7). Most proofs and additional technical material have been confined to separate appendices (A, B, C).

In summary, our main contributions are as follows: (1) A clean semantics for PPGs based on expectation taken over infinite-trace, which incorporates conditioning/reweighting; (2) a finite approximation theorem linking this semantics to finite traces and FK models, thereby establishing the consistency of PF for PPGs; (3) a vectorized version of the PF algorithm based on PPGs.

Related work The book [7] is an introduction to and survey of recent literature on PPLs. A more concise introduction can be found in the review article [26]. With few notable exceptions, most work on the semantics of PPL follows the denotational approach initiated by Kozen [32]. This include, among others, the works of Borgström, Gordon et al. [16], Staton et al. [45, 46] and Scibior et al. [44]. In this context, a general goal here is devising denotational semantics driven methods to combine and reason on densities, such as in [11, 27, 28, 48]. This goal is quite orthogonal to what we do here; in particular, we do not require that a PP induce a density on the probability space of infinite traces.

Relevant to our approach is a series of works by Lunden et al. on SMC inference applied to PPLs. In [34], for a lambda-calculus enriched with an explicit resample primitive, consistency of PF is shown to hold, under certain restrictions, independently of the placements of the resample's in the code. Operationally, their functional approach is very different from our automata-theoretic one. In particular, they handle suspension and resumption of particles in correspondence of resampling via an implicit use of *continuations*, in the style of webPPL [25] and other PPLs. The combination of functional style and continuations does not naturally lend itself to vectorization. For instance, ensuring that all particles are *aligned*, that is are at a resample point of their execution, is an issue that can impact negatively on performance or accuracy. On the contrary, in our automata-theoretic model, placement of resamples and alignment are not issues: resampling always happens after each (vectorized) transition step, so all particles are automatically aligned. Note that in PPGs a transition can group together complicated, conditioning free computations; in any case, consistency of PF is guaranteed. In a subsequent work [35, 36], Lunden al. study concrete implementation issues of SMC. In [35], they consider *PPL Control-Flow Graphs* (PCFGs), a structure intended as a target for the compilation of high-level PPLs, such as their CorePPL. The PCFG model is very similar in spirit to PPGs, however, it lacks a formal semantics. Lunden et al. also offer an implementation of this framework, designed to take advantage of the potential parallelism existing at the level of particles. We compare our implementation with theirs in Section 6.

PCFGs are also considered as the basic operational model in a series of works by K. Chatterjee et al., see e.g. [18] and references therein. The operational semantics they consider is similar to ours, but they do not consider an expected-based semantics, where we incorporate weights. More generally, in their works emphasis is on verification, like obtaining certified bounds on the termination probability of a given program. Issues related to conditioning and to consistency of sampling algorithms are not considered. Wang et al. [49] also consider a model very similar to PCFGs, but their focus is on exact/symbolic techniques (see further below).

Adiyta et al. prove consistency of Markov Chain Monte Carlo (MCMC) for their PPL R2 [40], which is based on a big-step sampling semantics that considers finite execution paths. No approximation results bridging finite and infinite traces hence unbounded loops, is provided. It is also unclear if a big-step semantics would effectively translate into a SIMD-parallel algorithm. Wu et al. [51] provide the PPL Blog with a rigorous measure-theoretic semantics, formulated in terms of Bayesian Networks, and a very efficient implementation of the PF algorithm tailored to such networks. Again, they do not offer results for unbounded loops. In our previous work [13], we have considered a measure theoretic semantics for a PPL with unbounded loops, and provided a finite approximation result and a SIMD-parallel implementation, with guarantees, of what is in effect a *rejection sampling* algorithm. Rejection may be effective for limited forms of conditioning; but it rapidly becomes wasteful and ineffective as conditioning becomes more demanding, so to speak: e.g. when it is repeated in a loop, or the observed data have a low likelihood in the model. Finally, SMCP3 [33] provides a rich measure-theoretic framework for extending the practical Gen language [20] with expressive proposal distributions.

A rich area in the field of PPL focus on symbolic, exact techniques [43, 23, 38, 8, 10, 42, 29] aiming to obtain termination certificates, or certified bounds on termination probability of PPs, or even exact representations the posterior distribution; see also [6, 9, 47, 3, 31, 49] for some recent works in this direction. Our goal and methodology, as already stressed, are rather different, as we focus on inference via sampling and the ensuing consistency issues. While certificates in the traditional sense are impossible to obtain in our framework, we hope to demonstrate that the resulting methodology is still practically useful and much more scalable.

2 Preliminaries on measure theory

We review a few basic concepts from measure theory following closely the presentation in the first two chapters of [2], which is a reference for whatever is not explicitly described below. Given a nonempty set Ω, a *sigma-field* $\mathcal{F}$ on Ω is a collection of subsets of Ω that contains Ω, and is closed under complement and under countable disjoint union. The pair $(\Omega, \mathcal{F})$ is called a *measurable space*. A (total) function $f : \Omega_1 \to \Omega_2$ is *measurable* w.r.t. the sigma-fields $(\Omega_1, \mathcal{F}_1)$ and $(\Omega_2, \mathcal{F}_2)$ if whenever $A \in \mathcal{F}_2$ then $f^{-1}(A) \in \mathcal{F}_1$. We let $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$ be the set of extended reals, assuming the standard arithmetic for $\pm\infty$ (cf. [2, Sect.1.5.2]), and $\overline{\mathbb{R}}^+$ the set of nonnegative reals including $+\infty$. The *Borel sigma-field* $\mathcal{F}$ on $\Omega = \overline{\mathbb{R}}^m$ is the minimal sigma-field that contains all rectangles of the form $[a_1, b_1] \times \cdots \times [a_n, b_n]$, with $a_i, b_i \in \overline{\mathbb{R}}$. An important case of measurable spaces $(\Omega, \mathcal{F})$ is when $\Omega = \overline{\mathbb{R}}^m$ for some $m \geq 1$ and $\mathcal{F}$ is the Borel sigma-field over Ω. Throughout the paper, *"measurable" means "Borel measurable"*, both for sets and for functions. On functions, Borel measurability is preserved by composition and other elementary operations on functions; continuous real functions are Borel measurable. We will let $\mathcal{F}_k$ denote the Borel sigma-field over $\overline{\mathbb{R}}^k$ $(k \geq 1)$ when we want to be specific about the dimension of the space.

A *measure* over a measurable space $(\Omega, \mathcal{F})$ is a function $\mu : \mathcal{F} \to \overline{\mathbb{R}}^+$ that is countably additive, that is $\mu(\cup_{j\geq 1} A_j) = \sum_{j\geq 1} \mu(A_j)$ whenever A_j's are pairwise disjoint sets in $\mathcal{F}$. The *Lebesgue integral* of a Borel measurable function f w.r.t. a measure μ [2, Ch.1.5], both defined over a measure space $(\Omega, \mathcal{F})$, is denoted by $\int_\Omega \mu(d\omega) f(\omega)$, with the subscript Ω omitted when clear from the context. When μ is the standard Lebesgue measure, we may omit μ and write the integral as $\int_\Omega d\omega f(\omega)$. For $A \in \mathcal{F}$, $\int_A \mu(d\omega) f(\omega)$ denotes $\int_\Omega \mu(d\omega) f(\omega) 1_A(\omega)$, where $1_A(\cdot)$ is the indicator function of the set A. We let δ_v denote Dirac's measure concentrated on v: for each set A in an appropriate sigma-field, $\delta_v(A) = 1$ if $v \in A$, $\delta_v(A) = 0$ otherwise. Otherwise said, $\delta_v(A) = 1_A(v)$. Another measure that arises (in connections with discrete distributions) is the counting measure, $\mu_C(A) := |A|$. In particular, for a nonnegative f, we have the equality $\int_A \mu_C(d\omega) f(\omega) =$

$\sum_{\omega \in A} f(\omega)$. A *probability measure* is a measure μ defined on $\mathcal{F}$ such that $\int \mu(du) = 1$. For a given nonnegative measurable function f defined over Ω, its *expectation* w.r.t. a probability measure ν is just its integral: $\mathrm{E}_\nu[f] = \int \nu(d\omega) f(\omega)$. The following definition is central.

Definition 1 (Markov kernel). *Let $(\Omega_1, \mathcal{F}_1)$ and $(\Omega_2, \mathcal{F}_2)$ be measurable spaces. A function $K : \Omega_1 \times \mathcal{F}_2 \longrightarrow \overline{\mathbb{R}}^+$ is a* Markov kernel *from Ω_1 to Ω_2 if it satisfies the following properties:*

1. *for each $\omega \in \Omega_1$, the function $K(\omega, \cdot) : \mathcal{F}_2 \to \overline{\mathbb{R}}^+$ is a probability measure on $(\Omega_2, \mathcal{F}_2)$;*
2. *for each $A \in \mathcal{F}_2$, the function $K(\cdot, A) : \Omega_1 \to \overline{\mathbb{R}}^+$ is measurable.*

Notationally, we will most often write $K(\omega, A)$ as $K(\omega)(A)$. The following is a standard result about the construction of finite product of measures over a product space[1] $\Omega^t = \Omega \times \cdots \times \Omega$ (t times) for $t \geq 1$ an integer(see Theorem A.1 in the Appendix for a more detailed statement).It is customary to denote the measure μ^t defined by the theorem also as $\mu^1 \otimes K_2 \otimes \cdots \otimes K_t$.

Theorem 1 (product of measures). *Let $t \geq 1$ be an integer. Let μ^1 be a probability measure on Ω and $K_2, ..., K_t$ be $t-1$ (not necessarily distinct) Markov kernels from Ω to Ω. Then there is a unique probability measure μ^t defined on $(\Omega^t, \mathcal{F}^t)$ such that for every $A_1 \times \cdots \times A_t \in \mathcal{F}^t$ we have: $\mu^t(A_1 \times \cdots \times A_t) = \int_{A_1} \mu^1(d\omega_1) \int_{A_2} K_2(\omega_1)(d\omega_2) \cdots \int_{A_t} K_t(\omega_{t-1})(d\omega_t)$.*

3 Probabilistic programs

We first introduce a general formalism for specifying programs, in the form of certain graphs that can be regarded as symbolic finite automata. For this formalism, we introduce an operational semantics in terms of Markov kernels.

Probabilistic Program Graphs In defining probabilistic programs, we will rely on a repertoire of basic distributions: continuous, discrete and mixed distributions will be allowed. A crucial point for expressiveness is that a measure may depend on *parameters*, whose value at runtime is determined by the state of the program. To ensure that the resulting programs define measurable functions (on a suitable space), it is important that the dependence between the parameters and the measure be in turn of measurable type. We will formalize this in terms of Markov kernels. Additionally, we will consider score functions, a generalization of 0/1-valued predicates. Formally, we will consider the two families of functions defined below. In the definitions, we will let $m \geq 1$ denote a fixed integer, representing the number of *variables* in the program, conventionally referred to as $x_1, ..., x_m$. We will let v range over $\overline{\mathbb{R}}^m$, the content of the program variables in a given state, or *store*.

- *Parametric measures*: Markov kernels $\zeta : \overline{\mathbb{R}}^m \times \mathcal{F}_m \to [0, 1]$.
- *Score functions*: measurable functions $\gamma : \overline{\mathbb{R}}^m \to [0, 1]$. A *predicate* is a special case of a score function $\varphi : \overline{\mathbb{R}}^m \to \{0, 1\}$. An Iverson bracket style notation will be often employed, e.g.: $[x_1 \geq 1]$ is the predicate that on input v yields 1 if $v_1 \geq 1$, 0 otherwise.

For a parametric measure ζ and a store $v \in \overline{\mathbb{R}}^m$, $\zeta(v)$ is a distribution, that can be used to sample a new store $v' \in \overline{\mathbb{R}}^m$ depending on the current program store v. Analytically, ζ may be expressed by, for instance, chaining together sampling of individual components of the store. This can be done by relying on *parametric densities*: measurable functions $\rho : \overline{\mathbb{R}}^m \times \overline{\mathbb{R}} \to \overline{\mathbb{R}}^+$ such that, for a designated measure μ_ρ, the function $(v, A) \mapsto \int_A \mu_\rho(dr)\, \rho(v, r)$ $(A \in \mathcal{F}_m)$ is a Markov kernel from $\overline{\mathbb{R}}^m$ to $\overline{\mathbb{R}}$. This is explained via the following example.

Example 1. Fix $m = 2$. Consider the Markov kernel defined as follows, for each $x_1, x_2 \in \overline{\mathbb{R}}$ and $A \in \mathcal{F}_2$

$$\zeta(x_1, x_2)(A) := \int \mu_1(dr_1) \left(\rho_1(x_1, x_2, r_1) \cdot \int \mu_2(dr_2) \rho_2(r_1, x_2, r_2) 1_A(r_1, r_2) \right) \tag{1}$$

where: $\mu_1 = \mu_C$ is the counting measure; $\rho_1(x_1, x_2, r) = \frac{1}{2} 1_{\{x_1\}}(r) + \frac{1}{2} 1_{\{x_2\}}(r)$ is the density of a discrete distribution on $\{x_1, x_2\}$; $\mu_2 = \mu_L$ is the ordinary Lebesgue measure; $\rho_1(x_1, x_2, r) = N(x_1, x_2, r) := \frac{1}{|x_2|\sqrt{2\pi}} \exp(-\frac{1}{2}(\frac{r - x_1}{|x_2|})^2)$ is the density of the Normal distribution of mean x_1 and standard deviation[2] $|x_2|$.

[1] We shall freely identify language-theoretic *words* with *tuples*, hence use the notations $A_1 \cdot A_2 \cdot \cdots \cdot A_k$ and $A_1 \times A_2 \times \cdots \times A_k$ interchangeably. This convention will also apply to infinite words (cf. Section 4).

[2] With the proviso that, when $x_2 = 0$ or $|x_1|, |x_2| = +\infty$, $N(x_1, x_2, r)$ denotes an arbitrarily fixed, default probability density.

Fig. 1: **Left**. The PPG of Example 2. The nil node (2) is distinguished with a double border. Constant 1 predicates and score functions, and the identity function are not displayed in transitions. The score function γ decorates node 3, that is $\mathbf{sc}(3) = \gamma$. **Right**. The PPG for the drunk man and mouse random walk of Example 3. The score function γ decorates node 1, that is $\mathbf{sc}(1) = \gamma$.

The function ζ is a parametric measure: concretely, it corresponds to first sampling uniformly r_1 from the the set $\{x_1, x_2\}$, then sampling r_2 from the Normal distribution of mean r_1 and s.d. $|x_2|$ (if $|x_2|$ is positive and finite, otherwise from a default distribution). Rather than via (1), we will describe ζ via the following more handy notation:

$$r_1 \sim \rho_1(x_1, x_2)\,;\, r_2 \sim \rho_2(r_1, x_2)$$

(or listed top-down). Note that the sampling order from left to right is relevant here.

In fact, as far as the formal framework of PPGs introduced below is concerned, how the parametric measures ζ's are analytically described is irrelevant. From the practical point of view, it is important we know how to (efficiently) sample from the measure $\zeta(v)$, for any v, in order for the inference algorithms to be actually implemented (see Section 5). In concrete terms, $\zeta(v)$ might represent the (possibly unknown) distribution of the outputs in $\overline{\mathbb{R}}^m$ returned by a piece of code, when invoked with input v. Another important special case of parametric measure is the following. For any $v = (v_1, \ldots, v_m) \in \overline{\mathbb{R}}^m$, $r \in \overline{\mathbb{R}}$ and $1 \leq i \leq m$, let $v[r@i] := (v_1, \ldots, r, \ldots, v_m)$ denote the tuple where v_i has been replaced by r. Consider the parametric measure $\zeta(v) = \delta_{v[g(v)@i]}$, where $g : \overline{\mathbb{R}}^m \to \overline{\mathbb{R}}$ is a measurable function. In programming terms, this corresponds to the deterministic *assignment* of the value $g(v)$ to the variable x_i. We will describe this ζ as: $x_i := g(x_1, \ldots, x_m)$.

In the definition of PPG below, one may think of the computation (sampling) taking place in successive stages on the edges (transitions) of the graph, with nodes serving as *checkpoints* (a term we have borrowed from [34]) between stages for conditioning on observed data — or, more generally, re-weighting the score assigned to a computation. The edges also accounts for the control flow among the different stages via predicates computed on the store of the source nodes.

Definition 2 (PPG). *Fix $m \geq 1$. A* Probabilistic Program Graph (PPG) *on $\overline{\mathbb{R}}^m$ is 4-tuple $\mathbf{G} = (\mathcal{P}, E, \mathsf{nil}, \mathbf{sc})$ satisfying the following.*

- *$\mathcal{P} = \{S_1, \ldots, S_k\}$ is a finite, nonempty set of* program checkpoints *(*programs*, for short).*
- *E is a finite, nonempty set of* transitions *of the form (S, φ, ζ, S'), where: $S, S' \in \mathcal{P}$ are called the* source *and* target *program checkpoint, respectively; $\varphi : \overline{\mathbb{R}}^m \to \{0,1\}$ is a predicate; and $\zeta : \overline{\mathbb{R}}^m \times \mathcal{F}_m \to [0,1]$ is a parametric measure.*
- *$\mathsf{nil} \in \mathcal{P}$ is a distinguished* terminated *program checkpoint, such that $(\mathsf{nil}, 1, \mathrm{id}, \mathsf{nil})$ (id = identity) is the only transition in E with nil as source.*
- *$\mathbf{sc}$ is a mapping from $\mathcal{P}$ to the set of score functions, s.t. $\mathbf{sc}(\mathsf{nil})$ is the constant 1.*

Additionally, denoting by E_S the set of transitions in E with S as a source checkpoint, the following consistency *condition is assumed: for each $S \in \mathcal{P}$, the function $\sum_{(S,\varphi,\zeta,S') \in E_S} \varphi$ is the constant 1.*

We first illustrate Definition 2 with a simple example. This will also serve to illustrate the finite approximation theorem later on in this section.

Example 2. Consider the PPG in Fig. 1, left. Here we have $m = 2$ and $B(p)$ is the Bernoulli distribution with success probability p. In a more conventional notation, the resulting program might be described as follows, where `skip` represents termination.

```
c~B(1/2);
if (c==0) skip
else {d~B(1/2); observe(d==1); skip}
```

We will not pursue a systematic formal translation from this program notation to PPGs, though.

The following example illustrate the use of scoring functions inside loops. It is a bit contrived, but close to the structure of more significant scenarios, such as the aircraft tracking example of [51], cf. Section 6.

Example 3 (of mice and drunk men). Consider the following variation on the classical drunk man's random walk. On a street, a drunk man and a mouse perform independent random walks starting at conventional positions -1 and 1 respectively. Initially, each of them samples a value from a uniform distribution, to be used as a standard deviation (s.d.) of the length of subsequent steps: the drunk man samples d from $(0, 2)$, the mouse r from $(0, 1)$. Then, at each discrete time step, they independently sample their own next position from a Normal distribution centered at the current position (x man, y mouse), with the s.d. (d man, r mouse) chosen at the beginning. The process is stopped as soon as the man and the mouse meet, which we take to mean the distance between them is $< 1/10$.

The man and the mouse' actions are independent. On the other hand, it has been suggested that in certain urban areas a man is never more than 3m away from a mouse [37]. Let us take this information at face value, and incorporate it in our model: we let $\mathbf{sc}(\cdot)$ associate the appropriate checkpoint in the graph with the score function $\gamma := [|x - y| \leq 3]$ — actually a predicate written in Iverson bracket notation. The resulting PPG, $\mathbf{G} = (\mathcal{P}, E, \mathsf{nil}, \mathbf{sc})$, has $m = 4$, three checkpoints and the transition structure described in Figure 1, right.

Remark 1. The definition of score function requires that γ takes values on $[0, 1]$. This is restriction is necessary for the weight function to be defined on infinite traces (cf. (5)) be well-defined. A similar restriction is found in e.g. [16]. In practice, provided that γ is nonnegative and bounded, we can always divide by an appropriate constant[3]. E.g., rather than $\gamma(x_1, x_2) = N(x_1, 0.1, x_2)$, we can consider $\gamma(x_1, x_2) = N(x_1, 0.1, x_2)/N(0, 0.1, 0)$. As far as the trace semantics is concerned (Section 4), this normalization will in fact be immaterial, as it will take place on both the numerator and the denominator. On the other hand, we cannot use unbounded score functions, like say, $\gamma(x_1, x_2) = N(x_1, x_2, x_2)$.

Operational semantics of PPGs For any given PPG $\mathbf{G}$, we will define a Markov kernel $\kappa_{\mathbf{G}}(\cdot, \cdot)$ that describes its operational semantics. From now on, we will consider one arbitrarily fixed PPG, $\mathbf{G} = (\mathcal{P}, E, \mathsf{nil}, \mathbf{sc})$ and just drop the subscript $_{\mathbf{G}}$ from the notation. Let us also remark that the scoring function $\mathbf{sc}(\cdot)$ will play no role in the definition of the Markov kernel — it will come into play in the trace based semantics of Section 4.

Some additional notational shorthand is in order. First, we identify $\mathcal{P}$ with the finite set of naturals $\{0, ..., |\mathcal{P}| - 1\}$. With this convention, we have that $\overline{\mathbb{R}}^m \times \mathcal{P} \subseteq \overline{\mathbb{R}}^{m+1}$. Henceforth, we define our state space and sigma-field as follows:

$$\Omega := \overline{\mathbb{R}}^{m+1} \qquad\qquad \mathcal{F} := \text{Borel sigma-field over } \overline{\mathbb{R}}^{m+1}.$$

We keep the symbol $\mathcal{F}_k$ for the Borel sigma-field over $\overline{\mathbb{R}}^k$, for any $k \geq 1$. For any $S \in \mathcal{P}$ and $A \in \mathcal{F}$, we let $A_S := \{v \in \overline{\mathbb{R}}^m \,:\, (v, S) \in A\}$ be the *section* of A at S. Note that $A_S \in \mathcal{F}_m$, as sections of measurable sets are measurable, see [2, Th.2.6.2,proof(1)].

Definition 3 (PPG Markov kernel). *The function $\kappa : \Omega \times \mathcal{F} \to \mathbb{R}^+$ is defined as follows, for each $\omega \in \Omega$ and $A \in \mathcal{F}$:*

$$\kappa(\omega)(A) := \begin{cases} \delta_\omega(A) & \text{if } \omega \notin \overline{\mathbb{R}}^m \times \mathcal{P} \\ \sum_{(S,\varphi,\zeta,S') \in E_S} \varphi(v) \cdot \zeta(v)\,(A_{S'}) & \text{if } \omega = (v, S) \in \overline{\mathbb{R}}^m \times \mathcal{P}. \end{cases} \tag{2}$$

Lemma 1. *The function κ is a Markov kernel from Ω to Ω.*

[3] Provided all the score functions appearing in the program are divided by the *same* constant, so as to avoid distorsive effects.

4 Trace semantics and finite approximation for PPGs

Trace semantics In what follows, we fix an arbitrary a PPG, $\mathbf{G} = (\mathcal{P}, E, \mathsf{nil}, \mathbf{sc})$ and let κ denote the induced Markov kernel, as per Definition 3. For any $t \geq 1$, we call Ω^t the set of *paths of length* t. Consider now the set of paths of infinite length, Ω^∞, that is the set of infinite sequences $\tilde{\omega} = (\omega_1, \omega_2, ...)$ with $\omega_i \in \Omega$. For any $\omega^t \in \Omega^t$ and $\tilde{\omega} \in \Omega^\infty$, we identify the pair $(\omega^t, \tilde{\omega})$ with the element of Ω^∞ in which the prefix ω^t is followed by $\tilde{\omega}$. For $t \geq 1$ and a measurable $B_t \subseteq \Omega^t$, we let $\mathfrak{c}(B_t) := B_t \cdot \Omega^\infty \subseteq \Omega^\infty$ be the *measurable cylinder* generated by B_t. We let C be the minimal sigma-field over Ω^∞ generated by all measurable cylinders. Under the same assumptions of Theorem 1 on the measure μ^1 and on the kernels $K_2, K_3, ...$ there exists a unique measure μ^∞ on C such that for each $t \geq 1$ and each measurable cylinder $\mathfrak{c}(B_t)$, it holds that $\mu^\infty(\mathfrak{c}(B_t)) = \mu^t(B_t)$: see [2, Th.2.7.2], also known as the *Ionescu-Tulcea theorem*. In the definition below, we let $0 = (0, ..., 0)$ (m times) and consider $\delta_{(0,S)}$, the Dirac's measure on Ω that concentrates all the probability mass in $(0, S)$.

Definition 4 (probability measure induced by S**).** *Let $S \in \mathcal{P}$. For each integer $t \geq 1$, we let μ_S^t be the probability measure over Ω^t uniquely defined by Theorem 1(a) by letting $\mu^1 = \delta_{(0,S)}$ and $K_2 = \cdots = K_t = \kappa$. We let μ_S^∞ be the unique probability measure on C induced by μ_1 and $K_2 = \cdots = K_t = \cdots = \kappa$, as determined by the Ionescu-Tulcea theorem.*

In other words, $\mu_S^t = \delta_{(0,S)} \otimes \kappa \otimes \cdots \otimes \kappa$ ($t-1$ times κ). By convention, if $t = 1$, $\mu_S^t = \delta_{(0,S)}$. The measure μ_S^∞ can be informally interpreted as the limit of the measures μ_S^t and represents the semantics of S.

The following is a general lemma useful to connect measure over sets of infinite and finite traces. In its statement, we let μ^∞ denote a generic measure on the cylindrical sigma-field, obtained as an infinite product of kernels in the sense of the Ionescu-Tulcea theorem, and by μ^t the corresponding finite product measures. We shall make use of the following notation. For $\tilde{\omega} = (\omega_1, \omega_2, ...) \in \Omega^\infty$, we let $\tilde{\omega}_{1:t} := (\omega_1, ..., \omega_t) \in \Omega^t$. For $h : \Omega^t \to \overline{\mathbb{R}}^+$ a nonnegative function, we let $\tilde{h} : \Omega^\infty \to \overline{\mathbb{R}}^+$ be defined as follows for each $\tilde{\omega} \in \Omega^\infty$:

$$\tilde{h}(\tilde{\omega}) := h(\tilde{\omega}_{1:t}) \,. \tag{3}$$

Lemma 2. *Let $h : \Omega^t \to \overline{\mathbb{R}}^+$ a nonnegative measurable function. Then $\tilde{h}$ as defined in* (3) *is measurable. Moreover, for each measurable cylinder $\mathfrak{c}(B_t) \subseteq \Omega^\infty$ ($B_t \subseteq \Omega^t$), we have $\int_{\mathfrak{c}(B_t)} \mu^\infty(d\tilde{\omega})\tilde{h}(\tilde{\omega}) = \int_{B_t} \mu^t(d\omega^t)h(\omega^t)$.*

Recall that the *support* of an (extended) real valued function f is the set $\mathrm{supp}(f) := \{z : f(z) \neq 0\}$. In what follows, *we shall concentrate on nonnegative measurable functions* f to avoid unnecessary complications with the existence of integrals. General functions can be dealt with by the usual trick of decomposing f as $f = f^+ - f^-$, where $f^+ = \max(0, f)$ and $f^- = -\min(0, f)$, and then dealing separately with f^+ and f^-. Let us introduce a *combined score function* $\mathrm{sc} : \Omega \to [0, 1]$ as follows, for each $\omega = (v, S)$:

$$\mathrm{sc}(\omega) := \begin{cases} \mathbf{sc}(S)(v) & \text{if } \omega = (v, S) \in \overline{\mathbb{R}}^m \times \mathcal{P} \\ 1 & \text{otherwise.} \end{cases} \tag{4}$$

The function $\mathrm{sc}(\cdot)$ is extended to a *weight function* on infinite traces, $\mathrm{w} : \Omega^\infty \to [0, 1]$ by letting[4], for any $\tilde{\omega} = (\omega_1, \omega_2, ...)$:

$$\mathrm{w}(\tilde{\omega}) := \Pi_{j\geq 1}\mathrm{sc}(\omega_j) \,. \tag{5}$$

For each $t \geq 1$, we define the weight function truncated at time t, $\mathrm{w}_t : \Omega^t \to [0, 1]$, by $\mathrm{w}_t(\omega^t) := \Pi_{j=1}^t \mathrm{sc}(\omega_j)$. Both w and w_t ($t \geq 1$) are measurable functions on the respective domains(see Lemma A.1). We arrive at the definition of the semantics of programs. We consider the ratio of the unnormalized semantics ($[S]f$) to the weight of all traces, terminated or not ($[S]\mathrm{w}$). In the special case when the score functions represent conditioning, this choice corresponds to quotienting over the probability of *non failed* traces. In PPL, quotienting over non failed states is somewhat standard: see e.g. the discussion in [30, Section 8.3.2].

Definition 5 (trace semantics). *Let f be a nonnegative measurable function defined on Ω^∞. We let the* unnormalized semantics of S and f *be $[S]f := \mathrm{E}_{\mu_S^\infty}[f]$ ($= \int \mu_S^\infty(d\tilde{\omega})f(\tilde{\omega})$). We let*

$$[[S]]f := \frac{[S]f \cdot \mathrm{w}}{[S]\mathrm{w}} \tag{6}$$

provided the denominator above is > 0; otherwise $[[S]]f$ is undefined.

[4] Note that $\mathrm{w}(\tilde{\omega})$ is well-defined because $0 \leq \mathrm{sc}(\omega_j) \leq 1$ for each $j \geq 0$, hence the series of partial products is monotonically nonincreasing.

Finite approximation We are mainly interested in $[[S]]f$ in cases where the value of f is, informally speaking, determined by a finite prefix of its argument: we call these functions *prefix-closed*, and will define them further below. We first have to introduce prefix-closed languages[5], for which some notation on languages of finite and infinite words is useful. Given two words $w, w' \in \Omega^*$, we write $w \prec w'$ if w is a prefix of w', i.e. there exists a word $w'' \in \Omega^*$ such that $ww'' = w'$; otherwise we write $w \nprec w'$. For $L, L' \subseteq \Omega^*$, we write $L \nprec L'$ if for any $w \in L$ and $w' \in L'$ we have $w' \nprec w'$. A sequence of languages $L_0, L_1, \ldots$ such that for each j, $L_j \subseteq \Omega^j$ (with $\Omega^0 := \{\epsilon\}$, the empty sequence) is said to be *prefix-free* if for each $i \neq j$, $L_i \nprec L_j$. Note that if $L_0 \neq \emptyset$ then $L_j = \emptyset$ for $j \geq 1$. For the sake of uniform notation, in what follows we convene that $\omega^0 := \epsilon$ and $\mathfrak{c}(\{\epsilon\}) := \Omega^\infty$. We say $A \subseteq \Omega^\infty$ is a *prefix closed set* if there is a prefix-free sequence of languages $L_0, L_1, \ldots$ such that $A = \cup_{j=0}^{\infty} \mathfrak{c}(L_j)$; we call L_j a j*-branch* of A, and refer to $L_0, L_1, \ldots$ collectively as *branches of* A. For any $t \geq 1$, we define the following subsets of Ω^t:

$$L^{\leq t} := \cup_{j=0}^{t} L_j \cdot \Omega^{t-j}, \qquad L^{>t} := \{\omega^t \ : \ \text{there is } t' > t \text{ and } \omega_{t'} \in L_{t'} \text{ s.t. } \omega^t \prec \omega^{t'}\}\,.$$

Informally speaking, $L^{\leq t}$ is the set of paths of length t that will become members of A however we extend them to infinite words. $L^{>t}$ is the set of paths of length t for which some infinite extensions, but not all, are in A — they are so to speak "undecided". Of special interest is the prefix-free sequence of languages defined below.

Definition 6 (termination). *Let* $\mathsf{T} := \overline{\mathbb{R}}^m \times \{\mathsf{nil}\}$ *be the set of* terminated *states. We let* $T_j \subseteq \Omega^j$ $(j \geq 0)$ *be the set of finite sequences that* terminate at time j*, that is:* $T_0 := \emptyset$ *and* $T_j := (\mathsf{T}^c)^{j-1} \cdot \mathsf{T}$*, for* $j \geq 1$*. We let* $T_\mathrm{f} := \cup_{t\geq 0}\mathfrak{c}(T_t) \subseteq \Omega^\infty$ *denote the set of infinite sequences that* terminate in finite time.

Note that $\{T_j : j \geq 0\}$ forms a prefix-free sequence, that $T^{\leq t} \subseteq \Omega^t$ is the set of all paths of length t that terminate within time t, while $\mathfrak{c}(T_t) \subseteq \Omega^\infty$ is the set of infinite execution paths with termination at time t.

The next definition introduces prefix-closed functions. These are functions f with a prefix-free support, condition (a), additionally satisfying two extra conditions. Condition (b) just states that the value of f on its support is determined by a finite prefix of the input sequence. Condition (c), T-respectfulness, means that a trace that terminates at time j ($\omega^j \in T_j$) cannot lead to $\mathrm{supp}(f)$ at a later time ($\omega^j \notin L^{>j}$). This is a consistency condition, formalizing that the value of f does not depend on, so to speak, what happens *after* termination.

Definition 7 (prefix-closed function). *Let* $f : \Omega^\infty \to \overline{\mathbb{R}}^+$ *be a nonnegative measurable function and* $(L_0, L_1, \ldots)$ *be a prefix-closed sequence. We say* f *is a* prefix-closed function *with branches* $L_0, L_1, \ldots$ *if the following conditions are satisfied.*

(a) $\mathrm{supp}(f)$ *is prefix-free with branches* L_j $(j \geq 0)$.
(b) for each $j \geq 0$ *and* $\omega^j \in L_j$, f *is constant on* $\mathfrak{c}(\{\omega^j\})$.
(c) $\mathrm{supp}(f)$ *is* T-respectful*: for each* $j \geq 0$, $L^{>j} \cap T_j = \emptyset$.

Note that there may be different prefix-free sequences w.r.t. which f is prefix-closed.

Example 4. The indicator function 1_{T_f} is clearly a prefix closed, measurable function with $\mathrm{supp}(1_{T_\mathrm{f}}) = T_\mathrm{f}$ and branches $L_j = T_j$. For more interesting examples, consider the PPG in Example 3 and the following functions.

- $f_1(\tilde{\omega}) = j$ if $\omega_j \in \mathsf{T}$ is the first terminated state occurring in $\tilde{\omega}$, if such a ω_j exists; 0 otherwise.
- $f_2(\tilde{\omega}) = d$ if $\omega = (d, r, x, y, \mathsf{nil}) \in \mathsf{T}$ is the first terminated state occurring in $\tilde{\omega}$ and $d \in [0, 2]$, if such a ω_j exists; 0 otherwise.

f_1 returns the time the process terminates: $\mathrm{supp}(f_1) = T_\mathrm{f}$ has branches $L_j = T_j$ $(j \geq 0)$. f_2 returns the value of d at termination. Here $\mathrm{supp}(f_2) = \{\tilde{\omega} \in T_\mathrm{f} \ : \ $ the first terminated state ω in $\tilde{\omega}$, if it exists, has $\omega(1) = d \in (0, 2]\}$, and $L_0 = \emptyset$, $L_j = (\mathsf{T}^c)^{j-1} \cdot (\mathsf{T} \cap ((0, 2] \times \overline{\mathbb{R}}^4))$ $(j \geq 1)$.

[5] In the context of model checking, these languages arise as complements of Safety properties; see e.g. [5, Def.3.22].

We will now study how to consistently approximate infinite computations (μ_S^∞ semantics) with finite ones (μ_S^t semantics). This will lead to the main result of this section (Theorem 2). As a first step, let us introduce an appropriate notion of finite approximation for functions f defined on the infinite product space Ω^∞. Fix an arbitrary element $\star \in \Omega$. For each $f : \Omega^\infty \to \overline{\mathbb{R}}^+$ and $t \geq 1$, let us define the function $f_t : \Omega^t \to \overline{\mathbb{R}}^+$ by

$$f_t(\omega^t) := f(\omega^t, \star^\infty) .$$

The intuition here is that, for a prefix-closed function f, the function f_t approximates correctly f for all finite paths in the L_j-branches of f, for $j \leq t$. Consider for instance the function $f = f_1$ in Example 4. On $L^{\leq t}$, the approximation f_t gives the correct value w.r.t. f in a precise sense: $f_t(\omega^t) = f(\omega^t, \star^\infty) = f(\omega^t, \tilde{\omega}')$ whatever $\star$ and $\tilde{\omega}'$. On the other hand, for finite paths $\omega^t \in L^{>t}$, f_t may not approximate f correctly: we may have $f_t(\omega^t) = f(\omega^t, \star^\infty) \neq f(\omega^t, \tilde{\omega}')$ depending on the specific $\star$ and $\tilde{\omega}'$. The catch is, as t grows large, the set $L^{>t}$ will become thinner and thinner — at least under reasonable assumptions on the measure μ_S^∞.

It is not difficult to check that, for any t, f_t is measurable over Ω^t(Lemma A.1 in the Appendix).The next result shows how to approximate $[\![S]\!]f$ with quantities defined *only in terms of* f_t, w_t *and* μ_S^t, which is the basis for the sampling-based inference algorithm in the next section. Formally, for $t \geq 1$ and a measurable function $h : \Omega^t \to \overline{\mathbb{R}}^+$, we let

$$[S]^t h := \mathrm{E}_{\mu_S^t}[h] \ \left(= \int \mu_S^t(d\omega^t) h(\omega^t)\right) .$$

The finite approximation theorem for $[\![S]\!]f$ relies on the following approximation lemma for the unnormalized semantics. The intuitive content of the lemma is as follows. Consider a prefix closed function f with branches $L_0, L_1, \ldots$. For any time t, it is not difficult to see that $\mathfrak{c}(L^{\leq t} \cap T^{\leq t}) \subseteq \mathrm{supp}(f) \subseteq \mathfrak{c}(L^{\leq t} \cap T^{\leq t}) \cup (\mathfrak{c}(T^{\leq t}))^c$ (the last inclusion involves T-respectfulness). As f_t approximates correctly f on $L^{\leq t}$ one sees that the first inclusion leads to the lower bound $[S]^t f_t \cdot 1_{L^{\leq t} \cap T^{\leq t}} \cdot \mathrm{w}_t \leq [S] f \mathrm{w}$. As for the upper bound, the intuition is that, over $(\mathfrak{c}(T^{\leq t}))^c$, f is upper-bounded by M.

Lemma 3. *Let $t \geq 1$ and let $f \leq M$ be a prefix-closed function with branches L_j ($j \geq 0$). Then*

$$[S]^t f_t \cdot 1_{L^{\leq t} \cap T^{\leq t}} \cdot \mathrm{w}_t \leq [S] f \cdot \mathrm{w} \leq [S]^t f_t \cdot 1_{L^{\leq t} \cap T^{\leq t}} \cdot \mathrm{w}_t + M \cdot [S]^t (1 - 1_{T^{\leq t}}) \cdot \mathrm{w}_t . \tag{7}$$

The proof of the main result follows by applying the above lemma to the numerators and denominators of the expressions involved in (8). In the formulation of the upper bound, we find it convenient to introduce a ‘correction factor’ $\alpha_t \geq 1$, the ratio of the weight of *all* traces to *terminated* traces at time t. We premise a technical lemma on convergence of integrals.

Lemma 4. *Let $S \in \mathcal{P}$. As $t \to +\infty$, we have $[S]^t 1_{\mathrm{T}^{\leq t}} \cdot \mathrm{w}_t \longrightarrow [S] 1_{\mathrm{T_f}} \cdot \mathrm{w}$. Moreover, the sequence $[S]^t 1_{\mathrm{T}^{\leq t}} \cdot \mathrm{w}_t$ ($t \geq 1$) is monotonically nondecreasing.*

Theorem 2 (finite approximation). *Consider $S \in \mathcal{P}$ and $t \geq 1$ such that $[S]^t 1_{T^{\leq t}} \cdot \mathrm{w}_t > 0$. Then for any prefix-closed function f with branches $L_0, L_1, \ldots$ we have that $[\![S]\!]f$ is well defined. Moreover, given an upper bound $f \leq M$ ($M \in \overline{\mathbb{R}}^+$), for each t large enough and $\alpha_t := \frac{[S]^t \mathrm{w}_t}{[S]^t 1_{T^{\leq t}} \cdot \mathrm{w}_t}$ we have:*

$$\frac{[S]^t f_t \cdot 1_{L^{\leq t} \cap T^{\leq t}} \cdot \mathrm{w}_t}{[S]^t \mathrm{w}_t} \leq [\![S]\!]f \leq \frac{[S]^t f_t \cdot 1_{L^{\leq t} \cap T^{\leq t}} \cdot \mathrm{w}_t}{[S]^t \mathrm{w}_t} \alpha_t + M \cdot (\alpha_t - 1) . \tag{8}$$

Proof. We first show that $[\![S]\!]f$ is well defined, that is that $[S]\mathrm{w} > 0$. Indeed, from Lemma 4, and from $[S]^t 1_{\mathrm{T}^{\leq t}} \cdot \mathrm{w}_t > 0$ for at least one t, we get $[S] 1_{\mathrm{T_f}} \cdot \mathrm{w} > 0$; since $1_{\mathrm{T_f}} \cdot \mathrm{w} \leq \mathrm{w}$, we get $[S] 1_{\mathrm{T_f}} \cdot \mathrm{w} \leq [S]\mathrm{w}$, hence the wanted statement.

Now consider $[\![S]\!]f = \dfrac{[S] f \cdot \mathrm{w}}{[S]\mathrm{w}}$, for f like in the hypothesis, and the inequalities in (8). Consider the following bounds for the numerator and denominator of this fraction.

$$[S]^t f_t \cdot 1_{L^{\leq t} \cap T^{\leq t}} \cdot \mathrm{w}_t \leq [S] f \cdot \mathrm{w} \leq [S]^t f_t \cdot 1_{L^{\leq t} \cap T^{\leq t}} \cdot \mathrm{w}_t + M([S]^t \mathrm{w}_t - [S]^t 1_{T^{\leq t}} \cdot \mathrm{w}_t) \tag{9}$$

$$[S]^t 1_{T^{\leq t}} \cdot \mathrm{w}_t \leq [S]\mathrm{w} \leq [S]^t \mathrm{w}_t . \tag{10}$$

The bounds in (9) are just those in Lemma 3, with the term $M \cdot (\cdots)$ written in an equivalent form. As to (10), first apply the bounds of Lemma 3 to the constant function $f = 1$. Note that this f is measurable, and is trivially prefix closed for the prefix-free sequence of languages $L_0 = \{\epsilon\}$ and $L_j = \emptyset$ for $j > 0$. As a consequence, for $t \geq 1$, over Ω^t we have $L^{\leq t} = \Omega^t$, hence $1_{L^{\leq t} \cap T^{\leq t}} = 1_{T^{\leq t}}$. Moreover $f_t = M = 1$ identically. From these facts, it is immediate to see that the bounds (7) of Lemma 3 specialize to (10). From the above established bounds (9) and (10) for the numerator and denominator of $[\![S]\!]f = \frac{[S]f \cdot \mathrm{w}}{[S]\mathrm{w}}$, it follows that

$$\frac{[S]^t f_t \cdot 1_{L^{\leq t} \cap T^{\leq t}} \cdot \mathrm{w}_t}{[S]^t \mathrm{w}_t} \leq [\![S]\!]f \leq \frac{[S]^t f_t \cdot 1_{L^{\leq t} \cap T^{\leq t}} \cdot \mathrm{w}_t}{[S]^t 1_{T^{\leq t}} \cdot \mathrm{w}_t} + M \cdot \left(\frac{[S]^t \mathrm{w}_t}{[S]^t 1_{T^{\leq t}} \cdot \mathrm{w}_t} - 1 \right). \tag{11}$$

Now, multiplying and dividing the first term of the above upper bound by $[S]^t \mathrm{w}_t$, positive by hypothesis, and recalling the definition of α_t, the wanted (8) follows.

When f is an indicator function, $f = 1_A$, we can of course take $M = 1$ in the theorem above. We first illustrate the above result with a simple example.

Example 5. Consider the PPG of Example 2 (Fig. 1, left). We ask what is the expected value of c upon termination of this program. Formally, we consider the program checkpoint $S = 0$, and the function f on traces that returns the value of c on the first terminated state, if any, and 0 elsewhere. This f is clearly prefix-closed with branches $L_j \subseteq T_j$. We apply Theorem 2 to $[\![S]\!]f$. Fixing the time $t = 4$, we can calculate easily the quantities involved in the approximation of $[\![S]\!]f$ in (8). In doing so, we must consider the finitely many paths of length t of nonzero probability and weight (there only two of them), their weights and the value of c on their final state when terminated[6].

$$[S]^t f_t \cdot 1_{T^{\leq t}} \cdot \mathrm{w}_t = 0 \cdot \tfrac{1}{2} + 1 \cdot \tfrac{1}{2} \cdot \tfrac{1}{2} = \tfrac{1}{4} \qquad [S]^t \mathrm{w}_t = \tfrac{1}{2} + \tfrac{1}{2} \cdot \tfrac{1}{2} = \tfrac{3}{4}$$
$$[S]^t 1_{T^{\leq t}} \cdot \mathrm{w}_t = \tfrac{1}{2} + \tfrac{1}{2} \cdot \tfrac{1}{2} = \tfrac{3}{4} \qquad \alpha_t = 1\,.$$

Then, with $M = 1$, the lower and upper bounds in (8) coincide and yield $[\![S]\!]f = \frac{1}{3}$. If we remove conditioning on node 4, then all the paths of length t have weight 1, and a similar calculation yields $[\![S]\!]f = \frac{1}{2}$.

In more complicated cases, we may not be able to calculate exactly the quantities involved in (8), but only to estimate them via sampling. To this purpose, we will introduce Feynman-Kac models and the Particle Filtering algorithm in the next section. For now, we content ourselves with the following example.

Example 6. Consider the PPG of Example 3 and $f = f_2$ from Example 4. Take $S = 0$. Then $[\![S]\!]f$ is the posterior expectation of the value of d, the drunk man's standard deviation. We can compute upper and lower bounds on $[\![S]\!]f$ using (8). Let us fix $t = 60$. By sampling from μ_S^t, we can compute separately the following estimates for each of the expected values involved in (8): $[S]^t f_t \cdot 1_{L^{\leq t} \cap T^{\leq t}} \cdot \mathrm{w}_t = 0.396$, $[S]^t \mathrm{w}_t = 0.987$ and $\alpha_t = 1.497$. Combining these estimates as in (8), with $M = 2$, we get the bounds: $0.402 \leq [\![S]\!]f \leq 1.596$. This relatively large interval can be narrowed down by considering higher values of t, hence α_t closer to 1. A more efficient and accurate method to compute the bounds in (8) will be introduced in Section 5, the Particle Filtering algorithm.

The theorem below confirms that the bounds established above are asymptotically tight, at least under the assumption that the program $S \in \mathcal{P}$ terminates with probability 1. In this case, in fact, the probability mass outside $\mathsf{T}^{\leq t}$ tends to 0, which leads the lower and the upper bound in (8) to coincide. Moreover, we get a simpler formula in the special case when termination is guaranteed to happen within a fixed time limit; for instance, in the case of acyclic[7] PPGs.

Theorem 3 (tightness). *Assume the same hypotheses as in Theorem 2. Further assume that $\mu_S^\infty(T_{\mathrm{f}}) = 1$. Then both the lower and the upper bounds in* (8) *tend to $[\![S]\!]f$ as $t \to +\infty$. In particular, if for some $t \geq 1$ we have $[S]^t 1_{T^{\leq t}} = 1$, then*

$$[\![S]\!]f = \frac{[S]^t f_t \cdot \mathrm{w}_t}{[S]^t \mathrm{w}_t}\,. \tag{12}$$

[6] Here, we also use the fact that $f_t \cdot 1_{T^{\leq t}} = f_t \cdot 1_{L^{\leq t} \cap T^{\leq t}}$, a consequence of $L_j \subseteq T_j$ for all js.

[7] Or, more accurately, PPGs where the only loop is the self-loop on the nil state.

Example 7. For the PPG of Example 2 one has $\mu_S^\infty(T_\mathrm{f}) = 1$. As already seen in Example 2, lower and upper bounds coincide for $t \geq 4$.

A practically relevant class of closed prefix functions are those where the result $f(\tilde{\omega})$ only depends on computing a function h, defined on Ω, on the first terminated state, if any, of the sequence $\tilde{\omega}$. This way h is *lifted* to Ω^∞. This case covers all the examples seen so far. We formally introduce lifting below. Recall that for $t \geq 1$, $T_t = (\mathsf{T}^c)^{t-1} \cdot \mathsf{T}$.

Definition 8 (lifting). *Let* $h : \Omega \to \overline{\mathbb{R}}^+$ *a nonnegative measurable function such that* $\mathrm{supp}(h) \subseteq \mathsf{T}$. *The* lifting of h *is the measurable function* $\check{h} : \Omega^\infty \to \overline{\mathbb{R}}^+$ *defined as follows for each* $\tilde{\omega} = (\omega_1, \omega_2, \ldots)$*:* $\check{h}(\tilde{\omega}) := \sum_{t \geq 1} 1_{\mathfrak{c}(T_t)}(\tilde{\omega}) \cdot h(\omega_t)$.

Clearly, any $\check{h}$ is prefix closed with branches $L_0 = \emptyset$ and $L_j = (\mathsf{T}^c)^{j-1} \cdot \mathrm{supp}(h) \subseteq T_j$ for $j \geq 1$. In particular, $\mathrm{supp}(\check{h}) \subseteq T_\mathrm{f}$. As an example, the indicator function for the set of paths that eventually terminate, $\check{h} = 1_{\mathsf{T}_\mathrm{f}}$, is clearly the lifting of $h = 1_\mathsf{T}$; the functions f_1, f_2 in Example 4 can also be obtained by lifting (details omitted). More complicated functions, that also look at sequences of events before termination, can often be encoded as lifted functions at the cost of introducing in the program extra variables to keep track of those events in the store.

5 Feynman-Kac models

In the field of Sequential Monte Carlo methods, Feynman-Kac (FK) models [19, Ch.9] are characterized by the use of *potential* functions. A potential in a Feynman-Kac model is a function that assigns a weight $G_t(x)$ to a *particle* (instance of a random process) in state x at time t. This weight represents how plausible or fit x is at time t based on some observable or conditioning. In other words, G_t modifies the *importance* of particles as the system evolves. For instance, in a model for tracking an object, the potential function could depend on the distance between the predicted particle position and the actual observed position. Particles closer to the observed position get higher weights.

FK models and probabilistic program semantics We first introduce FK models in a general context. Our formulation follows closely [19, Ch.9]. Throughout this and the next section, we let $t \geq 1$ be an arbitrary fixed integer.

Definition 9 (Feynman-Kac models). *A* Feynman-Kac (FC) model *is a tuple* FK $= (\mathcal{X}, t, \mu^1, \{K_i\}_{i=2}^t, \{G_i\}_{i=1}^t)$*, where* $\mathcal{X} = \overline{\mathbb{R}}^\ell$ *for some* $\ell \geq 1$*,* μ^1 *is a probability measure on* $\mathcal{X}$ *and, for* $i = 2, \ldots, t$*:* K_i *is a Markov kernel from* $\mathcal{X}$ *to* $\mathcal{X}$*, and* $G_i : \mathcal{X} \to \overline{\mathbb{R}}^+$ *is a measurable function.*

Let μ^t *denote the unique product measure on* $\mathcal{X}^t$ *induced by* $\mu_1, K_2, \ldots, K_t$ *as per Theorem 1. Let* $G := \Pi_{i=1}^t G_i$*. Provided* $0 < \mathrm{E}_{\mu^t}[G] < +\infty$*, the* Feynman-Kac measure *induced by* FK *is defined by the following, for every measurable* $A \subseteq \mathcal{X}^t$*:*

$$\phi_{\mathrm{FK}}(A) := \frac{\mathrm{E}_{\mu^t}[1_A \cdot G]}{\mathrm{E}_{\mu^t}[G]}\,. \tag{13}$$

We will refer to G in the above definition as the *global potential*. Equality (13) easily generalizes to expectations taken according to ϕ_{FK}. That is, for any measurable nonnegative function g on $\mathcal{X}^t$, we can easily show that:

$$\mathrm{E}_{\phi_{\mathrm{FK}}}[g] = \frac{\mathrm{E}_{\mu^t}[g \cdot G]}{\mathrm{E}_{\mu^t}[G]}\,. \tag{14}$$

In what follows, we will suppress the subscript $_{\mathrm{FK}}$ from ϕ_{FK} in the notation, when no confusion arises. Comparing (14) against the definition (6) suggests that the global potential G should play in FK models a role analogous to the weight function w in probabilistic programs. Note however that there is a major technical difference between the two, because FK models are only defined for a finite time horizon model given by t. A reconciliation between the two is possible thanks to the finite approximation theorem seen in the last section; this will be elaborated further below (see Theorem 4).

We will be particularly interested in the *t-th marginal* of ϕ, that is the probability measure on $\mathcal{X}$ defined as ($A \subseteq \mathcal{X}$ measurable):

$$\phi_t(A) := \phi(\mathcal{X}^{t-1} \times A) = \mathrm{E}_\phi[1_{\mathcal{X}^{t-1}\times A}]\,. \tag{15}$$

The measure ϕ_t is called *filtering* distribution (at time t), and can be effectively be computed via the Particle Filtering algorithm described in the next subsection.

Now let $\mathbf{G} = (\mathcal{P}, E, \mathsf{nil}, \mathbf{sc})$ be an arbitrary fixed PPG. Comparing (14) against e.g. the lower bound in (8) suggests considering the following FK model associated with $\mathbf{G}$ and a checkpoint S.

Definition 10 **(FK_S model).** *Let $t \geq 1$ be an integer and S a program checkpoint of* $\mathbf{G}$. *We define* FK_S *as the FK model where: $\mathcal{X} = \Omega$, $\mu^1 = \delta_{(0,S)}$, $K_i = \kappa$ ($i = 2, ..., t$) and $G_i = \mathrm{sc}$ ($i = 1, ..., t$). We let ϕ_S denote the measure on Ω^t induced by* FK_S.

We now restrict our attention to functions f that are the lifting of a nonnegative h defined on Ω. Let $\phi_{S,t}$ denote the filtering distributions of ϕ_S at time t obtained by (15). In the following proposition we express the bounds in (8) in terms of the measure $\phi_{S,t}$. The whole point and interest of this result is that the bounds are expressed directly as expectations; these are moreover taken w.r.t. a *1-dimensional* filtering distribution ($\phi_{S,t}$), rather than a t-dimensional one (μ^t_S). Importantly, there are well-known algorithms to estimate expectations under a filtering distribution, as we will see in the next subsection.

Theorem 4 **(filtering distributions and lifted functions).** *Under the same assumptions of Theorem 2, further assume that f is the lifting of h. Then $\alpha_t = \mathrm{E}_{\phi_{S,t}}[1_\mathsf{T}]^{-1}$ and*

$$\beta_L := \mathrm{E}_{\phi_{S,t}}[h] \leq [\![S]\!]f \leq \mathrm{E}_{\phi_{S,t}}[h]\cdot\alpha_t + M\cdot(\alpha_t - 1) =: \beta_U\,. \tag{16}$$

Example 8. Consider again the PPG of Example 2. We can re-compute $[\![S]\!]f$ relying on Theorem 4. Fix $t = 4$. We first compute the filtering distribution ϕ_t on $\mathcal{X} = \overline{\mathbb{R}}^3$ relying on its definition (15). Similarly to what we did in Example 5, we consider the nonzero-weight, nonzero-probability traces of length four. Then we project onto the final (fourth) state, and compute the weights of the resulting triples (c, d, S), then normalize. There are only two triples (c, d, S) of nonzero probability:

$$\phi_t(0,0,2) = \frac{2}{3} \qquad\qquad \phi_t(1,1,2) = \frac{1}{3}\,.$$

The function f considered in Example 5 is the lifting of the function $h(c, d, S) = c \cdot [S = 2]$ defined on $\mathcal{X} = \overline{\mathbb{R}}^3$. We apply Theorem 4 and get $\beta_L = \mathrm{E}_{\phi_t}[h] = \frac{1}{3} \leq [\![S]\!]f$. Moreover $\mathrm{E}_{\phi_t}[1_\mathsf{T}] = 1$, hence $\alpha_t = 1$ according to Theorem 4. Hence $\beta_L = \beta_U = [\![S]\!]f = \frac{1}{3}$. This coincides with what found in examples 5 and 7.

We can apply the above theorem to the functions described in Example 4 and to other computationally challenging cases: we will do so in Section 6, after introducing in the next section the Particle Filtering algorithm.

The Particle Filtering algorithm From a computational point of view, our interest in FK models lies in the fact that they allow for a simple, unified presentation of a class of efficient inference algorithms, known as *Particle Filtering (PF)* [19, 21, 50]. For the sake of presentation, we only introduce here the basic version, *Bootstrap* PF, following closely[8] [19, Ch.11]. Fix a generic FK model, $\mathrm{FK} = (\mathcal{X}, t, \mu^1, \{K_i\}_{i=2}^t, \{G_i\}_{i=1}^t)$. Fix $N \geq 1$, the number of *particles*, that is instances of the random process represented by the K_i's, we want to simulate. Let $W = W^{1:N} = (W^{(1)}, ..., W^{(N)})$ be a tuple of N real nonnegative random variables, the *weights*. Denote by $\widehat{W}$ the normalized version of W, that is $\widehat{W}^{(i)} = W^{(i)}/(\sum_{j=1}^N W^{(j)})$. A *resampling scheme* for (N, W) is a N-tuple of random variables $R = (R_1, ..., R_N)$ taking values on $1..N$ and depending on W, such that, for each $1 \leq i \leq N$, one has: $\mathrm{E}[\sum_{j=1}^N 1_{R(j)=i}\,|W] = N \cdot \widehat{W}^{(i)}$. In other words, each index $i \in 1..N$ on average is selected in R a number of times proportional to its weight in W. We shall write $R(W)$ to indicate that R depends on a given weight vector W. Various resampling schemes have been proposed in the literature, among which the

[8] Additional details in Appendix B.

Algorithm 1 A generic PF algorithm

Input: FK = $(\mathcal{X}, t, \mu^1, \{K_k\}_{k=2}^t, \{G_k\}_{k=1}^t)$, a FK model; $N \geq 1$, number of particles.
Output: $X_t^{1:N} \in \mathcal{X}^N$, $W_t^{1:N} \in \mathbb{R}^{+N}$.

$$\begin{array}{ll} 1: & X_1^{(j)} \sim \mu^1 \\ 2: & W_1^{(j)} := G_1(X_1^{(j)}) \end{array}\Bigg\} \; (j = 1, \dots, N)$$

1: ▹ state initialisation
2: ▹ weight initialisation
3: **for** $k = 2, \dots, t$ **do**
4: $\quad r_{1:N} \sim R(W_{k-1}^{1:N})$ ▹ resampling

$$\begin{array}{ll} 5: & \quad X_k^{(j)} \sim K_k(X_{k-1}^{(r_j)}) \\ 6: & \quad W_k^{(j)} := G_k(X_k^{(j)}) \end{array}\Bigg\} \; (j = 1, \dots, N)$$

5: ▹ state update
6: ▹ weight update
7: **end for**
8: **return** (X_t, W_t)

simplest is perhaps *multimomial resampling*; see e.g. [19, Ch.9] and references therein. Algorithm 1 presents a generic PF algorithm. Resampling here takes place at step 4: its purpose is to give more importance to particles with higher weight, when extracting the next generation of N particles, while discarding particles with lower weight.

The justification and usefulness of this algorithm is that, under mild assumptions, for any measurable function h defined on $\mathcal{X}$, expectation under ϕ_t, the filtering distribution on $\mathcal{X}$ at time t, in the limit can be expressed a weighted sum with weights $\widehat{W}_t^{(j)}$:

$$\sum_{j=1}^{N} \widehat{W}_t^{(j)} \cdot h(X_t^{(j)}) \longrightarrow \mathrm{E}_{\phi_t}[h] \quad \text{a.s. as } N \longrightarrow +\infty\,. \tag{17}$$

The practical implication here is that we can estimate quite effectively the expectations involved in (16), for $\phi_t = \phi_{S,t}$, as weighted sums like in (17). Note that in the above consistency statement t is held fixed — it is one of the parameter of the FK model — while the number of particles N tends to $+\infty$.

6 Implementation and experimental validation

Implementation The PPG model is naturally amenable to a vectorized implementation of PF that leverages the fine-grained, SIMD parallelism existing at the level of particles. At every iteration, the state of the N particles, $\omega^N = (\omega_1, \dots, \omega_N)$ with $\omega_i = (v_i, z_i) \in \overline{\mathbb{R}}^{m+1}$, will be stored using a pair of arrays (V, Z) of shape $N \times m$ and $N \times 1$, respectively. The weight vector is stored using another array W of shape $N \times 1$. We rely on vectorization of operations: for a function $f : \overline{\mathbb{R}}^k \to \overline{\mathbb{R}}$ and a $N \times k$ array U, $f(U)$ will denote the $N \times 1$ array obtained by applying f to each row of U. In particular, we denote by $(Z = s)$ (for any $s \in \mathbb{N}$) the $N \times 1$ array obtained applying element-wise the indicator function $1_{\{s\}}$ to Z element-wise, and by $\varphi(V)$ the $N \times 1$ array obtained by applying the predicate φ to V to the row-wise. For U a $N \times k$ array and W a $N \times 1$ array, $U * W$ denotes the $N \times k$ array obtained by multiplying the the jth row of U by the jth element of W, for $j = 1, \dots, N$: when W is a $0/1$ vector, this is an instance of *boolean masking*. Abstracting the vectorization primitives of modern CPUs and programming languages, we model the assignments of a vector to an array variable as a single instruction, written $U := Z$. The usual rules for broadcasting scalars to vectors apply, so e.g. $V := S$ for $S \in \overline{\mathbb{R}}$ means filling V with S. Likewise, for ζ a parametric distribution, $U \sim \zeta(V)$ means sampling N times independently from $\zeta(v_1), \dots, \zeta(v_N)$, and assigning the resulting matrix to U: this too counts as a single instruction.

Based on the above idealized model of vectorized computation, we present VPF, a vectorized version of the PF algorithm for PPGs, as Algorithm 2. Here it is assumed that $\mathcal{P} \subseteq \mathbb{N}$, while $\mathbf{sc}(s) = \gamma_s$. On line 4, Resampling$(\cdot)$ denotes the result of applying a generic resampling algorithm based on weights W to the current particles' state, represented by the pair of vectors (V, Z). With respect to the generic PF Algorithm 1, here in the returned output, (V, Z) corresponds to X_t and W to W_t. Note that there are no loops where the number of iterations depends on N; the **for** loop in lines 5–7 only scans the transitions set E, whose size is independent of N. Line 8 is just a vectorized implementation of sampling from the Markov kernel function in (13). Line 9 is a vectorized implementation of the combined score function (4). In the actual TensorFlow implementation, the sums in lines 8 and 9 are encoded via boolean masking and vectorized operations.

Algorithm 2 VPF, a Vectorized PF algorithm for PPGs.

Input: $\mathbf{G} = (\mathcal{P}, E, \mathsf{nil}, \mathbf{sc})$, a PPG; $S \in \mathcal{P}$, initial program checkpoint; $t \geq 1$, time horizon; $N \geq 1$, number of particles.
Output: $V \in \overline{\mathbb{R}}^{m\times N}$, $Z, W \in \overline{\mathbb{R}}^{1\times N}$.

$V := S$; $Z := S$ ▹ state initialisation
$W := \gamma_S(Z)$ ▹ weight initialisation
for $t-1$ **times do**
  $(V, Z) := \text{Resampling}((V, Z), W)$ ▹ resampling
  for $(s, \varphi, \zeta, s') \in E$ **do**
    $M_{s,\varphi} := \varphi(V) * (Z = s)$ ▹ mask computation
  end for
  $V \sim \sum_{(s,\varphi,\zeta,s')\in E} \zeta(V) * M_{s,\varphi}$; $Z := \sum_{(s,\varphi,\zeta,s')\in E} s' \cdot M_{s,\varphi}$ ▹ state update
  $W := \sum_{s\in\mathcal{P}} \gamma_s(V) * (Z = s)$ ▹ weight update
end for
return (V, Z, W)

Experimental validation We illustrate some experimental results obtained with a proof-of-concept TensorFlow-based [1] implementation of Algorithm 2. We still refer to this implementation as VPF. We consider five challenging probabilistic models that feature conditioning inside loops. For all the models, we will estimate $[[S]]f$, for suitable functions f, relying on the bounds provided by Theorem 4 in terms of expectations w.r.t. filtering distributions. Such expectations will be estimated via VPF. We also compare VPF with two state-of-the-art PPLs, webPPL [25] and CorePPL [35]. webPPL is a popular PPL supporting several inference algorithms, including SMC, where resampling is handled via continuation passing. We have chosen to consider CorePPL as it supports a very efficient implementation of PF. In [35], a comparison of CorePPL with webPPL, Pyro [12] and other PPLs in terms of performance shows the superiority of CorePPL SMC-based inference across a number of benchmarks. As discussed in the Introduction, CorePPL's implementation is based on a compilation into an intermediate format, conceptually similar to our PPGs. Direct compilation to GPU via the intermediate-level format RootPPL is also supported and will be also considered below.

For our experiments we consider the following five models: *Aircraft tracking* (AT, [51]), *Drunk man and mouse* (DMM, Example 3), *Hare and tortoise* (HT, e.g. [4]), *Bounded retransmission protocol* (BRP, [31]), *Non-i.i.d. loops* (NIID, e.g. [31]). These programs feature conditioning/scoring inside loops. In particular, DMM, HT and NIID feature unbounded loops: for these three programs, in the case of VPF we have chosen $t = 100$ as the time parameter of Theorem 4, which allows us to deduce bounds on the value of $[[S]]f$; for the other tools, we just consider the truncated estimate returned at the end of the 100 iterations. AT and BRP feature bounded loops, but are nevertheless quite challenging. In particular, AT features multiple conditioning inside a for-loop, sampling from a mix of continuous and discrete distributions, and noisy observations. A description of these programs, together with further details on the experimental set up, can be found in Appendix C and [15]; Table 1 summarizes the obtained experimental results. We report the execution time, the estimated expected value and the effective sample size (ESS; see Appendix C) for VPF, CorePPL, RootPPL and webPPL, as the number N of particles increases. At least for $N \geq 10^5$, the tools tend to yield very similar estimates of the expected value, which we take as an empirical evidence of accuracy. On the other hand, in terms of ESS and execution time, the difference across the tools is significant. VPF consistently yields ESS that are higher or on par with the other tools'. For larger values of N ($\geq 10^5$), VPF outperforms the other tools in terms of execution time; the difference is especially significant for $N = 10^6$ on all models. Overall, we take this as an evidence of the higher scalability of VPF over the other tools.

7 Conclusion

We study correct and efficient implementations of Sequential Monte Carlo inference algorithms for universal probabilistic programs. Building on a clean expectation-based operational semantics for PPGs, we prove a finite approximation theorem that allows us to establish a precise relation with FK models, and consistency of the PF algorithm for our semantics. Preliminary experiments conducted with VPF, a vectorized version of PF tailored to PPGs, show very promising results when compared to state-of-the-art tools for inference in PPs.

		AT				DMM				HT				BRP				NIID			
		VPF	CorePPL	RootPPL	webPPL	VPF	CorePPL	RootPPL	webPPL	VPF	CorePPL	RootPPL	webPPL	VPF	CorePPL	RootPPL	webPPL	VPF	CorePPL	RootPPL	webPPL
$N = 10^3$	*time*	**0.009**	0.014	0.034	0.190	0.295	**0.016**	0.184	0.140	0.274	**0.015**	0.159	0.152	0.676	**0.021**	1.014	0.155	0.240	**0.010**	0.150	0.061
	EV	6.805	6.955	8.653	6.696	0.787 ± 0.575	0.502	0.432	0.431	32.834	33.683	33.988	32.368	0.018	0.016	0.024	0.023	3.594	2.694	3.500	3.473
	ESS	**1000**	**1000**	**1000**	999	**996.0**	817.63	900.0	25.81	**955.0**	758.9	821.0	951.1	**1000**	**1000**	**1000**	974.5	**1000**	846.6	891.9	726.9
$N = 10^4$	*time*	**0.131**	0.194	18.806	3.842	0.388	**0.178**	34.325	2.693	0.290	**0.180**	1.352	3.839	0.786	**0.309**	-	1.328	0.323	**0.058**	9.777	0.490
	EV	6.817	6.967	6.168	6.760	0.890±0.497	0.507	0.435	0.448	32.725	33.474	33.581	32.702	0.029	0.025	-	0.024	3.364	2.766	3.395	3.417
	ESS	$\mathbf{10^4}$	$\mathbf{10^4}$	9999	*9975*	**9954**	7798.8	8234	9963	**9445**	7692.9	7795	9476.2	$\mathbf{10^4}$	$\mathbf{10^4}$	-	9745.8	$\mathbf{10^4}$	8555.6	9283	7560.5
$N = 10^5$	*time*	**0.354**	2.252	-	-	**0.598**	3.833	-	154.878	**0.379**	4.225	-	361.792	**0.797**	5.010	-	15.038	**0.445**	1.083	29.455	92.419
	EV	6.818	6.970	-	-	0.820±0.396	0.506	-	0.450	33.128	33.545	-	32.560	0.024	0.025	-	0.026	3.467	2.772	3.411	3.430
	ESS	$\mathbf{10^5}$	9.9e$\mathbf{10^5}$	-	-	**99316**	78119.9	-	1541.07	94856	77243	-	**94881.29**	$\mathbf{10^5}$	$\mathbf{10^5}$	-	97484	$\mathbf{10^5}$	85609.7	92708.9	75809.5
$N = 10^6$	*time*	**2.286**	26.481	-	-	**4.220** **38.978**	46.420 -	-	-	**3.749**	49.493	-	-	**10.155**	58.448	-	-	**2.916**	14.323	-	-
	EV	6.834	6.980	-	-	0.803±0.391 0.504±0.109	0.499 -	-	-	33.432	33.606	-	-	0.024	0.025	-	-	3.413	2.774	-	-
	ESS	$\mathbf{10^6}$	9.9e10^5	-	-	**993000** **999490**	778664 -	-	-	**947641.9**	771932.9	-	-	$\mathbf{10^6}$	$\mathbf{10^6}$	-	-	$\mathbf{10^6}$	855989	-	-

Table 1: Execution time ($time$) in seconds, estimated expected value (EV) and effective sample size ($ESS := (\sum_{i=1}^{N} W_i)^2/(\sum_{i=1}^{N} W_i^2)$; the higher the better, see e.g. [22]) as the number of particles (N) increases, for VPF, CorePPL, RootPPL and webPPL, when applied on Aircraft tracking (AT), Drunk man and mouse (DMM), Hare and tortoise (HT), Bounded retransmission protocol (BRP) and Non-i.i.d. loops (NIID). For VPF, with reference to Theorem 4: for AT, HT and BRP we have $EV = \beta_L = \beta_U$ (as $\alpha_t = 1$); for NIID, we only provide β_L, as β_U is vacuous. For DMM, we give the midpoint of the interval $[\beta_L, \beta_U]$ ± its half-width. In DMM, only for the case $N = 10^6$, we also report the results for $t = 1000$ (second row of each entry). Everywhere, − means no result due to out-of-memory or timeout (500s). Best results for $time$ and ESS for each example and value of N are marked in **boldface**.

References

1. Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, Xiaoqiang Zheng. TensorFlow: A System for Large-Scale Machine Learning. *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16). arXiv:1605.08695*, 2016.
2. Robert B. Ash. *Real Analysis and Probability.* Academic Press, Inc., New York, NY, USA, 1972.
3. Martin Avanzini, Georg Moser, and Michael Schaper. Automated Expected Value Analysis of Recursive Programs. *Proc. ACM Program. Lang.*, 7, PLDI, 2023.
4. Alexander Bagnall, Gordon Stewart, Anindya Banerjee. Formally Verified Samplers from Probabilistic Programs with Loops and Conditioning. *Proc. ACM Program. Lang.*, 7, 1–24, 2023.
5. Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT press, 2008.
6. Jialu Bao, Nitesh Trivedi, Drashti Pathak, Justin Hsu, Subhajit Roy. Data-driven invariant learning for probabilistic programs. *In: CAV. Lecture Notes in Computer Science*, vol. 13371, pp. 33–54. Springer, 2022.
7. Gilles Barthe, Joost-Pieter Katoen, and Alexandra Silva. *Foundations of Probabilistic Programming.* Cambridge University Press, 2020.
8. Ezio Bartocci, Laura Kovács, and Miroslav Stankovic. Mora-automatic generation of moment-based invariants. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 492–498,2020.
9. Kevin Batz, Mingshuai Chen, Sebastian Junges, Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja. Probabilistic Program Verification via Inductive Synthesis of Inductive Invariants. In: Sankaranarayanan, S., Sharygina, N. (eds) *Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2023. Lecture Notes in Computer Science*, vol 13994. Springer, 2023.
10. Raven Beutner, Luke Ong, Fabian Zaiser. Guaranteed bounds for posterior inference in universal probabilistic programming. *Proceedings of the 43rd ACM SIGPLAN international conference on programming language design and implementation, (PLDI 22)*, 2022.
11. Sooraj Bhat, Johannes Borgström, Andrew D. Gordon, and Claudio Russo. Deriving probability density functions from probabilistic functional programs. *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2013)*, 2013.
12. Eli Bingham, Jonathan P Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D Goodman. Pyro: Deep universal probabilistic programming. *The Journal of Machine Learning Research* 20, 1, 973–978, 2019.
13. Michele Boreale and Luisa Collodi. Guaranteed inference for probabilistic programs: a parallelisable, small-step operational approach. In *VMCAI 2024*, *Lecture Notes in Computer Science*, Vol. 14500, Springer, 2023.
14. Michele Boreale and Luisa Collodi. Full version of the present paper. Available from `https://github.com/Luisa-unifi/Vectorized-Particle-Filtering`, December 2024.
15. Michele Boreale and Luisa Collodi. Code for the experiments described in the present paper. `https://github.com/Luisa-unifi/Vectorized-Particle-Filtering`, December 2024.
16. Johannes Borgström, Ugo Dal Lago, Andrew D. Gordon, Marcin Szymczak. A lambda-calculus foundation for universal probabilistic programming. *ACM SIGPLAN Notices* 51.9: 33-46, 2016.
17. Bob Carpenter, Andrew Gelman, Matthew D. Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A probabilistic programming language. *Journal of Statistical Software* 76(1), 2017. 10.18637/jss.v076.i01
18. Krishnendu Chatterjee, Amir K. Goharshady, Tobias Meggendorfer, Đorđe Žikelic. Sound and Complete Certificates for Quantitative Termination Analysis of Probabilistic Programs. In: Shoham, S., Vizel, Y. (eds) *Computer Aided Verification. CAV 2022. Lecture Notes in Computer Science*, vol 13371. Springer, 2022.
19. Nicolas Chopin, Omiros Papaspiliopoulos. *An Introduction to Sequential Monte Carlo*. Springer Nature Switzerland AG, 2021
20. Marco F. Cusumano-Towner, Feras A. Saad, Alexander K. Lew, Vikash K. Mansinghka. Gen: A General-purpose Probabilistic Programming System with Programmable Inference. *In Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pp. 221–236. New York, NY, USA, 2019.
21. Pierre Del Moral, Arnaud Doucet, Ajay Jasra. Sequential Monte Carlo Samplers. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, Vol. 68, No. 3, pp. 411-436, 2006.
22. Víctor Elvira, Luca Martino, Christian P. Robert. Rethinking the Effective Sample Size. *arXiv:1809.04129*, 2018.
23. Timon Gehr, Sasa Misailovic, and Martin T. Vechev. Psi: Exact symbolic inference for probabilistic programs. *Proceedings of the 28th International Conference in Computer Aided Verification (CAV 2016), Toronto*, page 62–83, 2016.

24. Noah D. Goodman, Vikash Mansinghka, Daniel Roy, Keith Bonawitz, and Joshua Tenenbaum. Church: a language for generative models. *Proc. Uncertainty in Artificial Intelligence*. 2008.
25. Noah D. Goodman and Andreas Stuhlmüller. The design and implementation of probabilistic programming languages. Retrieved 2023-8-31 from `http://dippl.org`.
26. Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. Probabilistic programming. *Proceedings of Future of Software Engineering Proceedings (FOSE2014)*, pages 167–181, 2014.
27. Maria I. Gorinova, Andrew D. Gordon, Charles Sutton. Probabilistic programming with densities in SlicStan: efficient, flexible, and deterministic.*Proceedings of the ACM on Programming Languages 3, POPL*. 1-30, 2019.
28. Maria I. Gorinova, Andrew D. Gordon, Charles Sutton, Matthijs Vákár. Conditional independence by typing. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 44.1: 1-54, 2021.
29. Michikazu Hirata, Yasuhiko Minamide, Tetsuya Sato. Semantic Foundations of Higher-Order Probabilistic Programs in Isabelle/HOL. In *14th International Conference on Interactive Theorem Proving (ITP)*, 2023. *Leibniz International Proceedings in Informatics (LIPIcs)*, Volume 268, pp. 18:1-18:18, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.
30. Benjamin L. Kaminski. *Advanced weakest precondition calculi for probabilistic programs*. Doctoral thesis (Ph.D), RWTH Aachen University, 2019.
31. Lutz Klinkenberg, Christian Blumenthal, Mingshuai Chen, Darion Haase, and Joost-Pieter Katoen. Exact Bayesian Inference for Loopy Probabilistic Programs using Generating Functions. In *Proceedings of the ACM on Programming Languages (OPSLA)*, Volume 8: 127,923-953, 2024.
32. Dexter Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 22(3):328–350, 1981.
33. Alexander K. Lew, George Matheos, Tan Zhi-Xuan, Matin Ghavamizadeh, Nishad Gothoskar, Stuart Russell, Vikash K. Mansinghka. SMCP3: Sequential Monte Carlo with Probabilistic Program Proposals. *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics (AISTATS)*. PMLR 206:7061-7088, 2023.
34. Daniel Lundén, Johannes Borgström, David Broman. Correctness of sequential Monte Carlo inference for probabilistic programming languages. *Programming Languages and Systems*, pp. 404–431. Springer International Publishing, Cham, 2021.
35. Daniel Lundén, Joey Ohman, Jan Kudlicka, Viktor Senderov, Fredrik Ronquist, and David Broman. Compiling universal probabilistic programming languages with efficient parallel sequential Monte Carlo inference. In *ESOP*, pages 29–56, 2022.
36. Daniel Lundén, David Broman, Fredrik Ronquist, Lawrence M. Murray. Automatic alignment of sequential Monte Carlo inference in higher-order probabilistic programs. *ESOP*, pages 535-563, 2023.
37. Tshepo Mokoena. Urban Myths: Are you never more than 6ft from a rat in a city? The Guardian online, 2014. `https://www.theguardian.com/cities/2014/feb/13/urban-myths-6ft-from-a-rat`.
38. Praveen Narayanan, Jacques Carette, Chungchieh Shan Wren Romano, and Robert Zinkov. Probabilistic inference by program transformation in Hakaru (system description). *Proceedings of the 13th International Symposium on Functional and Logic Programming (FLOPS 2016)*, pages 62–79, 2016.
39. Praveen Narayanan, Chung-chieh Shan. Symbolic disintegration with a variety of base measures. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 42 (2), 1-60, 2020.
40. Aditya V. Nori, Chung-Kil Hur, Sriram K. Rajamani, and Selva Samuel. R2: An efficient mcmc sampler for probabilistic programs. *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, page 2476–2482, July 2014.
41. Alexey Radul, Boris Alexeev. The Base Measure Problem and its Solution. *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics (AISTATS)*, PMLR 130:3583-3591, 2021.
42. Francesca Randone, Luca Bortolussi, Emilio Incerto, Mirco Tribastone. Inference of Probabilistic Programs with Moment-Matching Gaussian Mixtures. *Proceedings of the ACM on Programming Languages (POPL)*:1882-1912, 2024.
43. Sriram Sankaranarayanan, Aleksandar Chakarov, and Sumit Gulwani. Static analysis for probabilistic programs: Inferring whole program properties from finitely many paths. *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation.*, 2013.
44. Adam Scibior, Ohad Kammar, Matthijs Vákár, Sam Staton, Hongseok Yang, Yufei Cai, Klaus Ostermann, Sean K. Moss, Chris Heunen, Zoubin Ghahramani. Denotational validation of higher-order Bayesian inference. *Proceedings of the ACM on Programming Languages 2. POPL*. 1-29, 2017.
45. Sam Staton. Commutative semantics for probabilistic programming. *Proceedings of the 26th European Symposium on Programming (ESOP2017), Uppsala, Sweden*, 2017.
46. Sam Staton, Frank Wood, Hongseok Yang, Chris Heunen, and Ohad Kammar. Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2016.

47. Zachary Susag, Sumit Lahiri, Justin Hsu, and Subhajit Roy. Symbolic execution for randomized programs. *Proc. ACM Program. Lang., 6, OOPSLA*, 1583–1612, 2022.
48. Joseph Tassarotti, Jean-Baptiste Tristan. Verified density compilation for a probabilistic programming language. *Proceedings of the ACM on Programming Languages 7, PLDI*. 615-637, 2023.
49. Peixin Wang, Tengshun Yang, Hongfei Fu, Guanyan Li, C.-H. Luke Ong. Static Posterior Inference of Bayesian Probabilistic Programming via Polynomial Solving. *Proceedings of the ACM on Programming Languages*, Volume 8, Issue *PLDI'24*, 202, 1361 - 1386, 2024.
50. Frank Wood, Jan Willem Meent, Vikash Mansinghka. A New Approach to Probabilistic Programming Inference. *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, PMLR 33:1024-1032, 2014.
51. Yi Wu, Siddharth Srivastava, Nicholas Hay, Simon Du, Stuart Russell. Discrete-Continuous Mixtures in Probabilistic Programming: Generalized Semantics and Inference Algorithms. *Proceedings of the 35th International Conference on Machine Learning*, PMLR 80:5343-5352, 2018.

A Proofs

The following result, which subsumes Theorem 1, is well known from measure theory. The formulation below is a specialization of [2, Th.2.6.7] to Markov kernels and nonnegative functions. Part (a) gives a way to construct a measure on the product space Ω^t, starting from an initial measure μ^1 and $t-1$ Markov kernels. The product space is, intuitively, the sample space of the paths of length t of a Markov chain. In particular, a path of length $t=1$ consists of just an initial state — no transition has been fired. Part (b) is a generalization of Fubini theorem, which allows one to express an integral over the product space w.r.t. the measure of part (a) in terms of iterated integrals over the component spaces. Below, we will let ω^t range over Ω^t.

Theorem A.1 (product of measures). *Let $t \geq 1$ be an integer. Let μ^1 be a probability measure on Ω and $K_2, \dots, K_t$ be $t-1$ (not necessarily distinct) Markov kernels from Ω to Ω.*

(a) There is a unique probability measure μ^t defined on $(\Omega^t, \mathcal{F}^t)$ such that for every $A_1 \times \cdots \times A_t \in \mathcal{F}^t$ we have:

$$\mu^t(A_1 \times \cdots \times A_t) = \int_{A_1} \mu^1(d\omega_1) \int_{A_2} K_2(\omega_1)(d\omega_2) \cdots \int_{A_t} K_t(\omega_{t-1})(d\omega_t)\,. \tag{18}$$

(b) (Fubini) Let f be a nonnegative measurable function defined on Ω^t. Then, letting $\omega^t = (\omega_1, \dots, \omega_t)$, we have

$$\int \mu^t(\omega^t) f(\omega^t) = \int \mu^1(d\omega_1) \int K_2(\omega_1)(d\omega_2) \cdots \int K_t(\omega_{t-1})(d\omega_t) f(\omega^t)\,. \tag{19}$$

In particular, on the right-hand side, for each $j = 1, \dots, t-1$ and $(\omega_1, \dots, \omega_{j-1})$, the function $\omega_j \mapsto \int K_{j+1}(\omega_j)(d\omega_{j+1}) \cdots \int K_t(\omega_{t-1})(d\omega_t) f(\omega^t)$ is measurable over Ω.

We now proceed to the proof of the results stated in the paper.

Proof of Lemma 1. As a function $\overline{\mathbb{R}}^{m+1} \times \mathcal{F} \to \overline{\mathbb{R}}^{+}$, κ can be written as follows:

$$\kappa(v,z)(A) = [z \notin \mathcal{P}] \cdot \delta_v(A_z) + \sum_{(S,\varphi,\zeta,S') \in E} [z = S] \cdot \varphi(v) \cdot \zeta(v)(A_{S'})\,. \tag{20}$$

We now check the two properties required by Definition 1.

- For each $(v,z) \in \Omega$, the function $A \mapsto \kappa(v,z)(A)$ is a probability measure. Consider the right-hand side of (20): since for each S, $\sum_{(S,\varphi,\zeta,S') \in E} \varphi(v) = 1$, for the given (v,z) exactly one of the summands is different from the constant 0 function. In particular, there is a probability measure ν on $\mathcal{F}_m$ such that for every $A \in \mathcal{F}$, we have $\kappa(v,z)(A) = \nu(A_z)$. Next, we note the following general property of sections of measurable sets, which can be shown by elementary set-theoretic reasoning: for any measurable set $A \in \mathcal{F}$ s.t. $A = \cup_{j \geq 0} A_j$ (disjoint union of measurable sets) and $z \in \overline{\mathbb{R}}$, we have $A_z = \cup_{j \geq 0} (A_j)_z$ (disjoint union of measurable sets). Applying the two facts just established and the additivity of the measure ν, we have: $\kappa(v,z)(A) = \nu((\cup_{j \geq 0} A_j)_z) = \nu(\cup_{j \geq 0} (A_j)_z) = \sum_{j \geq 0} \nu((A_j)_z) = \sum_{j \geq 0} \kappa(v,z)(A_j)$. This shows that $\kappa(v,z)$ is a measure. Moreover, $\kappa(v,z)(\Omega) = \nu(\Omega_z) = \nu(\overline{\mathbb{R}}^m) = 1$, which completes the prove that $\kappa(v,z)$ is a probability measure.
- For each $A \in \mathcal{F}$, the function $(v,z) \mapsto \kappa(v,z)(A)$ is nonnegative and measurable. Consider again the right-hand side of (20), but write $\delta_v(A_z)$ as the indicator function $1_{A_z}(v)$: as a function of v, this is measurable (as A_z is a measurable set). Moreover, for any A and S', $\zeta(v)(A_{S'})$ is a measurable function of v (as ζ is a Markov kernel); hence for each φ, also $\varphi(v) \cdot \zeta(v)(A_{S'})$ is a measurable function of v. But any measurable function of v alone, say $h(v)$, is also a measurable function of (v,z) (i.e. the function obtained by composing the projection $(v,z) \mapsto v$ with $v \mapsto h(v)$). Likewise, the predicates $[z \notin \mathcal{P}]$ and $[z = S]$ (for any fixed $S \in \mathcal{P}$) are measurable functions of z, hence of (v,z). As $\kappa(\cdot)(A)$ is obtained by products and sums of nonnegative measurable functions of (v,z), it is a measurable function of (v,z) [2, Ch.1,Th.1.5.6].

We now prove Lemma 2. In the what follows, we shall make use of the following properties of measurable cylinders: for measurable $A, B \subseteq \Omega^t$ ($t \geq 1$), we have $\mathfrak{c}(A \cup B) = \mathfrak{c}(A) \cup \mathfrak{c}(B)$ and $\mathfrak{c}(A \cap B) = \mathfrak{c}(A) \cap \mathfrak{c}(B)$.

Proof of Lemma 2. First, consider the case of indicator functions $h = 1_{A_t}$, for a measurable $A_t \subseteq \Omega^t$. Then $\tilde{h} = 1_{\mathfrak{c}(A_t)}$, the indicator function of the measurable cylinder generated by A_t, and the statement is obvious, because h is measurable, and $\int_{\mathfrak{c}(B_t)} \mu^\infty(d\tilde{\omega})\tilde{h}(\tilde{\omega}) = \int \mu^\infty(d\tilde{\omega})\tilde{h}(\tilde{\omega})1_{\mathfrak{c}(B_t)}(\tilde{\omega}) = \mu^\infty(\mathfrak{c}(B_t) \cap \mathfrak{c}(A_t)) = \mu^\infty(\mathfrak{c}(B_t \cap A_t)) = \mu^t(B_t \cap A_t) = \int_{B_t} \mu^t(d\omega^t)h(\omega^t)$. The statement for the general case of h follows then by standard measure-theoretic arguments (linearity, dominated convergence).

We can now readily establish measurability of various functions used throughout the paper.

Lemma A.1 (measurability of functions). *Let $t \geq 1$. The following functions are measurable: (1) $\mathrm{w}_t : \Omega^t \to [0,1]$; (2) $\mathrm{w} : \Omega^\infty \to [0,1]$; (3) $f_t : \Omega^t \to \overline{\mathbb{R}}^+$, provided $f : \Omega^\infty \to \overline{\mathbb{R}}^+$ is measurable.*

Proof. Concerning parts 1 and 2, first note one can write the score function on Ω (Definition (4)) as: $\mathrm{sc}(v,z) = [z \notin \mathcal{P}] + \sum_{S\in\mathcal{P}}[z = S]\cdot\mathbf{sc}(S)(v)$, where $\gamma = \mathbf{sc}(S)$ is a measurable score function on $\overline{\mathbb{R}}^m$. This easily implies that $\mathrm{sc}(\cdot)$ is measurable on Ω (cf. also the proof of Lemma 1, second item). As $\mathrm{w}_t(\omega^t) = \mathrm{sc}(\omega_1)\cdots\mathrm{sc}(\omega_t)$ is the product of measurable functions on Ω, it is a measurable function on Ω^t. Now consider $\widetilde{\mathrm{w}}_t : \Omega^\infty \to [0,1]$: applying Lemma 2 with $h = \mathrm{w}_t$, we deduce that $\widetilde{\mathrm{w}}_t$ is measurable. Finally, as $t \to +\infty$, it is seen that $\widetilde{\mathrm{w}}_t \to \mathrm{w}$ pointwise: then w is measurable as well, because it is the pointwise limit of a sequence of measurable functions functions, cf. [2, Th.1.5.4].

Concerning part 3, define the *t-section of $C \in \mathcal{C}$ at $\tilde{\omega} \in \Omega^\infty$* as $C^t_{\tilde{\omega}} := \{\omega^t \in \Omega^t : (\omega^t, \tilde{\omega}) \in C\}$. A proof very similar to that given in [2, Th.2.6.2(1)] for finite products shows that t-sections of elements in $\mathcal{C}$ are measurable. Now let A be any measurable set in $\overline{\mathbb{R}}$. By definition, $f^{-1}(A) = \{(\omega^t, \tilde{\omega}) : f(\omega^t, \tilde{\omega}) \in A\}$ is measurable. The t-section at $\tilde{\omega} = \star^\infty$ of this set is precisely $f_t^{-1}(A)$, hence it is measurable. This implies that f_t is measurable.

We now turn to the proof of Theorem 2. We first prove a few results about to the measures μ^∞_S and μ^t_S. In what follows, we shall consider the the notation $\int f\, d\mu$ for $\int \mu(d\omega)f(\omega)$. We shall use the two notations interchangeably; the second one is more convenient for expressing iterated integrals.

Definition A.1 (consistent paths). *We define the following measurable subsets of Ω^t ($t \geq 1$) and Ω^∞:*

$$\Theta^{\leq t} := \cup_{j=0}^{t-1}(\mathsf{T}^c)^j\cdot\mathsf{T}^{t-j} \qquad \Theta_t := \Theta^{\leq t}\cup(\mathsf{T}^c)^t \qquad \Theta := \left(\cup_{j\geq 0}(\mathsf{T}^c)^j\cdot\mathsf{T}^\infty\right)\cup(\mathsf{T}^c)^\infty. \tag{21}$$

Lemma A.2. *(a) $\mu^\infty_S(\Theta) = 1$. Hence for any measurable set A and nonnegative measurable f defined on Ω^∞, $\int_A f\, d\mu^\infty_S = \int_{A\cap\Theta} f\, d\mu^\infty_S$.*

(b) Let $t \geq 1$. Then $\mu^t_S(\Theta_t) = 1$. Hence for any measurable set A and nonnegative measurable f defined on Ω^t, $\int_A f\, d\mu^t_S = \int_{A\cap\Theta_t} f\, d\mu^t_S$.

Proof. Let us consider part (a). Consider $\Theta^c = \cup_{j\geq 0}\mathfrak{c}(\Omega^j\cdot\mathsf{T}\cdot\mathsf{T}^c)$; note that this union is in general not disjoint, but this is not relevant for the rest of the proof. For any $j \geq 0$, we will show that $\mu^\infty_S(\mathfrak{c}(\Omega^j\cdot\mathsf{T}\cdot\mathsf{T}^c)) = 0$, which implies the thesis. Indeed, $\mu^\infty_S(\mathfrak{c}(\Omega^j\cdot\mathsf{T}\cdot\mathsf{T}^c)) = \mu^{j+2}_S(\Omega^j\cdot\mathsf{T}\cdot\mathsf{T}^c)$, by definition of the product measure μ^∞_S. Theorem A.1(a) (Fubini) gives us

$$\mu^{j+2}_S(\Omega^j\mathsf{T}\mathsf{T}^c) = \int \delta_{(0,S)}(d\omega_1)\int \kappa(\omega_1)(d\omega_2)\int\cdots\int_{\mathsf{T}}\kappa(\omega_j)(d\omega_{j+1})\int_{\mathsf{T}^c}\kappa(\omega_{j+1})(d\omega_{j+2})1\,. \tag{22}$$

Considering the innermost two integrals in the above expression, let $J(\omega_j) := \int_{\mathsf{T}}\kappa(\omega_j)(d\omega_{j+1})\int_{\mathsf{T}^c}\kappa(\omega_{j+1})(d\omega_{j+2})1 = \int\kappa(\omega_j)(d\omega_{j+1})\int\kappa(\omega_{j+1})(d\omega_{j+2})1_{\mathsf{T}}(\omega_{j+1})\cdot 1_{\mathsf{T}^c}(\omega_{j+2})$. Suppose $\omega_{j+1} = (v, \mathsf{nil}) \in \mathsf{T}$: then considering the innermost integral in $J(\omega_j)$, by definition of κ we have $\int\kappa(\omega_{j+1})(d\omega_{j+2})1_{\mathsf{T}}(\omega_{j+1})\cdot 1_{\mathsf{T}^c}(\omega_{j+2}) = \int\kappa(v,\mathsf{nil})(d\omega_{j+2})1_{\mathsf{T}^c}(\omega_{j+2}) = \int\delta_{(v,\mathsf{nil})}(d\omega_{j+2})1_{\mathsf{T}^c}(\omega_{j+2}) = 1_{\mathsf{T}^c}(v,\mathsf{nil}) = 0$. Similarly, we have that the innermost integral is 0 if $\omega_{j+1} \in \mathsf{T}^c$. This implies that $J(\omega_j) = 0$, hence the integral in (22) is 0.

The proof of part (b) is similar.

Proof of Lemma 4. For each $t \geq 1$, consider the function $h_t = 1_\Theta\cdot(\widetilde{1_{\mathsf{T}^{\leq t}}\cdot\mathrm{w}_t})$, defined on Ω^∞ and measurable, being the product of two measurable functions (measurability of $\widetilde{\cdot}$ follows from Lemma 1). It is easy to check that: (1) $(h_t)_{t\geq 1}$ is a monotonically nondecreasing sequence of functions; and that (2) as $t \to +\infty$, $h_t \to$

$1_\Theta \cdot 1_{\mathsf{T}_\mathrm{f}} \cdot \mathrm{w}$ pointwise. By the Monotone Convergence Theorem [2, Th.1.6.2], $\int h_t d\mu_S^\infty \longrightarrow \int 1_\Theta \cdot 1_{\mathsf{T}_\mathrm{f}} \cdot \mathrm{w} d\mu_S^\infty$, where the sequence of integrals on the left is nondecreasing. Now, applying Lemma A.2(a) and Lemma 2 with $B_t = \Omega^t$, we get $\int h_t d\mu_S^\infty = \int_\Theta (\widetilde{1_{\mathsf{T}^{\le t}} \cdot \mathrm{w}_t}) d\mu_S^\infty = \int (\widetilde{1_{\mathsf{T}^{\le t}} \cdot \mathrm{w}_t}) d\mu_S^\infty = \int 1_{\mathsf{T}^{\le t}} \mathrm{w}_t d\mu_S^t$, where the last quantity is by definition $[S]^t 1_{\mathsf{T}^{\le t}} \mathrm{w}_t$. Similarly, by Lemma A.2(a) $\int 1_\Theta \cdot 1_{\mathsf{T}_\mathrm{f}} \cdot \mathrm{w} d\mu_S^\infty = \int 1_{\mathsf{T}_\mathrm{f}} \cdot \mathrm{w} d\mu_S^\infty = [S] 1_{\mathsf{T}_\mathrm{f}} \cdot \mathrm{w}$. This completes the proof.

We need a lemma on the support of prefix-closed functions.

Lemma A.3. *Let f be a prefix-closed function with branches L_j ($j \ge 0$) and $t \ge 1$. Then $L^{>t} \cap T^{\le t} = \emptyset$.*

Proof. For each $0 \le j \le t$, we have $L^{>j} \cap T_j = \emptyset$ (T-respectfulness), which implies $L^{>t} \cap T_j \cdot \Omega^{t-j} = \emptyset$ (as $L^{>t} \subseteq L^{>j} \cdot \Omega^{t-j}$). Therefore, recalling that $T^{\le t} = \cup_{j=0}^t T_j \cdot \Omega^{t-j}$, we have $L^{>t} \cap T^{\le t} = \emptyset$.

Proof of Lemma 3. We proceed by proving separately the upper bound and the lower bound in (7).

– (Upper bound). First, let us establish the inclusion $\mathrm{supp}(f) \subseteq \mathfrak{c}(L^{\le t} \cap T^{\le t}) \cup (\mathfrak{c}(T^{\le t}))^\mathrm{c}$. Indeed, consider $\tilde\omega \in \mathrm{supp}(f)$ such that $\tilde\omega \notin \mathfrak{c}(L^{\le t} \cap T^{\le t}) = \mathfrak{c}(L^{\le t}) \cap \mathfrak{c}(T^{\le t})$. Then either $\tilde\omega \in \mathfrak{c}(T^{\le t})^\mathrm{c}$, and there is nothing left to prove. Or $\tilde\omega \in \mathfrak{c}(L_j)$ for some $j > t$, hence $\tilde\omega \in \mathfrak{c}(L^{>t})$; by Lemma A.3, $L^{>t} \cap T^{\le t} = \emptyset$, hence $\mathfrak{c}(L^{>t}) \cap \mathfrak{c}(T^{\le t}) = \emptyset$; this implies that $\tilde\omega \notin \mathfrak{c}(T^{\le t})$, i.e. $\tilde\omega \in \mathfrak{c}(T^{\le t})^\mathrm{c}$, which completes the proof of the wanted inclusion. As a consequence of the inclusion just established,

$$[S]f \cdot \mathrm{w} \le \underbrace{[S]f \cdot 1_{\mathfrak{c}(L^{\le t} \cap T^{\le t})} \cdot \mathrm{w}}_{K_1} + \underbrace{[S]f \cdot 1_{\mathfrak{c}(T^{\le t})^\mathrm{c}} \cdot \mathrm{w}}_{K_2} . \tag{23}$$

We proceed now to separately bound K_1 and K_2.

- *Upper bound on K_1.* Using the notation introduced in (3), we first check that

$$\text{on } \mathfrak{c}(L^{\le t}), \text{ hence on } \mathfrak{c}(L^{\le t} \cap T^{\le t}), \text{ we have } f \cdot \tilde{\mathrm{w}}_t = (\widetilde{f_t \cdot \mathrm{w}_t}) . \tag{24}$$

In fact, for any $\omega^t \in L^{\le t}$ and $\tilde\omega \in \Omega^\infty$, we have: $(\widetilde{f_t \cdot \mathrm{w}_t})(\omega^t, \tilde\omega) = (f_t \cdot \mathrm{w}_t)(\omega^t) = f_t(\omega^t) \cdot \mathrm{w}_t(\omega^t) = f(\omega^t, \star^\infty) \cdot \tilde{\mathrm{w}}_t(\omega^t, \tilde\omega) = f(\omega^t, \tilde\omega) \cdot \tilde{\mathrm{w}}_t(\omega^t, \tilde\omega)$, where the equality $f(\omega^t, \star^\infty) = f(\omega^t, \tilde\omega)$ stems from f being prefix-closed and from $(\omega^t, \tilde\omega), (\omega^t, \star^\infty) \in \mathfrak{c}(L_j)$ for some $0 \le j \le t$; this proves (24). Now we have

$$K_1 = \int_{\mathfrak{c}(L^{\le t} \cap T^{\le t})} f \cdot \mathrm{w} \, d\mu_S^\infty \tag{25}$$
$$\le \int_{\mathfrak{c}(L^{\le t} \cap T^{\le t})} f \cdot \tilde{\mathrm{w}}_t \, d\mu_S^\infty \tag{26}$$
$$= \int_{\mathfrak{c}(L^{\le t} \cap T^{\le t})} (\widetilde{f_t \cdot \mathrm{w}_t}) \, d\mu_S^\infty \tag{27}$$
$$= \int_{L^{\le t} \cap T^{\le t}} f_t \cdot \mathrm{w}_t \, d\mu_S^t \tag{28}$$
$$= [S]^t f_t \cdot 1_{L^{\le t} \cap T^{\le t}} \cdot \mathrm{w}_t \tag{29}$$

where: (26) stems from $\mathrm{w} \le \tilde{\mathrm{w}}_t$; in (27) we have used (24), and in (28) we have applied Lemma 2 with $h = f_t \cdot \mathrm{w}_t$ and $B_t = L^{\le t} \cap T^{\le t}$.
- *Upper bound on K_2.* From $f \le M$ and $\mathrm{w} \le \tilde{\mathrm{w}}_t$, we obtain $K_2 \le M \cdot [S] 1_{\mathfrak{c}(T^{\le t})^\mathrm{c}} \cdot \tilde{\mathrm{w}}_t = M \cdot (\int \tilde{\mathrm{w}}_t d\mu_S^\infty - \int_{\mathfrak{c}(T^{\le t})} \tilde{\mathrm{w}}_t d\mu_S^\infty)$. Now, apply Lemma 2 to $h = \mathrm{w}_t$: first with $B_t = \Omega^t$, to obtain $\int \tilde{\mathrm{w}}_t d\mu_S^\infty = \int \mathrm{w}_t d\mu_S^t = [S]^t \mathrm{w}_t$; then with $B_t = T^{\le t}$, to obtain $\int_{\mathfrak{c}(T^{\le t})} \tilde{\mathrm{w}}_t d\mu_S^\infty = \int_{T^{\le t}} \mathrm{w}_t d\mu_S^t = [S]^t \mathrm{w}_t \cdot 1_{T^{\le t}}$. To sum up, we have:

$$K_2 \le M \cdot ([S]^t \mathrm{w}_t - [S]^t 1_{T^{\le t}} \cdot \mathrm{w}_t) = M \cdot [S]^t (1 - 1_{T^{\le t}}) \cdot \mathrm{w}_t . \tag{30}$$

– (Lower bound). Recall that, for any $j \geq 1$, $T_j = (\mathsf{T}^c)^{j-1} \cdot \mathsf{T}$ and that $T^{\leq t} = \cup_{j=1}^{t} T_j \cdot \Omega^{t-j} \subseteq \Omega^t$. Consider now $\Theta^{\leq t} = \cup_{j=1}^{t} T_j \cdot \mathsf{T}^{t-j} \subseteq \Omega^t$. For the sake of conciseness, let us use the following abbreviation:

$$A_t := \Theta^{\leq t} \cdot \mathsf{T}^\infty \cap \mathfrak{c}(L^{\leq t} \cap T^{\leq t})\,. \tag{31}$$

Clearly, $[S]f\mathrm{w} \geq [S]f\mathrm{w}1_{A_t} = \int_{A_t} f\mathrm{w}\, d\mu_S^\infty$. Now we check that:

$$\text{on } A_t\text{, we have } f\mathrm{w} = \widetilde{(f_t\mathrm{w}_t)}\,. \tag{32}$$

Indeed, for any $(\omega^t, \tilde{\omega}) \in A_t$, we have: $(f\mathrm{w})(\omega^t, \tilde{\omega}) = f(\omega^t, \tilde{\omega}) \cdot \mathrm{w}(\omega^t, \tilde{\omega}) = f(\omega^t, \star^\infty) \cdot \mathrm{w}(\omega^t, \tilde{\omega}) = f_t(\omega^t) \cdot \mathrm{w}_t(\omega^t) = (f_t \cdot \mathrm{w}_t)(\omega^t) = \widetilde{(f_t \cdot \mathrm{w}_t)}(\omega^t, \tilde{\omega})$, where: (i) $f(\omega^t, \tilde{\omega}) = f(\omega^t, \star^\infty)$ stems from f being prefix-closed, and $(\omega^t, \tilde{\omega}), (\omega^t, \star^\infty) \in \mathfrak{c}(L_j)$ for some $0 \leq j \leq t$; and, (ii) $\mathrm{w}(\omega^t, \tilde{\omega}) = \mathrm{w}_t(\omega^t)$ stems from $(\omega^t, \tilde{\omega}) \in T_j \cdot \mathsf{T}^\infty$ for some $0 \leq j \leq t$, and recalling that the basic weight function $\mathrm{sc}(\cdot)$ defined on Ω yields 1 on T; this proves (32). Now we have:

$$\begin{aligned}
[S]f \cdot \mathrm{w} &\geq \int_{A_t} f\mathrm{w}\, d\mu_S^\infty & (33)\\
&= \int_{A_t} \widetilde{(f_t\mathrm{w}_t)}\, d\mu_S^\infty & (34)\\
&= \int_{\mathfrak{c}(L^{\leq t} \cap T^{\leq t})} \widetilde{(f_t\mathrm{w}_t)}\, d\mu_S^\infty & (35)\\
&= \int_{L^{\leq t} \cap T^{\leq t}} (f_t\mathrm{w}_t)\, d\mu_S^t & (36)\\
&= [S]^t f_t\mathrm{w}_t 1_{L^{\leq t} \cap T^{\leq t}} & (37)
\end{aligned}$$

where: in the second step we have used (32); in the third step we have applied Lemma A.2 and the fact that $\mathfrak{c}(L^{\leq t} \cap T^{\leq t}) \cap \Theta = A_t$; and in the last but one step, Lemma 2 with $h = f_t\mathrm{w}_t$ and $B_t = L^{\leq t} \cap T^{\leq t}$.

The bounds in (29), (30) and (37) imply the wanted bounds (7).

Proof of Theorem 3. Write (8) in the form (11). We discuss the limit as $t \longrightarrow +\infty$ of each of the three distinct involved terms .

1. $[S]^t f_t 1_{L^{\leq t} \cap T^{\leq t}} \mathrm{w}_t \longrightarrow \int f\mathrm{w}\, d\mu_S^\infty = [S]f\mathrm{w}$. Consider the set $A_t = \Theta^{\leq t} \cdot \mathsf{T}^\infty \cap \mathfrak{c}(L^{\leq t} \cap T^{\leq t})$ introduced in (31). We have the equalities

$$\begin{aligned}
[S]^t f_t \cdot 1_{L^{\leq t} \cap T^{\leq t}} \cdot \mathrm{w}_t &= \int_{A_t} f \cdot \mathrm{w}\, d\mu_S^\infty\\
&= \int f \cdot \mathrm{w} \cdot 1_{\mathfrak{c}(T^{\leq t} \cap L^{\leq t})}\, d\mu_S^\infty & (38)
\end{aligned}$$

where: the first equality has been proven in (33)–(37), and the second one follows from Lemma A.2 and the equality $A_t \cap \Theta = \mathfrak{c}(T^{\leq t} \cap L^{\leq t}) \cap \Theta$. Now $1_{\mathfrak{c}(T^{\leq t} \cap L^{\leq t})}$ converges pointwise to $1_{T_\mathrm{f} \cap \mathrm{supp}(f)}$, by definition of $\mathrm{supp}(f) = \cup_{j \geq 0} \mathfrak{c}(L_j)$: in particular, for each $\tilde{\omega}$ we have that, for each t large enough, $1_{\mathfrak{c}(T^{\leq t} \cap L^{\leq t})}(\tilde{\omega}) = 1_{T_\mathrm{f} \cap \mathrm{supp}(f)}(\tilde{\omega})$. This in turn implies that $f \cdot \mathrm{w} \cdot 1_{\mathfrak{c}(T^{\leq t} \cap L^{\leq t})}$ converges pointwise to $f \cdot \mathrm{w} \cdot 1_{T_\mathrm{f} \cap \mathrm{supp}(f)} = f \cdot \mathrm{w} \cdot 1_{T_\mathrm{f}}$ (even if f takes on the value $+\infty$). Moreover, the sequence of functions $f \cdot \mathrm{w} \cdot 1_{\mathfrak{c}(T^{\leq t} \cap L^{\leq t})}$ is monotonically nondecreasing. By the Monotone Convergence Theorem [2, Th.1.6.2], the integral in (38) then converges to $\int f \cdot \mathrm{w} \cdot 1_{T_\mathrm{f}}\, d\mu_S^\infty = \int f \cdot \mathrm{w}\, d\mu_S^\infty$, where the last equality stems from $\mu_S^\infty(T_\mathrm{f}) = 1$ and again Lemma A.2.
2. $[S]^t \mathrm{w}_t \longrightarrow [S]\mathrm{w}$. By taking $B_t = \Omega^t$ in Lemma 2, we have: $[S]^t \mathrm{w}_t = \int \mathrm{w}_t\, d\mu_S^t = \int \widetilde{\mathrm{w}_t}\, d\mu_S^\infty$. Moreover, the sequence of functions $\widetilde{\mathrm{w}_t}$ converges pointwise to w, and all these function are dominated by e.g. 1, which is integrable. Applying the Dominated Convergence Theorem [2, 1.6.9], we obtain $\int \widetilde{\mathrm{w}_t}\, d\mu_S^\infty \longrightarrow \int \mathrm{w}_t\, d\mu^\infty = [S]\mathrm{w}$, which is the wanted statement.

3. $[S]^t 1_{T^{\leq t}} \cdot \mathrm{w}_t \to [S]\mathrm{w}$. Apply the first item above to the constant function $f = 1$. Note that this f is trivially prefix-closed with $L_0 = \{\epsilon\}$, hence $L^{\leq t} = \Omega^t$ for each $t \geq 1$.

As to (12), we have $[S]^t 1_{T^{\leq t}} \mathrm{w}_t = \int_{T^{\leq t}} \mathrm{w}_t \, d\mu_S^t = \int \mathrm{w}_t \, d\mu_S^t = [S]^t \mathrm{w}_t$, which follows from the hypothesis $\mu_S^t(T^{\leq t}) = 1$ and from elementary measure-theoretic reasoning. Therefore $\alpha_t = 1$ and from (8) we obtain that the lower and upper bounds on $[[S]]f$ coincide with $\frac{[S]^t f_t \cdot 1_{L^{\leq t} \cap T^{\leq t}} \cdot \mathrm{w}_t}{[S]^t \mathrm{w}_t}$. Now we check that

$$f_t \cdot 1_{T^{\leq t}} = f_t \cdot 1_{L^{\leq t} \cap T^{\leq t}} \, .$$

Indeed, as $\mathrm{supp}(f_t) \subseteq L^{\leq t} \cup L^{>t}$, one can consider two cases for $\omega^t \in \mathrm{supp}(f_t)$. Either $\omega^t \in \mathrm{supp}(f_t) \cap L^{\leq t}$: then we have by definition that $(f_t \cdot 1_{T^{\leq t}})(\omega^t) = (f_t \cdot 1_{L^{\leq t} \cap T^{\leq t}})(\omega^t)$. Or $\omega^t \in \mathrm{supp}(f_t) \cap L^{>t}$: then we have $\omega^t \notin T^{\leq t}$ (Lemma A.3), hence again $(f_t \cdot 1_{T^{\leq t}})(\omega^t) = (f_t \cdot 1_{L^{\leq t} \cap T^{\leq t}})(\omega^t) = 0$. Finally, we can compute as follows: $[S]^t f_t \cdot \mathrm{w}_t = [S]^t f_t \cdot 1_{T^{\leq t}} \cdot \mathrm{w}_t = [S]^t f_t \cdot 1_{L^{\leq t} \cap T^{\leq t}} \cdot \mathrm{w}_t$, where the first equality follows from $\mu(T^{\leq 1}) = 1$ and elementary measure-theoretic reasoning, and the second one from the above established equality. This completes the proof of (12).

The following result is useful to relate our semantics to the filtering distribution of a Feynman-Kac model. Here we let $\mathrm{pr}_j : \Omega^t \to \Omega$ $(1 \leq j \leq t)$ denote the projection on the jth component; this is a measurable function. The result basically says that taking the expectation of f_t on paths of length t is the same as taking the expectation of $h \circ \mathrm{pr}_t$, that only looks at the last state of a path.

Lemma A.4. *Let $f = \check{h}$ for a nonnegative h defined on Ω. For each S and $t \geq 1$, we have $[S]^t f_t \cdot 1_{L^{\leq t}} \cdot \mathrm{w}_t = [S]^t (h \circ \mathrm{pr}_t) \cdot \mathrm{w}_t$, where L_j $(j \geq 0)$ are the branches of f.*

Proof. We first prove a general statement about the measures μ_S^t. Recall that we use ω^t to range over tuples $(\omega_1, ..., \omega_t)$. Let $g : \Omega^t \to \overline{\mathbb{R}}^+$ be a measurable function. For each $1 \leq j \leq t$, let $\eta_j(\omega^t) := [\omega_{j+1} = \omega_j] \cdots \cdot [\omega_t = \omega_j]$ the predicate that yields 1 if and only if $\omega_j = \cdots = \omega_t$. Then, recalling that $T_j = (\mathsf{T}^c)^{j-1} \cdot \mathsf{T}$, we have

$$\int_{T_j \cdot \Omega^{t-j}} \mu_S^t(d\omega^t) \, g(\omega^t) = \int_{T_j \cdot \Omega^{t-j}} \mu_S^t(d\omega^t) \, g(\omega^t) \cdot \eta_j(\omega^t) \, . \tag{39}$$

This equality can be checked as follows. First, write the integral on the left- (resp. right-)hand side of (39) as an iterated integral, via Theorem A.1(b) (Fubini): in the resulting expression, call H_j (resp. K_j) the expression corresponding to the $t - j$ innermost iterated integrals. For each $\omega_1, ..., \omega_j$, we have the following equalities

$$\begin{aligned}
H_j &= \int \kappa(\omega_j)(d\omega_{j+1}) \int \kappa(\omega_{j+1})(d\omega_{j+2}) \cdots \int \kappa(\omega_{t-1})(d\omega_t) \, g(\omega^t) \cdot 1_{T_j \cdot \Omega^{t-j}}(\omega^t) \\
&= g(\omega_1, ...\omega_{j-1}, \omega_j, ..., \omega_j) \cdot 1_{\mathsf{T}^c}(\omega_1) \cdots 1_{\mathsf{T}^c}(\omega_{j-1}) \cdot 1_{\mathsf{T}}(\omega_j) && (40) \\
&= g(\omega_1, ...\omega_{j-1}, \omega_j, ..., \omega_j) \cdot 1_{\mathsf{T}^c}(\omega_1) \cdots 1_{\mathsf{T}^c}(\omega_{j-1}) \cdot 1_{\mathsf{T}}(\omega_j) \cdot \eta_j(\omega_1, ...\omega_{j-1}, \omega_j, ..., \omega_j) && (41) \\
&= \int \kappa(\omega_j)(d\omega_{j+1}) \int \kappa(\omega_{j+1})(d\omega_{j+2}) \cdots \int \kappa(\omega_{t-1})(d\omega_t) \, g(\omega^t) \cdot \eta_j(\omega^t) \cdot 1_{T_j \cdot \Omega^{t-j}}(\omega^t) && (42) \\
&= K_j
\end{aligned}$$

where: (40) is obvious if $\omega_j \notin \mathsf{T}$, as both sides are 0 in this case; if $\omega_j \in \mathsf{T}$, say $\omega_j = (v, \mathsf{nil})$, then $\kappa(\omega_j)(\cdot) = \delta_{\omega_j}(\cdot)$ by definition of κ, and (40) follows by a repeated application of the property $\int \delta_\omega(d\omega')q(\omega') = q(\omega)$ of Dirac's measures; (41) follows by definition of η_j; and (42) from the same reasoning as for (40). Now (39) follows by integrating H_j and K_j with respect to $\omega_1, ..., \omega_j$ (note that both sides are measurable functions of $\omega_1, ..., \omega_j$, by Fubini), and then applying Fubini on both sides, to rewrite the resulting iterated integrals as integrals over Ω^t. Now we have

$$\int_{T_j\cdot\Omega^{t-j}} \mu_S^t(d\omega^t)\,(f_t\cdot 1_{L^{\le t}}\cdot \mathrm{w}_t)(\omega^t) = \int_{T_j\cdot\Omega^{t-j}} \mu_S^t(d\omega^t)\,(f_t\cdot \mathrm{w}_t)(\omega^t) \tag{43}$$

$$= \int_{T_j\cdot\Omega^{t-j}} \mu_S^t(d\omega^t)\,(f_t\cdot \mathrm{w}_t\cdot\eta_j)(\omega^t) \tag{44}$$

$$= \int_{T_j\cdot\Omega^{t-j}} \mu_S^t(d\omega^t)\,(\mathrm{w}_t\cdot\eta_j)(\omega^t)\cdot h(\omega_j) \tag{45}$$

$$= \int_{T_j\cdot\Omega^{t-j}} \mu_S^t(d\omega^t)\,(\mathrm{w}_t\cdot\eta_j)(\omega^t)\cdot h(\omega_t) \tag{46}$$

$$= \int_{T_j\cdot\Omega^{t-j}} \mu_S^t(d\omega^t)\,(\mathrm{w}_t\cdot\eta_j\cdot(h\circ \mathrm{pr}_t))(\omega^t) \tag{47}$$

$$= \int_{T_j\cdot\Omega^{t-j}} \mu_S^t(d\omega^t)\,((h\circ \mathrm{pr}_t)\cdot \mathrm{w}_t)(\omega^t) \tag{48}$$

where one exploits the following equalities, for $\omega^t \in T_j\cdot\Omega^{t-j}$: in (43), $(1_{L^{\le t}}\cdot f_t)(\omega^t) = f_t(\omega^t)$, as $\mathrm{supp}(f_t)\cap T_j\cdot\Omega^{t-j} \subseteq L^{\le t}\cap T_j\cdot\Omega^{t-j}$; in (44), (39) with $g = f_t\cdot \mathrm{w}_t$; in (45), $f_t(\omega^t) = h(\omega_j)$; in (46), $\eta_j(\omega^t)h(\omega_j) = \eta_j(\omega^t)h(\omega_t)$; in (47), the definition of $h(\omega_t)$ and $\mathrm{pr}_t(\omega^t)$; in (48), again (39), with $g = (h\circ \mathrm{pr}_t)\cdot \mathrm{w}_t$. Now, we have:

$$[S]^t f_t\cdot 1_{L^{\le t}}\cdot \mathrm{w}_t = \int f_t\cdot 1_{L^{\le t}}\cdot \mathrm{w}_t\, d\mu^t$$

$$= \int_{T^{\le t}} f_t\cdot 1_{L^{\le t}}\cdot \mathrm{w}_t\, d\mu^t \tag{49}$$

$$= \sum_{j=1}^t \int_{T_j\cdot\Omega^{t-j}} f_t\cdot 1_{L^{\le t}}\cdot \mathrm{w}_t\, d\mu^t \tag{50}$$

$$= \sum_{j=1}^t \int_{T_j\cdot\Omega^{t-j}} (h\circ \mathrm{pr}_t)\cdot \mathrm{w}_t\, d\mu^t \tag{51}$$

$$= \int_{T^{\le t}} (h\circ \mathrm{pr}_t)\cdot \mathrm{w}_t\, d\mu^t \tag{52}$$

$$= \int_{\Theta^{\le t}} (h\circ \mathrm{pr}_t)\cdot \mathrm{w}_t\, d\mu^t \tag{53}$$

$$= \int_{\Theta^{\le t}} (h\circ \mathrm{pr}_t)\cdot \mathrm{w}_t\, d\mu^t + \int_{(\mathsf{T}^{\mathrm{c}})^t} (h\circ \mathrm{pr}_t)\cdot \mathrm{w}_t\, d\mu^t \tag{54}$$

$$= \int_{\Theta_t} (h\circ \mathrm{pr}_t)\cdot \mathrm{w}_t\, d\mu^t \tag{55}$$

$$= \int (h\circ \mathrm{pr}_t)\cdot \mathrm{w}_t\, d\mu^t \tag{56}$$

where: (49) follows from $L^{\le t}\subseteq T^{\le t}$; (50) from $T^{\le t} = \cup_{j=1}^t T_j\cdot\Omega^{t-j}$ (disjoint union) and basic properties of integrals; (51) from the equality established in (43)—(48); (52) again from $T^{\le t} = \cup_{j=1}^t T_j\cdot\Omega^{t-j}$; (53) from Lemma A.2(b) and $\Theta_t\cap T^{\le t} = \Theta^{\le t}$; (54) from the fact that $h\circ \mathrm{pr}_t$ is identically 0 on $(\mathsf{T}^{\mathrm{c}})^t$ as $\mathrm{supp}(h)\subseteq \mathsf{T}$, hence the second integral here is 0; (55) from definition of $\Theta_t = \Theta_{\le t}\cup(\mathsf{T}^{\mathrm{c}})^t$ (disjoint union); (56) again from Lemma A.2(b).

Proof of Theorem 4. The following equality is easy to check and will be useful below.

$$\mathrm{E}_{\phi_t}[h] = \frac{\mathrm{E}_{\mu^t}[(h\circ \mathrm{pr}_t)\cdot G]}{\mathrm{E}_{\mu^t}[G]}\,. \tag{57}$$

As for the actual proof, first note that $L^{\le t}\subseteq T^{\le t}$ by definition of lifting (Def. 8), hence $f_t\cdot 1_{L^{\le t}\cap T^{\le t}} = f_t\cdot 1_{L^{\le t}}$. Concerning β_L, from the lower bound in (8) we have that:

$$[\![S]\!]f \ge \frac{[S]^t f_t\cdot 1_{L^{\le t}\cap T^{\le t}}\cdot \mathrm{w}_t}{[S]^t \mathrm{w}_t} = \frac{[S]^t f_t\cdot 1_{L^{\le t}}\cdot \mathrm{w}_t}{[S]^t \mathrm{w}_t} = \frac{[S]^t (h\circ \mathrm{pr}_t)\cdot \mathrm{w}_t}{[S]^t \mathrm{w}_t} = \mathrm{E}_{\phi_{S,t}}[h]$$

where in the last but one step we have applied Lemma A.4 to the numerator of the fraction, and in the last step we have applied (57) to the model FK_S, with $G = \mathrm{w}_t$ as a global potential.

Concerning β_U, consider the function $f = \check{1}_{\mathsf{T}}$ (the lifting of the function $h = 1_{\mathsf{T}}$), which has branches $L_j = T_j$: it is immediate to check that for each $t \geq 1$, $1_{T^{\leq t}} \cdot \mathrm{w}_t = f_t \cdot 1_{T^{\leq t}} \cdot \mathrm{w}_t$. Then we can repeat the reasoning used above for β_L with this function f to prove that $\frac{[S]^t 1_{T^{\leq t}} \cdot \mathrm{w}_t}{[S]^t \mathrm{w}_t} = \mathrm{E}_{\phi_{S,t}}[1_{\mathsf{T}}]$, that is $\alpha_t = \mathrm{E}_{\phi_{S,t}}[1_{\mathsf{T}}]^{-1}$. Then the upper bound in (8) allows us to complete the proof.

B The Particle Filtering algorithm

From a computational point of view, our interest in FK models lies in the fact that they allow for a simple, unified presentation of a class of efficient inference algorithms, known as *Particle Filtering (PF)* [21, 19]. In what follows we present an algorithm to compute the filtering distribution ϕ_t. We will introduce below a general PF algorithm scheme following closely [19, Ch.11].

Fix a generic FK model, $\mathrm{FK} = (\mathcal{X}, t, \mu^1, \{K_i\}_{i=2}^t, \{G_i\}_{i=1}^t)$. Fix $N \geq 1$, the number of *particles*, that is instances of the random process represented by the K_i's, we want to simulate. For any tuple $W = W^{1:N} = (W^{(1)}, ..., W^{(N)})$ of real nonnegative random variables, the *weights*, denote by $\widehat{W}$ the normalized version of W, that is[9] $\widehat{W}^{(i)} = W^{(i)}/(\sum_{j=1}^N W^{(j)})$. A *resampling scheme* for (N, W) is N-tuple of random variables taking values on $1..N$, say $R = (R_1, ..., R_N)$, such that: for each $i \in 1..N$, letting F_i denote the number of occurrences of i in R, one has

$$\mathrm{E}[F_i|W] = N \cdot \widehat{W}_i \,.$$

In other words, R is a randomized selection process of N indices out of $1..N$, with repetitions, such that, on average, each index $i \in 1..N$ is selected a number of times proportional to its weight in W. We shall write $R(W)$ to indicate that R depends on a given weight vector W. Various resampling schemes have been proposed in the literature. Perhaps the simplest is letting R be N i.i.d. random variables each distributed according to $\widehat{W}$: this is known as *multimomial resampling*. We refer the reader to the specialized literature on PF for details and efficient implementation methods, see e.g. [19, Ch.9] and references therein.

Algorithm 1 is a generic PF algorithm. At the k-th iteration, for $k = 1, ..., t$, two N-tuples are extracted:

- a tuple of states $X_k = X_k^{1:N} = (X_k^{(1)}, ..., X_k^{(N)}) \in \mathcal{X}^N$;
- a tuple of (unnormalized) weights $W_k = W_k^{1:N} = (W_k^{(1)}, ..., W_k^{(N)}) \in (\mathbb{R}^+)^N$.

The elements of $X_k^{1:N}$ depend on the tuples $X_{k-1}^{1:N}, W_{k-1}$ of the previous iteration. The purpose of the resampling step 4 is to give more importance to particles with higher weight, when extracting the next tuple of particles, while discarding particles with lower weight. In case R is multinomial resampling, steps 4-5 amount to drawing each $X_k^{(j)}$ from the (empirical) distribution $\sum_{j=1}^N \widehat{W}_{k-1}^{(j)} \delta_{K_k(X_{k-1}^{(j)})}$. The weights $W_k^{1:N}$ are computed via the potential function G_k, and will be used in the resampling step at iteration $k+1$, if $k < t$, or returned as part of the algorithm's output.

The following theorem states consistency, in an asymptotic sense, of the PF algorithm with respect to the filtering distribution ϕ_t on $\mathcal{X}$. Its practical implication is that we can estimate expectations with respect to the filtering distribution as weighted sums. Note that in its statement t is held fixed — it is one of the parameter of the FK model — while the number of particles N tends to $+\infty$.

Theorem B.1 (convergence of PF, [19]). *Consider the random variables (X_t, W_t) $(t \geq 1)$ as returned by Algorithm 1. Suppose that the FK measure ϕ is well defined on $\mathcal{X}^t$. Further assume that $R(\cdot)$ is multinomial resampling and that the potential functions G_k all have a finite upper bound. For each nonnegative measurable function h defined on $\mathcal{X}$, we have: $\sum_{j=1}^N \widehat{W}_t^{(j)} \cdot h(X_t^{(j)}) \longrightarrow \mathrm{E}_{\phi_t}[h]$ almost surely[10] as $N \to +\infty$.*

[9] With the proviso that e.g. $\widehat{W}^{(i)} := 1/N$ in the event all the $W^{(i)}$'s are 0. In the execution of the PF algorithm this event will occur with probability $\to 0$ as $N \to +\infty$.

[10] See [19, Ch.11] for the precise definition of the probability space where this assertion makes sense.

C Additional details on experiments

We give a more detailed textual description of the considered examples.

1. *Aircraft tracking* (AT) [51]. An aircraft is modeled as a point moving on a 2D plan according to a Gaussian process, The aircraft is tracked by six radar: at each discrete time step, each radar noisily measures the distance of the aircraft from its own position; specific distances are being observed. We target the posterior expected value of the final horizontal position of the aircraft. This is by far the most complicated example among those considered here; we provide a detailed description of its coding in terms of a PPG at the end of this section.
2. *Drunk man and mouse* (DMM), Example 3. We target the posterior expected value of the drunk man variance.
3. *Hare and tortoise* (HT), see e.g. [4]. This model simulates a race between a hare and a tortoise along a one-dimensional line: the tortoise takes a step of length 1 every time step, while the hare occasionally takes a step whose length is Gaussian-distributed. Additionally, at each time step it is observed that the hare and the tortoise are never at a distance more than 10 from each other. The race is terminated as soon as the hare overtakes the turtle. We target the posterior expected value of the final position of the hare.
4. *Bounded retransmission protocol*, [31]. A number of packets must be transmitted over a lossy channel, and each packet can be lost with a probability of 0.02. Losses can be observed only during the transmission of the last 80 packets. The transmission is considered successful if none of the packets needs more than 4 retransmissions. We target the posterior expected value of failure probability.
5. *Non-i.i.d. loops*, [31]. This model describes the behaviour of a discrete sampler that keeps tossing two fair coins, until they both turn tails. Additionally, it is observed that at each iteration at least one of the coins yields the same outcome as in the previous iteration. This observation induce data dependencies across consecutive loop iterations. We target the posterior expected value of the number of iterations until termination.

For each model, we draw samples of N ($N \in \{10^3, 10^4, 10^5, 10^6\}$) particles, and the corresponding weights, with each of the considered tools/algorithms. With the drawn samples and weights, the tools compute the (posterior) expected value of the quantities of interest. For each tool, we are interested in assessing:

- the accuracy of the computed expected values;
- the quality of the drawn samples;
- the performance in terms of execution time.

Concerning accuracy, we do not know the exact value of the targeted expected values (but in one case, see below), so a direct comparison is not possible. Nevertheless, asymptotic consistency of PF and other SMC sampling algorithms, in the sense discussed in subsection 5, guarantees that as as $N \to +\infty$ the sample estimates will converge to the true expected value. For a specific values of N, there is no obvious way to judge how close we are to convergence: pragmatically, we will take the fact that the different tools yield estimates very close to one another as an empirical evidence of convergence and accuracy. As remarked above, differently from the other tools that return truncated point estimates, VPF provides in principle lower and upper bounds of $[\![S]\!]f$ as an application of Theorem 4. The upper bound will be vacuous ($+\infty$) whenever the target f is unbounded, which is the case for HT and NIID here. Beside, examples AT and BRP are bounded loops, for which $\alpha_t = 1$, hence $\beta_L = \beta_U = [\![S]\!]f$.

We measure empirically the quality of the drawn samples in terms of *effective sample size (ESS)* [22] of the corresponding weights $W_1, ..., W_N$:

$$ESS := \frac{(\sum_{i=1}^{N} W_i)^2}{\sum_{i=1}^{N} W_i^2} .$$

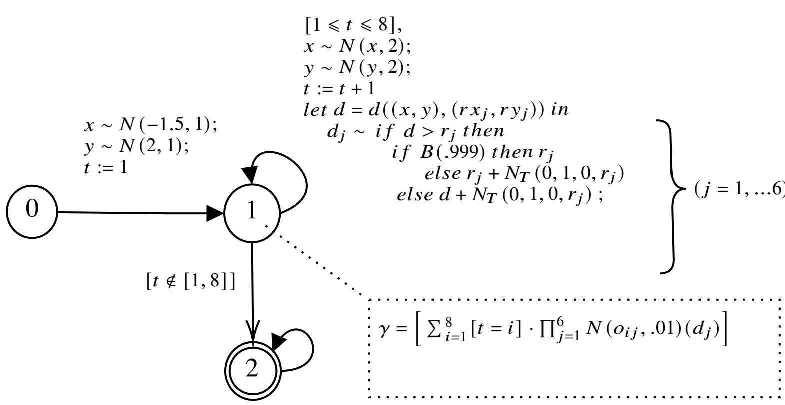
ESS is an empirical measure of efficiency of the sampled particles, the higher the better. Specifically, ESS quantifies the number of i.i.d. samples from the target distribution that would be required to achieve the same variance in the estimator as that obtained from the weighted samples. So a ESS close to N indicates that the N particles appear to be drawn i.i.d. from the target distribution.

The experiments have been run on a 2.8 GHz Intel Core i7 PC, with 16GB RAM and Nvidia T500 GPU. VPF and webPPL have been run under Windows 10 OS, with CUDA Toolkit v. 11.8, driver v. 522.06. CorePPL and RootPPL have been run under Ubuntu 22, with CUDA Toolkit v. 12.2, driver v. 535.86.

In Table 1 (Section 6), we report the execution time, the estimated expected value and the effective sample size for VPF, CorePPL, RootPPL and webPPL, as the number N of particles increases. In the case of VPF, a single value estimate is reported for all examples but DMM: for programs AT and BRP this is an estimate of $[[S]]f$; for examples HT and NIID (unbounded f), this is an estimate the lower bound β_L, being the upper bound vacuous as discussed above. We also remark that for NIID, it is known that $[[S]]f = \frac{24}{7} = 3.428\cdots$ [31]. We note that, at least for $N \geq 10^5$, the tools tend to yield very similar estimates of the expected value[11], as a consequence of the asymptotic consistency of PF and other SMC algorithms: we take this as an empirical evidence of accuracy.

We end this section with an explicit description of the PPG for AT. An aircraft is modeled as a point moving on a 2D plan according to a Gaussian process, for $t = 1, \ldots, 8$ discrete time instants. Throughout these time points, the airplane is tracked by six radar. Each radar is characterized by a radius: at each time, if the aircraft is within the radar's radius, the radar returns the noisily measured distance from the aircraft, otherwise the radar just returns a noisy version of its own radius. We aim to infer the final horizontal position of the aircraft, i.e. the value of x at time $t = 8$, conditioned on actual observed data obtained from the six radars at all eight time instants.

In the PPG below, o_{ij} is the observed distance at time i from radar j, for $1 \leq i \leq 8$ and $1 \leq j \leq 6$, while (rx_j, ry_j) and r_j are the coordinates and radius of radar j, respectively. The actual numerical data can be found in [51]. Moreover $B(p)$ is the Bernoulli distribution of parameter p, while $N_T(a, b, c, d)$ represents the Normal density of mean c and standard deviation d truncated at $[a, b]$; $N_T(a, b, c, d)(z)$ is the value at z of this density.



[11] An exception is represented by the result returned by CorePPL for the NIID example. The results of the other three tools agree with each other and with the exact value $\frac{24}{7} = 3.428\cdots$, though.