Approximations in Probabilistic Programs

Ekansh Sharma
ekansh@cs.toronto.edu
Department of Computer Science
University of Toronto
Vector Institute

Daniel M. Roy
droy@utstat.toronto.edu
Department of Statistical Sciences
University of Toronto
Vector Institute

Abstract

We study the first-order probabilistic programming language introduced by Staton et al. [14], but with an additional language construct, **stat**, that, like the fixpoint operator of Atkinson et al. [2], converts the description of the Markov kernel of an ergodic Markov chain into a sample from its unique stationary distribution. Up to minor changes in how certain error conditions are handled, we show that **norm** and **score** are eliminable from the extended language, in the sense of Felleisen [5]. We do so by giving an explicit program transformation and proof of correctness. In fact, our program transformation implements a Markov chain Monte Carlo algorithm, in the spirit of the "Trace-MH" algorithm of Wingate et al. [16] and Goodman et al. [6], but less sophisticated to enable analysis. We then explore the problem of approximately implementing the semantics of the language with potentially nested **stat** expressions, in a language without **stat**. For a single **stat** term, the error introduced by the finite unrolling proposed by Atkinson et al. vanishes only asymptotically. In the general case, no guarantees exist. Under uniform ergodicity assumptions, we are able to give quantitative error bounds and convergence results for the approximate implementation of the extended first-order language.

Keywords Probabilistic Programming, Nested Markov Chains, Quantitative Error Bounds

1 Introduction

Probabilistic programming languages (PPLs) for Bayesian modelling, like Stan and TensorFlow Probability, provide statisticians and data scientists a formal language to model observed data and latent (i.e., unobserved) variables [1, 4, 6]. In these languages, users specify a "prior" probability distribution that represents prior beliefs/assumptions about the data and latent variables. Then conditional statements are introduced to represent actual observed data. The semantics of a program in these probabilistic languages is the "posterior" (i.e., conditional) probability distribution, which represents the prior beliefs after being updated (conditioned) on observed data.

Conference'17, July 2017, Washington, DC, USA
2020. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00

Staton [13], Staton et al. [14] give precise measure theoretic semantics to an idealized first order probabilistic languages. Beyond the standard deterministic core language, the language provides three constructs for modeling data and latent variables: the **sample** construct introduces latent variables whose values are subject to uncertainty; the **score** construct introduces data into a model in the form of likelihoods; the **norm** construct combines the latent variables and data likelihoods to produce the (normalized) posterior distribution, representing updated beliefs given data.

Irrespective of the theoretical prospect of exact implementations of PPL semantics, approximations are ubiquitous in implementations of PPLs that are intended to scale to the demands of real-world Bayesian modelling. Indeed, in all but the simplest models, approximations necessarily arise because the key operation—computing the posterior distribution from the prior distribution and data—is computationally intractable. As a result, every scalable PPL implementation performs a nontrivial transformation that modifies the semantics of the original program in complex ways by introducing errors that tradeoff with computation. Often, these transformations are implicit within an inference engine, rather than producing a new program in a formal language. Since approximation is critical to scalable implementations of PPLs, it is of interest to define PPLs in which we can reason about the process of introducing approximations. In this work, we take a step in this direction.

Most practical implementations of PPLs use one (or a combination) of two classes of approximate inference methods: *Markov chain Monte Carlo* (MCMC) methods approximate the posterior via the simulation of a Markov chain whose stationary (target) distribution is the posterior distribution; whereas, *variational* methods use (possibly stochastic) optimization to search among a family of tractable distribution for one that best approximates the posterior distribution.

In this work, we focus on MCMC methods. While many existing PPLs are powerful enough to represent the MCMC implementation of any program in their own language, existing systems cannot reason about the error introduced by simulating Markov chains for a finite number of steps. This challenge grows when we ask how the error scales under composition of multiple approximate programs and nested MCMC approximations.

1.1 Contributions

In this paper, we bridge the gap between the semantics of a probabilistic programming language and that of a Markov chain Monte Carlo implementation of the language, building on the foundations laid by Staton [13], Staton et al. [14]. We start from their first-order PPL, whose **sample**, **score**, and **norm** constructs permit the specification of Bayesian models and conditioning. In Section 4, we append the language proposed by Staton et al. [14] to include **stat** construct, that takes a Markov kernel and returns the unique stationary distribution, if one exists. In Section 5, using the framework of eliminability [5], we show that the language with only **sample** and **stat** is equivalent to the original language. In Section 6, we give an approximate compiler for the **stat** construct, similar to Atkinson et al. [2]. We then identify some semantic constraints on the Markov kernel, given as argument to the **stat** construct, under which we can derive quantitative error bounds.

2 Related Work

Atkinson et al. [2] propose a language in which users can manually construct inference procedures for probabilistic models. In particular, users can specify (Markov transition) kernels and their stationary distributions using a fixpoint construct, like **stat**. They propose to approximately compute fixpoints by iteration, but leave open the problem of characterizing how this approximation affects the semantics, and whether one can control the approximation error. Our work provides the first such guarantees by exploiting uniform ergodicity.

Borgström et al. [3], Hur et al. [7], Ścibior et al. [12] introduce Markov Chain Monte Carlo based implementations of distinct but related PPLs. In each case, the authors provide theoretical guarantees, building on standard results in Markov chain theory. Informally speaking, when a program has the equivalent of a single, top-level **norm** expression, these results guarantee that the original semantics are recovered asymptotically. This line of work does not provide a framework for quantifying error. As a result, these results do not bear on the correctness, even asymptotically, of programs with nested **norm** terms. In Section 6, we give assumptions on the underlying Markov chains that allow us to quantify the error of the approximate inference scheme, even under composition and nesting.

Rainforth [10] studies the error associated with Monte Carlo approximations to nested **norm** terms, assuming that one can produce exact samples from the **norm** terms. In this work, we contend with the approximation error associated with nested Markov chains that are not assumed to have converged to their stationary distribution. In Section 6, we show that one can quantify the rate at which nested Markov chains converge, under additional hypotheses.

In the MCMC literature, Medina-Aguayo et al. [9], Roberts et al. [11] study the convergence of Markov chain with approximate transition kernels. Medina-Aguayo et al. [9] gives quantitative convergence bounds for approximate Metropolis–Hastings algorithm when the acceptance probability is approximate, e.g., due to the dynamics of the Markov chain being perturbed. In our setting, the perturbation of the Markov chain is caused by the dynamics of nested Markov chains. We posit assumptions of uniform ergodicity on nested Markov chains and give quantitative error bounds.

3 Measure-theoretic preliminaries

We assume the reader is familiar with the basics of measure and probability theory: σ-algebras, measurable spaces, measurable functions, random variables, (countably additive) measures, (Lebesgue) integration, conditional probability distributions. We review a few definitions here for completeness: Recall that a measure μ on a measurable space (X, Σ_X) is said to be *finite* when $\mu(X) < \infty$, said to be *S-finite* when $\mu = \sum_{i\in\mathbb{N}} \mu_i$, where μ_i is a finite measure for all i, and said to be *σ-finite* when $\mu = \sum_{i\in\mathbb{N}} \mu_i$, where μ_i is a finite measure for all i and μ_i, μ_j are *mutually singular* for all $i \neq j$. We say μ is a *probability measure* when $\mu(X) = 1$ and a *sub-probability measure* when $\mu(X) \leq 1$. We say a property P holds *μ-almost everywhere (or μ-a.e.)* if $\mu(\{x\colon \neg P(x)\}) = 0$.

Let (X, Σ_X) and (Y, Σ_Y) be measurable spaces. Recall that a *kernel (from X to Y)* is a function $k\colon X \times \Sigma_Y \to \mathbb{R}_+$ such that, for all $x \in X$, the function $k(x, \cdot)\colon \Sigma_X \to [0, \infty]$ is a measure, and, for all $A \in \Sigma_X$, the function $k(\cdot, A)\colon \to [0, \infty]$ is measurable. We say that k is *finite* (resp., *σ-finite*, and *S-finite*) if, for all x, $k(x, \cdot)$ is a finite (resp., *σ-finite*, and *S-finite*) measure. Similarly, k is a *(sub-)probability kernel* if for all x, $k(x, \cdot)$ is a (sub-)probability measure. In this paper we study approximations to probabilistic semantics and give rates of convergence. These notions require metric structure on the space of probabilistic distributions. Recall that the *total variation* distance between a pair μ, ν of probability distributions on a common measurable space (X, Σ_X) is defined to be

$$\|\mu - \nu\|_{\mathrm{tv}} = \sup_{A\in\Sigma_{[\![A]\!]}} |\mu(A) - \nu(A)| .$$

The total variation distance is a metric.

3.1 Markov Chain theory

We introduce a new language construct explicitly designed to refer to the long run behavior of repeating the same probabilistic computation over and over. The study of such phenomena falls within the remit of Markov chain theory. We summarize basic definitions and results here.

Fix a basic probability space, and write P for the associated probability measure. All random variables will be defined relative to this structure. Let (X, Σ_X) be a measurable space. A *Markov chain* (with *state space* X) is a sequence of X-valued random variables $(Y_0, Y_1, \dots)$ such that, for some probability

kernel $k : X \times \Sigma_X \to [0,1]$, with probability one, for all measurable subsets $A \in \Sigma_X$,

$$P(Y_i \in A | Y_0, Y_1, \ldots, Y_{i-1}) = k(Y_{i-1}, A).$$

The probability kernel k is called the *transition kernel* of the Markov chain. The transition kernel k, combined with an initial distribution $\mu = P(Y_0 \in \cdot)$, determines the distribution of the Markov chain. Note that our definition is somewhat stronger than the usual abstract definition in terms of conditional independence, though agrees with the usual definition for Borel spaces.

For notational convenience, define $k^n : X \times \Sigma_X \to [0,1]$ by

$$k^n(x, \cdot) \stackrel{\text{def}}{=} \int_X k(y, \cdot) k^{n-1}(x, dy).$$

The (probability) distribution (or law) of the random variable Y_n, denoted $\mathcal{L}(Y_n)$, satisfies

$$\mathcal{L}(Y_n)(\cdot) = \int k^n(x, \cdot)\mu(dx).$$

We say that a probability measure π is *invariant* with respect to k when

$$\left\| \pi(\cdot) - \int_X k(x, \cdot)\pi(dx) \right\|_{\text{tv}} = 0.$$

A Markov chain is *ergodic* if it admits a unique invariant distribution, π. Recall that μ is *absolutely continuous* with respect to π (equivalently, π dominates μ, or $\pi \gg \mu$), when, for all $E \in \Sigma_X$, $\pi(E) = 0$ implies $\mu(E) = 0$. When μ is absolutely continuous with respect to π, then

$$\left\| \int_X k^n(x, \cdot)\mu(dx) - \pi(\cdot) \right\|_{\text{tv}} \to 0 \qquad \text{as } n \to \infty. \tag{1}$$

In this case, π is called the *stationary distribution*. The Markov chain is *uniformly ergodic* if there exist constants $C \in \mathbb{R}_+$, $\rho \in (0,1)$ such that

$$\sup_{x \in X} \|k^n(x, \cdot) - \pi(\cdot)\|_{\text{tv}} \leq C\rho^n. \tag{2}$$

3.2 The Metropolis–Hastings algorithm

The Metropolis–Hastings (MH) algorithm is a framework for constructing Markov chains with a specified ("target") stationary distribution. In particular, given a probability kernel Q from a measurable space (X, Σ_X) to itself, called the *proposal kernel*, the MH algorithm describes how to modify the proposal kernel in order to get a new kernel that defines a Markov chain with the desired stationary distribution. In particular, the *Metropolis–Hastings kernel* proposes to transition from state $x \in X$ by sampling a potential state transition y from $Q(x, \cdot)$. The kernel accepts the transition with probability $\alpha(x, y)$, where $\alpha : X \times X \to [0,1]$ is a carefully crafted measurable function. If the potential transition is rejected, the algorithm instead chooses the self-transition from x to itself. The Metropolis–Hastings kernel leaves π invariant if and only if, for all bounded measurable functions f,

$$\begin{aligned} &\iint f(x,y)\pi(dx)Q(x,dy) \\ &= \iint f(x,y)\pi(dy)Q(y,dx)\alpha(y,x). \end{aligned} \tag{3}$$

The above equation is known as *detailed balance*. It implies that the Markov chain is time reversible.

Theorem 3.1 (Radon–Nikodym for kernels). *Let (X, Σ_X) be a measurable space where Σ_X is Borel, $\mu : X \times \Sigma_X \to [0, \infty]$ be an σ-finite kernel, and $\nu : X \times \Sigma_X$ be a σ-finite kernel. If μ is absolutely continuous with respect to ν, then there exists a measurable function $f : X \times X \to [0, \infty)$ such that, for all $x \in X$ and $A \in \Sigma_X$,*

$$\mu(x, A) = \int_A f(x, y)\nu(dy).$$

Furthermore, f uniquely defined up to a ν-null set, i.e., if $\mu(x, A) = \int_A g\, d\nu$ for all $A \in \Sigma_X$, then $f = g$ ν-almost everywhere.

By the almost everywhere uniqueness, we are justified in referring to f as *the* Radon–Nikodym derivative of μ with respect to ν.

Theorem 3.2 (Tierney [15]). *For the Metropolis–Hastings algorithm with proposal kernel Q and target stationary distribution π, assume there exists a measure ν that dominates both π and $Q(x, \cdot)$, for all $x \in X$. Writing p and $q(x, \cdot)$ for the Radon–Nikodym derivatives of π and $Q(x, \cdot)$ with respect to ν. Let $r(x, y) = \frac{p(x)q(x,y)}{p(y)q(y,x)}$, let*

$$R = \{(x, y) : p(x)q(x, y) > 0 \text{ and } p(y)q(y, x) > 0\},$$

and define

$$\alpha(x, y) = \begin{cases} \min\{1, r(y, x)\}, & \text{if } (x, y) \in R, \\ 0, & \text{otherwise.} \end{cases}$$

Then the Metropolis–Hastings algorithm satisfies Equation (3).

4 Language Syntax and Semantics

In this section, we present an idealized first-order probabilistic language with a construct **stat** that takes as input a transition kernel for a Markov chain on some state space and returns the stationary distribution associated with the Markov chain. The language we present is based on the first-order probabilistic language introduced and studied by Staton et al. [14] and Staton [13], which has constructs for sampling, soft constraints, and normalization. The key differences, which we highlight again below, are (i) a syntactic distinction between probabilistic terms with and without soft constraints, which affects also typing, (ii) error cases in the denotation of **norm**, and (iii) the introduction of the new construct, **stat**.

Types:

$$\mathbb{A}_0, \mathbb{A}_1 ::= \mathbb{R} \mid \mathbf{1} \mid \mathrm{P}(\mathbb{A}) \mid \mathbb{A}_0 \times \mathbb{A}_1 \mid \sum_{i\in\mathbb{N}} \mathbb{A}_i$$

Terms:

deterministic:

$$a_0, a_1 ::= x \mid * \mid (a_0, a_1) \mid (i, a) \mid \pi_j(a) \mid f(a)$$
$$\mid \textbf{case } a \textbf{ of } \{(i, x) \Rightarrow a_i\}_{i\in I}$$

purely probabilistic:

$$t_0, t_1 ::= \textbf{sample}(a) \mid \textbf{return}(a) \mid \textbf{let } x = t_0 \textbf{ in } t_1$$
$$\mid \textbf{case } a \textbf{ of } \{(i, x) \Rightarrow t_i\}_{i\in I}$$
$$\mid \textbf{stat}(t_0, \lambda x.t_1) \mid \textbf{norm}(v)$$

probabilistic:

$$v_0, v_1 ::= t \mid \textbf{let } x = v_0 \textbf{ in } v_1$$
$$\mid \textbf{case } a \textbf{ of } \{(i, x) \Rightarrow v_i\}_{i\in I}$$
$$\mid \textbf{score}(a)$$

Program:

t is a program if t is purely probabilistic with no free variables

Figure 1. Syntax for the probabilistic language: $\mathcal{L}_{\mathbf{norm}}$

4.1 Probabilistic language with construct for stationary

We begin with types and syntax of the language, presented in Figure 1. For the remainder of this paper, we call this language $\mathcal{L}_{\mathbf{norm}}$.

Types

We study a typed probabilistic programming language where every term in the language has a type generated by the following grammar:

$$\mathbb{A}_0, \mathbb{A}_1 ::= \mathbb{R} \mid \mathbf{1} \mid \mathrm{P}(\mathbb{A}) \mid \mathbb{A}_0 \times \mathbb{A}_1 \mid \sum_{i\in I} \mathbb{A}_i,$$

where I is some countable, non-empty set. The language has a standard unit type, product types, and sum types. Denotationally, types are interpreted as measurable spaces, i.e., each type $\mathbb{A}$ represents some set $[\![\mathbb{A}]\!]$ coupled with a σ-algebra $\Sigma_{[\![\mathbb{A}]\!]}$ associated with the set $[\![\mathbb{A}]\!]$. We'll use $[\![\mathbb{A}]\!]$ as shorthand for the measurable space $([\![\mathbb{A}]\!], \Sigma_{[\![\mathbb{A}]\!]})$. The language has a special type $\mathbb{R}$ for real numbers and, for each type $\mathbb{A}$, a type $P(\mathbb{A})$ for probability distributions on (the measurable space denoted by) $\mathbb{A}$.

Now we give the space and the σ-algebra associated to each type and type constructor:

- For the type $\mathbb{R}$, the underlying space $[\![\mathbb{R}]\!]$ is the space of real numbers; $\Sigma_{[\![\mathbb{R}]\!]}$ is the (standard Borel) σ-algebra generated by the set of open subsets on the real line.
- For the unit type $\mathbf{1}$, the underlying space is $\{()\}$; $\Sigma_{[\![\mathbf{1}]\!]} = \{\{()\}, \varnothing\}$ is the corresponding σ-algebra.
- For the type $\mathrm{P}(\mathbb{A})$, the underlying space is the set of probability measure on $[\![A]\!]$, denoted $\mathcal{M}([\![\mathbb{A}]\!])$; $\Sigma_{\mathcal{M}([\![\mathbb{A}]\!])}$ is the the σ-algebra generated by sets $\{\mu \mid \mu(A) \le r\}$ for all $A \in \Sigma_{[\![\mathbb{A}]\!]}$ and $r \in [0, 1]$.
- For the product type $\mathbb{A} \times \mathbb{B}$, the underlying space is the product space $[\![\mathbb{A} \times \mathbb{B}]\!] = [\![\mathbb{A}]\!] \times [\![\mathbb{B}]\!]$; $\Sigma_{[\![\mathbb{A}\times\mathbb{B}]\!]}$ is the (product) σ-algebra generated by the rectangles $U \times V$, for $U \in \Sigma_{[\![\mathbb{A}]\!]}$ and $V \in \Sigma_{[\![\mathbb{B}]\!]}$.
- For the sum type $\sum_{i\in I} \mathbb{A}_i$, the underlying space is the co-product space $[\![\sum_{i\in I} \mathbb{A}_i]\!] = \biguplus_{i\in I} [\![\mathbb{A}_i]\!]$; informally, $\Sigma_{[\![\sum_{i\in I} \mathbb{A}_i]\!]}$ is the σ-algebra generated by the sets of the form $\{(i, a) \mid a \in U\}$ for $U \in \Sigma_{[\![\mathbb{A}_i]\!]}$.

Terms

As in Staton et al. [14], each term in the language is either *deterministic* or *probabilistic*, satisfying typing judgments of the form $\Gamma \vdash_d t : \mathbb{A}$ and $\Gamma \vdash_p t : \mathbb{A}$, respectively, given some environment/context $\Gamma = (x_1 : \mathbb{A}_1, ..., x_n : \mathbb{A}_n)$. Letting $[\![\Gamma]\!] = \prod_{i=1}^n [\![\mathbb{A}_i]\!]$, a deterministic term denotes a measurable function from the environment $[\![\Gamma]\!]$ to $[\![\mathbb{A}]\!]$. As in Staton [13], a probabilistic term denotes an S-finite kernel from $[\![\Gamma]\!]$ to $[\![\mathbb{A}]\!]$. Different from Staton [13], Staton et al. [14], we distinguish a subset of probabilistic terms we call *purely probabilistic*, which satisfy an additional typing judgment $\Gamma \vdash_{p1} t : \mathbb{A}$. A purely probabilistic term denotes a probability kernel from $[\![\Gamma]\!]$ to $[\![\mathbb{A}]\!]$. Departing again from Staton [13], Staton et al. [14], a *program* in our language is a purely probabilistic term with no free variables.

4.1.1 Variables, measurable functions, constructors, and destructors

As in Staton et al. [14], the language contains standard variables, constructors, and destructors, and constant terms f for all the measurable functions $f : [\![\mathbb{A}_0]\!] \to [\![\mathbb{A}_1]\!]$. The typing rules and semantics are unchanged and reproduced here for completeness:

$$\frac{}{\Gamma, x : \mathbb{A}, \Gamma' \vdash_d x : \mathbb{A}} \qquad \frac{}{\Gamma \vdash_d () : \mathbf{1}}$$

$$\frac{\Gamma \vdash_d t : \mathbb{A}_i}{\Gamma \vdash_d (i, t) : \sum_{j\in I} \mathbb{A}_j} \ (i \in I)$$

$$\frac{\Gamma \vdash_d t : \sum_{i\in I} \mathbb{A}_i \quad (\Gamma, x : \mathbb{A}_i \vdash_z u_i : \mathbb{B})_{i\in I}}{\Gamma \vdash_z \textbf{case } t \textbf{ of } \{(i, x) \Rightarrow u_i\}_{i\in I} : \mathbb{B}} \ (z \in \{d, p, p1\})$$

$$\frac{\Gamma \vdash_d t_0 : \mathbb{A}_0 \quad \Gamma \vdash_d t_1 : \mathbb{A}_1}{\Gamma \vdash_d (t_0, t_1) : \mathbb{A}_0 \times \mathbb{A}_1} \qquad \frac{\Gamma \vdash_d t : \mathbb{A}_0}{\Gamma \vdash_d f(t) : \mathbb{A}_1}$$

$$\frac{\Gamma \vdash_d t : \mathbb{A}_0 \times \mathbb{A}_1}{\Gamma \vdash_d \pi_j(t) : \mathbb{A}_j} \ (j \in \{0, 1\}).$$

$$[\![x]\!]_{\gamma,d,\gamma'} \stackrel{\text{def}}{=} d, \quad [\![()]\!]_\gamma \stackrel{\text{def}}{=} (), \quad [\![(i,t)]\!]_\gamma \stackrel{\text{def}}{=} (i, [\![t]\!]_\gamma),$$
$$[\![(t_0,t_1)]\!]_\gamma \stackrel{\text{def}}{=} ([\![t_0]\!]_\gamma, [\![t_1]\!]_\gamma), \quad [\![f(t)]\!]_\gamma \stackrel{\text{def}}{=} f([\![t]\!]_\gamma),$$
$$[\![\pi_j(t)]\!]_\gamma \stackrel{\text{def}}{=} d_j \text{ if } [\![t]\!]_\gamma \stackrel{\text{def}}{=} (d_0,d_1).$$

For case statements, if the kind judgement gives a deterministic term, the semantics is

$$[\![\textbf{case } t \textbf{ of } \{(i,x) \Rightarrow u_i\}_{i\in I}]\!]_\gamma \stackrel{\text{def}}{=} [\![u_i]\!]_{\gamma,d} \text{ if } [\![t]\!]_\gamma = (i,d).$$

If the kind judgement gives a probabilistic term, the term denotes a S-finite kernel and the semantics is

$$[\![\textbf{case } t \textbf{ of } \{(i,x) \Rightarrow u_i\}_{i\in I}]\!]_{\gamma,A} \stackrel{\text{def}}{=} [\![u_i]\!]_{\gamma,d,A} \text{ if } [\![t]\!]_\gamma = (i,d),$$

where $A \in \Sigma_{[\![\mathbb{B}]\!]}$.

4.1.2 Sequencing and sampling terms

In addition to standard **let** statements and **return** statements for sequencing, the language has a construct for producing a random sample from a probability distribution. The typing rules are as follows:

$$\frac{\Gamma \vdash_{z_1} t_1 : \mathbb{A} \quad \Gamma, x:\mathbb{A} \vdash_{z_2} t_2 : \mathbb{B}}{\Gamma \vdash_{z_3} \textbf{let } x = t_1 \textbf{ in } t_2 : \mathbb{B}},$$

where $z_3 = \begin{cases} p1 & \text{if } z_1 = p1 \wedge z_2 = p1 \\ p & \text{if } z_1 = p \vee z_2 = p \end{cases}$.

$$\frac{\Gamma \vdash_d t : \mathbb{A}}{\Gamma \vdash_{p1} \textbf{return}(t) : \mathbb{A}} \qquad \frac{\Gamma \vdash_d t : \mathrm{P}(\mathbb{A})}{\Gamma \vdash_{p1} \textbf{sample}(t) : \mathbb{A}}$$

Now all the terms for sequencing are judged as probabilistic terms, where **sample** and **return** are in fact probabilistic measure one terms. Also, if both t_1 and t_2 of the **let** statements are judged as purely probabilistic terms then so is the **let** statement.

As in Staton [13], the semantics of the **let** construct is defined in terms of integration as follows:

$$[\![\textbf{let } x = t_1 \textbf{ in } t_2]\!]_{\gamma,A} \stackrel{\text{def}}{=} \int_{[\![\mathbb{A}]\!]} [\![t_2]\!]_{\gamma,x,A} [\![t_1]\!]_{\gamma,dx}$$

Since both t_1 and t_2 are probabilistic terms, both are interpreted as S-finite kernels; The category of S-finite kernels is closed under composition, thus the term **let** $x = t_1$ **in** t_2 is also interpreted as an S-finite kernel.

The semantics of the **return** statement is given by the kernel $[\![\textbf{return}(t)]\!] \colon [\![\Gamma]\!] \times \Sigma_{[\![\mathbb{A}]\!]} \to [0,1]$

$$[\![\textbf{return}(t)]\!]_{\gamma,A} \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } [\![t]\!]_\gamma \in A \\ 0 & \text{otherwise} \end{cases}$$

Finally the **sample** statement takes in as argument a deterministic term of type $\mathrm{P}(\mathbb{A})$ that is it takes in as argument a probability measure on the space $[\![\mathbb{A}]\!]$. Thus the semantics is given as:

$$[\![\textbf{sample}(t)]\!]_{\gamma,A} = [\![t]\!]_{\gamma,A},$$

where $[\![t]\!]_\gamma \in \mathcal{M}([\![\mathbb{A}]\!])$.

4.1.3 Soft constraints and normalization terms

We are studying a probabilistic language for Bayesian inference; We have terms in the language that is used to scale the prior by the likelihood of some observed data and a term that re-normalizes the scaled measure to return the posterior distribution over the return type. The constructs **score** and **norm** are the constructs that, respectively, scale the prior program, and normalize the program to return the posterior probability distribution on the output type, if there exists one.

$$\frac{\Gamma \vdash_d t : \mathbb{R}}{\Gamma \vdash_p \textbf{score}(t) : \mathbf{1}} \qquad \frac{\Gamma \vdash_p t : \mathbb{A}}{\Gamma \vdash_{p1} \textbf{norm}(t) : (\mathbb{A} + \mathbf{1})}.$$

The semantics for the **score** construct are given by a S-finite kernel on the **1** type as follows:

$$[\![\textbf{score}(a)]\!]_{\gamma,A} = \begin{cases} \left|[\![t]\!]_\gamma\right| & \text{if } A = \{()\} \\ 0 & \text{otherwise} \end{cases}$$

The main difference between this semantics and the denotational semantics of the language proposed in Staton [13], Staton et al. [14] is in the semantics of **norm**. We interpret the semantics of **norm** terms as a probability kernel on the sum space given as $[\![\textbf{norm}(t)]\!] \colon [\![\Gamma]\!] \times \Sigma_{\mathbb{A}+1} \to [0,1]$ defined as

$$[\![\textbf{norm}(t)]\!]_{\gamma,A} = \begin{cases} \frac{[\![t]\!]_{\gamma,\{u \mid (0,u)\in A\}}}{[\![t]\!]_{\gamma,[\![\mathbb{A}]\!]}} & \text{if } [\![t]\!]_{\gamma,[\![\mathbb{A}]\!]} \in (0,\infty) \\ 0 & \text{else if } (1,()) \notin A \\ 1 & \text{else if } (1,())(1,()) \in A \end{cases},$$

where $A \in [\![\mathbb{A}+1]\!]$. The key distinction is that we are not able to determine if the term $[\![t]\!]_\gamma$ is an infinite measure or a null measure.

4.1.4 Stationary terms

One of the main contributions of this paper is that we propose a new feature in the probabilistic language that takes as argument an initial distribution and a Markov chain transition kernel on some state space, and returns the corresponding stationary distribution that leaves the kernel invariant. We allow the users to define a transition kernel on some measurable space using a standard lambda expression. The following is the syntax and typing rules for the stationary term:

$$\frac{\Gamma \vdash_{p1} t_0 : \mathbb{A} \quad \Gamma, x:\mathbb{A} \vdash_{p1} t_1 : \mathbb{A}}{\Gamma \vdash_{p1} \textbf{stat}(t_0, \lambda x.t_1) : \mathbb{A} + 1}.$$

First note that $[\![t_0]\!]_\gamma$ represents the initial distribution and $[\![t_1]\!]_\gamma$ represents the transition kernel for the Markov

chain. The denotational semantics of $[\![\mathbf{stat}(t_0, \lambda x.t_1)]\!] : [\![\Gamma]\!] \times \Sigma_{[\![\mathbb{A}+1]\!]} \to [0,1]$ is given as:

$$[\![\mathbf{stat}(t_0, \lambda x.t_1)]\!]_{\gamma,A} = \\ = \begin{cases} \mu(\{u : (0,u) \in A\}) & \text{if } \exists! \mu \in \mathcal{M}([\![\mathbb{A}]\!]) : \\ & \int_{[\![\mathbb{A}]\!]} [\![t_1]\!]^{(n)}(x,\cdot) \to_n \mu \\ & \text{for } x \text{ a.e. } [\![t_0]\!]_\gamma . \\ \delta_{(1,())}(A) & \text{otherwise} \end{cases}$$

The function $[\![\mathbf{stat}(t_0, \lambda x.t_1)]\!]$ is jointly measurable. See Appendix A for more details.

5 Eliminability of soft constraints and normalization terms

We introduced a new language construct **stat**, which takes a Markov chain transition kernel and returns the corresponding stationary distribution, and added this construct to the language proposed in Staton et al. [14]. In this section, we compare the original language and our extension, using the framework of *expressibility* due to Felleisen [5]. In particular, we show that the soft constraint and normalization terms, presented in Section 4.1.3, are *eliminable* from the language $\mathcal{L}_{\mathbf{norm}}$.

5.1 Expressibility of programming languages

In this section, we summarize the framework of relative expressibility of programming languages. For a detailed account, see Felleisen [5]. We begin by giving a formal definition of a programming language and conservative restrictions:

Definition 5.1 (Programming Language)**.** *A programming language $\mathcal{L}$ consists of*

- *a set of $\mathcal{L}$-phrases generated from a pre-specified syntax grammar, which consists of possibly infinite number of function symbols $\mathbb{F}_1, \mathbb{F}_2, \ldots$ with arities $a_1, a2, \ldots$ respectively;*
- *a set of $\mathcal{L}$-programs is a non-empty subset of $\mathcal{L}$-phrases;*
- *a semantics to the terms of the language.*

A programming language $\mathcal{L}'$ is a conservative restriction of a language $\mathcal{L}$ if

- *the set of constructors of the language $\mathcal{L}'$ is a subset of constructors of $\mathcal{L}$, i.e., there exists a set $\{\mathbb{F}_1, \mathbb{F}_2, \ldots\}$ that is subset of the constructors of the language $\mathcal{L}$ but not in the language $\mathcal{L}'$.*
- *the set of $\mathcal{L}'$-phrases is the full subset of $\mathcal{L}$-phrases that do not contain any constructs in $\{\mathbb{F}_1, \ldots, \mathbb{F}_n, \ldots\}$;*
- *the set of $\mathcal{L}'$-programs is the full subset of $\mathcal{L}$-programs that do not contain any constructs in $\mathbb{F}_1, \ldots, \mathbb{F}_n, \ldots$;*
- *the semantics of $\mathcal{L}'$ is a restriction of $\mathcal{L}$'s semantics.*

Informally speaking, a language construct from the language is *eliminable* if we can find a *computable* function that: 1) maps every program in the original language to a program in its conservative restriction; 2) is "local" for all the language constructs that exists in the conservative restriction. We make this notion precise in the following definition:

Types:

$$\mathbb{A}_0, \mathbb{A}_1 ::= \mathbb{R} \mid \mathbf{1} \mid \mathrm{P}(\mathbb{A}) \mid \mathbb{A}_0 \times \mathbb{A}_1 \mid \sum_{i \in \mathbb{N}} \mathbb{A}_i$$

Terms:

deterministic:

$$a_0, a_1 ::= x \mid * \mid (a_0, a_1) \mid (i, a) \mid \pi_j(a) \mid f(a) \\ \mid \mathbf{case}\ a\ \mathbf{of}\ \{(i,x) \Rightarrow a_i\}_{i \in I}$$

purely probabilistic:

$$t_0, t_1 ::= \mathbf{sample}(a) \mid \mathbf{return}(a) \mid \mathbf{let}\ x = t_0\ \mathbf{in}\ t_1 \\ \mid \mathbf{case}\ a\ \mathbf{of}\ \{(i,x) \Rightarrow t_i\}_{i \in I} \\ \mid \mathbf{stat}(t_0, \lambda x.t_1)$$

Program:

t is a program if t is purely probabilistic with no free variables

Figure 2. Syntax for the probabilistic language: $\mathcal{L}_{\mathbf{stat}}$

Definition 5.2 (Eliminability of programming constructs)**.** *Let $\mathcal{L}$ be a programming language and let $A = \{\mathbb{F}_1, \ldots, \mathbb{F}_n, \ldots\}$ be a subset of its constructors such that $\mathcal{L}'$ is a conservative restriction of $\mathcal{L}$ without the constructors in A. The programming constructs $\mathbb{F}_i \in A$ are* eliminable *if there is a computable mapping Φ from $\mathcal{L}$-phrases to $\mathcal{L}'$-phrases such that:*

1. *$\Phi(t)$ is an $\mathcal{L}'$-program for all $\mathcal{L}$-programs t;*
2. *$\Phi(\mathbb{F}(t_1, \ldots, t_d)) = \mathbb{F}(\Phi(t_1), \ldots, \Phi(t_d))$ for any construct $\mathbb{F}$ of $\mathcal{L}'$, i.e., Φ is homomorphic in all constructs of $\mathcal{L}'$.*
3. *For all programs t, $[\![t]\!] = [\![\Phi(t)]\!]$.*

Note that the first two properties of eliminability (Definition 5.2) are syntactic, while the third property is semantic.

5.2 Eliminability of the constructs "norm" and "score"

Consider the conservative restriction of $\mathcal{L}_{\mathbf{norm}}$ obtained by removing the **norm** and **score** constructs. (We denote this restriction by $\mathcal{L}_{\mathbf{stat}}$, and provide its syntax in Figure 2.) The main result of this section shows that $\mathcal{L}_{\mathbf{norm}}$ and $\mathcal{L}_{\mathbf{stat}}$ are equally expressive:

Theorem 5.3 (Eliminability of **norm** and **score**)**.** *The constructs* **norm** *and* **score** *are eliminable from $\mathcal{L}_{\mathbf{norm}}$.*

To prove Theorem 5.3, we give an explicit compiler $\Phi : \mathcal{L}_{\mathbf{norm}} \to \mathcal{L}_{\mathbf{stat}}$ that meets the requirements of eliminability. This compiler is given in its entirety in Figure 3. The compiler can be viewed as a simplified version of the Trace-MH algorithm first described in Wingate et al. [16]. The primary difference is the proposal kernel.

$$
\begin{aligned}
&\textbf{Tracer}(t) := \textbf{match } t \textbf{ with}\\
&\quad \{\textbf{sample}(t) \mapsto \textbf{sample}(t),\ \textbf{return}(t) \mapsto \textbf{return}(t),\ \textbf{stat}(t_0, \lambda x.t_1) \mapsto \textbf{stat}(t_0, \lambda x.t_1),\\
&\quad \textbf{score}(t) \mapsto \textbf{score}(t),\ \textbf{norm}(t) \mapsto \textbf{norm}(t),\\
&\quad \textbf{let } x = t_0 \textbf{ in } t_1 \mapsto \begin{array}{l}\textbf{let } trace_x = \textbf{Tracer}(t_0) \textbf{ in}\\ \textbf{let } trace_y = \textbf{Tracer}(t_1[x \backslash \pi_{-1}(trace_x)]) \textbf{ in}\\ \textbf{return}(trace_x, trace_y)\end{array},\\
&\quad \textbf{case } a \textbf{ of } \{(i,x) \Rightarrow u_i\} \mapsto \textbf{case } a \textbf{ of } \{(i,x) \Rightarrow \textbf{Tracer}(u_i)\}\}
\end{aligned}
$$

(a) Tracer Transformation

$$
\begin{aligned}
&\textbf{Prior} \circ \textbf{Tracer}(t) := \textbf{match } t \textbf{ with}\\
&\quad \{\textbf{sample}(t) \mapsto \textbf{sample}(t),\ \textbf{return}(t) \mapsto \textbf{return}(t),\ \textbf{stat}(t_0, \lambda x.t_1) \mapsto \textbf{stat}(t_0, \lambda x.t_1),\\
&\quad \textbf{score}(t) \mapsto \textbf{return}(()),\ \textbf{norm}(t) \mapsto \textbf{norm}(t),\\
&\quad \textbf{let } x = t_0 \textbf{ in } t_1 \mapsto \begin{array}{l}\textbf{let } trace_x = \textbf{Prior} \circ \textbf{Tracer}(t_0) \textbf{ in}\\ \textbf{let } trace_y = \textbf{Prior} \circ \textbf{Tracer}(t_1[x \backslash \pi_{-1}(trace_x)]) \textbf{ in}\\ \textbf{return}(trace_x, trace_y)\end{array},\\
&\quad \textbf{case } a \textbf{ of } \{(i,x) \Rightarrow u_i\} \mapsto \textbf{case } a \textbf{ of } \{(i,x) \Rightarrow \textbf{Prior} \circ \textbf{Tracer}(u_i)\}\}
\end{aligned}
$$

(b) Prior Transformation

$$
\begin{aligned}
&\textbf{Lhd} \circ \textbf{Tracer}(t) := \textbf{match } t \textbf{ with}\\
&\quad \{\textbf{sample}(t) \mapsto 1, \textbf{return}(t) \mapsto 1,\ \textbf{stat}(t_0, \lambda x.t_1) \mapsto 1,\\
&\quad \textbf{score}(t) \mapsto |t|, \textbf{norm}(t) \mapsto 1\\
&\quad \textbf{let } x = t_0 \textbf{ in } t_1 \mapsto \textbf{Lhd} \circ \textbf{Tracer}(t_0) * \textbf{Lhd} \circ \textbf{Tracer}(t_1),\\
&\quad \textbf{case } a \textbf{ of } \{(i,x) \Rightarrow t_i\}_{i \in I} \mapsto \textbf{case } a \textbf{ of } \{(i,x) \Rightarrow \textbf{Lhd}(t_i)\}_{i \in I}\}
\end{aligned}
$$

(c) Likelihood Transformation

$$
\begin{aligned}
&\text{MH}(\textbf{norm}(t)) := \textbf{stat}\left(\textbf{Prior} \circ \textbf{Tracer}(t), \begin{array}{l}\lambda x.\textbf{let } x' = \textbf{Prior} \circ \textbf{Tracer}(t) \textbf{ in}\\ \quad \textbf{case sample}(\text{Bern}(\alpha(x,x'))) \textbf{ of } \{(0,T) \Rightarrow \textbf{return}(x'), (1,F) \Rightarrow \textbf{return}(x))\}\end{array}\right),\\
&\text{where } \alpha(x,x') = \begin{cases}\min\left\{1, \frac{\textbf{Lhd} \circ \textbf{Tracer}(t)(x')}{\textbf{Lhd} \circ \textbf{Tracer}(t)(x)}\right\} & \text{if } \textbf{Lhd} \circ \textbf{Tracer}(t)(x) > 0 \text{ and } \textbf{Lhd} \circ \textbf{Tracer}(t)(x') > 0\\ 0 & \text{otherwise}\end{cases},
\end{aligned}
$$

(d) Metropolis–Hastings Transformation

$$
\begin{aligned}
&\Phi(t) := \textbf{match } t \textbf{ with}\\
&\quad \{\textbf{sample}(t) \mapsto \textbf{sample}(t),\ \textbf{return}(t) \mapsto \textbf{return}(t),\ \textbf{stat}(t_0, \lambda x.t_1) \mapsto \textbf{stat}(t_0, \lambda x.t_1),\\
&\quad \textbf{norm}(t) \mapsto \pi_{-1} \circ \text{MH}(\textbf{norm}(t)), \textbf{let } x = t_1 \textbf{ in } t_2 \mapsto \textbf{let } x = \Phi(t_1) \textbf{ in } \Phi(t_2),\\
&\quad \textbf{case } a \textbf{ of } \{(i,x) \Rightarrow t_i\}_{i \in I} \mapsto \textbf{case } a \textbf{ of } \{(i,x) \Rightarrow \Phi(t_i)\}_{i \in I}\},
\end{aligned}
$$

(e) Compiler Transformation

Figure 3. Here we give a formal description of the compiler Φ that maps programs in language $\mathcal{L}_{\textbf{norm}}$ to the programs in language $\mathcal{L}_{\textbf{stat}}$. In Figure 3a we give the **Tracer** transformation that modifies the sequencing term to give a joint distribution rather than marginalizing out the intermediate terms. The projection function, π_{-1}, returns the last element of a tuple. In Figure 3b we give a transformation that given a $\mathcal{L}_{\textbf{norm}}$-term returns the prior term associated with it. In Figure 3c we give a description of the function that for a term t, represents Radon–Nikodym derivative of **Tracer**(t) with respect to the **Prior** $\circ$ **Tracer**(t). In Figure 3d we give *independence Trace-MH* algorithm. In Figure 3e we finally give the complete description of the compiler.

First, we show that this compiler maps every program in $\mathcal{L}_{\text{norm}}$ to a program in $\mathcal{L}_{\text{stat}}$ and is homomorphic in all the features of $\mathcal{L}_{\text{stat}}$.

Lemma 5.4. *Let Φ be the compiler described in Figure 3.*

1. *Φ maps every $\mathcal{L}_{\text{norm}}$-program to a $\mathcal{L}_{\text{stat}}$ program.*
2. *Φ is homomorphic in all constructs of $\mathcal{L}_{\text{stat}}$.*

We then show Φ preserves semantics:

Lemma 5.5. *Let Φ be the compiler described in Figure 3 and let t be a $\mathcal{L}_{\text{norm}}$-program. Then, under the semantics given in Section 4:*

$$\|[\![t]\!] - [\![\Phi(t)]\!]\|_{\text{tv}} = 0,$$

i.e., Φ preserves semantics.

Theorem 5.3 follows immediately from these two lemma. The proof of Lemma 5.4 can be found in Appendix B. The proof of Lemma 5.5 occupies the remainder of the section.

5.2.1 Compiler preserves semantics

We begin by introducing and analyzing two source-to-source transformations, **Tracer** and **Prior**, and one program transformation, **Lhd**, that produces a measurable function. (See Figure 3 for their definition.)

In Figure 3a, we define the *tracer* transformation, **Tracer**, which produces programs that return all probabilistic choices encountered in evaluation of the program as a tuple. The behavior of this transformation is straightforward in all cases other than sequences. On sequences, the **Tracer** transformation satisfies the following typing rule:

$$\frac{\Gamma \mid\!\!\frac{}{p}\ \mathbf{Tracer}(t_0) \colon \mathbb{A}_0 \quad \Gamma, x : \mathbb{A} \mid\!\!\frac{}{p}\ \mathbf{Tracer}(t_1) \colon \mathbb{A}_1}{\Gamma \mid\!\!\frac{}{p}\ \mathbf{Tracer}(\mathbf{let}\ x = t_0\ \mathbf{in}\ t_1) : \mathbb{A}_0 \times \mathbb{A}_1}.$$

Remark 5.6. *Let $\mu \colon X \rightsquigarrow Y$ and $\nu \colon X \times Y \rightsquigarrow Z$ be σ-finite kernels. For $x \in X$, $A \in \Sigma_Y$, and $B \in \Sigma_Z$, define*

$$(\mu \otimes \nu)(x, A \times B) = \int_A \int_B \mu(x, dy)\nu(x, y, dz).$$

By [8, Lem. 1.17], $(\mu \otimes \nu)$ is a σ-finite kernel. Therefore, while the semantics of a probabilistic term $\Gamma \mid\!\!\frac{}{p}\ t \colon \mathbb{A}$ is, in general, a S-finite kernel, the semantics of $[\![\mathbf{Tracer}(t)]\!]$ can be taken to be a σ-finite kernel.

In Figure 3b, we define the *prior* transformation, **Prior**, which removes all occurrences of **score** terms that are not enclosed within a **norm** term. The next result establishes the key property we require of the resulting semantics:

Lemma 5.7. *Let $\Gamma \mid\!\!\frac{}{p}\ \mathbf{Tracer}(t) : \mathbb{A}$ be a probabilistic term in the language $\mathcal{L}_{\text{norm}}$. If $\mathbf{Prior} \circ \mathbf{Tracer}$ is the program transformation give in Figure 3b, then $\Gamma \mid\!\!\frac{}{p1}\ \mathbf{Prior} \circ \mathbf{Tracer}(t) : \mathbb{A}$ is a purely probabilistic term such that, for all γ, $[\![\mathbf{Tracer}(t)]\!]_\gamma \ll [\![\mathbf{Prior} \circ \mathbf{Tracer}(t)]\!]_\gamma$, i.e.,*

$$[\![\mathbf{Prior} \circ \mathbf{Tracer}(t)]\!]_{\gamma,A} = 0 \Rightarrow [\![\mathbf{Tracer}(t)]\!]_{\gamma,A} = 0.$$

The proof can be found in Appendix C.

Finally, in Figure 3c, we define the likelihood transformation, **Lhd**, which produces a (deterministic) measurable function and satisfies the following typing rule:

$$\frac{\Gamma \mid\!\!\frac{}{p}\ \mathbf{Tracer}(t) \colon \mathbb{A}}{\Gamma \mid\!\!\frac{}{d}\ \mathbf{Lhd} \circ \mathbf{Tracer}(t) \colon \mathbb{A} \to \mathbb{R}_+}.$$

The following lemma establishes that the measurable function obtained by the likelihood transformation is the Radon–Nikodym derivative of (i) the semantics of tracer transformation with respect to (ii) the semantics of the prior transformation.

Lemma 5.8. *Let $\Gamma \mid\!\!\frac{}{p}\ t : \mathbb{A}$ and $\gamma \in [\![\Gamma]\!]$. Then $[\![\mathbf{Lhd} \circ \mathbf{Tracer}(t)]\!]_\gamma$ is the Radon–Nikodym derivative of $[\![\mathbf{Tracer}(t)]\!]_\gamma$ with respect to $[\![\mathbf{Prior} \circ \mathbf{Tracer}(t)]\!]_\gamma$.*

Proof. We proceed via induction on the grammar of probabilistic terms. The only interesting base case is $\mathbf{score}(t)$. First, notice that

$$[\![\mathbf{Lhd} \circ \mathbf{Tracer}(\mathbf{score}(t))]\!]_\gamma = \left|[\![t]\!]_\gamma\right|.$$

Then, for $A \in \Sigma_{[\![\mathbb{A}]\!]}$,

$$\begin{aligned}
[\![\mathbf{Tracer}(\mathbf{score}(t))]\!]_{\gamma,A} &= [\![\mathbf{score}(t)]\!]_{\gamma,A} \\
&= \int_A \left|[\![t]\!]_\gamma\right| [\![\mathbf{return}(())]\!]_{\gamma,dx} \\
&= \int_A \left|[\![t]\!]_\gamma\right| [\![\mathbf{Prior} \circ \mathbf{Tracer}(\mathbf{score}(t))]\!]_{\gamma,dx}
\end{aligned}$$

Thus, $[\![\mathbf{Lhd} \circ \mathbf{Tracer}(\mathbf{score}(t))]\!]_\gamma$ is the Radon–Nikodym derivative with respect to $\mathbf{Prior} \circ \mathbf{Tracer}(\mathbf{score}(t))_\gamma$.

For the inductive step, we first consider the **let** construct. Let $A \times B \in \Sigma_{[\![\mathbb{A}]\!]} = \Sigma_{[\![\mathbb{A}_0]\!] \times [\![\mathbb{A}_1]\!]8}$ be a rectangle. Then

$$\begin{aligned}
&[\![\mathbf{Tracer}(\mathbf{let}\ x = t_0\ \mathbf{in}\ t_1)]\!]_{\gamma,A\times B} \\
&= \int_{[\![\mathbb{A}_0]\!]} \int_{[\![\mathbb{A}_1]\!]} [\![\mathbf{return}(x', y')]\!]_{\gamma,x',y',A\times B} \\
&\qquad [\![\mathbf{Tracer}(t_1)]\!]_{\gamma,x',dy'} [\![\mathbf{Tracer}(t_0)]\!]_{\gamma,dx'} \\
&= \int_A \int_B [\![\mathbf{Tracer}(t_1)]\!]_{\gamma,x',dy'} [\![\mathbf{Tracer}(t_0)]\!]_{\gamma,dx'} \\
&= \int_A \int_B [\![\mathbf{Lhd} \circ \mathbf{Tracer}(t_0)(x')]\!]_{\gamma,x'} [\![\mathbf{Prior} \circ \mathbf{Tracer}(t_0)]\!]_{\gamma,dx'} \\
&\qquad [\![\mathbf{Lhd} \circ \mathbf{Tracer}(t_1)(y')]\!]_{\gamma,x',y'} [\![\mathbf{Prior} \circ \mathbf{Tracer}(t_1)]\!]_{\gamma,x',dy'} \\
&= \int_{A\times B} [\![\mathbf{Lhd} \circ \mathbf{Tracer}(\mathbf{let}\ x = t_0\ \mathbf{in}\ t_1)(x', y')]\!]_{\gamma,x',y'} \\
&\qquad [\![\mathbf{Prior} \circ \mathbf{Tracer}(\mathbf{let}\ x = t_0\ \mathbf{in}\ t_1)]\!]_{\gamma,dx'\times dy'}.
\end{aligned}$$

The third equality follows from the inductive hypothesis.

It remains to consider **case** statements:

$$
\begin{aligned}
&[\![\mathbf{Tracer}(\mathbf{case}\ a\ \mathbf{of}\ \{t_i\}_{i\in I})]\!]_{\gamma,A}\\
&= [\![\mathbf{case}\ a\ \mathbf{of}\ \{(i,x)\Rightarrow \mathbf{Tracer}(t_i)\}_{i\in I})]\!]_{\gamma,A}\\
&= [\![\mathbf{Tracer}(t_i)]\!]_{\gamma,v,A} \quad \text{if } [\![a]\!]_\gamma = (i,v)\\
&= \int_A [\![\mathbf{Lhd}\circ\mathbf{Tracer}(t_i)]\!]_{\gamma,v,x} \quad \text{if } [\![a]\!]_\gamma = (i,v)\\
&\qquad\quad [\![\mathbf{Prior}\circ\mathbf{Tracer}(t_i)]\!]_{\gamma,v,dx}\\
&= \int_A [\![\mathbf{Lhd}\circ\mathbf{Tracer}(\mathbf{case}\ a\ \mathbf{of}\ \{(i,x)\Rightarrow t_i\}_{i\in I})]\!]_{\gamma,x}\\
&\qquad\quad [\![\mathbf{Prior}\circ\mathbf{Tracer}(\mathbf{case}\ a\ \mathbf{of}\ \{(i,x)\Rightarrow t_i\}_{i\in I})]\!]_{\gamma,dx}.
\end{aligned}
$$

The third equality follows from the inductive hypothesis. □

Given the properties of the program transformations, we show in the following proposition that the **norm** terms can be compiled into a **stat** term by constructing the Metropolis–Hastings kernel with the correct stationary distribution.

Proposition 5.9. *Let* $\Gamma \vdash_{p1} \mathbf{norm}(t)$ *and* $\gamma \in [\![\Gamma]\!]$. *Then*

$$\left\| [\![\mathbf{norm}(t)]\!]_\gamma - [\![\pi_{-1}\circ \mathrm{MH}(\mathbf{norm}(t))]\!]_\gamma \right\|_{\mathrm{tv}} = 0,$$

i.e., the MH*-transformation preserves semantics.*

Proof. First, note that it is straightforward to verify that $[\![\mathbf{norm}(t)]\!]_\gamma = [\![\pi_{-1}\circ \mathbf{norm}(\mathbf{Tracer}(t))]\!]_\gamma$..

Further, $[\![\mathbf{norm}(\mathbf{Tracer}(t))]\!]$ is a probability kernel that maps a context to a measure on the sum type $[\![\mathbb{A}+1]\!]$. We can rewrite

$$
\begin{aligned}
[\![\mathbf{norm}(\mathbf{Tracer}(t))]\!] = {}& [\![\mathbf{norm}(\mathbf{Tracer}(t))]\!]|_{\{(0,u):u\in[\![\mathbb{A}]\!]\}} +\\
& [\![\mathbf{norm}(\mathbf{Tracer}(t))]\!]|_{\{(1,())\}}.
\end{aligned}
$$

We now show that

$$
[\![\mathbf{norm}(\mathbf{Tracer}(t))]\!] = \left[\!\!\left[\mathbf{stat}\left(\mathbf{Prior}\circ\mathbf{Tracer}(t),\ \begin{array}{r} \lambda x.\mathbf{let}\ x' = \mathbf{Prior}\circ\mathbf{Tracer}(t)\ \mathbf{in}\\ \mathbf{case}\ \mathbf{sample}(\mathrm{Bern}(\alpha(x,x')))\ \mathbf{of}\\ (0,T)\Rightarrow \mathbf{return}(x')\\ \mid (1,F)\Rightarrow \mathbf{return}(x)) \end{array} \right) \right]\!\!\right]
$$

Case 1: $[\![\mathbf{Tracer}(t)]\!]_{\gamma,[\![\mathbb{A}]\!]} \in \{0,\infty\}$:
This is the case when $[\![\mathbf{Tracer}(t)]\!]$ is either a null measure or an infinite measure. Recall that $\mathbf{norm}(\mathbf{Tracer}(t))$ satisfies

$$
\begin{aligned}
[\![\mathbf{norm}(\mathbf{Tracer}(t))]\!]_\gamma(A) &= [\![\mathbf{norm}(\mathbf{Tracer}(t))]\!]_\gamma\big|_{\{(1,())\}}(A)\\
&= \begin{cases} 1 & \text{if } (1,()) \in A\\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

When $[\![\mathbf{norm}(\mathbf{Tracer}(t))]\!]_\gamma$ is a null measure, $\alpha(x,x') = 0$. Thus, there is no *unique* probability measure that leaves the distribution invariant, thus **stat** returns a point measure on $(1,())$. When $[\![\mathbf{norm}(\mathbf{Tracer}(t))]\!]_\gamma$ is an infinite measure, the MH kernel is reversible with respect to the specified σ-finite measure. Since there is no *probability* measure that leaves this distribution invariant, **stat** returns a point measure on $(1,())$.

Case 2: $[\![t]\!]_{\gamma,[\![\mathbb{A}]\!]} \in (0,\infty)$
Semantically, the Markov transition kernel in the stationary term satisfies

$$
\begin{aligned}
&\left[\!\!\left[\begin{array}{l} \mathbf{let}\ x' = \mathbf{Prior}(\mathbf{Tracer}(t))\ \mathbf{in}\\ \mathbf{case}\ \mathbf{sample}(\mathrm{Bern}(\alpha(x,x')))\ \mathbf{of}\\ \quad (0,T)\Rightarrow \mathbf{return}(x')\\ \quad \mid (1,F)\Rightarrow \mathbf{return}(x)) \end{array} \right]\!\!\right]_{\gamma,v,dv'}\\
&\quad \alpha(v,v')[\![\mathbf{Prior}(t)]\!]_{\gamma,dv'} +\\
&= \quad \delta_v(dv')\int_{[\![\mathbb{A}]\!]}(1-\alpha(v,y))[\![\mathbf{Prior}(t)]\!]_\gamma(dy),
\end{aligned}
$$

where for

$$
R = \{(x,x') : \forall y \in \{x,x'\}\ \mathbf{Lhd}\circ\mathbf{Tracer}(t)(y) \in (0,\infty)\}
$$

$$
\alpha(x,x') \mapsto \begin{cases} \min\left\{1, \frac{\mathbf{Lhd}\circ\mathbf{Tracer}(t)(x')}{\mathbf{Lhd}\circ\mathbf{Tracer}(t)(x)}\right\} & \text{if } (x,x')\in R\\ 0 & \text{otherwise} \end{cases}
$$

Note that the Markov kernel constructed by the program transformation is exactly the Markov kernel associated with the Metropolis–Hastings algorithm discussed in Section 3.1. Here the proposal kernel is $[\![\mathbf{Prior}(\mathbf{Tracer}(t))]\!]_\gamma$. The acceptance probability $\alpha(x,x')$ satisfies the hypotheses of Theorem 3.2, and thus Metropolis–Hastings kernel is reversible with respect to $\frac{[\![\mathbf{Tracer}(t)]\!]_\gamma}{[\![\mathbf{Tracer}(t)]\!]_{\gamma,[\![\mathbb{A}]\!]}}$. □

Proof of Lemma 5.5. We proceed by induction on purely probabilistic terms, since a program is a purely probabilistic without free variables. The only interesting base case in the $\mathbf{norm}(t)$ term. Proposition 5.9 shows that Φ preserves semantics.

Case 1: **let** constructor:

$$
\begin{aligned}
[\![\Phi(\mathbf{let}\ x = t_0\ \mathbf{in}\ t_1)]\!]_\gamma &= [\![\mathbf{let}\ x = \Phi(t_0)\ \mathbf{in}\ \Phi(t_1)]\!]_\gamma\\
&= \int [\![\Phi(t_1)]\!]_{\gamma,x}[\![\Phi(t_0)]\!]_{\gamma,dx}\\
&= \int [\![t_1]\!]_{\gamma,x}[\![t_0]\!]_{\gamma,dx}\\
&= \int [\![\mathbf{let}\ x = t_0\ \mathbf{in}\ t_1)]\!]_{\gamma,dx},
\end{aligned}
$$

where third inequality is due to the inductive hypothesis.

Case 2: **case** constructor:

$$
\begin{aligned}
&[\![\Phi(\mathbf{case}\ a\ \mathbf{of}\ \{(i,x)\Rightarrow t_i\}]\!]_{\gamma,A} =\\
&\qquad = [\![\mathbf{case}\ a\ \mathbf{of}\ \{(i,x)\Rightarrow \Phi(t_i)\}]\!]_{\gamma,A}\\
&\qquad = [\![\Phi(t_i)]\!]_{\gamma,v,A} \quad \text{if } [\![a]\!]_\gamma = (i,v)\\
&\qquad = [\![t_i]\!]_{\gamma,v,A} \quad \text{if } [\![a]\!]_\gamma = (i,v)\\
&\qquad = [\![\mathbf{case}\ a\ \mathbf{of}\ \{(i,x)\Rightarrow t_i\}]\!]_{\gamma,A},
\end{aligned}
$$

where second inequality is due to the inductive hypothesis. □

6 Approximate compilation of probabilistic programs

Staton et al. [14] proposed a probabilistic programming language with constructs for **sample**, **score** and **norm** where the semantics of the language assumes an ideal implementations for the **norm** construct. As we saw in Section 4, to compute the purely probabilistic term $\Gamma \vdash_{p1} \mathbf{norm}(t) : \mathbb{A}$, we need to compute the normalization factor $[\![t]\!]_{\gamma,[\![\mathbb{A}]\!]}$; This is computationally intractable. In Section 4, we introduced a new programming construct **stat** that takes a description of an initial distribution and a transition kernel, and represents the limiting distribution, if there exists one, of the associated Markov chain. Theorem 5.3 tells us that for a language with **stat**, the constructs **norm** and **score** are eliminable. Even though computing the stationary distribution of a Markov chain is also computationally intractable, if the Markov chain specified is *ergodic*, we can iterate the Markov kernel starting from the specified initial distribution to approximate the stationary distribution. The error associated with this approximation depends on rate at which the Markov chain is converging.

In this section we concretely define this approximate compilation scheme based on iteration and state the assumption under which the Markov chain converges. A similar iteration scheme was also given in Atkinson et al. [2] to approximate the fixpoint of a Markov kernel. In Theorem 6.3 we show that if the Markov chains represented by the **stat** terms of a program in the language $\mathcal{L}_{\mathbf{stat}}$ were uniformly ergodic, then we can give a quantitative bound on the error associated with this approximate compilation scheme—even when we allow for "nested" **stat** terms.

6.1 Approximate implementation for stat terms

In this section we give a simple approximate compilation scheme for **stat** terms similar to the approximation scheme proposed by Atkinson et al. [2]. First, we inductively define syntactic sugar **Iterate** as follows:

$$\mathbf{Iterate}^0(t_0, \lambda x.t_1) := t_0$$

$$\mathbf{Iterate}^N(t_0, \lambda x.t_1) := \mathbf{let}\ x = \mathbf{Iterate}^{N-1}(t_0, \lambda x.t_1)\ \mathbf{in}\ t_1$$

The semantics of the approximate program transformation is given as:

$$[\![\mathbf{Iterate}^N(t_0, \lambda x.t_1)]\!]_\gamma = \int [\![t_1]\!]^N_{\gamma,x} [\![t_0]\!]_\gamma(dx)$$

Given these syntactic sugar, now we give the program transformation ϕ as follows:

$$\phi(\mathbf{stat}(t_0, \lambda x.t_1), n) \stackrel{\text{def}}{=} \mathbf{Iterate}^n(t_0, \lambda x.t_1)$$

We make the following assumption on the semantics of the Markov chain specified by the **stat** terms:

Assumption 1. *For a term $\Gamma \vdash_{p1} \mathbf{stat}(t_0, \lambda x.t_1) : \mathbb{A} + 1$ in the language, for all γ, the transition kernel, $[\![t_1]\!]_\gamma$, and the initial distribution, $[\![t_0]\!]_\gamma$, specifies an ergodic Markov chain with stationary distribution $\pi_\gamma \in \mathcal{M}[\![\mathbb{A}]\!]$ such that:*

$$\lim_{N\to\infty} \left\| \int_{[\![\mathbb{A}]\!]} [\![t_1]\!]^N_{\gamma,x}(\cdot) [\![t_0]\!]_{\gamma,dx} - \pi_\gamma(\cdot) \right\|_{\text{tv}} = 0$$

Remark 6.1. *Note that if Assumption 1 does not hold, i.e., for the* **stat** *term $\Gamma \vdash_{p1} \mathbf{stat}(t_0, \lambda x.t_1') : \mathbb{A} + 1$, there is some γ, such that $[\![t_1]\!]_\gamma$ is not ergodic, then $[\![\mathbf{stat}(t_0, \lambda x.t_1')]\!]_\gamma$ is a point measure on the $\{((1, ()))\}$. But the program transformation $[\![\phi(\mathbf{stat}(t_0, \lambda x.t_1)]\!]_\gamma$ is still a probability measure on $[\![\mathbb{A}]\!]$.*

Under Assumption 1, for a program with a single **stat** term, we know by Equation (1) that the iteration scheme asymptotically converges to the invariant distribution. But if we allow for possibly nested **stat** terms, there is no guarantee that the approximate compilation scheme converges to something meaningful. This problem is highlighted below.

Problem 1. *One of the main challenges when studying approximate implementation of the* **stat** *construct is that it is not continuous, i.e., for some term $\Gamma \vdash_{p1} \mathbf{stat}(t_0, \lambda x.t_1) : \mathbb{A} + 1$ if we know that $[\![t_1]\!]_{\gamma,x}$ is an ergodic kernel that has a unique stationary distribution, it is possible to construct an approximate implementation of the Markov transition kernel $\lambda x.t_1'$ such that*

$$\exists \delta \in (0,1) \forall \gamma, x. \left\| [\![t_1]\!]_{\gamma,x} - [\![t_1']\!]_{\gamma,x} \right\| \leq \delta,$$

but the Markov kernel $[\![t_1']\!]_{\gamma,x}$ does not have a stationary distribution. Such an example is given in Proposition 1 of Roberts et al. [11].

To side step the issue stated in Problem 1, we need to make further semantic restrictions on the Markov chains specified by the **stat** terms. We identify that if the transition kernel given to the **stat** construct is *uniformly ergodic*, then **stat** construct is continuous.

We define this semantic restriction as relation that is assumed to be true when we give the quantitative error bounds to probabilistic program.

Definition 6.2. *Let $R \subseteq \mathcal{M}(\mathbb{A})^{[\![\mathbb{A}]\!]} \times \mathcal{M}[\![\mathbb{A}]\!] \times \mathbb{R}_+ \times [0,1)$ be a relation such that $R(P, \pi, C, \rho)$ holds if and only if P is a uniformly ergodic Markov chain with stationary distribution π such that for all $x \in [\![\mathbb{A}]\!]$*

$$\left\| P^N(x, \cdot) - \pi(\cdot) \right\|_{\text{tv}} \leq C\rho^N$$

6.2 Quantitative error bounds

Here, we derive the quantitative error bounds associated with the approximation compilation scheme of probabilistic programs where each **stat** term specifies a Markov chain that is uniformly ergodic. The main theorem we prove in this section is the following:

Theorem 6.3 (Quantitative error bound for probabilistic programs)**.** *Let P be a probabilistic program in the language $\mathcal{L}_{\mathbf{stat}}$. Let $\{\mathbf{stat}(t_{0i}, \lambda x.t_{1i})\}_{i\in I}$ be the set of all stationary terms in the*

program $\varnothing \vdash_{p1} P : \mathbb{B}$ *such that for all* γ*, there exist constants* $\{C_i\}$ *and* $\{\rho_i\}$ *such that* $R([\![t_{1i}]\!]_\gamma, [\![\mathbf{stat}(t_{0i}, \lambda x.t_{1i})]\!]_\gamma, C_i, \rho_i)$ *holds. Let* P' *be a program where for all* $i \in I$ *and* $N_i \in \mathbb{N}$, $\mathbf{stat}(t_{0i}, \lambda x.t_{1i})$ *is replaced by* $\phi(\mathbf{stat}(t_{0i}, \lambda x.t_{1i}), N_i)$*. Then, there exist constants* $\{C_i'\}_{i\in I}$ *such that*

$$\left\|[\![P]\!]_\gamma - [\![P']\!]_\gamma\right\|_{\text{tv}} \leq \sum_{i\in I} C_i' \rho_i^{N_i}.$$

In the remainder of this section we prove the result above. We begin by first establishing the uniform continuity of **let** and **case** constructs.

Proposition 6.4. *The following statements hold:*

1. *Let* $\Gamma \vdash_{p1} t_0 : \mathbb{A}$, $\Gamma \vdash_{p1} t_0' : \mathbb{A}$, $\Gamma, x : \mathbb{A} \vdash_{p1} t_1 : \mathbb{B}$*, and* $\Gamma, x : \mathbb{A} \vdash_{p1} t_1' : \mathbb{B}$ *be purely probabilistic terms. If for all* $\gamma \in \Gamma$, $\left\|[\![t_0']\!]_\gamma - [\![t_0]\!]_\gamma\right\|_{\text{tv}} \leq \alpha$ *and for all* $[\![x]\!]_\gamma \in [\![\mathbb{A}]\!]$,

$$\left\|[\![t_1']\!]_\gamma([\![x]\!]_\gamma) - [\![t_1]\!]_\gamma([\![x]\!]_\gamma)\right\|_{\text{tv}} \leq \beta$$

then

$$\left\|[\![\mathbf{let}\ x = t_0\ \mathbf{in}\ t_1]\!]_\gamma - [\![\mathbf{let}\ x = t_0'\ \mathbf{in}\ t_1']\!]_\gamma\right\|_{\text{tv}} \leq \alpha + \beta$$

2. *Let* $\Gamma, x : \mathbb{A}_i \vdash_{p1} t_i : \mathbb{B}$ *and* $\Gamma, x : \mathbb{A}_i \vdash_{p1} t_i' : \mathbb{B}$*, such that*

$$\forall i \in I, \forall x \in [\![\mathbb{A}_i]\!], \left\|[\![t_i']\!]_{\gamma,x} - [\![t_i]\!]_{\gamma,x}\right\|_{\text{tv}} \leq \alpha_i$$

$$\left\|\begin{array}{l}[\![\mathbf{case}\ a\ \mathbf{in}\ \{(i,x) \Rightarrow t_i'\}_{i\in I}]\!]_\gamma - \\ \qquad [\![\mathbf{case}\ a\ \mathbf{in}\ \{(i,x) \Rightarrow t_i\}_{i\in I}]\!]_\gamma\end{array}\right\|_{\text{tv}} \leq \sup\{\alpha_i\}_{i\in I}$$

Proof. Proof in Appendix D □

The next theorem quantifies the error associated with the N step **Iterate** transformation of the **stat** term with an *approximate* transition kernel to the **stat** term with the *correct* transition kernel.

Theorem 6.5. *Let* $\Gamma, x \vdash_{p1} t_1$ *and* $\Gamma, x \vdash_{p1} t_1'$ *be probabilistic terms. If there exist* π*,* C*, and* ρ *such that* $R([\![t_1]\!]_\gamma, \pi, C, \rho)$ *holds and*

$$\left\|[\![t_1']\!]_\gamma - [\![t_1]\!]_\gamma\right\|_{\text{tv}} \leq \varepsilon,$$

then

$$\left\|[\![\mathbf{Iterate}^N(t_0, \lambda x.t_1')]\!] - [\![\mathbf{stat}(t_0, \lambda x.t_1)]\!]\right\|_{\text{tv}} \leq \frac{\varepsilon C}{1-\rho} + C\rho^N.$$

To prove the theorem above, we begin by first giving a simple contraction lemma that quantifies the total variation distance between the laws of the n^{th} random variable of Markov chains with different initial distribution but same transition kernel under the assumption of uniform ergodicity.

Lemma 6.6 (Contraction)**.** *Let* $\Gamma \vdash_{p1} t_0 : \mathbb{A}$ *and* $\Gamma, x : \mathbb{A} \vdash_{p1} t_1 : \mathbb{A}$ *be purely probabilistic terms. If these terms are such that* $R\left([\![t_1]\!]_\gamma, ([\![\mathbf{stat}(t_0, \lambda x.t_1)]\!]_\gamma), C, \rho\right)$ *holds, then for all*

$$\left\|\begin{array}{l}[\![\mathbf{Iterate}^N(m_1, \lambda x.t_1)]\!]_\gamma - \\ \qquad [\![\mathbf{Iterate}^N(m_2, \lambda x.t_1)]\!]_\gamma\end{array}\right\|_{\text{tv}} \leq C\rho^N \left\|[\![m_1]\!]_\gamma - [\![m_2]\!]_\gamma\right\|.$$

Proof. The proof of this lemma follows directly from linearity of integration. □

To prove Theorem 6.5, we now give following theorem that quantifies the distance between the semantics of the **Iterate** transformation when the transition kernels are close.

Lemma 6.7. *Let* $\Gamma, x \vdash_{p1} t_1$ *and* $\Gamma, x \vdash_{p1} t_1'$ *be probabilistic terms. If there exist* $\pi \in \mathcal{M}(\mathbb{A})$*,* $C \in \mathbb{R}_+$*, and* $\rho \in [0, 1)$ *such that* $R([\![t_1]\!]_\gamma, \pi, C, \rho)$ *and*

$$\left\|[\![t_1']\!]_\gamma - [\![t_1]\!]_\gamma\right\|_{\text{tv}} \leq \varepsilon$$

then,

$$\left\|[\![\mathbf{Iterate}^N(t_0, \lambda x.t_1')]\!] - [\![\mathbf{Iterate}^N(t_0, \lambda x.t_1)]\!]\right\|_{\text{tv}} \leq \frac{\varepsilon C}{1-\rho},$$

Proof. We show this by first noting that

$$\left\|[\![\mathbf{Iterate}^N(t_0, \lambda x.t_1))]\!]_\gamma - [\![\mathbf{Iterate}^N(t_0, \lambda x.t_1')]\!]_\gamma\right\|_{\text{tv}}$$

$$= \left\|\begin{array}{l}\displaystyle\sum_{i=0}^{N-1} [\![\mathbf{Iterate}^{N-i}(\mathbf{Iterate}^i(t_0, \lambda x.t_1'), \lambda x.t_1)]\!]_\gamma \\ \qquad - [\![\mathbf{Iterate}^{N-i-1}(\mathbf{Iterate}^{i+1}(t_0, \lambda x.t_1'), \lambda x.t_1)]\!]_\gamma\end{array}\right\|_{\text{tv}}$$

$$= \left\|\begin{array}{l}\displaystyle\sum_{i=0}^{N-1} [\![\mathbf{Iterate}^{N-i-1}(\mathbf{let}\ x = \mathbf{Iterate}^i(t_0, \lambda x.t_1')\ \mathbf{in}\ n, \lambda x.t_1))]\!] \\ \qquad - [\![\mathbf{Iterate}^{N-i-1}(\mathbf{Iterate}^{i+1}(t_0, \lambda x.t_1'), \lambda x.t_1)]\!]_\gamma\end{array}\right\|_{\text{tv}}$$

Applying the contraction lemma,

$$\leq \sum_{i=0}^{N-1} C\rho^{N-i-1} \left(\left\|\begin{array}{l}[\![\mathbf{let}\ x = \mathbf{Iterate}^i(t_0, \lambda x.t_1')\ \mathbf{in}\ n]\!]_\gamma \\ \qquad - [\![\mathbf{let}\ x = \mathbf{Iterate}^i(t_0, \lambda x.t_1')\ \mathbf{in}\ n']\!]_\gamma\end{array}\right\|_{\text{tv}}\right)$$

$$\leq \sum_{i=0}^{N-1} C\rho^i\varepsilon \leq \frac{\varepsilon C}{1-\rho}.$$

□

Proof of Theorem 6.5. Given the Lemma 6.7 and the assumption that $R([\![t_1]\!]_\gamma, \pi, C, \rho)$ holds, the result follows by the triangle inequality. □

Now we prove the main theorem of this section.

Proof of Theorem 6.3. We proceed by induction on probabilistic terms.

- *Base case:*
  Leaf node for the induction is the terms of the form $\mathbf{stat}(t_0', \lambda x.t_1')$. By the assumption of uniform ergodicity there exist C', ρ' such that the following holds

  $$[\![\mathbf{stat}(t_0, \lambda x.t_1)]\!] - [\![\mathbf{Iterate}^{N'}(t_0, \lambda x.t_1)]\!] \leq C'\rho'^{N'}.$$

  Thus the base case holds.
- *Inductive Step:*
  For the inductive step we show the hypothesis holds for all constructor of probabilistic terms in our language.
  - **case** terms: By the inductive hypothesis,

    $$\left\|[\![t_j']\!]_\gamma - [\![t_j']\!]_\gamma\right\|_{\mathrm{tv}} \leq \sum_{i\in I_j} C_i\rho^{N_i}.$$

    Now, we show

    $$\left\|\begin{array}{l}[\![\mathbf{case}\ a\ \mathbf{in}\ \{(j,x) \Rightarrow t_j'\}_{j\in J}]\!] - \\ \quad [\![\mathbf{case}\ a\ \mathbf{in}\ \{(j,x) \Rightarrow t_j\}_{j\in J}]\!]\end{array}\right\|_{\mathrm{tv}} \leq \sum_{i\in\bigcup_{j\in J} I_j} C_i'\rho_i^{N_i}$$

    From Proposition 6.4 and the inductive hypothesis, we know

    $$\left\|\begin{array}{l}[\![\mathbf{case}\ a\ \mathbf{in}\ \{(j,x) \Rightarrow t_j'\}_{j\in J}]\!] - \\ \quad - [\![\mathbf{case}\ a\ \mathbf{in}\ \{(j,x) \Rightarrow t_j\}_{j\in J}]\!]\end{array}\right\|_{\mathrm{tv}} \leq \sup_{j\in J}\sum_{i\in I_j} C_i\rho^{N_i} \leq \sum_{i\in\bigcup_j I_j} C_i\rho^{N_i}$$

  - **let** term: We need to show

    $$\left\|[\![\mathbf{let}\ x = t_1\ \mathbf{in}\ t_2]\!] - [\![\mathbf{let}\ x = t_1'\ \mathbf{in}\ t_2']\!]\right\|_{\mathrm{tv}} \leq \sum_{i\in I_1\cup I_2} C_i\rho_i^{N_i}$$

    This follows from inductive hypothesis and Proposition 6.4.
  - **stat** term: From Theorem 6.5 and inductive hypothesis it follows that

    $$\left\|\begin{array}{l}[\![\mathbf{stat}(t_0, \lambda x.t_1)]\!] - \\ \quad [\![\mathbf{Iterate}^{N'}(t_0, \lambda x.t_1')]\!]\end{array}\right\|_{\mathrm{tv}} \leq C'\rho'^{N_i} + \sum_i \frac{C'C_i}{1-\rho'}\rho_i^{N_i}.$$

□

7 Summary and Discussion

MCMC algorithms are workhorses for approximate inference in probabilistic models. MCMC algorithms are popular because they give us asymptotic convergence guarantees and are commonly used as "approximate" compilers in probabilistic programming languages. In this paper we proposed a language construct **stat** that allows us give a formal description for such compilers. We then gave a simple compiler description that, at its core, implements an MCMC algorithm. Typically quantifying the rate at which Markov chain, with a given transition kernel, converges is an open problem and the one we do not attempt to solve in this paper. We make a semantic assumption that the user using our language provides us with a description of the Markov kernel that converges uniformly to the corresponding target distribution. We show that under this uniform convergence property, we can derive rates at which the approximate compiler converges to the original program.

The assumption for uniform ergodicity is crucial for us to derive the quantitative bound. The main difficulty we found in relaxing the uniform ergodicity assumption is the fact that our language allows us to nest the **stat**. We leave as open problem if we can relax the uniform ergodicity assumption.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] Eric Atkinson, Cambridge Yang, and Michael Carbin. 2018. Verifying Handcoded Probabilistic Inference Procedures. *arXiv preprint arXiv:1805.01863* (2018).

[3] Johannes Borgström, Ugo Dal Lago, Andrew D. Gordon, and Marcin Szymczak. 2016. A Lambda-calculus Foundation for Universal Probabilistic Programming. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming (ICFP 2016)*. ACM, New York, NY, USA, 33–46. https://doi.org/10.1145/2951913.2951942

[4] Bob Carpenter, Andrew Gelman, Matthew Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. 2017. Stan: A Probabilistic Programming Language. *Journal of Statistical Software, Articles* 76, 1 (2017), 1–32. https://doi.org/10.18637/jss.v076.i01

[5] Matthias Felleisen. 1991. On the expressive power of programming languages. *Science of Computer Programming* 17, 1 (1991), 35 – 75. https://doi.org/10.1016/0167-6423(91)90036-W

[6] Noah D. Goodman, Vikash K. Mansinghka, Daniel Roy, Keith Bonawitz, and Joshua B. Tenenbaum. 2008. Church: A Language for Generative Models. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence (UAI'08)*. AUAI Press, Arlington, Virginia, United States, 220–229. http://dl.acm.org/citation.cfm?id=3023476.3023503

[7] Chung-Kil Hur, Aditya Nori, and Sriram Rajamani. 2015. A Provably Correct Sampler for Probabilistic Programs. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)* (foundations of software technology and theoretical computer science (fsttcs) ed.). Leibniz International Proceedings in Informatics. https://www.microsoft.com/en-us/research/publication/a-provably-correct-sampler-for-probabilistic-programs/

[8] Olav Kallenberg. 2017. *Random measures, theory and applications*. Vol. 77. Springer International Publishing. https://doi.org/10.1007/978-3-319-41598-7

[9] Felipe Medina-Aguayo, Daniel Rudolf, and Nikolaus Schweizer. 2019. Perturbation bounds for Monte Carlo within Metropolis via restricted approximations. *Stochastic Processes and their Applications* (2019). https://doi.org/10.1016/j.spa.2019.06.015

[10] Tom Rainforth. 2018. Nesting Probabilistic Programs. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence,*

UAI 2018, Monterey, California, USA, August 6-10, 2018. 249–258. http://auai.org/uai2018/proceedings/papers/92.pdf

[11] Gareth O. Roberts, Jeffrey S. Rosenthal, and Peter O. Schwartz. 1998. Convergence Properties of Perturbed Markov Chains. *Journal of Applied Probability* 35, 1 (1998), 1–11. http://www.jstor.org/stable/3215541

[12] Adam Ścibior, Ohad Kammar, Matthijs Vákár, Sam Staton, Hongseok Yang, Yufei Cai, Klaus Ostermann, Sean K. Moss, Chris Heunen, and Zoubin Ghahramani. 2017. Denotational Validation of Higher-order Bayesian Inference. *Proc. ACM Program. Lang.* 2, POPL, Article 60 (Dec. 2017), 29 pages. https://doi.org/10.1145/3158148

[13] Sam Staton. 2017. Commutative Semantics for Probabilistic Programming. In *Proceedings of the 26th European Symposium on Programming Languages and Systems - Volume 10201*. Springer-Verlag New York, Inc., New York, NY, USA, 855–879. https://doi.org/10.1007/978-3-662-54434-1_32

[14] Sam Staton, Frank Wood, Hongseok Yang, Chris Heunen, and Ohad Kammar. 2016. Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. In *2016 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, 1–10.

[15] Luke Tierney. 1998. A note on Metropolis-Hastings kernels for general state spaces. *Ann. Appl. Probab.* 8, 1 (02 1998), 1–9. https://doi.org/10.1214/aoap/1027961031

[16] David Wingate, Andreas Stuhlmueller, and Noah Goodman. 2011. Lightweight Implementations of Probabilistic Programming Languages Via Transformational Compilation. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research)*, Geoffrey Gordon, David Dunson, and Miroslav Dudík (Eds.), Vol. 15. PMLR, Fort Lauderdale, FL, USA, 770–778. http://proceedings.mlr.press/v15/wingate11a.html

A Measurability of stat

The map $[\![\mathbf{stat}(t_0, \lambda x.t_1)]\!] : [\![\Gamma]\!] \to \mathcal{M}([\![\mathbb{A}]\!])$ is measurable. We show this by constructing a test function ϕ_n as follows:

$$\phi_n(x,x') = \left\| [\![t_1]\!]^{(n)}(x,\cdot) - [\![t_1]\!]^{(n)}(x',\cdot) \right\|_{\mathrm{tv}}$$

such that $\exists! \mu \in \mathcal{M}([\![\mathbb{A}]\!]) : \int_{[\![\mathbb{A}]\!]} [\![t_1]\!]^{(n)}(x,\cdot) \to_n \mu$ for x a.e. $[\![t_0]\!]_\gamma$ if and only if $\int [\![t_1]\!]^{(n)}(x,\cdot)$ is Cauchy in the total variation metric and $\lim_{n\to\infty} \phi_n(x,x') = 0$ (x,x') a.e. $[\![t_0]\!]_\gamma \otimes [\![t_0]\!]_\gamma$.

B Proof of Lemma 5.4

We need show that the syntactic manipulations Φ defined in Figure 3 satisfy the following two properties:

1. $\Phi(t)$ is a $\mathcal{L}_{\mathbf{stat}}$-program for all $\mathcal{L}_{\mathbf{norm}}$-program t.
2. $\Phi(\mathbb{F}(a_1, ..., a_n)) = \mathbb{F}(\Phi(a_1), ..., \Phi(a_n))$ for all constructors in $\mathcal{L}_{\mathbf{stat}}$

Recall that a program in $\mathcal{L}_{\mathbf{norm}}$ and $\mathcal{L}_{\mathbf{stat}}$ is a purely probabilistic term with no free variables. To verify the first property, we need to make sure that for every program $\vdash_{p1} t : \mathbb{A}$, $\Phi(t)$ is a also a purely probabilistic term with no free variables. This is an easy fact to verify by inducting on purely probabilistic terms. You can also witness this fact by noticing that every term in $\mathcal{L}_{\mathbf{stat}}$ is purely probabilistic and the compiler does not introduce any free variables.

For the second property, first recall compiler does not modify any deterministic term and that the set of probabilistic constructs in $\mathcal{L}_{\mathbf{stat}}$ are as follows:

$$\mathcal{F} = \{\mathbf{sample}, \mathbf{return}, \mathbf{let}\ , \mathbf{case}\ , \mathbf{stat}\}\ .$$

By examining the the compiler transformation for all of these terms in Figure 3, we notice that the compiler is homomorphic in all constructs of $\mathcal{L}_{\mathbf{stat}}$.

C Proof of Lemma 5.7

The proof that $\mathbf{Prior}(t)$ is a purely probabilistic term can be witnessed by the fact that **Prior** transforms all **score** statements to **return** statements. Since probabilistic terms are closed under composition, it is a simple exercise to show that $\mathbf{Prior}(t)$ is purely probabilistic. We now show that for all γ

$$[\![\mathbf{Prior}(t)]\!]_{\gamma,A} = 0 \Rightarrow [\![t]\!]_{\gamma,A} = 0$$

This is shown by induction on terms. For the base cases the only change is in the **score**-term where for some arbitrary γ,

$$[\![\mathbf{Prior}(\mathbf{score}(t))]\!]_{\gamma,A} = [\![\mathbf{return}(())]\!]_{\gamma,A} = \begin{cases} 1 & \text{if } () \in A \\ 0 & \text{if } () \notin A \end{cases}$$

$$[\![\mathbf{score}(t)]\!]_{\gamma,A} = \begin{cases} \left|[\![t]\!]_\gamma\right| & \text{if } () \in A \\ 0 & \text{if } () \notin A. \end{cases}$$

Thus, if $[\![\mathbf{Prior}(\mathbf{score}(t))]\!]_{\gamma,A} = 0$ implies $() \notin A$ which means $[\![\mathbf{score}(t)]\!]_{\gamma,A} = 0$. Hence we have shown $[\![\mathbf{score}(t)]\!]_\gamma \ll [\![\mathbf{Prior}(\mathbf{score}(t))]\!]_\gamma$. This finishes the bases cases for the induction.

For the inductive step, we need to show that **let** terms and **case** terms is absolutely continuous with respect to the prior transformation for **let** terms and **case** terms respectively.

First we prove the let statements. For $A \in [\![\mathbb{A}_1]\!]$,

$$\begin{aligned} &[\![\mathbf{Prior}(\mathbf{let}\ x = t_0\ \mathbf{in}\ t_1)]\!]_{\gamma,A} \\ &= [\![\mathbf{let}\ x = \mathbf{Prior}(t_0)\ \mathbf{in}\ \mathbf{Prior}(t_1)]\!]_{\gamma,A} \\ &= \int_A \int_{[\![\mathbb{A}_0]\!]} [\![\mathbf{Prior}(t_1)]\!]_{\gamma,x,dy} [\![\mathbf{Prior}(t_0)]\!]_{\gamma,dx} \end{aligned}$$

By IH, $\left\{[\![t_i]\!]_\gamma \ll [\![\mathbf{Prior}(t_i)]\!]_\gamma\right\}_{i\in\{0,1\}}$. Theorem 3.1 guarantees us that there exist measurable functions $f_0 : [\![\Gamma]\!] \times [\![\mathbb{A}_0]\!] \to [0,\infty]$, and $f_1 : [\![\Gamma]\!] \times [\![\mathbb{A}_0]\!] \times [\![\mathbb{A}_1]\!] \to [0,\infty]$ such that

$$[\![t_0]\!]_{\gamma,A} = \int_A f_0(\gamma,x)[\![\mathbf{Prior}(t_0)]\!]_{\gamma,dx}, \text{ and}$$
$$[\![t_1]\!]_{\gamma,x,A} = \int_A f_1(\gamma,x,y)[\![\mathbf{Prior}(t_1)]\!]_{\gamma,dy}.$$

Now, let $A \in \Sigma_{[\![\mathbb{A}]\!]}$ be such that:

$$\int_A \int_{[\![\mathbb{A}_0]\!]} [\![\mathbf{Prior}(t_1)]\!]_{\gamma,x,dy} [\![\mathbf{Prior}(t_0)]\!]_{\gamma,dx} = 0$$

Also, we know

$$\begin{aligned} &[\![\mathbf{let}\ x = t_0\ \mathbf{in}\ t_1]\!]_{\gamma,A} \\ &= \int_A \int_{[\![\mathbb{A}_0]\!]} f_0(\gamma,x) f_1(\gamma,x,y) [\![\mathbf{Prior}(t_1)]\!]_{\gamma,x,dy} [\![\mathbf{Prior}(t_0)]\!]_{\gamma,dx}. \end{aligned}$$

Since the integration of any measurable function on a null set is 0; Thus the following statement holds:

$$\int_A \int_{[\![\mathbb{A}_0]\!]} f_0(\gamma,x) f_1(\gamma,x,y) [\![\mathbf{Prior}(t_1)]\!]_{\gamma,x,dy} [\![\mathbf{Prior}(t_0)]\!]_{\gamma,dx} = 0.$$

This concludes the proof for **let** statements. Now we show for the **case** statements:

$$\begin{aligned} &[\![\mathbf{Prior}(\mathbf{case}\ a\ \mathbf{of}\ \ \{(i,x) \Rightarrow t_i\})]\!]_{\gamma,A} \\ &= [\![\mathbf{case}\ a\ \mathbf{of}\ \ \{(i,x) \Rightarrow \mathbf{Prior}(t_i)\}]\!]_{\gamma,A} \\ &= [\![\mathbf{Prior}(t_i)]\!]_{\gamma,d,A} \quad \text{if } [\![a]\!]_\gamma = (i,d) \end{aligned}$$

We know,

$$\begin{aligned} &[\![\mathbf{case}\ a\ \mathbf{of}\ \ \{(i,x) \Rightarrow t_i\}]\!]_{\gamma,A} \\ &= [\![t_i]\!]_{\gamma,d,A} \quad \text{if } [\![a]\!]_\gamma = (i,d) \end{aligned}$$

But by IH, $[\![t_i]\!]_{\gamma,d} \ll [\![\mathbf{Prior}(t_i)]\!]_{\gamma,d}$. Thus,

$$\begin{aligned} &[\![\mathbf{case}\ a\ \mathbf{of}\ \ \{(i,x) \Rightarrow t_i\}]\!]_\gamma \\ &\ll [\![\mathbf{Prior}(\mathbf{case}\ a\ \mathbf{of}\ \ \{(i,x) \Rightarrow t_i\})]\!]_\gamma, \end{aligned}$$

completing the proof.

D Proof of Proposition 6.4

1. For the **let** construct:

$$
\begin{aligned}
&\left\| \int [\![t_1]\!]_{\gamma,x} [\![t_0]\!]_{\gamma,dx} - \int [\![t_1']\!]_{\gamma,x} [\![t_0']\!]_{\gamma,dx} \right\|_{\text{tv}} \\
\leq &\left\| \int [\![t_1]\!]_{\gamma,x} [\![t_0]\!]_{\gamma,dx} - \int [\![t_1']\!]_{\gamma,x} [\![t_0]\!]_{\gamma,dx} \right\|_{\text{tv}} \\
&\qquad + \left\| \int [\![t_1']\!]_{\gamma,x} [\![t_0]\!]_{\gamma,dx} - \int [\![t_1']\!]_{\gamma,x} [\![t_0']\!]_{\gamma,dx} \right\|_{\text{tv}} \\
= &\left\| \int \left([\![t_1]\!]_{\gamma,x} - [\![t_1']\!]_{\gamma,x} \right) [\![t_0]\!]_{\gamma,dx} \right\|_{\text{tv}} \\
&\quad + \sup_A \left| \int [\![t_1']\!]_\gamma(x, A) [\![t_0]\!]_{\gamma,dx} - \int [\![t_1']\!]_\gamma(x, A) [\![t_0']\!]_{\gamma,dx} \right| \\
\leq &\int \sup_{x'} \left\| [\![t_1]\!]_\gamma(x') - [\![t_1']\!]_\gamma(x') \right\|_{\text{tv}} [\![t_0]\!]_{\gamma,dx} \\
&\qquad + \sup_{f \leq 1} \left| \int f(x) [\![t_0]\!]_{\gamma,dx} - \int f(x) [\![t_0']\!]_{\gamma,dx} \right| \\
= &\int \beta [\![t_0]\!]_{\gamma,dx} + \alpha \\
= &\alpha + \beta
\end{aligned}
$$

2. For the **case** construct:

$$
\begin{aligned}
&\left\| [\![\textbf{case } a \textbf{ in } \left\{(i,x) \Rightarrow t_i'\right\}_{i \in I}]\!]_\gamma - \right. \\
&\qquad \left. [\![\textbf{case } a \textbf{ in } \{(i,x) \Rightarrow t_i\}_{i \in I}]\!]_\gamma \right\|_{\text{tv}} \\
&= \left\| [\![t_i']\!]_{v,\gamma} - [\![t_i]\!]_{v,\gamma} \right\|_{\text{tv}} && \text{if } (i,v) = [\![a]\!]_\gamma \\
&\leq \alpha_i && \text{if } (i,v) = [\![a]\!]_\gamma \\
&\leq \sup_i \{\alpha_i\}_{i \in I}
\end{aligned}
$$