
CHALMERS
UNIVERSITY OF TECHNOLOGY

# GradInf: Gradient Estimation as Probabilistic Inference


# GradInf: Gradient Estimation as Probabilistic Inference

GAURAV ARYA, Carnegie Mellon University, USA
MATHIEU HUOT, Massachusetts Institute of Technology, USA
MORITZ SCHAUER, Chalmers University of Technology & University of Gothenburg, Sweden
ALEXANDER K. LEW, Yale University, USA
FERAS A. SAAD, Carnegie Mellon University, USA

*Gradient estimation*—the task of computing the gradient of the expected value of a probabilistic program—has diverse applications in scientific computing, but is notoriously difficult because of issues such as high-dimensional integration, discrete random choices, and complex stochastic dependencies. This article introduces *gradient inference*, a new approach to developing sound and efficient gradient estimators for probabilistic programs. Gradient inference rests on a formal reduction from a gradient estimation problem to a closely related probabilistic inference problem, whose solution can be differentiated to obtain a gradient estimator. This inference problem is obtained by applying two powerful statistical operations—*coupling* and *factorization*—to the input probabilistic program. Our reduction lets us leverage the rich toolkit of probabilistic inference algorithms to design novel gradient estimators that extend and improve upon existing methods.

We introduce GradInf, a probabilistic programming system that facilitates the sound and automated implementation of gradient inference. GradInf is centered around programmable source-to-source transformations for coupling and factorizing higher-order probabilistic programs, whose soundness is proven in terms of a denotational semantics. Key to our development is the use of information-flow typing to allow random choices in a probabilistic program to be factored out and *partially evaluated*, which improves our ability to deploy sophisticated probabilistic inference algorithms. The resulting system offers practitioners a principled framework for designing gradient estimators. We apply GradInf to several challenging case studies, showing that it can express prominent gradient estimators from the literature and enables the construction of new state-of-the-art estimators that outperform the best existing baselines.

CCS Concepts: • **Mathematics of computing** → *Probabilistic algorithms*; *Probabilistic inference problems*; *Statistical software*; *Automatic differentiation*; • **Theory of computation** → *Probabilistic computation.*

Additional Key Words and Phrases: probabilistic programming, gradient estimation, couplings

**ACM Reference Format:**
Gaurav Arya, Mathieu Huot, Moritz Schauer, Alexander K. Lew, and Feras A. Saad. 2026. GradInf: Gradient Estimation as Probabilistic Inference. *Proc. ACM Program. Lang.* 10, PLDI, Article 243 (June 2026), 27 pages. https://doi.org/10.1145/3808321

## 1 Introduction

Suppose we are given a function $\mu : \mathbb{R}^n \to P(\mathbb{R})$ that maps a parameter vector $\theta \coloneqq (\theta_1, \ldots, \theta_n)$ to a probability distribution $\mu_\theta$ over the reals. The expectation of $\mu_\theta$ is given by

$$\mathbb{E}_{x\sim\mu_\theta}[x] = \int_{\mathbb{R}} x\mu_\theta(\mathrm{d}x) =: \ell(\theta). \tag{1.1}$$

Authors' Contact Information: Gaurav Arya, Carnegie Mellon University, Pittsburgh, USA, gauravar@cmu.edu; Mathieu Huot, Massachusetts Institute of Technology, Cambridge, USA, mhuot@mit.edu; Moritz Schauer, Chalmers University of Technology & University of Gothenburg, Gothenburg, Sweden, smoritz@chalmers.se; Alexander K. Lew, Yale University, New Haven, USA, alexander.lew@yale.edu; Feras A. Saad, Carnegie Mellon University, Pittsburgh, USA, fsaad@cmu.edu.


ACM 2475-1421/2026/6-ART243
https://doi.org/10.1145/3808321

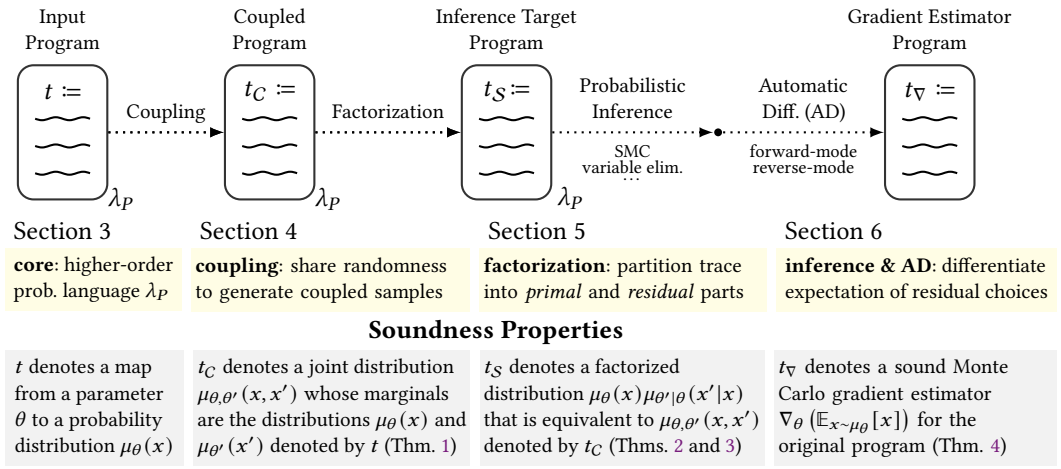



Fig. 1. Overview of gradient inference workflow using GradInf. The purpose of each component and its corresponding soundness criteria are summarized informally along the top and bottom rows, respectively.

*Gradient estimation* is the problem of computing the gradient $\nabla\ell(\theta):\mathbb{R}^n$ of the expectation of $\mu_\theta$, which is a vector whose $i$th coordinate is the partial derivative of $\ell$ with respect to $\theta_i$. Gradient estimation is a fundamental computational problem, with applications in diverse domains such as biochemical modeling [4], queuing theory [25], statistical epidemiology [19], particle detector simulations [40], reinforcement learning [80], neural networks [13], and large language models [76]. Use cases for gradient estimation include optimizing a stochastic objective function and assessing the sensitivity of a stochastic process to perturbations in its parameters.

### 1.1 The Challenge of Gradient Estimation

Computerized methods for differentiation such as symbolic differentiation [20] or automatic differentiation (AD) [11, 68] are given the source code of a target function $\ell:\mathbb{R}^n\to\mathbb{R}$ and return a new program for its gradient $\nabla\ell:\mathbb{R}^n\to\mathbb{R}^n$. The key challenge presented by gradient estimation is that our target function (1.1) is typically an intractable integral, rendering the direct application of standard symbolic manipulation rules infeasible. Moreover, the probability distribution $\mu_\theta$ being integrated may be specified by a probabilistic program that includes discrete randomness, stochastic control flow, and other challenging constructs that cause standard AD to fail [55].

The dominant approach to gradient estimation is grounded in Monte Carlo methods [25, 65], which construct a random vector $L(\theta)$ that estimates the gradient $\nabla\ell(\theta)$. An ideal Monte Carlo gradient estimator is *computationally efficient* (i.e., fast to compute), *sound* (i.e., unbiased), and *statistically efficient* (i.e., low variance, or equivalently low mean squared error):

$$\underline{\text{Unbiased}}:\ \mathbb{E}\left[L(\theta)\right]=\nabla\ell(\theta),\qquad \underline{\text{Low Variance}}:\ \mathbb{E}\left[||L(\theta)-\nabla\ell(\theta)||^2\right]\ \text{small}. \tag{1.2}$$

Over the decades, researchers from a range of fields have developed gradient estimators that aim to be sound and efficient. Examples include score function estimation [30], also known as REINFORCE [87]; infinitesimal perturbation analysis (IPA [29]), also known as the reparameterization trick [43]; smoothed perturbation analysis (SPA [31]); and measure-valued derivatives (MVD [36]). Practitioners, however, know that no single estimator works well (or even at all) in all settings. Developing an effective gradient estimator requires a careful understanding of the characteristics of the probability distribution $\mu_\theta$. But the diversity of existing estimation strategies and their varying mathematical formalisms make it difficult to interpret, apply, combine, or extend these methods.

Table 1. Gradient inference can be used to reinterpret a variety of existing gradient estimators from the literature in terms of a factorized coupling and a probabilistic inference algorithm. This framework lets us develop novel estimators that harness more powerful inference algorithms to achieve lower variance.

| | Factorized Coupling | Probabilistic Inference Algorithm | Reference |
|---|---|---|---|
| ***Existing Estimators*** | | | |
| [29] Pathwise / IPA | $\langle\bullet\rangle_{\texttt{CRN}}$ (e.g., **normal**$_{\texttt{CRN}}$) | Simple Monte Carlo | Appendix B.1 |
| [31] Phantom (SPA) | $\langle\bullet\rangle_{\texttt{CRN}}$ (e.g., **flip**$_{\texttt{CRN}}$) | Stratified Importance Sampling | Appendix B.2 |
| [25] Randomized Phantom (SPA) | $\langle\bullet\rangle_{\texttt{CRN}}$ (e.g., **flip**$_{\texttt{CRN}}$) | Stratified Importance Resampling | Appendix B.2 |
| [36] Measure-Valued Derivative | $\langle\bullet\rangle_{\texttt{FIXED-PROP}}$ | Stratified Importance Sampling | Appendix B.3 |
| [30] Score (w/ Control Variates) | $\langle\bullet\rangle_{\texttt{SHARED-SAMPLE}}$ | Varies by Control Variate Method | Appendix B.4 |
| [64] RLOO | **flip-layer**$_{\texttt{SAME-OR-INDEP}}$ | Stratified Importance Sampling | Appendix B.5 |
| [24] DisARM | **flip-layer**$_{\texttt{SAME-OR-ANTI}}$ | Stratified Importance Sampling | Appendix B.5 |
| [47] BitFlip1 | **flip-layer**$_{\texttt{FIXED-PROP}}$ | Stratified Importance Resampling | Appendix B.5 |
| [13] Straight-Through | **flip-layer**$_{\texttt{FIXED-PROP}}$ | Moment Closure | Appendix B.5 |
| ***Novel Estimators*** | | | |
| GradInf-CRN-VE | **flip**$_{\texttt{CRN}}$ | Variable Elimination | Sections 2 and 7.2.1 |
| GradInf-CRN-TSMC | **categorical**$_{\texttt{CRN}}$ | Twisted Sequential Monte Carlo | Section 7.2.2 |
| GradInf-MI-SIR | **categorical**$_{\texttt{MAX-INDEP}}$ | Stratified Importance Resampling | Section 7.2.3 |
| GradInf-MI-TSMC | **categorical**$_{\texttt{MAX-INDEP}}$ | Twisted Sequential Monte Carlo | Section 7.2.3 |

### 1.2 Our Approach

We introduce *gradient inference*, which is a new approach to gradient estimation that aims to

**(A1)** provide a modular and compositional workflow for developing gradient estimators;
**(A2)** enable the construction of novel gradient estimators that are sound and efficient.

Gradient inference rests on a reduction of a gradient estimation problem to a probabilistic inference problem. This reduction is enabled by two programmable techniques for statistical variance reduction: *coupling* (i.e., common random variables [69, §8.6]) and *factorization* (i.e., conditioning [69, §8.7]). Table 1 shows how gradient inference lets us interpret existing estimators as a combination of a factorized coupling and a probabilistic inference algorithm, which we can then generalize and improve upon by using more powerful programmable probabilistic inference methods [21, 62, 78].

**GradInf.** We contribute a probabilistic programming system, GradInf, that supports the sound and automated construction of gradient inference schemes. Figure 1 shows an overview of GradInf. The input is a probabilistic program $t$ denoting the target measure $\mu_\theta$ over $\mathbb{R}$ whose expectation we seek to differentiate. The final output is a new probabilistic program $t_\nabla$ that defines a sound (and typically efficient) Monte Carlo estimator $L(\theta)$ for the gradient $\nabla_\theta\left(\mathbb{E}_{x\sim\mu_\theta}[x]\right)$. The program $t_\nabla$ is obtained by first synthesizing an intermediate *inference target* program $t_S$ that exhibits a semantic equivalence to $t$, but incorporates coupling and factorization to enable low-variance gradient estimation. Next, a probabilistic inference algorithm—such as variable elimination [45, §9], sequential Monte Carlo (SMC [18]), or importance sampling [69, §9]—is used to estimate expectations over the return value of $t_S$. Finally, the overall gradient estimator $t_\nabla$ is obtained by applying *standard* AD to the probabilistic inference algorithm itself, which is constructed so as to avoid the usual pitfalls of naïvely differentiating a probabilistic program (e.g., [55, Fig. 1]).

**Applications.** We evaluate GradInf on several challenging case studies, testing its ability to compositionally express gradient estimators (A1) and produce novel schemes (A2). In a queuing theory application (§2), we generalize the classic SPA [31] estimator by using a variable elimination inference algorithm that achieves up to 16x reductions in time-adjusted variance. In case studies from mathematical finance (§7.2.2) and systems biology (§7.2.3), we use gradient inference with sequential Monte Carlo inference to produce novel estimators that deliver up to 370x improvements compared to existing baselines. Finally, in both these applications and supplementary studies in

Appendix B, we show that gradient inference can also express and unify a number of existing state-of-the-art estimators (cf. Table 1), surfacing new connections between diverse methodologies.

### 1.3 Contributions

In summary, this paper makes the following contributions.

- **Gradient Inference**: We introduce *gradient inference*, an algorithmic technique that converts a gradient estimation problem into a probabilistic inference problem and interoperates with standard forward- and reverse-mode automatic differentiation.
- **Language Constructs**: We present GradInf, a higher-order probabilistic programming system that facilitates gradient inference. GradInf is based on source-to-source program transformations that automate two statistical variance-reduction techniques: *coupling* and *factorization*.
- **Formalism**: We characterize the key soundness properties for gradient inference and prove the correctness of the program transformations in GradInf via proof-relevant logical relations. We leverage information-flow typing to ensure that the factored out choices in a probabilistic program can be *partially evaluated*, which enables accurate probabilistic inference algorithms.
- **Case Studies**: We demonstrate the effectiveness of gradient inference on challenging problems from queuing theory, finance, and genetics. GradInf can both express existing estimators and deliver new estimators that significantly outperform existing state-of-the-art baselines.

## 2 Overview of Gradient Inference

In this section, we give an overview of the key ideas in gradient inference using a probabilistic model from queuing theory (§2.1), and outline the key programming language challenges that GradInf addresses to facilitate sound and automated gradient inference workflows (§2.2).

### 2.1 How Gradient Inference Delivers Efficient Gradient Estimators

In the usual approach to gradient estimation, a Monte Carlo estimator for the target gradient is derived directly, with the key variance-reduction and algorithmic choices built into a single construction [65]. In contrast, our approach to gradient estimation, called gradient inference, modularly decomposes the problem into multiple programmable steps. We describe each step below, showing how gradient inference both recovers established estimators and yields novel estimators with lower variance.

**Example Model.** Figure 2d shows a probabilistic model called an M/M/c queue, which describes the flow of packets in a network queue. The discrete-time stochastic process $(X_0, X_1, \ldots, X_n)$ gives the number of packets $X_i$ in the queue after the $i$th queuing event, where packets either join or leave the queue with probability dependent on the number of routers $c = 25$ and the packet arrival rate $\theta : \mathbb{R}$. Figure 2e shows 10 sample paths of $X_{0:n}$ for $n = 50$ queuing events and $\theta \in \{10, 15\}$. Our goal is to assess the sensitivity of the *expected* final queue length after $n$ queuing events with respect to $\theta$, which is the gradient $\nabla_\theta \mathbb{E}[X_n]$. We will write $X_n$ as $X(\theta)$ to emphasize the $\theta$-dependence, dropping the subscript when it is clear from context.

**Probabilistic Program.** Figure 2a shows a user-written probabilistic program $t$ in GradInf that implements the model in Fig. 2d. The program uses standard probabilistic constructs such as Monte Carlo sampling from a Bernoulli distribution (line 3) and iteration of a probabilistic kernel (line 7). The return value $x$ on line 8 is the final queue size $X_n$. The semantics $[\![t]\!] : \mathbb{R} \to P(\mathbb{R})$ of this program is precisely the function $\mu : \mathbb{R} \to P(\mathbb{R})$ from §1, which maps the input parameter $\theta$ to the *marginal probability distribution* $[\![t]\!](\theta)$ of the return value $x$. In this notation, we seek to compute

(a) User-written input program $t$

```
let kernel = λ(θ : Real).λ(x : Int).
  let p = θ/(θ + min(25, x))
  b ←P flip_CRN p
  let x′ = if b then x + 1 else x − 1
  return_P x′
λ(θ : Real).
  x ←P iterate_{P_{n,Int}} (kernel θ) 0
  return_P x
```

probabilistic primitive

(b) Synthesized coupled program $t_C$

```
let kernel = λ(θ_A : Real, θ_B : Real).
             λ(x_A : Int, x_B : Int).
  let p_A = θ_A/(θ_A + min(25, x_A))
  let p_B = θ_B/(θ_B + min(25, x_B))
  ω ←P uniform (0, 1)
  let b_A = ω < p_A
  let b_B = ω < p_B
  let x′_A = if b_A then x_A + 1 else x_A − 1
  let x′_B = if b_B then x_B + 1 else x_B − 1
  return_P (x′_A, x′_B)
λ(θ : Real × Real).
  (x_A, x_B) ←P iterate_{P_{n,Int²}} (kernel θ) (0, 0)
  return_P (x_A, x_B)
```

result of coupling

(c) Synthesized inference target $t_S$

```
let kernel = λ(θ_A : Real, θ_B : Real).
             λ(x_A : Int, x_B : Int, s : Seed).
  let p_A = θ_A/(θ_A + min(25, x_A))
  let p_B = θ_B/(θ_B + min(25, x_B))
  let (s_0, s_1) = split s
  let b_A = s_0 < p_A  ▷ fixed via seed
  b_B ←P flip_ENUM ( if b_A then min(p_B/p_A, 1)
                     else max((p_B − p_A)/(1 − p_A), 0) )
  let x′_A = if b_A then x_A + 1 else x_A − 1
  let x′_B = if b_B then x_B + 1 else x_B − 1
  return_PP (x′_A, x′_B, s_1)
λ(θ : Real × Real).λ(s : Seed).
  (x_A, x_B) ←P iterate_{P_{n,Int×Int×Seed}}
               (kernel θ) (0, 0, s)
  return_PP (x_A, x_B)
```

result of partial evaluation

(d) Gradient estimation problem

**Generative Model:**

$X_0 \coloneqq 0 \quad c \coloneqq 25$

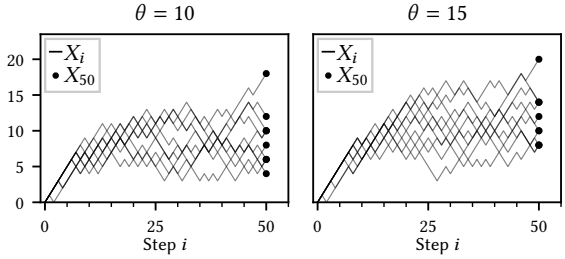
$B_i \sim \text{Bernoulli}\left(\theta/(\theta + \min(c, X_{i-1}))\right)$

$$X_i \coloneqq \begin{cases} X_{i-1} + 1 & \text{if } B_i = 1 \\ X_{i-1} - 1 & \text{otherwise} \end{cases}$$

**Goal:** Compute $\nabla_\theta \mathbb{E}[X_n]$, where

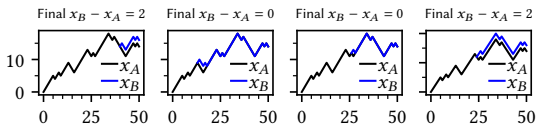
$\mathbb{E}[X_n] = \mathbb{E}[\cdots \mathbb{E}[\mathbb{E}[X_n \mid X_{n-1}]] \cdots \mid X_0]$

(e) Sample paths of $(X_0, X_1, \ldots, X_{50})$




(f) Applying two inference algorithms to $t_S$

Stratified Importance Resampling




4-Sample SIR Estimator = $0.83\varepsilon \times (2 + 0 + 0 + 2)/4 = 0.83\varepsilon$

prob. of stratified event

Variable Elimination

| Prob. \ Iter. | 0 | ⋯ | 24 | 25 | ⋯ | 50 |
|---|---|---|---|---|---|---|
| $\mathbb{P}(x_B = x_A)$ | 1 | ⋯ | $1 - 0.22\varepsilon$ | $1 - 0.20\varepsilon$ | ⋯ | $1 - 0.39\varepsilon$ |
| $\mathbb{P}(x_B = x_A + 2)$ | 0 | ⋯ | $0.22\varepsilon$ | $0.20\varepsilon$ | ⋯ | $0.39\varepsilon$ |

VE Estimator = $(1 - 0.39\varepsilon) \times 0 + 0.39\varepsilon \times 2 = 0.78\varepsilon$

(g) Variance of gradient estimators

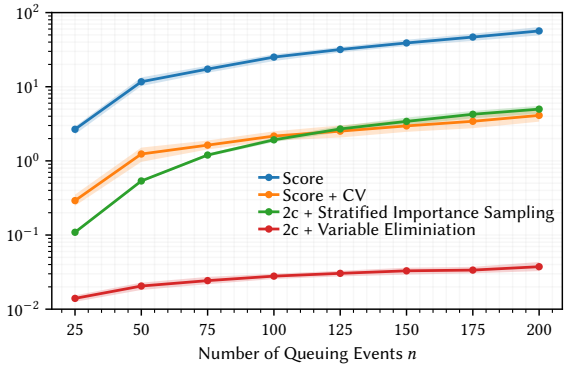



Fig. 2. Applying GradInf to a gradient estimation problem in a discrete-time M/M/c queuing model.

$\nabla_\theta \left[\mathbb{E}_{x\sim[\![t]\!](\theta)}[x]\right] = \nabla_\theta \mathbb{E}[X_n]$, which for $\theta : \mathbb{R}$ is just a scalar:

$$\nabla_\theta \mathbb{E}[X_n] = \lim_{\varepsilon\to 0} \frac{\mathbb{E}\left[X(\theta+\varepsilon)\right] - \mathbb{E}\left[X(\theta)\right]}{\varepsilon} = \lim_{\varepsilon\to 0} \frac{\mathbb{E}_{x\sim[\![t]\!](\theta+\varepsilon)}\left[x\right] - \mathbb{E}_{x\sim[\![t]\!](\theta)}\left[x\right]}{\varepsilon}. \tag{2.1}$$

Momentarily ignoring the $\varepsilon$-limit, (2.1) captures the essence of the gradient estimation challenge: the numerator requires an accurate estimate of the *difference* $\mathbb{E}\left[X(\theta+\varepsilon)\right] - \mathbb{E}\left[X(\theta)\right]$ of two closely-related expectations. We will thus turn our attention to this subproblem by temporarily assuming $\varepsilon$ is a small fixed perturbation, before taking the limit to eliminate $\varepsilon$.

Given only the program $t$ in Fig. 2a, we can obtain independent samples using inputs $\theta$ and $\theta+\varepsilon$ to estimate $\mathbb{E}\left[X(\theta+\varepsilon)\right] - \mathbb{E}\left[X(\theta)\right]$ via simple Monte Carlo. However, the resulting Monte Carlo estimator of $\left(\mathbb{E}\left[X(\theta+\varepsilon)\right] - \mathbb{E}\left[X(\theta)\right]\right)/\varepsilon$ using $k$ i.i.d. samples has unbounded variance $(\mathbb{V}(X(\theta)) + \mathbb{V}(X(\theta+\varepsilon)))/(k\varepsilon^2)$ as $\varepsilon \to 0$, rendering this naïve approach ineffective.

**Step 1: Forming a Coupling.** The variance explosion of simple Monte Carlo motivates forming a *coupling*. The key idea behind a coupling is that for any pair of random variables $(X_A, X_B)$ whose marginal distributions agree with $(X(\theta), X(\theta+\varepsilon))$, the linearity of expectation implies that $\mathbb{E}\left[X_B - X_A\right] = \mathbb{E}\left[X(\theta+\varepsilon)\right] - \mathbb{E}\left[X(\theta)\right]$. If $X_A$ and $X_B$ have positive covariance, then coupling achieves a (often substantial) variance reduction [69, §8.6] compared to using independent samples.

Figure 2b shows a coupled program $t_C$ synthesized from $t$, in which each original value has been replaced with a *pair* of values. This term has semantics $[\![t_C]\!] : (\mathbb{R}\times\mathbb{R}) \to P(\mathbb{R}\times\mathbb{R})$, which defines a joint distribution over pairs of outputs. Lines 4–6 of Fig. 2b show how the coupled variables are generated using a common random number (CRN) strategy, corresponding to the annotated **flip**$_{\text{CRN}}$ primitive on line 3 of Fig. 2a. The return value $(x_A, x_B)$ of $[\![t_C]\!](\theta, \theta+\varepsilon)$ on line 12 of Fig. 2b is guaranteed to be a valid coupling of $[\![t]\!](\theta)$ and $[\![t]\!](\theta+\varepsilon)$, retaining the unbiasedness of simple Monte Carlo estimation using $t$:

$$\mathbb{E}_{(x_A,x_B)\sim[\![t_C]\!](\theta,\theta+\varepsilon)}\left[x_B - x_A\right] = \mathbb{E}_{x\sim[\![t]\!](\theta+\varepsilon)}\left[x\right] - \mathbb{E}_{x\sim[\![t]\!](\theta)}\left[x\right]. \tag{2.2}$$

**Step 2: Factorizing the Coupling.** To form the inference target, GradInf synthesizes a *factorized* version of the coupling in Fig. 2b. As motivation, note that only samples where $x_B \neq x_A$ contribute to the expectation in (2.2). However, consider lines 2–6 of Fig. 2b: as $\varepsilon \to 0$ we have $\theta_B \to \theta_A$ and thus $p_B \to p_A$. Hence, the samples $b_A$ and $b_B$ on lines 5–6 will be equal with high probability, yielding an estimate of $x_B - x_A = 0$. We can improve our estimator by using a probabilistic inference algorithm, such as *importance sampling* [69, §9], to generate traces where $x_B \neq x_A$.

Let $X_{A,0:n}$ and $X_{B,0:n}$ be random variables corresponding to traces of $x_A$ and $x_B$ across all $n$ steps of the coupled program $t_C$ in Fig. 2b. By the chain rule, their joint distribution can be factorized as

$$P\left(X_{A,0:n} = x_{A,0:n}, X_{B,0:n} = x_{B,0:n}\right) = P\left(X_{A,0:n} = x_{A,0:n}\right) P\left(X_{B,0:n} = x_{B,0:n} \,\middle|\, X_{A,0:n} = x_{A,0:n}\right). \tag{2.3}$$

We call $X_{A,0:n}$ the *primal* trace and $X_{B,0:n}$ the *residual* trace. Figure 2c shows a synthesized program $t_{\mathcal{S}}$ that implements the factorization in (2.3). The program $t_{\mathcal{S}}$ is constructed to allow a probabilistic form of *partial evaluation* of the coupled program $t_C$ in Fig. 2b: all sampling steps determining the primal trace can be fixed in $t_{\mathcal{S}}$ by evaluating the program at a fixed random seed $s$. This property enables the subsequent use of generic probabilistic inference algorithms to infer a residual trace $X_{B,0:n}(\theta+\varepsilon) \mid X_{A,0:n}(\theta) = x_{A,0:n}$ that is distinct from a fixed primal trace $x_{A,0:n}$ with high probability.

**Step 3: Applying Probabilistic Inference.** Equipped with the *inference target* program $t_{\mathcal{S}}$ (Fig. 2c), the goal of probabilistic inference is to return an efficient estimator $Z$ of the conditional expectation

$$\mathbb{E}\left[Z(\theta, \theta+\varepsilon, x_{A,0:n})\right] = \mathbb{E}\left[X_{B,n} - X_{A,n} \,\middle|\, X_{A,0:n} = x_{A,0:n}\right] = \mathbb{E}_{(x_A,x_B)\sim[\![t_{\mathcal{S}}]\!](\theta,\theta+\varepsilon)(s)}\left[x_B - x_A\right]. \tag{2.4}$$

This step of gradient inference is related to Rao-Blackwellization, a variance reduction technique that replaces an estimator by its conditional expectation given a subset of random variables [69,

§8.7]. Our criterion (2.4) is a form of *approximate* Rao-Blackwellization, since the estimator $Z$ must be unbiased but not necessarily exhibit zero variance. Figure 2f shows two inference algorithms applied to $t_{\mathcal{S}}$ for $\theta = 15$ and $\varepsilon = 10^{-10}$. We develop a custom estimator $Z_{\text{SIR}}$ that uses stratified importance resampling (SIR), where the algorithm stratifies (i.e., separately samples from) the events $A_i$ where the trace of $x_B$ diverges from the fixed trace of $x_A$ at some iteration $i$. For this queuing model, we can develop an improved estimator $Z_{\text{VE}}$ that uses a *linear-time* variable elimination (VE) inference algorithm. The efficiency of VE is enabled by coupling and factorization: because the coupling is monotone, the value of $x_B$ is either $x_A$ or $x_A + 2$ at each iteration (where $x_A$ is *fixed* in the primal sample), greatly simplifying the inference target. Applying VE to the model in Fig. 2b would instead scale quadratically, highlighting the benefits of factorization.

**Step 4: Forming the Gradient Estimator.** Having performed the variance-reduction steps, we now eliminate $\varepsilon$ from (2.1). Given an estimator $Z$ satisfying (2.4), the law of total expectation gives

$$\mathbb{E}\left[X(\theta+\varepsilon)\right] - \mathbb{E}\left[X(\theta)\right] = \mathbb{E}\left[Z\left(\theta, \theta+\varepsilon, X_{A,0:n}\right)\right]. \tag{2.5}$$

If the random choices made to simulate the estimator $Z$ during probabilistic inference do not depend directly on the perturbation $\varepsilon$, we may first apply the chain rule of differentiation and then (under standard conditions) interchange the derivative and expectation:

$$\nabla_\theta \mathbb{E}\left[X(\theta)\right] = \nabla_\varepsilon \left[\mathbb{E}\left[Z\left(\theta, \theta+\varepsilon, X_{A,0:n}\right)\right]\right]_{\varepsilon=0} = \mathbb{E}\left[\nabla_\varepsilon \left[Z\left(\theta, \theta+\varepsilon, X_{A,0:n}\right)\right]_{\varepsilon=0}\right]. \tag{2.6}$$

Eq. (2.6) gives the final gradient estimator. The inner derivative on the right-hand side can be obtained using standard (forward- or reverse-mode) AD, concluding the gradient inference workflow.

**Empirical Results.** Figure 2g shows a comparison of the variance of four gradient estimators in the M/M/c model (Fig. 2d) at $\theta = 15$, with $n$ varied from 25 to 200. The blue and orange curves show the standard score function (REINFORCE [87]) gradient estimator without and with a control variate (CV). The green curve corresponds to our gradient inference scheme using SIR, which turns out to recover the randomized phantom estimator from the smoothed perturbation analysis literature [25]. The red curve corresponds to our gradient inference scheme using VE, which produces a novel gradient estimator that improves over the existing baselines by up to two orders of magnitude.

### 2.2 Soundness and Automation via Types and Transformations

GradInf facilitates the implementation of gradient inference workflows for expressive, higher-order probabilistic programs written in our core language $\lambda_P$, which is formalized in §3. Implementing the GradInf workflow in Fig. 1 requires novel solutions to three key challenges:

**(C1)** *How do we form couplings of arbitrary higher-order probabilistic programs?* In §4, adapting ideas from relational probabilistic program logics [10], we present a programmable coupling transformation $C\{\cdot\}$ for $\lambda_P$ programs and use a proof-relevant logical relations argument to prove that it synthesizes sound couplings of entire programs (Theorem 1).

**(C2)** *How do we partially evaluate random choices in the coupled program?* In §5, we present an extended language $\lambda_{PP}$ for factorizing probabilistic programs into *primal* and *residual* choices, whose type system builds on the dependency core calculus of Abadi et al. [1]. We present a transformation $C^*\{\cdot\}$ for forming factorized couplings, which targets $\lambda_{PP}$ but simplifies to a sound coupling transformation in $\lambda_P$ when the $\lambda_{PP}$-specific constructs are erased using the transformation $\mathcal{E}\{\cdot\}$ (Theorem 2); and a transformation $\mathcal{S}\{\cdot\}$ from $\lambda_{PP}$ to $\lambda_P$ for probabilistic partial evaluation (in the sense described in §2.1) that produces sound factorizations of coupled programs (Theorem 3).

**(C3)** *How do we ensure that the final synthesized gradient estimator is sound?* In §6, we complete the gradient inference workflow by explaining how the program synthesized by $C^*\{\cdot\}$ and $\mathcal{S}\{\cdot\}$

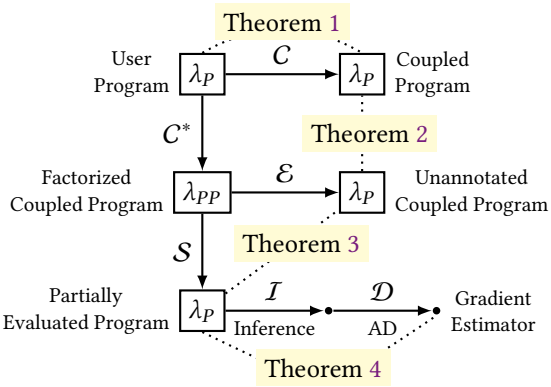



Fig. 3. Overview of program transformations in GradInf. Dotted lines indicate program equivalences, whose properties are stated in each of the respective theorems. The user program is written in $\lambda_P$. The probabilistic language $\lambda_{PP}$ serves as an intermediate representation for synthesized coupled programs that contain annotations for factorization. Erasing these annotations yields a valid coupled $\lambda_P$ program. The annotations are used to synthesize an equivalent partially evaluated $\lambda_P$ program to target with inference and AD, which gives a final sound gradient estimator.

can be composed with existing rigorous formalisms of probabilistic inference [78] and AD transformations [39, 46] to obtain an overall sound gradient estimator (Theorem 4).

Figure 3 summarizes the program transformations used in GradInf and related theorems that together establish the soundness of gradient inference. Sections 3–6 unpack the details of Fig. 3.

## 3 Core Probabilistic Language

We begin our technical development by introducing the syntax and semantics of our core probabilistic language $\lambda_P$. We use a blue shade to highlight language constructs related to probability.

**Syntax.** Listing 1 gives the syntax of $\lambda_P$. The deterministic core of $\lambda_P$ is the standard simply-typed $\lambda$-calculus, with functions, products, and built-in base types for booleans, integers, reals, and unit. For example, $\lambda(x:\tau).t$ and $t'\ t$ represent function abstraction and application, respectively, while $(t_1, t_2)$ creates a pair and $\mathbf{prj}_1\ t$ and $\mathbf{prj}_2\ t$ extract its components. To this core, we add the type $\texttt{Prob}\ \tau$ for measures over $\tau$. Measures are created by probabilistic primitives or the monadic return $\mathbf{return_P}\ t$; these measures are propagated using the bind $x \leftarrow_{\mathbf{P}} t;\ t'$. The set of primitives $p$ is extensible to support different applications and gradient estimation strategies (e.g., $\mathbf{flip}_{\mathsf{CRN}}$ in §2). The $\lambda_P$ language is equipped with a typing judgement $\Gamma \vdash t : \tau$. Listing 1 gives typing rules for base types and the probabilistic constructs; the full set of rules and details on syntactic sugar (e.g., for tuple unpacking and iteration) are given in Appendix C.

**Denotational Model.** A denotational model provides an interpretation of the syntactic types and well-typed terms of a language in a chosen "semantic space" category. As in recent works with higher-order probabilistic languages [12, 54, 55, 78], the semantic space for $\lambda_P$ is the category **QBS** of quasi-Borel spaces [37]. The semantics of a type $\tau$ is a **QBS**-object $[\![\tau]\!]$ and the semantics of a well-typed term $\Gamma \vdash t : \tau$ is a **QBS**-morphism $[\![\Gamma \vdash t : \tau]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$, where a context $\Gamma = x_1 : \tau_1, \ldots, x_n : \tau_n$ is interpreted as a labeled product of $[\![\tau_1]\!], \ldots, [\![\tau_n]\!]$.

We make use of two notational shorthands: (1) when the typing judgement is clear from context, we shorten $[\![\Gamma \vdash t : \tau]\!]$ to $[\![t]\!]$, and (2) in **QBS**, we write $u : U$ as a shorthand for $u : \mathbf{unit} \to U$. Using this notation, a closed term $\bullet \vdash t : \tau$ is interpreted as a *value* $[\![t]\!] : [\![\tau]\!]$, while a general open term $\Gamma \vdash t : \tau$ is interpreted as a map from an *environment* $\gamma : [\![\Gamma]\!]$ to a value $[\![t]\!](\gamma) : [\![\tau]\!]$. We denote environment extension by $\gamma[x := u]$. In the following, we use **black boldface** for semantic space constructs mirroring language syntax in **blue boldface**, e.g., $\mathbf{prj}_1((x, y)) = x$.

A review of **QBS** is given in Appendix C, but in the main text it will generally suffice to analogize to the usual category **Meas** of measurable spaces [27], so that we can intuitively think of $[\![\tau]\!]$ as a measurable space and $[\![\Gamma \vdash t : \tau]\!]$ as a measurable function. The utility of **QBS** is that the category both supports higher-order functions and admits a monad $P$ of measures [78]. We can

**Types**

$\sigma$ (∈ BaseTypes) ::= $\texttt{Bool} \mid \texttt{Int} \mid \texttt{Real} \mid \texttt{Unit}$

$\tau$ (∈ Types) ::= $\sigma \mid \tau_1 \times \tau_2 \mid \tau \to \tau' \mid \texttt{Prob}\ \tau$

**Contexts**

$\Gamma$ (∈ Contexts) ::= $\bullet \mid \Gamma, x : \tau$

**Terms**

$r \in \mathbb{R};\ n \in \mathbb{Z};\ x \in$ Variables

$c$ (∈ Constants) ::= $r \mid n \mid \mathbf{unit} \mid \mathbf{true} \mid \mathbf{false}$

$t$ (∈ Terms) ::= $x \mid c \mid p \mid (t_1, t_2) \mid \lambda(x : \tau).t \mid t'\ t$
$\mid \mathbf{let}\ x = t;\ t' \mid \mathbf{prj}_1 \mid \mathbf{prj}_2$
$\mid \mathbf{if}\ t\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2$
$\mid \mathbf{return_P}\ t \mid x \leftarrow_{\mathbf{P}} t;\ t'$

$p$ (∈ Primitives) ::= ⟨extensible⟩

**Typing Rules (Selected)**

$$\Gamma \vdash \mathbf{true} : \texttt{Bool} \quad \Gamma \vdash \mathbf{false} : \texttt{Bool} \quad \Gamma \vdash n : \texttt{Int} \quad \Gamma \vdash r : \texttt{Real} \quad \Gamma \vdash \mathbf{unit} : \texttt{Unit} \quad \Gamma \vdash p : \tau_p$$

$$\frac{\Gamma \vdash t : \tau}{\Gamma \vdash \mathbf{return_P}\ t : \texttt{Prob}\ \tau} \qquad \frac{\Gamma \vdash t : \texttt{Prob}\ \tau \quad \Gamma, x : \tau \vdash t' : \texttt{Prob}\ \tau'}{\Gamma \vdash x \leftarrow_{\mathbf{P}} t;\ t' : \texttt{Prob}\ \tau'}$$

Listing 1. Types and terms for $\lambda_P$, a core calculus for higher-order probabilistic programs.

gain intuition for the key operations supported by $P$ by analogizing to the Giry monad in **Meas**: (1) as a functor, $P$ sends a space $U$ to the space $P(U)$ of measures over $U$ and a function $f : U \to U'$ to the pushforward $P(f) : P(U) \to P(U')$; (2) $\mathbf{return_P}(u) : P(U)$ constructs a Dirac distribution at $u : U$; and (3) the bind $(\mu \gg\!=_{\mathbf{P}} f) : P(U')$ operation composes a measure $\mu : P(U)$ with a kernel $f : U \to P(U')$ in the usual way [71]: $(\mu \gg\!=_{\mathbf{P}} f)(A) = \int f(u)(A)\mu(\mathrm{d}u)$.

Using $P$, we can define the model $[\![\cdot]\!]$ on $\lambda_P$. The model interprets base types using their corresponding **QBS** spaces $\mathbb{B}$, $\mathbb{Z}$, $\mathbb{R}$, and $\mathbf{1}$, interprets products and function types using products and exponentials in **QBS**, and interprets $\texttt{Prob}\ \tau$ using the measure monad $P$:

$$[\![\texttt{Bool}]\!] \coloneqq \mathbb{B} \quad [\![\texttt{Int}]\!] \coloneqq \mathbb{Z} \quad [\![\texttt{Real}]\!] \coloneqq \mathbb{R} \quad [\![\texttt{Unit}]\!] \coloneqq \mathbf{1} \tag{3.1}$$

$$[\![\tau_1 \times \tau_2]\!] \coloneqq [\![\tau_1]\!] \times [\![\tau_2]\!] \quad [\![\tau_1 \to \tau_2]\!] \coloneqq [\![\tau_1]\!] \to [\![\tau_2]\!] \quad [\![\texttt{Prob}\ \tau]\!] \coloneqq P\left([\![\tau]\!]\right). \tag{3.2}$$

The model acts in the standard way on deterministic terms (e.g., the term $\lambda(x : \tau).t$ is interpreted as a function). The terms $\mathbf{return_P}\ t$ and $x \leftarrow_{\mathbf{P}} t;\ t'$ are interpreted using the return and bind of $P$: $[\![\mathbf{return_P}\ t]\!](\gamma) \coloneqq \mathbf{return_P}\left([\![t]\!](\gamma)\right)$ and $[\![\{x \leftarrow_{\mathbf{P}} t;\ t'\}]\!](\gamma) \coloneqq [\![t]\!](\gamma) \gg\!=_{\mathbf{P}} \left(u \mapsto [\![t']\!](\gamma[x \coloneqq u])\right)$. Since a primitive $p$ does not depend on its environment, we let $[\![p]\!](\gamma) \coloneqq [\![p]\!]$ where $[\![p]\!] : [\![\tau_p]\!]$ is given for each primitive. Appendix C supplies the full definition of the model $[\![\cdot]\!]$ and further detail.

**Program Transformations.** A program transformation $\mathcal{T}\{\cdot\}$ between two languages $\lambda_1$ and $\lambda_2$ maps the types and well-typed terms of a language $\lambda_1$ to those of a language $\lambda_2$, with the property that a well-typed term $\Gamma \vdash t : \tau$ of $\lambda_1$ transforms to a well-typed term $\mathcal{T}\{\Gamma \vdash t : \tau\}$ of $\lambda_2$ with context $\mathcal{T}\{\Gamma\}$ and type $\mathcal{T}\{\tau\}$. As with the denotational model, we will abbreviate $\mathcal{T}\{\Gamma \vdash t : \tau\}$ to $\mathcal{T}\{t\}$ when the typing judgement is clear from context.

Our program transformations will typically act in an "interesting" way on only a few types and terms, while preserving the structure of the rest. Accordingly, we say that $\mathcal{T}\{\cdot\}$ acts *homomorphically* on a type or term when it recurses on its structure. For example, $\mathcal{T}\{\tau_1 \times \tau_2\} \coloneqq \mathcal{T}\{\tau_1\} \times \mathcal{T}\{\tau_2\}$, $\mathcal{T}\{\texttt{Prob}\ \tau\} \coloneqq \texttt{Prob}\ \mathcal{T}\{\tau\}$, and $\mathcal{T}\{\mathbf{let}\ x = t;\ t'\} \coloneqq \{\mathbf{let}\ x = \mathcal{T}\{t\};\ \mathcal{T}\{t'\}\}$ specify homomorphic actions on products, measures, and the let bind. The full definition is given in Appendix C.

**Example Program.** The extensible set of primitives in $\lambda_P$ includes standard arithmetic operations, listed in Appendix C. Let us register a probabilistic **flip** primitive in $\lambda_P$ with type $\tau_{\mathbf{flip}} = \texttt{Real} \to \texttt{Prob Bool}$ and semantics $[\![\mathbf{flip}]\!] \coloneqq r \mapsto \mathrm{Ber}(r)$, where $\mathrm{Ber}(r) : P(\mathbb{B})$ is a Bernoulli distribution with

**Coupling Transformation $C\{\cdot\}$ on $\lambda_P$**

$$C\{\sigma\} \coloneqq \sigma \times \sigma \qquad C\{c\} \coloneqq (c, c) \qquad C\{p\} \coloneqq \langle\text{defined per-primitive}\rangle$$
$$C\{\Gamma \vdash \textbf{if } t \textbf{ then } t_1 \textbf{ else } t_2 : \tau\} \coloneqq \textbf{let } (b_A, b_B) = C\{t\};\ \mathcal{M}_\tau\{\textbf{if } b_A \textbf{ then } C\{t_1\} \textbf{ else } C\{t_2\}\}\{\textbf{if } b_B \textbf{ then } C\{t_1\} \textbf{ else } C\{t_2\}\},$$

**Merge Macro $\mathcal{M}$ for Conditional Expressions**

$$\mathcal{M}_\sigma\{t_1\}\{t_2\} \coloneqq \textbf{let } ((x_A : \sigma, x_B : \sigma), (y_A : \sigma, y_B : \sigma)) = (t_1, t_2);\ (x_A, y_B)$$
$$\mathcal{M}_{\tau_1 \times \tau_2}\{t_1\}\{t_2\} \coloneqq (\mathcal{M}_{\tau_1}\{\textbf{prj}_1\ t_1\}\{\textbf{prj}_1\ t_2\}, \mathcal{M}_{\tau_2}\{\textbf{prj}_2\ t_1\}\{\textbf{prj}_2\ t_2\})$$
$$\mathcal{M}_{\tau_1 \to \tau_2}\{t_1\}\{t_2\} \coloneqq \lambda(x : C\{\tau_1\}).\mathcal{M}_{\tau_2}\{t_1\ x\}\{t_2\ x\}$$
$$\mathcal{M}_{\texttt{Prob}\ \tau}\{t_1\}\{t_2\} \coloneqq \{x_A \leftarrow_{\mathbf{P}} t_1;\ x_B \leftarrow_{\mathbf{P}} t_2;\ \textbf{return}_{\mathbf{P}}\ \mathcal{M}_\tau\{x_A\}\{x_B\}\},$$

Listing 2. The coupling transformation for $\lambda_P$. The transformation acts homomorphically on all omitted types and terms.

success probability $r$. We can now write an example probabilistic program term in $\lambda_P$,

$$t \coloneqq \{b \leftarrow_{\mathbf{P}} \textbf{flip } 0.5;\ \textbf{return}_{\mathbf{P}}\ (\textbf{if } b \textbf{ then } (3.5 + 1.2) \textbf{ else } 8.2)\}. \tag{3.3}$$

Treating $t$ as a closed term, we have $\bullet \vdash t : \texttt{Prob Real}$, where $[\![t]\!] : P(\mathbb{R})$ is the probability distribution that assigns probability 0.5 to each of the values 4.7 and 8.2.

## 4 Programmable Couplings of Higher-Order Programs

To address challenge (C1), we introduce a coupling transformation $C\{\cdot\}$ on the syntax of $\lambda_P$, as well as a logical relation $R_C$ that governs the soundness of this transformation.

**Coupling Transformation.** Listing 2 defines the coupling program transformation $C\{\cdot\}$. The transformation replaces base types $\sigma$ with pairs thereof and duplicates constant literals $c$. Its action on **if** $t$ **then** $t_1$ **else** $t_2$ uses a helper macro $\mathcal{M}_\tau\{\cdot\}\{\cdot\}$, where $\mathcal{M}_\tau\{s_1\}\{s_2\}$ is a coupled term in which each pair draws its first component from $s_1$ and its second component from $s_2$. On all other types and terms, the transformation acts homomorphically. The programs produced by $C\{\cdot\}$ can be viewed as probabilistic product programs, in the sense of Barthe et al. [10].

**Coupling Logical Relation.** To reason about the soundness of $C\{\cdot\}$ on terms of arbitrary type, we introduce the coupling logical relation $R_C$. The relation relates tuples $(x_A, x_B, x_C)$, where $x_A : [\![\tau]\!]$, $x_B : [\![\tau]\!]$, and $x_C : [\![C\{\tau\}]\!]$. If $(x_A, x_B, x_C) \in R_C(\tau)$, then $x_C$ is a sound coupling of $x_A$ and $x_B$.

To specify $R_C$, we define a *proof-relevant* version $R_C^\square$ of the relation. The relation $R_C^\square$ associates each type $\tau$ of $\lambda_P$ with a quasi-Borel space $R_C^\square(\tau)$, consisting of tuples $(x_A, x_B, x_C, \tilde{x})$ where $x_A : [\![\tau]\!]$, $x_B : [\![\tau]\!]$, $x_C : [\![C\{\tau\}]\!]$, and $\tilde{x}$ is a **QBS**-value that *proves* the relatedness $(x_A, x_B, x_C)$. As in a usual logical relations argument [3, 8, 39, 79], we define $R_C^\square$ inductively on types. At a base type $\sigma$,

$$R_C^\square(\sigma) \coloneqq \{(x_A, x_B, x_C, \textbf{unit}) \mid x_C = (x_A, x_B)\}. \tag{4.1}$$

In words, a coupling of two values of base type is a pair thereof. The definition of $R_C^\square$ on product and function types is standard, and supplied in Appendix D. To lift the relation to measures [9, 49], we let $R_C^\square(\texttt{Prob}\ \tau)$ be the set of tuples $(\mu_A, \mu_B, \mu_C, \tilde{\mu})$ for which the proof value $\tilde{\mu}$ is a probability distribution whose first three marginals recover $\mu_A, \mu_B$ and $\mu_C$:

$$\textstyle\int \tilde{\mu}(\mathrm{d}x) = 1 \wedge P(\mathbf{prj}_1)(\tilde{\mu}) = \mu_A \wedge P(\mathbf{prj}_2)(\tilde{\mu}) = \mu_B \wedge P(\mathbf{prj}_3)(\tilde{\mu}) = \mu_C. \tag{4.2}$$

Finally, we define $R_C(\tau) \coloneqq \{(x_A, x_B, x_C) \mid \exists \tilde{x}.(x_A, x_B, x_C, \tilde{x}) \in R_C^\square(\tau)\}$. The proof-relevance of $R_C^\square$ is crucial to proving our soundness theorem for couplings (Theorem 1), as it ensures that proof values can be constructed and propagated using **QBS**-morphisms (cf. [74]).

```
C{flip_INDEP} ≔
λ(p_A : Real, p_B : Real).
  b_A ←_P flip p_A
  b_B ←_P flip p_B
  return_P (b_A, b_B)

C{flip_CRN} ≔
λ(p_A : Real, p_B : Real).
  ω ←_P uniform (0, 1)
  let b_A = ω < p_A
  let b_B = ω < p_B
  return_P (b_A, b_B)

C{normal_CRN} ≔
λ((μ_A, μ_B), (σ_A, σ_B))
   : (Real² × Real²)).
  ω ←_P normal (0, 1)
  let x_A = μ_A + σ_A × ω
  let x_B = μ_B + σ_B × ω
  return_P (x_A, x_B)

C{categorical_MAX-INDEP α,n} ≔
λ(z : (α²)^n, p : (Real²)^n).
  let (p_A, p_B) = (map prj_1 p, map prj_2 p)
  i_A ←_P categorical ((1, ..., n), p_A)
  let x_A = index (map prj_1 z) i_A
  let x'_A = index (map prj_2 z) i_A
  b ←_P flip (min(1, (index p_B i_A)/(index p_A i_A)))
  if b then (return_P (x_A, x'_A)) else
    let r = map (λ(p_A, p_B). max(0, p_B − p_A)) p
    x_B ←_P categorical (map prj_2 z, normalize r)
    return_P (x_A, x_B)
```

Listing 3. Example definitions of the coupling transformation for four probabilistic primitives.

**Example Coupling Primitives.** For $\Gamma \vdash t : \tau$, we say that $C\{t\}$ is sound if it *preserves* the coupling logical relation, i.e., $(\gamma_A, \gamma_B, \gamma_C) \in R_C(\Gamma)$ implies that $\big([\![t]\!](\gamma_A), [\![t]\!](\gamma_B), [\![C\{t\}]\!](\gamma_C)\big) \in R_C(\tau)$. Since a primitive $p$ is environment-independent, $C\{p\}$ is sound if $\big([\![p]\!], [\![p]\!], [\![C\{p\}]\!]\big) \in R_C(\tau_p)$.

For the deterministic primitives of $\lambda_P$, there is usually just one sound coupling, e.g., $C\{+\} \coloneqq \lambda((x_A, x_B), (y_A, y_B)).(x_A{+}y_A, x_B{+}y_B)$. On the other hand, the probabilistic primitives can be soundly coupled using many different strategies. Listing 3 shows four examples of coupling primitives in $\lambda_P$, implementing an independent coupling for a Bernoulli, common random number couplings for a Bernoulli and normal distribution, and a maximal coupling with independent residuals [84] for a categorical distribution. Here, the primitives for normal and categorical distributions have types $\tau_{\mathbf{normal}} = \texttt{Real}^2 \to \texttt{Prob Real}$ and $\tau_{\mathbf{categorical}_{\alpha,n}} = \alpha^n \to \texttt{Real}^n \to \texttt{Prob}\ \alpha$ and semantics $[\![\mathbf{normal}]\!] \coloneqq (\mu, \sigma) \mapsto \mathrm{N}(\mu, \sigma)$; $[\![\mathbf{categorical}_{\alpha,n}]\!] \coloneqq (x_{1:n}, p_{1:n}) \mapsto \sum_i p_i \cdot \mathbf{return}_{\mathbf{P}}(x_i)$.

As a concrete example, consider the $\mathbf{flip}_{\mathsf{CRN}}$ primitive, whose type is $\tau_{\mathbf{flip}_{\mathsf{CRN}}} = \texttt{Real} \to \texttt{Prob Bool}$. Applying the coupling transform to this type gives

$$C\{\tau_{\mathbf{flip}_{\mathsf{CRN}}}\} = \texttt{Real} \times \texttt{Real} \to \texttt{Prob}\ (\texttt{Bool} \times \texttt{Bool}). \tag{4.3}$$

The semantics of $\mathbf{flip}_{\mathsf{CRN}}$ are defined to match $\mathbf{flip}$, i.e., $[\![\mathbf{flip}_{\mathsf{CRN}}]\!] = [\![\mathbf{flip}]\!]$. However, the CRN annotation specifies that the primitive should be coupled using *common random numbers*. Accordingly, the implementation of $C\{\mathbf{flip}_{\mathsf{CRN}}\}$ uses a shared uniform $\omega$ to generate both flips. By unpacking the definition of $R_C(\texttt{Real} \to \texttt{Prob Bool})$, it can be seen that the soundness requirement imposed on $C\{\mathbf{flip}_{\mathsf{CRN}}\}$ is equivalent to the statement that for all $p_A : \mathbb{R}$ and $p_B : \mathbb{R}$,

$$P(\mathbf{prj}_1)([\![C\{\mathbf{flip}_{\mathsf{CRN}}\}]\!](p_A, p_B)) = [\![\mathbf{flip}_{\mathsf{CRN}}]\!](p_A), \tag{4.4}$$

$$P(\mathbf{prj}_2)([\![C\{\mathbf{flip}_{\mathsf{CRN}}\}]\!](p_A, p_B)) = [\![\mathbf{flip}_{\mathsf{CRN}}]\!](p_B), \tag{4.5}$$

which is precisely the condition we intend: for all $p_A : \mathbb{R}$ and $p_B : \mathbb{R}$, $[\![C\{\mathbf{flip}_{\mathsf{CRN}}\}]\!](p_A, p_B)$ is a joint probability distribution with marginals of $[\![\mathbf{flip}_{\mathsf{CRN}}]\!](p_A)$ and $[\![\mathbf{flip}_{\mathsf{CRN}}]\!](p_B)$.

**Soundness of Program Couplings.** The main theorem of this section states that if $C\{\cdot\}$ is soundly defined on each primitive, then the transformation forms a sound coupling of any input program. (All proofs are given in the appendices.)

Theorem 1 (Soundness of Coupling Transformation). *Let $\Gamma \vdash t : \tau$ be a well-typed term of $\lambda_P$. If each primitive $p$ appearing in $t$ satisfies $([\![p]\!], [\![p]\!], [\![C\{p\}]\!]) \in R_C(\tau_p)$, then*

$$(\gamma_A, \gamma_B, \gamma_C) \in R_C(\Gamma) \implies \big([\![t]\!](\gamma_A), [\![t]\!](\gamma_B), [\![C\{t\}]\!](\gamma_C)\big) \in R_C(\tau). \tag{4.6}$$

## 5 Partially Evaluating a Coupled Probabilistic Program

Having defined a sound coupling transformation, we now address challenge (C2): how can we soundly factorize a coupled probabilistic program, so that it can be partially evaluated in the sense explained in §2? In this section, we present a factorized coupling transformation $C^*\{\cdot\}$ that

**Types and Terms**

$\tau$ (∈ Types) ::= $\cdots$ | $\texttt{Residual}\ \tau$ | $\texttt{PProb}\ \tau$

$t$ (∈ Terms) ::= $\cdots$ | $\textbf{hide}\ t$ | $x \leftarrow_{\mathrm{R}} t;\ t'$ | $\textbf{primal}_{\mathrm{PP}}\ t$ | $\textbf{residual}_{\mathrm{PP}}\ t$ | $\textbf{return}_{\mathrm{PP}}\ t$ | $x \leftarrow_{\mathrm{PP}} t;\ t'$

**Typing Rules**

$$\frac{\Gamma \vdash t : \tau}{\Gamma \vdash \textbf{hide}\ t : \texttt{Residual}\ \tau} \qquad \frac{\Gamma \vdash t : \texttt{Residual}\ \tau \quad \Gamma, x : \tau \vdash t' : \tau' \quad \text{Hidden}\ \tau'}{\Gamma \vdash x \leftarrow_{\mathrm{R}} t;\ t' : \tau'}$$

$$\frac{\Gamma \vdash t : \texttt{Prob}\ \tau \quad \text{Transparent}\ \tau}{\Gamma \vdash \textbf{primal}_{\mathrm{PP}}\ t : \texttt{PProb}\ \tau} \qquad \frac{\Gamma \vdash t : \texttt{Residual}\ (\texttt{Prob}\ \tau) \quad \text{Hidden}\ \tau}{\Gamma \vdash \textbf{residual}_{\mathrm{PP}}\ t : \texttt{PProb}\ \tau}$$

$$\frac{\Gamma \vdash t : \tau}{\Gamma \vdash \textbf{return}_{\mathrm{PP}}\ t : \texttt{PProb}\ \tau} \qquad \frac{\Gamma \vdash t : \texttt{PProb}\ \tau \quad \Gamma, x : \tau \vdash t' : \texttt{PProb}\ \tau'}{\Gamma \vdash x \leftarrow_{\mathrm{PP}} t;\ t' : \texttt{PProb}\ \tau'}$$

**Hidden and Transparent Typing Judgements**

$$\frac{}{\text{Transparent}\ \sigma} \qquad \frac{\text{Transparent}\ \tau_1 \quad \text{Transparent}\ \tau_2}{\text{Transparent}\ (\tau_1 \times \tau_2)} \qquad \frac{\text{Transparent}\ \tau_2}{\text{Transparent}\ (\tau_1 \to \tau_2)}$$

$$\frac{\text{Transparent}\ \tau}{\text{Transparent}\ (\texttt{Prob}\ \tau)} \qquad \frac{}{\text{Hidden}\ (\texttt{Residual}\ \tau)} \qquad \frac{\text{Hidden}\ \tau_1 \quad \text{Hidden}\ \tau_2}{\text{Hidden}\ (\tau_1 \times \tau_2)}$$

$$\frac{\text{Hidden}\ \tau_2}{\text{Hidden}\ (\tau_1 \to \tau_2)} \qquad \frac{\text{Hidden}\ \tau}{\text{Hidden}\ (\texttt{Prob}\ \tau)} \qquad \frac{\text{Hidden}\ \tau}{\text{Hidden}\ (\texttt{PProb}\ \tau)}$$

Listing 4. Types and terms for $\lambda_{PP}$, a core calculus for factorized probabilistic programs.

partitions a coupled probabilistic program into *primal* and *residual* random choices, and a *partial probability evaluation* transformation $\mathcal{S}\{\cdot\}$ that allows the primal choices to be fixed ahead of time, ensuring that only residual choices are targeted by probabilistic inference.

### 5.1 Forming Factorized Couplings

To ensure sound factorizations, we desire that residual choices in a program do not affect primal choices: a *noninterference* property. To enforce this property, we introduce a new intermediate probabilistic language $\lambda_{PP}$ based on the classic dependency core calculus (DCC) of Abadi et al. [1], that will be the target of the transformation $C^*\{\cdot\}$.

**New Syntax for Factorization.** The language $\lambda_{PP}$ is an extension of $\lambda_P$ from §3. Listing 4 gives new syntactic constructs, using a gray shade to highlight constructs imported from the DCC and a yellow shade to highlight new probabilistic constructs. The first new type is $\texttt{Residual}\ \tau$, whose role is to represent residual values that are prohibited from influencing primal random choices. The type is equipped with an introduction form $\textbf{hide}\ t$ and a bind $x \leftarrow_{\mathrm{R}} t;\ t'$. These constructs are derived from specializing the DCC to the case of a two-point lattice. The second new type in $\lambda_{PP}$ is $\texttt{PProb}\ \tau$. As with $\texttt{Prob}\ \tau$, the type $\texttt{PProb}\ \tau$ is equipped with a monadic return $\textbf{return}_{\mathrm{PP}}\ t$ and bind $x \leftarrow_{\mathrm{PP}} t;\ t'$. The role of $\texttt{PProb}\ \tau$ is to represent probability distributions that *distinguish* primal random choices from residual ones. Accordingly, the type is equipped with two additional introduction forms $\textbf{primal}_{\mathrm{PP}}\ t$ and $\textbf{residual}_{\mathrm{PP}}\ t$ for lifting a term of type $\texttt{Prob}\ \tau$ into a term of type $\texttt{PProb}\ \tau$, marking the term as primal and residual, respectively. Finally, the Transparent and Hidden judgements identify types that carry only primal values and only residual values, respectively.

**Factorized Coupling Transformation.** The factorized coupling program transformation $C^*\{\cdot\}$, defined in Listing 5, acts similarly to $C\{\cdot\}$ from Listing 2, with two main differences: (i) the second

**Factorized Coupling Transformation $C^*\{\cdot\}$ from $\lambda_P$ to $\lambda_{PP}$**

$$C^*\{\sigma\} \coloneqq \sigma \times \texttt{Residual}\ \sigma \quad C^*\{\texttt{Prob}\ \tau\} \coloneqq \texttt{PProb}\ (C^*\{\tau\}) \quad C^*\{c\} \coloneqq (c, \textbf{hide}\ c) \quad C^*\{p\} \coloneqq \langle\text{defined per-primitive}\rangle$$
$$C^*\{\textbf{return}_{\texttt{P}}\ t\} \coloneqq \textbf{return}_{\texttt{PP}}\ C^*\{t\} \quad C^*\{x \leftarrow_{\texttt{P}} t;\ t'\} \coloneqq \{x \leftarrow_{\texttt{PP}} C^*\{t\};\ C^*\{t'\}\}$$
$$C^*\{\Gamma \vdash \textbf{if}\ t\ \textbf{then}\ t_1\ \textbf{else}\ t_2 : \tau\} \coloneqq \textbf{let}\ (b_A, \bar{b}_B) = C^*\{t\}; \mathcal{M}^*_\tau\{\textbf{if}\ b_A\ \textbf{then}\ C^*\{t_1\}\ \textbf{else}\ C^*\{t_2\}\}$$
$$\{b_B \leftarrow_{\texttt{R}} \bar{b}_B;\ \textbf{hide}\ (\textbf{if}\ b_B\ \textbf{then}\ C^*\{t_1\}\ \textbf{else}\ C^*\{t_2\})\}$$

**Erasure Transformation $\mathcal{E}\{\cdot\}$ from $\lambda_{PP}$ to $\lambda_P$**

$$\mathcal{E}\{\sigma\} \coloneqq \sigma \quad \mathcal{E}\{\texttt{Residual}\ \tau\} \coloneqq \mathcal{E}\{\tau\} \quad \mathcal{E}\{\texttt{PProb}\ \tau\} \coloneqq \texttt{Prob}\ \mathcal{E}\{\tau\} \quad \mathcal{E}\{p\} \coloneqq p$$
$$\mathcal{E}\{\textbf{hide}\ t\} \coloneqq \mathcal{E}\{t\} \quad \mathcal{E}\{\textbf{return}_{\texttt{PP}}\ t\} \coloneqq \mathcal{E}\{\textbf{return}_{\texttt{P}}\ t\} \quad \mathcal{E}\{\textbf{primal}_{\texttt{PP}}\ t\} \coloneqq \mathcal{E}\{t\} \quad \mathcal{E}\{\textbf{residual}_{\texttt{PP}}\ t\} \coloneqq \mathcal{E}\{t\}$$
$$\mathcal{E}\{x \leftarrow_{\texttt{R}} t;\ t'\} \coloneqq \{\textbf{let}\ x = \mathcal{E}\{t\};\ \mathcal{E}\{t'\}\} \quad \mathcal{E}\{x \leftarrow_{\texttt{PP}} t;\ t'\} \coloneqq \{x \leftarrow_{\texttt{P}} \mathcal{E}\{t\};\ \mathcal{E}\{t'\}\},$$

**Partial Probability Evaluation Transformation $\mathcal{S}\{\cdot\}$ from $\lambda_{PP}$ to $\lambda_P$**

$$\mathcal{S}\{\sigma\} \coloneqq \sigma \quad \mathcal{S}\{\texttt{Residual}\ \tau\} \coloneqq \mathcal{S}\{\tau\}$$
$$\mathcal{S}\{\texttt{Prob}\ \tau\} \coloneqq \texttt{Prob}\ \mathcal{S}\{\tau\} \times (\texttt{Seed} \to \mathcal{S}\{\tau\}) \quad \mathcal{S}\{\texttt{PProb}\ \tau\} \coloneqq \texttt{Seed} \to \texttt{Prob}\ \mathcal{S}\{\tau\}$$
$$\mathcal{S}\{\textbf{hide}\ t\} \coloneqq \mathcal{S}\{t\} \quad \mathcal{S}\{x \leftarrow_{\texttt{R}} t;\ t'\} \coloneqq \{\textbf{let}\ x = \mathcal{S}\{t\};\ \mathcal{S}\{t'\}\}$$
$$\mathcal{S}\{\textbf{primal}_{\texttt{PP}}\ t\} \coloneqq \lambda s.\{\textbf{return}_{\texttt{P}}\ ((\textbf{prj}_2\ \mathcal{S}\{t\})\ s)\} \quad \mathcal{S}\{\textbf{residual}_{\texttt{PP}}\ t\} \coloneqq \lambda\_.\{\textbf{prj}_1\ \mathcal{S}\{t\}\}$$
$$\mathcal{S}\{\textbf{return}_{\texttt{P}}\ t\} \coloneqq (\textbf{return}_{\texttt{P}}\ \mathcal{S}\{t\}, \lambda\_.\mathcal{S}\{t\}) \quad \mathcal{S}\{\textbf{return}_{\texttt{PP}}\ t\} \coloneqq \lambda\_.\{\textbf{return}_{\texttt{P}}\ \mathcal{S}\{t\}\}$$
$$\mathcal{S}\{x \leftarrow_{\texttt{P}} t;\ t'\} \coloneqq (\{x \leftarrow_{\texttt{P}} \textbf{prj}_1\ \mathcal{S}\{t\};\ \textbf{prj}_1\ \mathcal{S}\{t'\}\}, \lambda s.\{\textbf{let}\ (s_0, s_1) = \textbf{split}\ s;\ \textbf{let}\ x = (\textbf{prj}_2\ \mathcal{S}\{t\})\ s_0;\ (\textbf{prj}_2\ \mathcal{S}\{t'\})\ s_1\})$$
$$\mathcal{S}\{x \leftarrow_{\texttt{PP}} t;\ t'\} \coloneqq \lambda s.\{\textbf{let}\ (s_0, s_1) = \textbf{split}\ s;\ \{x \leftarrow_{\texttt{P}} \mathcal{S}\{t\}\ s_0;\ \mathcal{S}\{t'\}\ s_1\}\}$$

Listing 5. The factorized coupling, erasure, and partial probability transformations for $\lambda_P$ and $\lambda_{PP}$. The program transformations act homomorphically on all omitted types and terms.

component of each coupled pair is marked as residual; and (ii) the return and bind for $\texttt{Prob}\ \tau$ are replaced with the return and bind for $\texttt{PProb}\ \tau$. The macro $\mathcal{M}^*_\tau\{\cdot\}\{\cdot\}$ (defined in Appendix E) used in the **if** $t$ **then** $t_1$ **else** $t_2$ rule is a straightforward extension of the merge macro in Listing 2.

The soundness of $C^*\{\cdot\}$ can be justified by viewing it as an *annotated* version of the coupling transformation $C\{\cdot\}$. To formalize this idea, we also introduce the erasure program transformation $\mathcal{E}\{\cdot\}$ in Listing 5, which erases all type and term constructs responsible for separating primal and residual values. We can now state the following analogue of Theorem 1 for $C^*\{\cdot\}$.

Theorem 2 (Soundness of Factorized Coupling Transformation). *Let $\Gamma \vdash t : \tau$ be a well-typed term of $\lambda_P$. If each primitive $p$ appearing in $t$ satisfies $(\llbracket p \rrbracket, \llbracket p \rrbracket, \llbracket \mathcal{E}\{C^*\{p\}\} \rrbracket) \in R_C(\tau_p)$, then*

$$(\gamma_A, \gamma_B, \gamma_C) \in R_C(\Gamma) \implies \big(\llbracket t \rrbracket(\gamma_A), \llbracket t \rrbracket(\gamma_B), \llbracket \mathcal{E}\{C^*\{t\}\} \rrbracket(\gamma_C)\big) \in R_C(\tau). \tag{5.1}$$

**Example Factorized Coupling Primitives.** As with $C\{\cdot\}$, there is usually one clear definition for deterministic primitives, e.g., $C^*\{+\} \coloneqq \lambda((x_A, \bar{x}_B), (y_A, \bar{y}_B)).(x_A + y_A, \{x_B \leftarrow_{\texttt{R}} \bar{x}_B;\ y_B \leftarrow_{\texttt{R}} \bar{y}_B;\ \textbf{hide}\ (x_B + y_B)\})$. Listing 6 defines factorized couplings for the $\textbf{flip}_{\texttt{CRN}}$ and $\textbf{categorical}_{\texttt{MAX-INDEP}}$ primitives from §4. Let us unpack the case of $C^*\{\textbf{flip}_{\texttt{CRN}}\}$, whose type is given by

$$C^*\{\tau_{\textbf{flip}_{\texttt{CRN}}}\} = \texttt{Real} \times \texttt{Residual}\ \texttt{Real} \to \texttt{PProb}\ (\texttt{Bool} \times \texttt{Residual}\ \texttt{Bool}). \tag{5.2}$$

In the implementation of $C^*\{\textbf{flip}_{\texttt{CRN}}\}$, the flip operation that samples $b_A$ is wrapped in $\textbf{primal}_{\texttt{PP}}$. That the typing rule for $\textbf{primal}_{\texttt{PP}}\ t$ in Listing 4 expects a *transparent* type ensures that this sample does not depend on any residual values. In contrast, the operations that sample $\bar{b}_B$ are wrapped in $\textbf{residual}_{\texttt{PP}}\ t$. That the typing rule for $\textbf{residual}_{\texttt{PP}}\ t$ in Listing 4 expects a *hidden* type ensures that this sample does not determine any primal values. Recall that this residual sampling step will be targeted by inference: the ENUM annotation is a hint to the inference algorithm (explained in §6).

$C^*\{\mathbf{flip}_{\mathsf{CRN}}\} \coloneqq \lambda(p_A : \mathtt{Real}, \bar{p}_B : \mathtt{Residual\ Real}).$
$\quad b_A \leftarrow_{\mathbf{PP}} \mathbf{primal}_{\mathbf{PP}}\ (\mathbf{flip}\ p_A)$
$\quad \bar{b}_B \leftarrow_{\mathbf{PP}} \mathbf{residual}_{\mathbf{PP}}$
$\quad\quad p_B \leftarrow_{\mathbf{R}} \bar{p}_B$
$\quad\quad \mathbf{let}\ r = \mathbf{if}\ b_A\ \mathbf{then}\ \min\left(\frac{p_B}{p_A}, 1\right)\ \mathbf{else}\ \max\left(\frac{p_B - p_A}{1 - p_A}, 0\right)$
$\quad\quad \mathbf{hide}\ (\mathbf{flip}_{\mathsf{ENUM}}\ r)$
$\quad \mathbf{return}_{\mathbf{PP}}\ (b_A, \bar{b}_B)$

$\mathcal{E}\{C^*\{\mathbf{flip}_{\mathsf{CRN}}\}\} \coloneqq \lambda(p_A : \mathtt{Real}, p_B : \mathtt{Real}).$
$\quad b_A \leftarrow_{\mathbf{P}} \mathbf{flip}\ p_A$
$\quad \mathbf{let}\ r = \mathbf{if}\ b_A\ \mathbf{then}\ \min\left(\frac{p_B}{p_A}, 1\right)\ \mathbf{else}\ \max\left(\frac{p_B - p_A}{1 - p_A}, 0\right)$
$\quad b_B \leftarrow_{\mathbf{P}} \mathbf{flip}_{\mathsf{ENUM}}\ r$
$\quad \mathbf{return}_{\mathbf{P}}\ (b_A, b_B)$

$C^*\{\mathbf{categorical}_{\mathsf{MAX\text{-}INDEP}\,\alpha,n}\} \coloneqq$
$\lambda(z : (\alpha \times \mathtt{Residual}\ \alpha)^n, p : (\mathtt{Real} \times \mathtt{Residual\ Real})^n).$
$\quad \mathbf{let}\ (z_A, p_A) = (\mathbf{map}\ \mathbf{prj}_1\ z, \mathbf{map}\ \mathbf{prj}_1\ p)$
$\quad i_A \leftarrow_{\mathbf{PP}} \mathbf{primal}_{\mathbf{PP}}\ (\mathbf{categorical}\ ((1, \ldots, n), p_A))$
$\quad \mathbf{let}\ x_A = \mathbf{index}\ z_A\ i_A$
$\quad \bar{x}_B \leftarrow_{\mathbf{PP}} \mathbf{residual}_{\mathbf{PP}}$
$\quad\quad p_B \leftarrow_{\mathbf{R}} \mathbf{sequence}_{\mathbf{R}}\ (\mathbf{map}\ \mathbf{prj}_2\ p)$
$\quad\quad z_B \leftarrow_{\mathbf{R}} \mathbf{sequence}_{\mathbf{R}}\ (\mathbf{map}\ \mathbf{prj}_2\ z)$
$\quad\quad b \leftarrow_{\mathbf{P}} \mathbf{flip}\ (\min(1, (\mathbf{index}\ p_B\ i_A)/(\mathbf{index}\ p_A\ i_A)))$
$\quad\quad \mathbf{if}\ b\ \mathbf{then}\ (\mathbf{hide}\ (\mathbf{return}_{\mathbf{P}}\ (\mathbf{index}\ z_B\ i_A)))\ \mathbf{else}$
$\quad\quad\quad \mathbf{let}\ f = (\lambda(p_A, p_B).\max(0, p_B - p_A))$
$\quad\quad\quad \mathbf{let}\ r = \mathbf{map}\ f\ (\mathbf{zip}\ p_A\ p_B)$
$\quad\quad\quad \mathbf{hide}\ (\mathbf{categorical}_{\mathsf{ENUM}}\ (z_B, \mathbf{normalize}\ r))$
$\quad \mathbf{return}_{\mathbf{PP}}\ (x_A, \bar{x}_B)$

Listing 6. Example definitions of the factorized coupling transformation for two primitives from Listing 3.

Listing 6 also shows the erasure of $C^*\{\mathbf{flip}_{\mathsf{CRN}}\}$. It is readily verified that $\mathcal{E}\{C^*\{\mathbf{flip}_{\mathsf{CRN}}\}\}$ has the same denotation as $C\{\mathbf{flip}_{\mathsf{CRN}}\}$ in Listing 3, and thus $C^*\{\mathbf{flip}_{\mathsf{CRN}}\}$ is a sound factorized coupling.

## 5.2 Partial Probability Evaluation

To make use of the factorized couplings produced by $C^*\{\cdot\}$ in gradient inference, we need a way to compile them into $\lambda_P$ programs in which only the residual choices remain probabilistic.

**Partial Probability Evaluation Transformation.** The *partial probability evaluation* program transformation $\mathcal{S}\{\cdot\}$ from $\lambda_{PP}$ to $\lambda_P$ is defined in Listing 5. The key idea is to use a shared seed to determine the output of all primal choices, but to ignore this seed when sampling residual choices. The definition uses a Seed type and helper primitives **seed** and **split** for sampling and splitting seeds, where $\tau_{\mathbf{seed}} = \mathtt{Prob\ Seed}$ and $\tau_{\mathbf{split}} = \mathtt{Seed} \to \mathtt{Seed} \times \mathtt{Seed}$. Concretely, we set $[\![\mathtt{Seed}]\!] = \mathbb{R}$, $[\![\mathbf{seed}]\!] = \mathrm{Unif}(0, 1)$, and $[\![\mathbf{split}]\!]$ to any measure-preserving map from $\mathbb{R}$ to $\mathbb{R} \times \mathbb{R}$.

The transformation $\mathcal{S}\{\cdot\}$ behaves similarly to the erasure transformation on nonprobabilistic terms. However, the key difference is that $\mathcal{S}\{\mathtt{PProb}\ \tau\}$ is defined as a function *from seeds to probability distributions*. The translation of the introduction form $\mathbf{residual}_{\mathbf{PP}}\ t$ *ignores* the seed, preserving the original distribution. The translation of $\mathbf{primal}_{\mathbf{PP}}\ t$ uses *only* the seed, forming a Dirac probability distribution given any fixed seed. To enable this behavior, the transformation on $\mathtt{Prob}\ \tau$ is modified to create and propagate both ordinary and seeded versions of each distribution: the former is of type $\mathtt{Prob}\ \tau$ and used in $\mathcal{S}\{\mathbf{residual}_{\mathbf{PP}}\ t\}$, and the latter is of type $\mathtt{Seed} \to \tau$ and used in $\mathcal{S}\{\mathbf{primal}_{\mathbf{PP}}\ t\}$.

**Soundness of Partial Probability Evaluation.** The following theorem guarantees that if $\mathcal{S}\{\cdot\}$ is soundly implemented on each primitive, then the transformation forms a sound factorization of any input program. In particular, sampling a seed (which fixes the primal choices) followed by sampling the residual choices in the transformed program $\mathcal{S}\{t\}$ is equivalent (in distribution) to sampling all choices jointly using the unannotated original program $\mathcal{E}\{t\}$. The theorem is established using a logical relation $R_{\mathcal{S}}$ (defined in Appendix E), where $(x_{\mathcal{E}}, x_{\mathcal{S}}) \in R_{\mathcal{S}}(\tau)$ when $x_{\mathcal{S}} : [\![\mathcal{S}\{\tau\}]\!]$ is a sound partial probability evaluation of $x_{\mathcal{E}} : [\![\mathcal{E}\{\tau\}]\!]$.

Theorem 3 (Soundness of Partial Probability Evaluation). *Let $\Gamma \vdash t : \mathtt{PProb}\ \tau$ be a well-typed term of $\lambda_{PP}$, where the type $\tau$ is constructed from base types, products, functions, and the* Residual *$\tau$ constructor. If each primitive $p$ appearing in $t$ satisfies $([\![p]\!], [\![\mathcal{S}\{p\}]\!]) \in R_{\mathcal{S}}(\tau_p)$, then*

$$(\gamma_{\mathcal{E}}, \gamma_{\mathcal{S}}) \in R_{\mathcal{S}}(\Gamma) \implies [\![\mathcal{E}\{t\}]\!](\gamma_{\mathcal{E}}) = \mathbf{seed} \gg\!=_{\mathbf{P}} [\![\mathcal{S}\{t\}]\!](\gamma_{\mathcal{S}}). \tag{5.3}$$

## 6 Gradient Inference via Composition with Inference and AD

We complete the gradient inference workflow by addressing challenge (C3), formalizing how the transformations $C^*\{\cdot\}$ and $\mathcal{S}\{\cdot\}$ from §5 can be composed with inference and automatic differentiation (AD) to yield a sound gradient estimator.

**Soundness of Inference.** Define the map **expect** : $P(\mathbb{R}) \to \mathbb{R}$ by $\mathbf{expect}(\mu) \coloneqq \int_{\mathbb{R}} x\mu(\mathrm{d}x)$. An *inference macro* $\mathcal{I}\{\cdot\}$ acts on a term $\Gamma \vdash t : \texttt{Prob Real}$ of $\lambda_P$ and produces a term $\Gamma \vdash \mathcal{I}\{t\} : \texttt{Prob Real}$. We say that $\mathcal{I}\{\cdot\}$ is *sound* on $t$ if it preserves its expectation, i.e., for all $\gamma : [\![\Gamma]\!]$,

$$\mathbf{expect}\left([\![\mathcal{I}\{t\}]\!](\gamma)\right) = \mathbf{expect}\left([\![t]\!](\gamma)\right). \tag{6.1}$$

The partial evaluation transformation $\mathcal{S}\{\cdot\}$ ensures that the inference target program does not contain any intractable conditioning or "observe" operations. Thus, (6.1) requires an inference algorithm that soundly estimates the expectation of the return value of a probabilistic program, rather than a *conditional* expectation involving an intractable normalization constant. This flexibility lets us reuse many standard inference algorithms to produce provably unbiased gradient estimators. We briefly review three such inference algorithms below, employed in our applications.

- *Variable elimination* [45, §9] is an exact inference algorithm based on successively marginalizing latent variables. When a variable $x$ is eliminated, the joint distribution over its dependencies $\mathbf{y}$ is updated by summing out all possible values of $x$: $p(\mathbf{y}) = \sum_x p(\mathbf{y}, x)$. The final result is the marginal distribution over the program's return value, from which the expectation is computed.
- *Stratified importance resampling* [69, §8] partitions the probability space into events $A_0, A_1, \ldots, A_n$, known as *strata*. The expectation $\mathbb{E}[X]$ of the program's return value can be unbiasedly estimated by separately sampling $X_i \sim X \mid A_i$ from each stratum and computing $\sum_{i=0}^{n} X_i \mathbb{P}(A_i)$. The computational cost can be reduced by resampling, wherein a random subset of the summands are evaluated and reweighted to preserve unbiasedness.
- *Sequential Monte Carlo* [18] propagates a set of weighted particles through the program, performing a combination of transition and resampling steps. These moves are designed to preserve *proper weightedness* [59, §2], which is a property that enables unbiased estimation of integrals. Once a set of properly weighted particles $(x_1, w_1), \ldots, (x_k, w_k)$ for the program's return value has been obtained, the expectation can be unbiasedly estimated by $\sum_{i=1}^{k} x_i w_i$.

To support the gradient inference applications in §7, we use the monad-bayes library developed by Ścibior et al. [77], which is backed by a rigorous denotational semantics for functional and higher-order probabilistic programs [78]. We also include inference annotations in the output of the GradInf transformations that can be used by monad-bayes inference algorithms. For example, the primitive $\mathbf{flip}_{\textsf{ENUM}}$ indicates to enumerate (i.e., stratify) the two possible outputs of **flip**.

**Soundness of AD.** After probabilistic inference is applied, the final step of gradient inference is to apply AD. An *AD macro* $\mathcal{D}\{\cdot\}$ acts on a term $\Gamma \vdash t : \texttt{Real}^n \to \texttt{Prob Real}$ of $\lambda_P$ and produces a term $\Gamma \vdash \mathcal{D}\{t\} : \texttt{Real}^n \to \texttt{Prob Real}^n$. We say that $\mathcal{D}\{\cdot\}$ is *sound* on $t$ if for all $\gamma : [\![\Gamma]\!]$ and $\theta : \mathbb{R}^n$,

$$\mathbf{expect}\left([\![\mathcal{D}\{t\}]\!](\gamma)(\theta)\right) = \nabla_\theta \left[\mathbf{expect}\left([\![t]\!](\gamma)(\theta)\right)\right]. \tag{6.2}$$

Gradient inference workflows rely only on *standard* AD algorithms. Several works formalize and prove AD transformations correct, including for higher-order programs [39, 46]. We use the Haskell libraries ad [44] for forward-mode and backprop [42] for reverse-mode. In our setting, these AD transformations can safely ignore the random choices made by the probabilistic inference algorithm (i.e., treat their outputs as constants) so long as they do not depend on the finite perturbation $\varepsilon$ being differentiated with respect to (cf. (2.6)). Prior work has developed soundness guarantees for AD in such settings [41, 52, 53, 55], e.g., the smoothness analysis system of Lee et al. [52].

(a) Original Program $t$

$\lambda(\theta : \texttt{Real}).$
  $b \leftarrow_{\mathbf{P}} \mathbf{flip}_{\mathsf{CRN}}\ \theta$
  $\mathbf{return}_{\mathbf{P}}\ (\mathbf{if}\ b\ \mathbf{then}\ 0.5\ \mathbf{else}\ 0.3)$

(b) Inference Target $t_{\mathcal{S}} \coloneqq \mathcal{S}\{C^*\{t\}\}$

$\lambda(\theta_A : \texttt{Real}, \theta_B : \texttt{Real}).\lambda(s : \texttt{Seed}).$
  $\mathbf{let}\ b_A = s < \theta_A$
  $\mathbf{let}\ r = \mathbf{if}\ b_A\ \mathbf{then}\ \min(\theta_B/\theta_A, 1)$
    $\mathbf{else}\ \max((\theta_B - \theta_A)/(1 - \theta_A), 0)$
  $b_B \leftarrow_{\mathbf{P}} \mathbf{flip}_{\mathsf{ENUM}}\ r$
  $\mathbf{return}_{\mathbf{P}}\ (\mathbf{if}\ b_A\ \mathbf{then}\ 0.5\ \mathbf{else}\ 0.3, \mathbf{if}\ b_B\ \mathbf{then}\ 0.5\ \mathbf{else}\ 0.3)$

(c) Difference Estimator $\mathcal{G}_\Delta\{t\}$

$\lambda(\theta_A : \texttt{Real}, \theta_B : \texttt{Real}).$
  $s \leftarrow_{\mathbf{P}} \mathbf{seed}$
  $\mathbf{let}\ b_A = s < \theta_A$
  $\mathbf{let}\ r = \mathbf{if}\ b_A\ \mathbf{then}\ \min(\theta_B/\theta_A, 1)$
    $\mathbf{else}\ \max((\theta_B - \theta_A)/(1 - \theta_A), 0)$
  $\mathbf{let}\ (b_{B,1}, w_1) = (\mathbf{true}, r)$
  $\mathbf{let}\ (b_{B,2}, w_2) = (\mathbf{false}, 1 - r)$
  $\mathbf{return}_{\mathbf{P}}\ w_1 \times (\mathbf{if}\ b_{B,1}\ \mathbf{then}\ 0.5\ \mathbf{else}\ 0.3)$
    $+ w_2 \times (\mathbf{if}\ b_{B,2}\ \mathbf{then}\ 0.5\ \mathbf{else}\ 0.3)$
    $- (w_1 + w_2) \times (\mathbf{if}\ b_A\ \mathbf{then}\ 0.5\ \mathbf{else}\ 0.3)$

(d) Gradient Estimator $\mathcal{G}\{t\}$

$\lambda(\theta : \texttt{Real}).$
  $s \leftarrow_{\mathbf{P}} \mathbf{seed}$
  $\mathbf{let}\ b_A = s < \theta$
  $\mathbf{let}\ (r, \dot{r}) = \begin{pmatrix}\mathbf{if}\ b_A\ \mathbf{then}\ 1\ \mathbf{else}\ 0,\\ \mathbf{if}\ b_A\ \mathbf{then}\ 0\ \mathbf{else}\ 1/(1-\theta)\end{pmatrix}$
  $\mathbf{let}\ (b_{B,1}, (w_1, \dot{w}_1)) = (\mathbf{true}, (r, \dot{r}))$
  $\mathbf{let}\ (b_{B,2}, (w_2, \dot{w}_2)) = (\mathbf{false}, (1 - r, -\dot{r}))$
  $\mathbf{return}_{\mathbf{P}}\ \dot{w}_1 \times (\mathbf{if}\ b_{B,1}\ \mathbf{then}\ 0.5\ \mathbf{else}\ 0.3)$
    $+ \dot{w}_2 \times (\mathbf{if}\ b_{B,2}\ \mathbf{then}\ 0.5\ \mathbf{else}\ 0.3)$
    $- (\dot{w}_1 + \dot{w}_2) \times (\mathbf{if}\ b_A\ \mathbf{then}\ 0.5\ \mathbf{else}\ 0.3)$

Listing 7. Example derivation of a gradient estimator via gradient inference, for a simple input program $t$ that flips a biased coin and returns a value based on the outcome.

**Gradient Inference Macro.** The overall gradient inference workflow is enacted by the macro $\mathcal{G}\{\cdot\}$. Given an inference macro $\mathcal{I}\{\cdot\}$, we first define a helper *difference macro* $\mathcal{G}_\Delta\{\cdot\}$, acting on a term $\bullet \vdash t : \texttt{Real}^n \to \texttt{Prob Real}$ and producing a term $\bullet \vdash \mathcal{G}_\Delta\{t\} : (\texttt{Real} \times \texttt{Real})^n \to \texttt{Prob Real}$,

$$\mathcal{G}_\Delta\{t\} \coloneqq \lambda\theta.\{s \leftarrow_{\mathbf{P}} \mathbf{seed};\ \mathcal{I}\{(x_A, x_B) \leftarrow_{\mathbf{P}} \mathcal{S}\{C^*\{t\}\}\ \theta\ s;\ \mathbf{return}_{\mathbf{P}}\ (x_B - x_A)\}\}. \tag{6.3}$$

Given also an AD macro $\mathcal{D}\{\cdot\}$, we now define the *gradient inference macro* $\mathcal{G}\{\cdot\}$, acting on a term $\bullet \vdash t : \texttt{Real}^n \to \texttt{Prob Real}$ and producing a term $\bullet \vdash \mathcal{G}\{t\} : \texttt{Real}^n \to \texttt{Prob Real}^n$,

$$\mathcal{G}\{t\} \coloneqq \lambda(\theta_1, \ldots, \theta_n).\big(\mathcal{D}\{\lambda(\varepsilon_1, \ldots, \varepsilon_n).\mathcal{G}_\Delta\{t\}\ ((\theta_1, \theta_1 + \varepsilon_1), \ldots, (\theta_n, \theta_n + \varepsilon_n))\}\ (0, \ldots, 0)\big). \tag{6.4}$$

The term $\mathcal{G}_\Delta\{t\}$ in (6.3) first samples a seed $s$ to fix the values of the primal random choices in $t_{\mathcal{S}} = \mathcal{S}\{C^*\{t\}\}$. Then, inference is applied to the residual choices in $t_{\mathcal{S}}$ to produce an estimator of the expected difference $x_B - x_A$ of the coupled pair of return values $(x_A, x_B)$. To form the gradient estimator, the term $\mathcal{G}\{t\}$ in (6.4) constructs an anonymous function that accepts perturbations $\varepsilon_1, \ldots, \varepsilon_n$ to the components of the parameter $\theta$. AD is then applied to this function at $\varepsilon_1 = \cdots = \varepsilon_n = 0$. Listing 7 illustrates each step of gradient inference on a toy program, using forward-mode AD via dual numbers. The inference step (Listing 7b to Listing 7c) enumerates the two possible outcomes of the sample from $\mathbf{flip}_{\mathsf{ENUM}}$ in the inference target program $t_{\mathcal{S}}$, while the AD step (Listing 7c to Listing 7d) differentiates the enumerated weights to produce a gradient estimator. The soundness theorem for the full GradInf workflow is as follows.

Theorem 4 (Soundness of Gradient Inference Macro). *Let $\bullet \vdash t : \texttt{Real}^n \to \texttt{Prob Real}$ be a well-typed term of $\lambda_P$, where each primitive $p$ appearing in $t$ satisfies $([\![p]\!], [\![p]\!], [\![\mathcal{E}\{C^*\{p\}\}]\!]) \in R_C(\tau_p)$ and $([\![\mathcal{E}\{p\}]\!], [\![\mathcal{S}\{p\}]\!]) \in R_{\mathcal{S}}(\tau_p)$. If the inference macro applied in (6.3) is sound (per (6.1)) and the AD macro applied in (6.4) is sound (per (6.2)), then for any $\theta : \mathbb{R}^n$,*

$$\mathbf{expect}\left([\![\mathcal{G}\{t\}]\!](\theta)\right) = \nabla_\theta \left[\mathbf{expect}([\![t]\!](\theta))\right]. \tag{6.5}$$

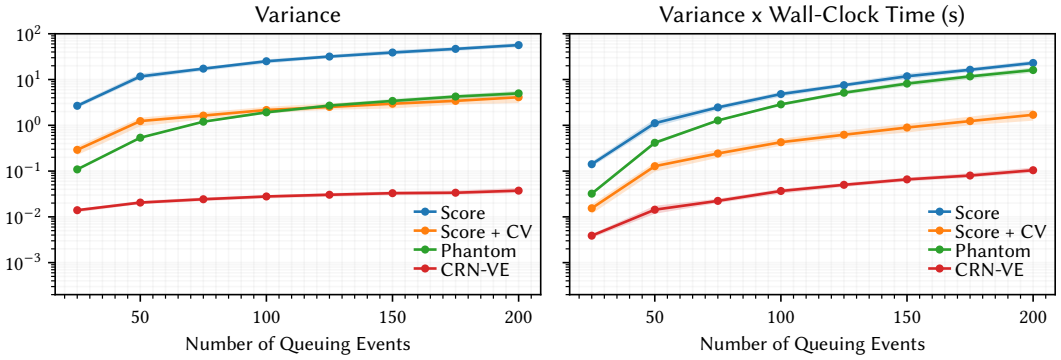



Fig. 4. Variance scaling plots for gradient estimators expressed by GradInf on M/M/c queue model.

## 7 Applications

We investigate the extent to which GradInf achieves its design goals A1 and A2 from §1.2, by considering the following research questions:

- **(Q1)** Can GradInf express novel estimators that deliver lower variance than state-of-the-art baseline estimators in the literature?
- **(Q2)** Do the estimators in GradInf empirically exhibit zero bias, as the theory predicts?
- **(Q3)** How large is the constant-factor runtime overhead in the prototype implementation of GradInf (Haskell) compared to ADEV [55] (Haskell) and Storchastic [83] (PyTorch)?

We study these questions in the context of three challenging case studies: the M/M/c queuing model (§7.2.1); an option pricing model (§7.2.2); and a gene transcription model (§7.2.3). The results confirm that GradInf meets its design goals, where GradInf is able to express state-of-the-art existing gradient estimators and improve upon them with novel estimators that achieve up to 370x reductions in time-adjusted variance. Appendix B presents a number of additional examples.

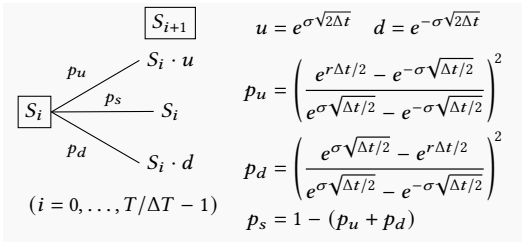
### 7.1 Experimental Setup

We implemented a prototype of GradInf in Haskell (https://github.com/probsys/grad-inf). The workflow begins with a user-written probabilistic program $t$ similar to the one in Fig. 2a, which uses GradInf probabilistic primitives to define a probability distribution over a real output. GradInf synthesizes an inference target program $t_S$ from $t$, to which we apply probabilistic inference algorithms using the monad-bayes library [78]. The final automatic differentiation step uses the ad library [44] (for forward-mode AD) or the backprop library [42] (for reverse-mode AD). To measure the performance of gradient estimators, we empirically measure both per-sample variance and the product of variance and wall-clock time. Because variance decreases linearly with the number of samples, the latter metric measures the computational work needed to obtain an estimate with a desired Monte Carlo error. Error bands with 95% confidence intervals are provided for variance plots, which are computed via bootstrapping. Appendix G gives a full list of experiment parameters.

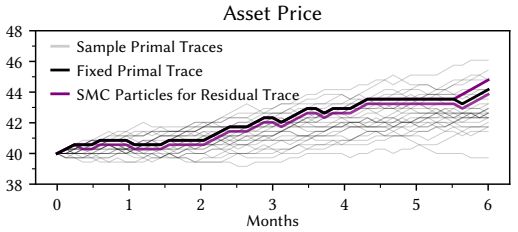
### 7.2 (Q1) Developing Novel Estimators with Low Variance

*7.2.1 Sensitivity Analysis of Queuing Model (Overview Example).* We first revisit the application of GradInf to the M/M/c queue model from §2.1. By combining a $\mathbf{flip}_{\mathsf{CRN}}$ primitive with different inference algorithms, GradInf can express the randomized phantom estimator [25] and enable a novel low-variance estimator (GradInf-CRN-VE) based on variable elimination.

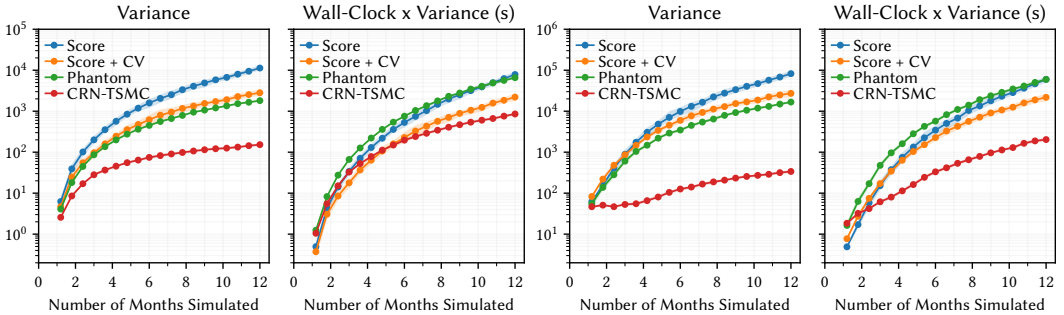
Figure 4 provides variance and time-adjusted variance asymptotics against the number of queuing events $n$ for four estimators expressed by GradInf: two score-function based estimators, with and




(a) Generative model




(b) Twisted SMC estimator via gradient inference




(c) Variance scaling asymptotics for $\kappa$

(d) Variance scaling asymptotics for $\rho$

Fig. 5. Applying GradInf to an option pricing model from computational finance.

without a control variate (CV) set equal to the empirical mean of the objective; the randomized phantom estimator; and the novel estimator GradInf-CRN-VE. The results show that GradInf-CRN-VE exhibits reduced variance relative to existing methods (e.g., 110x reduction at $n = 200$). Since all four estimators have runtime linear in the number of queuing events $n$, this variance reduction translates into improvements in time-adjusted variance (e.g., 16x reduction at $n = 200$).

*7.2.2 Estimating Greeks of Stochastic Option Pricing Models.* Sensitivities of the expected payoff of a financial option are of fundamental interest in mathematical finance, where they are known as "Greeks" [28, §7]. Figure 5a illustrates a widely used option pricing model known as the trinomial model [16]. The model evolves the price $S_i$ of an asset at discrete time steps $i\Delta T$ ($i = 1, 2, \ldots$) via a multiplicative random walk, whose kernel depends on the interest rate $r$ and price volatility $\sigma$. We consider a European-style option, in which an asset can be bought at a fixed time $T$ at price $K$. The quantity $\ell \coloneqq \mathbb{E}[e^{-rT} \max(S_{T/\Delta T} - K, 0)]$ gives the expected payoff of the option. We apply GradInf to the trinomial model to estimate the Greeks $\kappa \coloneqq \partial\ell/\partial\sigma$ and $\rho \coloneqq \partial\ell/\partial r$, with parameters $r = 15\%/\text{year}$, $\sigma = 5\%/\text{year}$, $S_0 = 40$, and $K = 41$.

In GradInf, we express the kernel of the random walk using a **categorical**$_{\text{CRN}}$ primitive, which uses a CRN-based factorized coupling (see Appendix B). Using the same stratification-based scheme as for the M/M/c queue from §7.2.1, GradInf produces an SPA phantom estimator, which is state-of-the-art for discrete option pricing models [28]. However, as Figs. 5c and 5d show, the phantom estimator gives limited performance improvements compared to a simpler score-based method with control variates. We thus swap in a more sophisticated inference algorithm based on twisted SMC [18, §10.3.3] (details in Appendix B.7). Figure 5b depicts the final SMC particles from a particular inference run, with opacities corresponding to particle weights. Figures 5c and 5d show that the resulting novel estimator (GradInf-CRN-TSMC) achieves significant reductions in time-adjusted variance as the time period $T$ grows (e.g., 11x reduction for $\rho$ at $T = 1.0$ year).

$$S \coloneqq \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}$$

$(M_0, P_0) \coloneqq$ <initial values>

$t_0 \coloneqq 0$

for $i = 1, 2, 3, \ldots$

$\mathbf{r}_i \coloneqq (\alpha, \beta M_{i-1}, \gamma M_{i-1}, \delta P_{i-1})$

$s \coloneqq \mathbf{r}_{i,0} + \mathbf{r}_{i,1} + \mathbf{r}_{i,2} + \mathbf{r}_{i,3}$

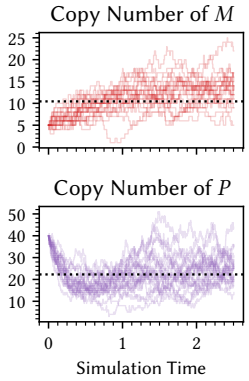
$k_i \sim \mathrm{Cat}(\mathbf{r}_i) \quad \Delta t_i \sim \mathrm{Exp}(s)$

$t_i \coloneqq t_{i-1} + \Delta t_i$

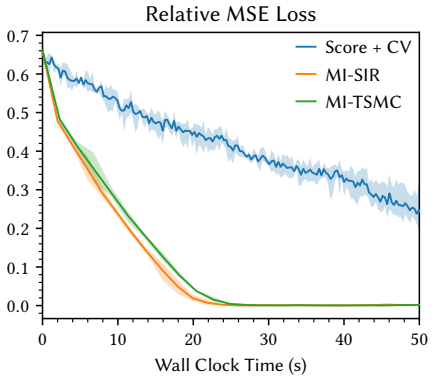
$M_i \coloneqq M_{i-1} + S_{0,k_i}$

$P_i \coloneqq P_{i-1} + S_{1,k_i}$

(a) Generative model




(b) Model traces




(c) Stochastic gradient descent traces

Fig. 6. Applying GradInf to infer the hidden parameters of a probabilistic gene transcription model.

Table 2. Variance of gradient estimators on gene transcription model at $\alpha = \beta = \gamma = \delta = 1$ with $k = 100$. Reductions in variance and time-adjusted variance are given with respect to the Score + CV baseline.

| | Variance | | | Variance Reduction | | Variance Reduction (Time Adj.) | |
|---|---|---|---|---|---|---|---|
| Rate | Score + CV | MI-SIR | MI-TSMC | of MI-SIR | of MI-TSMC | of MI-SIR | of MI-TSMC |
| $\alpha$ | $1.7 \cdot 10^{-1}$ | $1.6 \cdot 10^{-4}$ | $5.8 \cdot 10^{-5}$ | **1100x** | **2900x** | **160x** | **370x** |
| $\beta$ | $2.0 \cdot 10^{-1}$ | $5.2 \cdot 10^{-4}$ | $4.1 \cdot 10^{-4}$ | **380x** | **490x** | **61x** | **67x** |
| $\gamma$ | $2.0 \cdot 10^{-1}$ | $1.0 \cdot 10^{-3}$ | $4.4 \cdot 10^{-4}$ | **190x** | **450x** | **30x** | **61x** |
| $\delta$ | $3.9 \cdot 10^{-1}$ | $3.1 \cdot 10^{-3}$ | $2.6 \cdot 10^{-3}$ | **130x** | **150x** | **19x** | **20x** |

*7.2.3 Parameter Inference of Gene Transcription Model.* We apply GradInf to the task of parameter inference of a chemical reaction network (RN) model of gene transcription. RN models have seen broad adoption in systems biology for simulating the dynamics of biochemical reactions [17]. Following Anderson [4], we consider a chemical RN model of the copy numbers of mRNA $M$ and protein $P$, with four reactions $\emptyset \to^{\alpha} M$, $M \to^{\beta} M + P$, $M \to^{\gamma} \emptyset$, and $P \to^{\delta} \emptyset$, corresponding to gene transcription, mRNA translation, mRNA degradation, and protein degradation, respectively. Each reaction determines a Poisson process in time, with rate given as the product of the rate parameter ($\alpha, \beta, \gamma$ or $\delta$) and the current copy number of the reactant species. We simulate the RN using the Gillespie algorithm [26] shown in Fig. 6a, which relies on a *discrete* sampling step from a categorical distribution over the possible reactions. Our goal is to estimate the four parameters $(\alpha, \beta, \gamma, \delta)$ that reproduce observed mean copy numbers $\hat{M}$ and $\hat{P}$ of mRNA and protein (Fig. 6b, dotted lines), produced at hidden ground truth parameters. We minimize a relative mean squared error (MSE) loss $\ell \coloneqq \mathbb{E}\left[(\tilde{M} - \hat{M})^2/\hat{M}^2 + (\tilde{P} - \hat{P})^2/\hat{P}^2\right]$ between $k$-sample Monte Carlo estimates $\tilde{M}$ and $\tilde{P}$ of the mean copy numbers and their ground truth values $\hat{M}$ and $\hat{P}$.

The score function estimator, called the "Girsanov transformation method" in the context of chemical RNs, is the state-of-the-art unbiased estimator in this setting but still exhibits high variance [85]. We apply GradInf to contribute two novel estimators to the literature. The first combines the **categorical**$_{\texttt{MAX-INDEP}}$ primitive with stratified sampling (GradInf-MI-SIR). The second employs a twisted sequential Monte Carlo algorithm (GradInf-MI-TSMC), whose details are given in Appendix B.9. Table 2 shows the variance of these gradient estimation schemes when differentiating the objective $\ell$ at fixed parameters, using synthetic observation data generated at hidden parameters $\alpha = 18$, $\beta = 8$, $\gamma = 1.5$, and $\delta = 4$ with simulation time period $T = 2.5$. GradInf-MI-SIR improves efficiency by 19x–160x, while GradInf-MI-TSMC improves efficiency by 20x–370x. These variance reductions lead to better optimization performance using stochastic gradient descent (SGD) on the

loss $\ell$ (Fig. 6c). With a fixed computational budget, both GradInf-MI-SIR and GradInf-MI-TSMC find parameters that reproduce the observed data (relative MSEs of 0.2% and 0.1%, respectively) more accurately than the parameters found using the score-based estimator (relative MSE of 26%).

### 7.3 (Q2) Verifying Unbiased Estimation

We empirically verify that the novel estimators, while lower variance, remain unbiased as guaranteed by Theorem 4. Figure 7 shows box plots of samples of gradient estimates from the novel estimators and standard baselines. The case study parameters are scaled down to ensure that the baselines are sufficiently low variance to serve as accurate ground truths. The empirical means of the estimates are labeled in each plot. A paired equivalence test [48] verifies that the empirical means of the novel estimators GradInf-CRN-VE, GradInf-CRN-TSMC, GradInf-MI-SIR, and GradInf-MI-TSMC are all statistically indistinguishable from those of the baselines ($p < 1\%$, tolerance $\Delta = 5\%$).

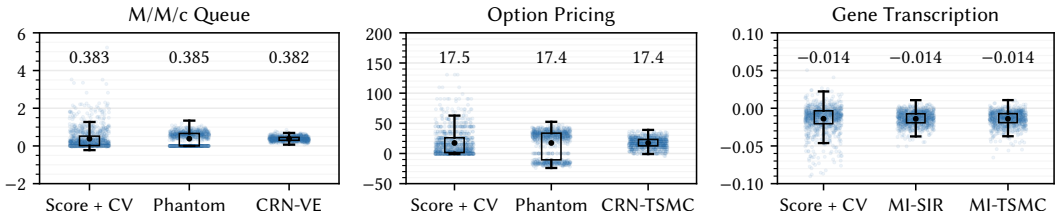



Fig. 7. Gradient estimate samples from the estimators in each case study, with their empirical means reported.

### 7.4 (Q3) Quantifying Constant-Factor Runtime Overhead

We quantify the runtime overhead of our prototype implementation of GradInf in Haskell by analyzing the score estimator, which is a standard baseline that is expressible in systems such as ADEV [55] and Storchastic [83]. Figure 8 compares the runtime of the score estimator across the systems in two settings: the M/M/c queue model (§7.2.1) and the option pricing model (§7.2.2). Both models feature a large number of loop iterations, leading to deep computation graphs. The runtime performance is considered as a function of problem size. GradInf has improved runtime scaling relative to Storchastic and is competitive with ADEV (within 3x runtime overhead for the queue model, and 6x for the option pricing model).

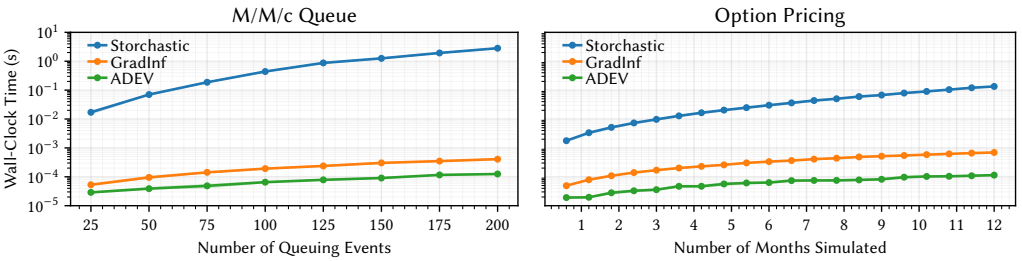



Fig. 8. Runtime scaling of the score estimator for different gradient estimation systems.

## 8 Related Work

**Differentiable Programming Systems.** Widely used libraries for probabilistic and differentiable programming such as Pyro [14] and TensorFlow Probability [23] provide users with predefined gradient estimators based on REINFORCE or the reparameterization trick, but do not support the design or application of most estimators in Table 1. Research libraries such as ADEV [12, 55], StochasticAD.jl [6], and Storchastic [83] provide modular approaches to deriving gradient estimators of probabilistic programs. These libraries build gradient estimators for whole programs by

composing gradient estimators attached to each primitive. However, many estimators of interest do not arise as compositions of per-primitive estimators (e.g., the randomized phantom estimator [25]). A key insight in GradInf is to attach to each primitive a factorized coupling, rather than an entire gradient estimator. This modular design permits the subsequent application of probabilistic inference and the construction of novel estimators as in §7.

**Gradient Estimation Methods.** Gradient estimation has a vast literature, across which the design, practice, and even naming of different gradient estimation schemes vary widely [65]. In this literature, coupling, factorization (or conditioning more generally), and probabilistic inference have been recognized as important ideas in varying capacities. For instance, the application of REINFORCE to stochastic computation graphs [75] has incorporated inference methods based on control variates [86], importance sampling [15, 51], and Rao-Blackwellization [60]. However, these applications are restricted to REINFORCE-based estimators, preventing the use of inference to improve upon the gradient estimation schemes shown in Table 1. Smoothed perturbation analysis (SPA) uses coupling and conditioning for gradient estimation [22, 25, 31]. However, SPA uses simple Monte Carlo or subsampling-based methods. It has not been recognized that richer probabilistic inference schemes can be modularly inserted into the gradient estimation workflow.

**Couplings of Probabilistic Programs.** Coupling is a central technique in Monte Carlo methods [58], and has been used by the programming languages community for relational reasoning about probabilistic programs [9, 10], including in the higher-order context [34, 35, 73, 81]. In GradInf we use coupling for a different purpose, namely programmable variance reduction. Specifically, rather than proving a relational property between programs, we aim to form low-variance estimators of the difference in expectation of a single probabilistic program evaluated at two different inputs. The insights from couplings used in the relational context could prove complementary to our goal, where techniques in the literature such as shift couplings [2] may be repurposed to produce lower variance gradient estimators.

**Differentiating Probabilistic Inference Algorithms.** AD techniques [70, 82] have been applied to optimize the likelihoods from probabilistic programming languages with exact inference [38, 72]. Other works apply gradient estimation to optimize the parameters of proposal distributions specified by inference algorithms such as sequential Monte Carlo (SMC) [50, 61, 66]. In contrast, our work is about using probabilistic inference to construct new gradient estimation schemes. Our inference algorithms, which operate on coupled and factorized programs, are designed to admit a simple interchange of expectation and derivative with respect to the finite perturbation $\varepsilon$ in the final AD step (2.6). We do not aim to learn the parameters of the probabilistic inference schemes themselves.

**Noninterference in Probabilistic Programs.** Prior work has used information-flow typing to automatically detect conditional independence in user models and exploit it for improved posterior inference in Bayesian models, e.g., the imperative while-loop language SlicStan [32, 33], and the higher-order language MaPPL [56]. MaPPL performs automated variable elimination (VE) via a language based on security labels, in which a candidate variable to be eliminated is labeled "high" and the labels of all other variables are inferred accordingly. In contrast, $\lambda_{PP}$ builds on the syntax of the dependency core calculus [1] to provide constructs for a probabilistic version of *partial evaluation*, in which a subset of random variables in a coupled program are sampled and fixed so that the remainder can be targeted by a generic probabilistic inference algorithm. While VE is one such algorithm, its use is as a generic inference algorithm applied subsequent to partial evaluation, of which Table 1 gives several other examples.

## 9 Limitations

**System Expressiveness.** GradInf supports programs featuring higher-order functions, finite iteration, and both discrete and continuous probability distributions, matching the expressiveness of prominent techniques in the PL literature such as ADEV [55]. These programs already cover many challenging problems; Table 1 covers several key classes of gradient estimators [65], including specialized gradient estimators used for applications such as discrete variational autoencoders. However, certain applications still pose a challenge to gradient inference:

- Differentiating programs featuring *unbounded recursion* or *infinite data structures*, which arise in simulations for particle showers in high-energy physics [40]. A challenge for such programs is forming statistically effective couplings that handle variable-size loops and data structures. Addressing this challenge would require extending our coupling transformation $C\{\cdot\}$ with more sophisticated coupling strategies for stochastic control flow.
- Handling *parametric discontinuities* involving continuous random variables [63], which arise in queuing theory [25] and differentiable rendering [7, 57]. The challenge here is that there is no direct combination of coupling, factorization, and probabilistic inference algorithm to which standard AD can be correctly applied (cf. Listing 7).
- Expressing gradient estimation strategies based on *sophisticated control variates*, which are common in reinforcement learning [65]. These control variates are often learned during optimization using approaches such as actor-critic methods [80], which GradInf does not support.

**Automating Primitives.** GradInf requires users to annotate each primitive in their program to specify a suitable factorized coupling. While this design enables flexibility, automating the selection of factorized couplings, or even the design of factorized coupling primitives themselves (e.g., by leveraging work on automatic symbolic disintegration [67]), could improve automation.

**GPU Acceleration and Batching.** The prototype implementation of GradInf does not support GPU-accelerated execution and batched evaluation. It would be useful to implement support for these features, following the approach of systems such as Storchastic [83] and ADEV JAX [12].

**Higher-Order Gradients.** This work has focused on estimating first-order gradients. The problem of estimating higher-order gradients has been studied for several classes of gradient estimators [65]. Generalizing these ideas to the gradient inference setting is a promising future direction.

## 10 Conclusion

We present *gradient inference*, a new approach to gradient estimation that enables the modular construction of sound and efficient gradient estimators. Our technique converts a gradient estimation problem into a probabilistic inference problem, to which powerful probabilistic inference algorithms can be applied, followed by standard automatic differentiation. This reduction rests on synthesizing an intermediate probabilistic program that serves as an inference target, leveraging provably sound program transformations for coupling and factorization to reduce variance. We evaluate our system, GradInf, on challenging case studies, showing it can express diverse estimators and enables the construction of new estimators that outperform state-of-the-art baselines.

Designing effective couplings, factorizations, and probabilistic inference algorithms is key to our approach to gradient estimation. In this paper, we demonstrate several examples of how these techniques can be combined to arrive at effective estimators. Because gradient inference is compositional, future improvements to any one of these three components will automatically deliver benefits for the entire workflow.

## Data-Availability Statement

A Haskell library with all the algorithms described in this article is available at https://github.com/probsys/grad-inf. A reproducible artifact for the evaluation in §7 is available on Zenodo [5].

## Acknowledgments

The authors thank the referees for helpful feedback. This material is based upon work supported by the National Science Foundation under Grant No. 2311983. Any opinions, findings, and conclusions or recommendations in this material are those of the authors and do not necessarily reflect the views of the NSF.

## References

[1] Martín Abadi, Anindya Banerjee, Nevin Heintze, and Jon G. Riecke. 1999. A Core Calculus of Dependency. In *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. Association for Computing Machinery, New York, NY, 147–160. https://doi.org/10.1145/292540.292555

[2] Alejandro Aguirre, Gilles Barthe, Lars Birkedal, Aleš Bizjak, Marco Gaboardi, and Deepak Garg. 2018. Relational Reasoning for Markov Chains in a Probabilistic Guarded Lambda Calculus. In *Proceedings of the 27th European Symposium on Programming*. Springer, Cham, 214–241. https://doi.org/10.1007/978-3-319-89884-1_8

[3] Amal Ahmed. 2006. Step-Indexed Syntactic Logical Relations for Recursive and Quantified Types. In *Proceedings of the 15th European Symposium on Programming*, Peter Sestoft (Ed.), Vol. 3924. Springer Berlin Heidelberg, Berlin, Heidelberg, 69–83. https://doi.org/10.1007/11693024_6

[4] David F. Anderson. 2012. An Efficient Finite Difference Method for Parameter Sensitivities of Continuous Time Markov Chains. *SIAM J. Numer. Anal.* 50, 5 (Jan. 2012), 2237–2258. https://doi.org/10.1137/110849079

[5] Gaurav Arya, Mathieu Huot, Moritz Schauer, Alexander Lew, and Feras Saad. 2026. *Artifact for "GradInf: Gradient Estimation as Probabilistic Inference"*. https://doi.org/10.5281/zenodo.19673572

[6] Gaurav Arya, Moritz Schauer, Frank Schäfer, and Christopher Rackauckas. 2022. Automatic Differentiation of Programs with Discrete Randomness. In *Proceedings of the 36th International Conference on Neural Information Processing Systems (Advances in Neural Information Processing Systems, Vol. 35)*. Curran Associates, Inc., Red Hook, NY, Article 758, 13 pages. https://doi.org/10.5555/3600270.3601028

[7] Sai Praveen Bangaru, Michael Gharbi, Fujun Luan, Tzu-Mao Li, Kalyan Sunkavalli, Milos Hasan, Sai Bi, Zexiang Xu, Gilbert Bernstein, and Fredo Durand. 2022. Differentiable Rendering of Neural SDFs through Reparameterization. In *SIGGRAPH Asia 2022 Conference Papers*. Association for Computing Machinery, New York, NY, USA, Article 5, 9 pages. https://doi.org/10.1145/3550469.3555397

[8] Gilles Barthe, Raphaëlle Crubillé, Ugo Dal Lago, and Francesco Gavazzo. 2020. On the Versatility of Open Logical Relations: Continuity, Automatic Differentiation, and a Containment Theorem. In *Proceedings of the 29th European Symposium on Programming*. Springer-Verlag, Berlin, 56–83. https://doi.org/10.1007/978-3-030-44914-8_3

[9] Gilles Barthe, Thomas Espitau, Benjamin Grégoire, Justin Hsu, Léo Stefanesco, and Pierre-Yves Strub. 2015. Relational Reasoning via Probabilistic Coupling. In *Proceedings of the 20th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*. Springer-Verlag, Berlin, Heidelberg, 387–401. https://doi.org/10.1007/978-3-662-48899-7_27

[10] Gilles Barthe, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2017. Coupling Proofs Are Probabilistic Product Programs. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*. Association for Computing Machinery, New York, NY, USA, 161–174. https://doi.org/10.1145/3009837.3009896

[11] Atılım Günes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. 2017. Automatic Differentiation in Machine Learning: A Survey. *Journal of Machine Learning Research* 18, 1 (Jan. 2017), 5595—-5637. https://doi.org/10.5555/3122009.3242010

[12] McCoy R. Becker, Alexander K. Lew, Xiaoyan Wang, Matin Ghavami, Mathieu Huot, Martin C. Rinard, and Vikash K. Mansinghka. 2024. Probabilistic Programming with Programmable Variational Inference. *Proceedings of the ACM on Programming Languages* 8, PLDI (June 2024), 2123–2147. https://doi.org/10.1145/3656463

[13] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. arXiv:1308.3432

[14] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D. Goodman. 2019. Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research* 20, 28 (2019), 1–6. https://doi.org/10.5555/3322706.3322734

[15] Jörg Bornschein and Yoshua Bengio. 2015. Reweighted Wake-Sleep. In *International Conference on Learning Representations*. 12 pages. arXiv:1406.2751

[16] Phelim P. Boyle. 1988. A Lattice Framework for Option Pricing with Two State Variables. *Journal of Financial and Quantitative Analysis* 23, 1 (March 1988), 1–12. https://doi.org/10.2307/2331019

[17] D. Chandran, W.B. Copeland, S.C. Sleight, and H.M. Sauro. 2008. Mathematical Modeling and Synthetic Biology. *Drug Discovery Today: Disease Models* 5, 4 (2008), 299–309. https://doi.org/10.1016/j.ddmod.2009.07.002

[18] Nicolas Chopin and Omiros Papaspiliopoulos. 2020. *An Introduction to Sequential Monte Carlo.* Springer, Cham. https://doi.org/10.1007/978-3-030-47845-2

[19] Ayush Chopra, Alexander Rodríguez, Jayakumar Subramanian, Arnau Quera-Bofarull, Balaji Krishnamurthy, B. Aditya Prakash, and Ramesh Raskar. 2023. Differentiable Agent-based Epidemiology. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 1848–1857. https://doi.org/10.5555/3545946.3598851

[20] Joel S. Cohen. 2002. *Computer Algebra and Symbolic Computation: Elementary Algorithms.* A K Peters, Natick, MA.

[21] Marco F. Cusumano-Towner, Feras A. Saad, Alexander K. Lew, and Vikash K. Mansinghka. 2019. Gen: A General-Purpose Probabilistic Programming System with Programmable Inference. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Design and Implementation*. Association for Computing Machinery, New York, NY, 221–236. https://doi.org/10.1145/3314221.3314642

[22] Liyi Dai. 2000. Perturbation Analysis via Coupling. *IEEE Trans. Automat. Control* 45, 4 (April 2000), 614–628. https://doi.org/10.1109/9.847099

[23] Joshua V. Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Vasudevan Srinivas, Dave Moore, Brian Patton, Alex Alemi, Matthew D. Hoffman, and Rif A. Saurous. 2017. TensorFlow Distributions. arXiv:1711.10604

[24] Zhe Dong, Andriy Mnih, and George Tucker. 2020. DisARM: An Antithetic Gradient Estimator for Binary Latent Variables. In *Proceedings of the 34th International Conference on Neural Information Processing Systems (Advances in Neural Information Processing Systems, Vol. 33)*. Curran Associates, Inc., Red Hook, NY, 18637–18647.

[25] Michael Fu and Jian-Qiang Hu. 1997. *Conditional Monte Carlo: Gradient Estimation Optimization and Applications.* Springer International Series in Engineering and Computer Science, Vol. 392. Springer, New York. https://doi.org/10.1007/978-1-4615-6293-1

[26] Daniel T. Gillespie. 1977. Exact Stochastic Simulation of Coupled Chemical Reactions. *The Journal of Physical Chemistry* 81, 25 (Dec. 1977), 2340–2361. https://doi.org/10.1021/j100540a008

[27] Michèle Giry. 1982. A Categorical Approach to Probability Theory. In *Categorical Aspects of Topology and Analysis*, B. Banaschewski (Ed.). Springer, Berlin, Heidelberg, 68–85. https://doi.org/10.1007/BFb0092872

[28] Paul Glasserman. 2003. *Monte Carlo Methods in Financial Engineering.* Number 53 in Stochastic Modelling and Applied Probability. Springer, New York, NY. https://doi.org/10.1007/978-0-387-21617-1

[29] Paul Glasserman and Yu-Chi Ho. 1991. *Gradient Estimation via Perturbation Analysis.* Number 116 in Kluwer International Series in Engineering and Computer Science Discrete Event Dynamic Systems. Kluwer Academic Publ, Boston.

[30] Peter W. Glynn. 1990. Likelihood Ratio Gradient Estimation for Stochastic Systems. *Commun. ACM* 33, 10 (1990), 75–84. https://doi.org/10.1145/84537.84552

[31] Wei-Bo Gong and Yu-Chi Ho. 1987. Smoothed (Conditional) Perturbation Analysis of Discrete-Event Dynamic Systems. *IEEE Trans. Automat. Control* 32, 10 (Oct. 1987), 858–866. https://doi.org/10.1109/TAC.1987.1104464

[32] Maria I. Gorinova, Andrew D. Gordon, and Charles Sutton. 2019. Probabilistic Programming with Densities in SlicStan: Efficient, Flexible, and Deterministic. *Proceedings of the ACM on Programming Languages* 3, POPL (Jan. 2019), 1–30. https://doi.org/10.1145/3290348

[33] Maria I. Gorinova, Andrew D. Gordon, Charles Sutton, and Matthijs Vákár. 2022. Conditional Independence by Typing. *ACM Transactions on Programming Languages and Systems* 44, 1 (2022), 1–54. https://doi.org/10.1145/3490421

[34] Simon Oddershede Gregersen, Alejandro Aguirre, Philipp G. Haselwarter, Joseph Tassarotti, and Lars Birkedal. 2024. Asynchronous Probabilistic Couplings in Higher-Order Separation Logic. *Proceedings of the ACM on Programming Languages* 8, POPL (Jan. 2024), 753–784. https://doi.org/10.1145/3632868

[35] Philipp G. Haselwarter, Kwing Hei Li, Alejandro Aguirre, Simon Oddershede Gregersen, Joseph Tassarotti, and Lars Birkedal. 2025. Approximate Relational Reasoning for Higher-Order Probabilistic Programs. *Proceedings of the ACM on Programming Languages* 9, POPL (Jan. 2025), 1196–1226. https://doi.org/10.1145/3704877

[36] B. Heidergott and F. J. Vázquez-Abad. 2008. Measure-Valued Differentiation for Markov Chains. *Journal of Optimization Theory and Applications* 136, 2 (Feb. 2008), 187–209. https://doi.org/10.1007/s10957-007-9297-7

[37] Chris Heunen, Ohad Kammar, Sam Staton, and Hongseok Yang. 2017. A Convenient Category for Higher-Order Probability Theory. In *Proceedings of the 32nd Annual Symposium on Logic in Computer Science*. IEEE Press, Piscataway, NJ, 12 pages. https://doi.org/10.1109/LICS.2017.8005137

[38] Steven Holtzen, Guy Van den Broeck, and Todd Millstein. 2020. Scaling Exact Inference for Discrete Probabilistic Programs. *Proceedings of the ACM on Programming Languages* 4, OOPSLA, Article 140 (Nov. 2020), 31 pages. https://doi.org/10.1145/3428208

[39] Mathieu Huot, Sam Staton, and Matthijs Vákár. 2020. Correctness of Automatic Differentiation via Diffeologies and Categorical Gluing. In *Foundations of Software Science and Computation Structures*, Jean Goubault-Larrecq and Barbara König (Eds.). Springer International Publishing, Cham, 319–338. https://doi.org/10.1007/978-3-030-45231-5_17

[40] Michael Kagan and Lukas Heinrich. 2023. Branches of a Tree: Taking Derivatives of Programs with Discrete and Branching Randomness in High Energy Physics. arXiv:2308.16680

[41] Basim Khajwal, C.-H. Luke Ong, and Dominik Wagner. 2023. Fast and Correct Gradient-Based Optimisation for Probabilistic Programming via Smoothing. In *Proceedings of the 32nd European Symposium on Programming*. Springer-Verlag, Berlin, 479–506. https://doi.org/10.1007/978-3-031-30044-8_18

[42] Lê Anh Khoa. 2025. mstksg/backprop: Heterogeneous Automatic Differentiation ("backpropagation") in Haskell. https://github.com/mstksg/backprop

[43] Diederik P. Kingma and Max Welling. 2013. Auto-Encoding Variational Bayes. arXiv:1312.6114

[44] Edward Kmett. 2025. ekmett/ad: Automatic Differentiation. https://github.com/ekmett/ad

[45] Daphne Koller and Nir Friedman. 2009. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, Cambridge, MA.

[46] Faustyna Krawiec, Simon Peyton Jones, Neel Krishnaswami, Tom Ellis, Richard A. Eisenberg, and Andrew Fitzgibbon. 2022. Provably Correct, Asymptotically Efficient, Higher-Order Reverse-Mode Automatic Differentiation. *Proceedings of the ACM on Programming Languages* 6, POPL, Article 48 (Jan. 2022), 30 pages. https://doi.org/10.1145/3498710

[47] Russell Z. Kunes, Mingzhang Yin, Max Land, Doron Haviv, Dana Pe'er, and Simon Tavaré. 2023. Gradient Estimation for Binary Latent Variables via Gradient Variance Clipping. *Proceedings of the AAAI Conference on Artificial Intelligence* 37, 7 (June 2023), 8405–8412. https://doi.org/10.1609/aaai.v37i7.26013

[48] Daniël Lakens, Anne M. Scheel, and Peder M. Isager. 2018. Equivalence Testing for Psychological Research: A Tutorial. *Advances in Methods and Practices in Psychological Science* 1, 2 (2018), 259–269. https://doi.org/10.1177/2515245918770963

[49] Kim G. Larsen and Arne Skou. 1991. Bisimulation Through Probabilistic Testing. *Information and Computation* 94, 1 (1991), 1–28. https://doi.org/10.1016/0890-5401(91)90030-6

[50] Tuan Anh Le, Maximilian Igl, Tom Rainforth, Tom Jin, and Frank Wood. 2018. Auto-Encoding Sequential Monte Carlo. In *International Conference on Learning Representations*. 10 pages. arXiv:1705.10306

[51] Tuan Anh Le, Adam R. Kosiorek, N. Siddharth, Yee Whye Teh, and Frank Wood. 2020. Revisiting Reweighted Wake-Sleep for Models with Stochastic Control Flow. In *Proceedings of the 35th Uncertainty in Artificial Intelligence Conference*. PMLR, Norfolk, MA, 1039–1049.

[52] Wonyeol Lee, Xavier Rival, and Hongseok Yang. 2023. Smoothness Analysis for Probabilistic Programs with Application to Optimised Variational Inference. *Proceedings of the ACM on Programming Languages* 7, POPL, Article 12 (Jan. 2023), 32 pages. https://doi.org/10.1145/3571205

[53] Wonyeol Lee, Hangyeol Yu, Xavier Rival, and Hongseok Yang. 2020. On Correctness of Automatic Differentiation for Non-Differentiable Functions. In *Proceedings of the 34th International Conference on Neural Information Processing Systems (Advances in Neural Information Processing Systems, Vol. 33)*. Curran Associates Inc., Red Hook, NY, USA, Article 564, 12 pages.

[54] Alexander K. Lew, Marco F. Cusumano-Towner, Benjamin Sherman, Michael Carbin, and Vikash K. Mansinghka. 2020. Trace Types and Denotational Semantics for Sound Programmable Inference in Probabilistic Languages. *Proceedings of the ACM on Programming Languages* 4, POPL (Jan. 2020), 1–32. https://doi.org/10.1145/3371087

[55] Alexander K. Lew, Mathieu Huot, Sam Staton, and Vikash K. Mansinghka. 2023. ADEV: Sound Automatic Differentiation of Expected Values of Probabilistic Programs. *Proceedings of the ACM on Programming Languages* 7, POPL (Jan. 2023), 121–153. https://doi.org/10.1145/3571198

[56] Jianlin Li, Eric Wang, and Yizhou Zhang. 2024. Compiling Probabilistic Programs for Variable Elimination with Information Flow. *Proceedings of the ACM on Programming Languages* 8, PLDI (June 2024), 1755–1780. https://doi.org/10.1145/3656448

[57] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. 2018. Differentiable Monte Carlo Ray Tracing Through Edge Sampling. *ACM Trans. Graph.* 37, 6, Article 222 (Dec. 2018), 11 pages. https://doi.org/10.1145/3272127.3275109

[58] Torgny Lindvall. 2002. *Lectures on the Coupling Method*. Dover Publications, Mineola, NY.

[59] Jun S. Liu. 2004. *Monte Carlo Strategies in Scientific Computing* (1 ed.). Springer, New York, NY, USA. https://doi.org/10.1007/978-0-387-76371-2

[60] Runjing Liu, Jeffrey Regier, Nilesh Tripuraneni, Michael Jordan, and Jon Mcauliffe. 2019. Rao-Blackwellized Stochastic Gradients for Discrete Distributions. In *Proceedings of the 36th International Conference on Machine Learning*. PMLR, Norfolk, MA, 4023–4031.

[61] Chris J. Maddison, Dieterich Lawson, George Tucker, Nicolas Heess, Mohammad Norouzi, Andriy Mnih, Arnaud Doucet, and Yee Whye Teh. 2017. Filtering Variational Objectives. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (Advances in Neural Information Processing Systems, Vol. 30)*. Curran Associates,

Inc., Red Hook, NY, 6576–6586.

[62] Vikash K. Mansinghka, Ulrich Schaechtle, Shivam Handa, Alexey Radul, Yutian Chen, and Martin Rinard. 2018. Probabilistic Programming with Programmable Inference. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Design and Implementation*. Association for Computing Machinery, New York, NY, 603–616. https://doi.org/10.1145/3192366.3192409

[63] Jesse Michel, Kevin Mu, Xuanda Yang, Sai Praveen Bangaru, Elias Rojas Collins, Gilbert Bernstein, Jonathan Ragan-Kelley, Michael Carbin, and Tzu-Mao Li. 2024. Distributions for Compositionally Differentiating Parametric Discontinuities. *Proceedings of the ACM on Programming Languages* 8, OOPSLA1 (April 2024), 893–922. https://doi.org/10.1145/3649843

[64] Andriy Mnih and Danilo J. Rezende. 2016. Variational Inference for Monte Carlo Objectives. arXiv:1602.06725

[65] Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. 2020. Monte Carlo Gradient Estimation in Machine Learning. *Journal of Machine Learning Research* 21, 132 (2020), 1–62. https://doi.org/10.5555/3455716.3455848

[66] Christian Naesseth, Scott Linderman, Rajesh Ranganath, and David Blei. 2018. Variational Sequential Monte Carlo. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 84)*, Amos Storkey and Fernando Perez-Cruz (Eds.). PMLR, 968–977.

[67] Praveen Narayanan and Chung-chieh Shan. 2020. Symbolic Disintegration with a Variety of Base Measures. *ACM Transactions on Programming Languages and Systems* 42, 2, Article 9 (2020), 60 pages. https://doi.org/10.1145/3374208

[68] Uwe Naumann. 2011. *The Art of Differentiating Computer Programs*. Society for Industrial and Applied Mathematics, Philadelphia. https://doi.org/10.1137/1.9781611972078

[69] Art B. Owen. 2013. *Monte Carlo Theory, Methods and Examples*. https://artowen.su.domains/mc/

[70] Viktor Pfanschilling, Hikaru Shindo, Devendra Singh Dhami, and Kristian Kersting. 2022. Sum-Product Loop Programming: From Probabilistic Circuits to Loop Programming. In *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning*. IJCAI Organization, 453–462. https://doi.org/10.24963/kr.2022/47

[71] Norman Ramsey and Avi Pfeffer. 2002. Stochastic Lambda Calculus and Monads of Probability Distributions. In *Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. Association for Computing Machinery, New York, NY, USA, 154–165. https://doi.org/10.1145/503272.503288

[72] Feras A. Saad, Martin C. Rinard, and Vikash K. Mansinghka. 2021. SPPL: Probabilistic Programming with Fast Exact Symbolic Inference. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. Association for Computing Machinery, New York, NY, USA, 804–819. https://doi.org/10.1145/3453483.3454078

[73] Tetsuya Sato, Alejandro Aguirre, Gilles Barthe, Marco Gaboardi, Deepak Garg, and Justin Hsu. 2019. Formal Verification of Higher-Order Probabilistic Programs: Reasoning about Approximation, Convergence, Bayesian Inference, and Optimization. *Proceedings of the ACM on Programming Languages* 3, POPL (Jan. 2019), 1–30. https://doi.org/10.1145/3290351

[74] Tetsuya Sato, Gilles Barthe, Marco Gaboardi, Justin Hsu, and Shin-ya Katsumata. 2019. Approximate Span Liftings: Compositional Semantics for Relaxations of Differential Privacy. In *Proceedings of the 34th Annual Symposium on Logic in Computer Science (LICS '19)*. IEEE Press, 1–14. https://doi.org/10.1109/LICS.2019.8785668

[75] John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. 2015. Gradient Estimation Using Stochastic Computation Graphs. In *Proceedings of the 29th International Conference on Neural Information Processing Systems (Advances in Neural Information Processing Systems, Vol. 28)*. Curran Associates, Inc., Red Hook, NY, 3528–3536.

[76] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv:arXiv:1707.06347

[77] Adam Ścibior, Ohad Kammar, and Zoubin Ghahramani. 2018. Functional Programming for Modular Bayesian Inference. *Proceedings of the ACM on Programming Languages* 2, ICFP, Article 83 (July 2018), 29 pages. https://doi.org/10.1145/3236778

[78] Adam Ścibior, Ohad Kammar, Matthijs Vákár, Sam Staton, Hongseok Yang, Yufei Cai, Klaus Ostermann, Sean K. Moss, Chris Heunen, and Zoubin Ghahramani. 2018. Denotational Validation of Higher-Order Bayesian Inference. *Proceedings of the ACM on Programming Languages* 2, POPL, Article 60 (Jan. 2018), 29 pages. https://doi.org/10.1145/3158148

[79] Richard Statman. 1985. Logical Relations and the Typed $\lambda$-Calculus. *Information and Control* 65, 2 (May 1985), 85–97. https://doi.org/10.1016/S0019-9958(85)80001-2

[80] Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement Learning: An Introduction*. Bradford Books, Cambridge.

[81] Joseph Tassarotti and Robert Harper. 2019. A Separation Logic for Concurrent Randomized Programs. *Proceedings of the ACM on Programming Languages* 3, POPL (Jan. 2019), 1–30. https://doi.org/10.1145/3290377

[82] Ryan Tjoa, Poorva Garg, Harrison Goldstein, Todd Millstein, Benjamin C. Pierce, and Guy Van den Broeck. 2025. Tuning Random Generators: Property-Based Testing as Probabilistic Programming. *Proceedings of the ACM on Programming Languages* 9, OOPSLA2, Article 304 (Oct. 2025), 28 pages. https://doi.org/10.1145/3763082

[83] Emile van Krieken, Jakub M Tomczak, and Annette ten Teije. 2021. Storchastic: A Framework for General Stochastic Automatic Differentiation. In *Proceedings of the 35th International Conference on Neural Information Processing Systems (Advances in Neural Information Processing Systems, Vol. 34)*. Curran Associates, Inc., Red Hook, NY, 7574–7587.

[84] Guanyang Wang, John O'Leary, and Pierre Jacob. 2021. Maximal Couplings of the Metropolis-Hastings algorithm. In *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 130)*. PMLR, 1225–1233.

[85] Ting Wang and Muruhan Rathinam. 2016. Efficiency of the Girsanov Transformation Approach for Parametric Sensitivity Analysis of Stochastic Chemical Kinetics. *SIAM/ASA Journal on Uncertainty Quantification* 4, 1 (Jan. 2016), 1288–1322. https://doi.org/10.1137/140998111

[86] Théophane Weber, Nicolas Heess, Lars Buesing, and David Silver. 2019. Credit Assignment Techniques in Stochastic Computation Graphs. In *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 89)*, Kamalika Chaudhuri and Masashi Sugiyama (Eds.). PMLR, 2650–2660.

[87] Ronald J. Williams. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning* 8 (1992), 229–256. https://doi.org/10.1007/BF00992696

Received 2025-11-13; accepted 2026-04-03