Amortized Rejection Sampling in
Universal Probabilistic Programming

Saeid Naderiparizi[1], **Adam Ścibior**[1], **Andreas Munk**[1], **Mehrdad Ghadiri**[2]
Atılım Güneş Baydin[3], **Bradley Gram-Hansen**[3], **Christian Schroeder de Witt**[3]
Robert Zinkov[3], **Philip H.S. Torr**[3], **Tom Rainforth**[3], **Yee Whye Teh**[3], **Frank Wood**[1]
[1]University of British Columbia, [2]Georgia Institute of Technology, [3]University of Oxford

Abstract

Existing approaches to amortized inference in probabilistic programs with unbounded loops can produce estimators with infinite variance. An instance of this is importance sampling inference in programs that explicitly include rejection sampling as part of the user-programmed generative procedure. In this paper we develop a new and efficient amortized importance sampling estimator. We prove finite variance of our estimator and empirically demonstrate our method's correctness and efficiency compared to existing alternatives on generative programs containing rejection sampling loops and discuss how to implement our method in a generic probabilistic programming framework.

1 INTRODUCTION

It is now understood how to apply probabilistic programming inference techniques to generative models written in *existing* "universal" languages (van de Meent et al., 2018). While the expressivity of such languages allows users to write generative procedures naturally, this flexibility introduces complexities, some of surprising and subtle character. For instance there is nothing to stop users from using rejection sampling loops to specify all or part of their generative model. While existing inference approaches may asymptotically produce correct inference results for such programs, the reality, which we discuss at length in this paper, is murkier.

The specific problem we address, that of efficient amortized importance-sampling-based inference in models

with user-defined rejection sampling loops is more prevalent than it might seem on first consideration. Our experience suggests that rejection sampling within generative model specification is actually the rule rather than the exception when programmers use universal languages for model specification. To generate a single draw from anything more complex than standard distribution effectively requires either adding a new probabilistic primitive to the language (beyond most users), hard conditioning on constraint satisfaction (inefficient under most forms of universal PPL inference), or a user-programmed rejection loop. A quick example of this is sampling from a truncated normal. If the model specification language does not have a truncated normal primitive then the most natural way to generate such a variate is via user-programmed rejection. More sophisticated examples abound in simulators used in the physical sciences (Baydin et al., 2018, 2019), chemistry (Cai, 2007; Ramaswamy and Sbalzarini, 2010; Slepoy et al., 2008), and other domains (Stuhlmüller and Goodman, 2014). Note that the issue we address here is not related to hard rejection via conditioning, i.e. (Ritchie et al., 2015) and related work. Ours is specifically about rejection sampling loops within the generative model program, whereas the latter is about developing inference engines that are reasonably efficient even when the user specified program has a constraint-like observation that produces an extremely peaked posterior.

The first inference algorithms for languages that allowed generative models containing rejection sampling loops to be written revolved around Markov chain Monte Carlo (MCMC) (Goodman et al., 2008; Wingate et al., 2011) and sequential importance sampling (SIS) (Wood et al., 2014) using the prior as the proposal distribution and then mean-field variational inference (Wingate and Weber, 2013). Those methods were very inefficient, prompting extensions of PPLs providing programmable inference capabilities (Mansinghka et al., 2014; Ścibior, 2019). Efforts to speed

1: $x \sim p(x)$ 2: 3: **for** $k \in \mathbb{N}^+$ **do** 4: $\quad z^k \sim p(z\|x)$ 5: 6: 7: $\quad$ **if** $c(x, z^k)$ **then** 8: $\quad\quad z = z^k$ 9: $\quad\quad$ **break** 10: observe$(y, p(y\|z, x))$	$x \sim q(x\|y)$ $w \leftarrow \frac{p(x)}{q(x\|y)}$ **for** $k \in \mathbb{N}^+$ **do** $\quad z^k \sim q(z\|x, y)$ $\quad w^k \leftarrow \frac{p(z^k\|x)}{q(z^k\|x,y)}$ $\quad w \leftarrow w\, w^k$ $\quad$ **if** $c(x, z^k)$ **then** $\quad\quad z = z^k$ $\quad\quad$ **break** $w \leftarrow wp(y\|z, x)$
(a) Original program	(b) Inference compilation
1: $x \sim p(x)$ 2: 3: $z \sim p(z\|x, c(x, z))$ 4: 5: observe$(y, p(y\|z, x))$	$x \sim q(x\|y)$ $w \leftarrow \frac{p(x)}{q(x\|y)}$ $z \sim q(z\|x, y, c(x, z))$ $w \leftarrow w\, \frac{p(z\|x,c(x,z))}{q(z\|x,y,c(x,z))}$ $w \leftarrow w\, p(y\|z, x)$
(c) Equivalent to above	(d) Our IS estimator

Figure 1: (a) illustrates the problem we are addressing. Existing approaches to inference compilation use trained proposals for the importance sampler shown in (b), where w can have infinite variance, even when each w^k individually has finite variance, as k is unbounded. There exists a simplified program (c) equivalent to (a) and ideally we would like to perform inference using the importance sampler in (d). While this is not directly possible, since we do not have access to the conditional densities required, our method approximates this algorithm, guaranteeing finite variance of w.

inference since then have revolved around amortized inference (Gershman and Goodman, 2014), where a slow initial off-line computation is traded against fast and accurate test-time inference. Such methods work by training neural networks that quickly map a dataset either to a variational posterior (Ritchie et al., 2016) or a to a sequence of proposal distributions for SIS (Le et al., 2017). This paper examines and builds on the latter "Inference Compilation" (IC) approach.

Unbounded loops potentially require integrating over infinitely many latent variables. With IC each of these variables has its own importance weight and the product of all the weights can have infinite variance, resulting in a divergent importance sampler. Furthermore, the associated self-normalizing importance sampler can converge to an arbitrary point, giving wrong inference results without warning. It is therefore necessary to take extra steps to ensure consistency of the importance sampler resulting from IC when unbounded loops are present.

In this paper we present a solution to this problem

for the common case of rejection sampling loops. We establish, both theoretically and empirically, that computing importance weights naïvely in this situation can lead to arbitrarily bad posterior estimates. To remedy this problem, we develop a collection of novel estimators and discuss when each of them is best to apply. A preview of the problem and the proposed solution is shown in Figure 1.

2 PROBLEM FORMULATION

Our formulation of the problem will be presented concretely starting from the example probabilistic program shown in Figure 1a. Even though both the problem and our solution are more general, applying to all probabilistic programs with rejection sampling loops regardless of how complicated they are, this simple example captures all the important aspects of the problem. As a reminder, inference compilation refers to offline training of importance sampling proposal distributions for all random variables in a program (Le et al., 2017). Existing approaches to inference compilation for the program in Figure 1a correspond to the importance sampler shown in Figure 1b where there is some proposal learned for every random choice in the program. While the weighted samples produced by this method result in unbiased estimates, the variance of the weights can be very high and potentially infinite due to the unbounded number of w^k's. To show this, we start by more precisely defining the meaning of the sampler in Figure 1b.

Definition 2.1 (Naive weighing). Let $p(x, y, z)$ be a probability density such that all conditional densities exist. Let $c(x, z)$ be a Boolean condition, and A be the event that c is satisfied, such that $p(A|x, y) \geq \epsilon$ for all (x, y) and some $\epsilon > 0$. For each y, let $q(x, z|y)$ be a probability density absolutely continuous with respect to $p(x, z|y)$ and $q(A|x, y) \geq \epsilon$ for all (x, y). Let $x \sim q(x|y)$ and let $z^k \sim_{i.i.d.} q(z|x, y)$ and $w^k = \frac{p(z^k|x)}{q(z^k|x,y)}$ for all $k \in \mathbb{N}^+$. Let $L = \min\{k|c(x, z^k)\}$, $z = z^L$ and $w_{IC} = \frac{p(x)}{q(x|y)}p(y|x, z)\prod_{k=1}^{L} w^k$.

We assert that definition 2.1 corresponds to the program in Figure 1b. A rigorous correspondence could be established using formal semantics methods, such as the construction of Ścibior et al. (2017), but this is beyond the scope of this paper. Although the resulting importance sampler correctly targets the posterior distribution, the variance of w_{IC} is a problem, and it is this specific problem that we tackle in this paper.

Intuitively, a large number of rejections in the loop leads to a large number of w_k being included in w_{IC} and the variance of their product tends to grow quickly. In the worst case, this variance may be in-
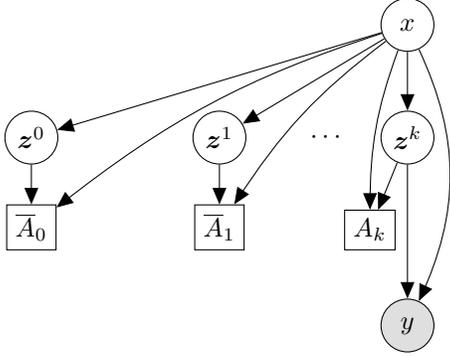
Figure 2: Dependency graph for a trace from Program 1a. Here, A_i is the event of accepting a sampled $\boldsymbol{z}$ i.e. $c(x, \boldsymbol{z}^i)$ holds and similarly, $\overline{A}_i$ is defined for rejection. Circles and squares represent stochastic and deterministic nodes, respectively.

finite, even when each w_k has finite variance individually. This happens when the proposed samples are rejected too often, which is formalized in the following theorem.

Theorem 1. *Under assumptions of Definition 2.1, if the following condition holds with positive probability under $x \sim q(x|y)$*

$$\mathbb{E}_{\boldsymbol{z} \sim q(\boldsymbol{z}|x,y)} \left[\frac{p(\boldsymbol{z}|x)^2}{q(\boldsymbol{z}|x,y)^2} (1 - p(A|x, \boldsymbol{z})) \right] \geq 1 \qquad (1)$$

then the variance of w_{IC} is infinite.

Proof. In Appendix A. □

What this means is that importance sampling with proposals other than the prior may hurt more than help in the case of rejection sampling loops and it is no trivial way to ensure Equation 1 does not hold or to detect if it holds for a proposal. Furthermore, existing IC schemes are effectively useless under the conditions of Theorem 1, since the consistency of the self-normalizing importance sampler depends on the variance of weights being finite. Worse, even when the variance is finite but large, it may render the effective sample size too low for practical applications, a phenomenon we have observed repeatedly in practice. What remains is to derive an alternative way to compute w_{IC} that guarantees finite variance and in practice leads to larger effective sample sizes than existing methods.

3 APPROACH

A starting point to the presentation of our algorithm is to observe that the program in Figure 1a is equivalent to the program 1c, where $\boldsymbol{z}$ satisfying the condition c

is sampled directly. Figure 1d presents an importance sampler targeting 1c, obtained by sampling $\boldsymbol{z}$ directly from q under the condition c. Note that the sampling processes in 1b and 1d are the same, only the weights are computed differently.

Definition 3.1 (Collapsed weighing)**.** Extending Definition 2.1, let

$$w_C = \frac{p(x)}{q(x|y)} \underbrace{\frac{p(\boldsymbol{z}|x, A)}{q(\boldsymbol{z}|x, A, y)}}_{\text{\textcircled{T}}} p(y|x, \boldsymbol{z}) \qquad (2)$$

Note that $\mathbb{E}[w_{IC}] = \mathbb{E}[w_C]$, which we prove in Theorem 3. However, since w_C only involves a fixed number (three) of terms, we can expect it to avoid problems with exploding variance. Unfortunately, we can not directly compute w_C.

In equation 2 we can directly evaluate all terms except \textcircled{T}, since, $p(\boldsymbol{z}|x, A)$ and $q(\boldsymbol{z}|x, A, y)$ are defined implicitly by the rejection sampling loop. Applying Bayes' rule to this term gives the following equality:

$$\text{\textcircled{T}} = \underbrace{\frac{q(A|x, y)}{p(A|x)}}_{\text{\textcircled{1}}} \underbrace{\frac{p(\boldsymbol{z}|x)}{q(\boldsymbol{z}|x, y)}}_{\text{\textcircled{2}}} \underbrace{\frac{p(A|\boldsymbol{z}, x)}{q(A|\boldsymbol{z}, x, y)}}_{\text{\textcircled{3}}} \qquad (3)$$

The term \textcircled{2} can be directly evaluated, since we have access to both conditional densities and the term \textcircled{3} is always equal to 1, since A is deterministic given x and $\boldsymbol{z}$. However, the term \textcircled{1}, which is the ratio of acceptance probabilities under q and p, can not be computed directly and we need to estimate it. We provide separate unbiased estimators for $q(A|x, y)$ and $\frac{1}{p(A|x)}$.

For $q(A|x, y)$ we use straightforward Monte Carlo estimation of the following expectation:

$$\begin{aligned} q(A|x, y) &= \int q(A|\boldsymbol{z}, x, y) q(\boldsymbol{z}|x, y) d\boldsymbol{z} \\ &= \int c(\boldsymbol{z}, x) q(\boldsymbol{z}|x, y) d\boldsymbol{z} \\ &= \mathbb{E}_{\boldsymbol{z} \sim q(\boldsymbol{z}|x,y)} [c(\boldsymbol{z}, x)] \end{aligned} \qquad (4)$$

For $\frac{1}{p(A|x)}$ we use Monte Carlo estimation after applying the following standard lemma:

Lemma 2. *Let A be an event that occurs in a trial with probability p. The expectation of the number of trials to the first occurrence of A is equal to $\frac{1}{p}$.*

It is important that these estimators are constructed independently of $\boldsymbol{z}$ being sampled to ensure that we obtain an unbiased estimator for w_C specified in Equation 2. We put together all these elements to obtain

our final method in Algorithm 1. More formally, the weight obtained our method is defined as follows.

Algorithm 1 Pseudocode for our algorithm applied to the probabilistic program from Figure 1a.

1: $x \sim q(x|y)$
2: $w \leftarrow \frac{p(x)}{q(x|y)}$
3: **for** $k \in \mathbb{N}^+$ **do**
4:      $\boldsymbol{z}^k \sim q(\boldsymbol{z}|x,y)$
5:      **if** $c(x, \boldsymbol{z}^k)$ **then**
6:          $\boldsymbol{z} = \boldsymbol{z}^k$
7:          **break**
8: $w \leftarrow w \frac{p(\boldsymbol{z}|x)}{q(\boldsymbol{z}|x,y)}$
9: ▷ estimate $q(A|x,y)$ using Equation 4
10: $K \leftarrow 0$
11: **for** $i \in 1, \dots N$ **do**
12:      $\boldsymbol{z}_i' \leftarrow q(\boldsymbol{z}|x,y)$
13:      $K \leftarrow K + c(\boldsymbol{z}, x)$
14: ▷ estimate $\frac{1}{p(A|x)}$ using Lemma 2
15: **for** $j \in 1, \dots M$ **do**
16:      **for** $l \in \mathbb{N}^+$ **do**
17:          $\boldsymbol{z}_{j,l}'' \leftarrow p(\boldsymbol{z}|x)$
18:          **if** $c(x, \boldsymbol{z}_{j,l}'')$ **then**
19:              $T_j \leftarrow l$
20:              **break**
21: $T \leftarrow \frac{1}{M} \sum_{j=1}^{M} T_j$
22: $w \leftarrow w \frac{KT}{N}$
23: $w \leftarrow w\, p(y|\boldsymbol{z}, x)$

Definition 3.2 (Our weighing)**.** Extending Definition 2.1, let $\boldsymbol{z}_i' \sim_{i.i.d} q(\boldsymbol{z}|x,y)$ for $i \in 1, \dots, N$ and K be the number of $\boldsymbol{z}_i'$ for which $c(x, \boldsymbol{z}_i')$ holds. Let $(\boldsymbol{z}_{j,1}'', \dots, \boldsymbol{z}_{j,T_j}'')$ be sequences of potentially varying length for $j \in 1, \dots, M$ with $\boldsymbol{z}_{j,l}'' \sim_{i.i.d.} p(\boldsymbol{z}|x)$ such that for all j, T_j is the smallest index l for which $c(x, \boldsymbol{z}_{j,l}'')$ holds. Let $T = \frac{1}{M} \sum_{j=1}^{M} T_j$. Finally, let

$$w = \frac{p(x)}{q(x|y)} \frac{KT}{N} \frac{p(\boldsymbol{z}|x)}{q(\boldsymbol{z}|x,y)} p(y|x, \boldsymbol{z}). \tag{5}$$

Throughout this section we have only informally argued that the three importance samplers presented target the same distribution. With all the definitions in place we can make this argument precise in the following theorem.

Theorem 3. *For any $N \geq 1$ and $M \geq 1$, and all values of $(x, y, \boldsymbol{z})$,*

$$\mathbb{E}\left[w_{IC}|x, y, \boldsymbol{z}\right] = w_C = \mathbb{E}\left[w|x, y, \boldsymbol{z}\right]. \tag{6}$$

Proof. For the second equality, use Equation 4, then Lemma 2, Equation 3, and finally Equation 2.

$$\mathbb{E}\left[w|x, y, \boldsymbol{z}\right] = \tag{7}$$

$$\frac{p(x)}{q(x|y)} \frac{p(\boldsymbol{z}|x)}{q(\boldsymbol{z}|x,y)} p(y|x, \boldsymbol{z}) \frac{1}{N} \mathbb{E}_{\boldsymbol{z}'}\left[K\right] \mathbb{E}_{\boldsymbol{z}''}\left[T\right] = \tag{8}$$

$$\frac{p(x)}{q(x|y)} \frac{p(\boldsymbol{z}|x)}{q(\boldsymbol{z}|x,y)} p(y|x, \boldsymbol{z}) q(A|x,y) \frac{1}{p(A|x)} = \tag{9}$$

$$\frac{p(x)}{q(x|y)} \frac{p(\boldsymbol{z}|x, A)}{q(\boldsymbol{z}|x, A, y)} p(y|x, \boldsymbol{z}) = w_C \tag{10}$$

For the first equality, use Equation 20 in Appendix A to get

$$\mathbb{E}\left[w_{IC}|x, y, \boldsymbol{z}\right] = \tag{11}$$

$$\frac{p(x)}{q(x|y)} p(y|x, z) \, w_L \, \mathbb{E}_{\boldsymbol{z}^{1:L-1}}\left[\prod_{k=1}^{L-1} w^k\right] = \tag{12}$$

$$\frac{p(x)}{q(x|y)} p(y|x, z) \frac{p(\boldsymbol{z}|x)}{q(\boldsymbol{z}|x,y)} \frac{q(A|x,y)}{p(A|x)} = w_C \tag{13}$$

$\square$

Since all three importance samplers use the same proposal distributions for $(x, \boldsymbol{z})$, Theorem 3 shows that they all target the same distribution, which is the posterior distribution specified by the original probabilistic program in Figure 1a.

Finally, we can prove that our method handles inference in rejection sampling loops without introducing infinite variance. Note that variance may still be infinite for reasons not having to do with the rejection sampling loop, if $q(x|y)$ and $q(\boldsymbol{z}|x,y)$ are poorly chosen.

Theorem 4. *For any $N \geq 1$ and $M \geq 1$, if w_C from Definition 3.1 has finite variance, then w from Definition 3.2 has finite variance.*

Proof. Note that conditionally on $(x, y, \boldsymbol{z})$ K follows a binomial distribution, so $Var[\frac{K}{N}|x, y, \boldsymbol{z}] < 1 < \infty$, while T follows a geometric distribution and $Var[T|x, y, \boldsymbol{z}] < \frac{1}{p(A|x)^2} \leq \frac{1}{\epsilon^2} < \infty$. Also, conditionally on $(x, y, \boldsymbol{z})$, $\frac{K}{N}$ and T are independent, so $Var[\frac{KT}{N}] < B$ for some constant $B < \infty$. Then see that $w = w_C \frac{p(A|x)}{q(A|x,y)} \frac{KT}{N}$. Then, using the law of total variance, we get

$$Var[w] = \mathbb{E}\left[Var[w|x, y, \boldsymbol{z}]\right] + Var[\mathbb{E}\left[w|x, y, \boldsymbol{z}\right]] \tag{14}$$

$$= \mathbb{E}\left[\left(w_C \frac{p(A|x)}{q(A|x,y)}\right)^2 Var\left[\frac{KT}{N}\bigg|x, y, \boldsymbol{z}\right]\right] + Var[w_C] \tag{15}$$

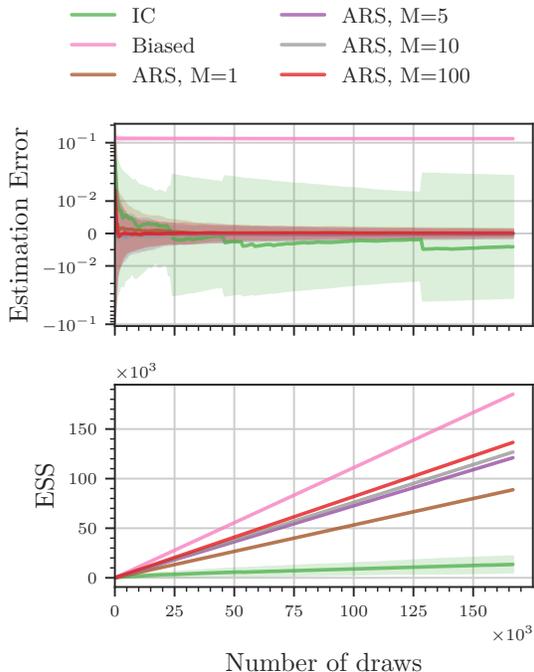$$\leq \mathbb{E}\left[w_C \frac{1}{\epsilon^2} B\right] + Var[w_C] < \infty \tag{16}$$

$\square$

Figure 3: Inference results for the Gaussian unknown mean program. We estimate $\mathbb{E}_{p(\mu|y)}[\mu]$ which in this instance has an analytic ground truth value. The plots show an aggregation of 100 different runs of the experiment. (top) Estimation error of 3 different methods (IC, Biased, and (ours) ARS with different values of M). Our method reaches estimation error approaching zero faster and with significantly less variance than the alternatives. Larger values of M lead to lower variance and faster convergence. (bottom) Effective sample size (ESS) as a function of number of IS proposals for each estimator. The multiplicative inclusion of importance weights in the existing IC approach significantly adversely affects ESS compared to our method regardless of the chosen value of M.

4   EXPERIMENTS

We illustrate our method by performing inference in two example probabilistic programs that include rejection sampling loops. We designed these programs so that baseline posteriors can be derived analytically.

As is now nearly standard in writing about probabilistic programming, in the following programs `sample` means to draw a random value from the given distribution and `observe` means to condition the program on having observed the given value under the given distribution.

In our example models, Programs 1 and 2, we either explicitly introduce rejection sampling or replace some

sample statements with rejection sampling loops that generate samples from the same distribution as the one replaced. We evaluate the efficacy of our approach in several ways including computing the effective sample size (ESS) of IS posterior estimates and empirically comparing the convergence rates of different methods to analytically computed ground truth values.

The specific methods we compare are:

- Existing (IC): Like the approach in Figure 1b this uses a proposal for all iterations of rejection sampling loops. Final importance weights are computed by multiplying all the weights for all samples in the trace, accepted and rejected.

- Ours (ARS, M=m): Similar to the approach in Figure 1d this uses our method (Algorithm 1) with $M = m$ and $N = \max(n, 10)$ as defined in Definition 3.2, not multiplying in the weights of all of the rejected samples on the trace.

- Existing (Biased): Implements the incorrect variant of ARS that appears currently in PyProb. It does not multiply in all the weights of the rejected samples, however, it does not estimate nor multiple in the correction factor $\frac{q(A|x,y)}{p(A|x)}$.

Program 1: Gaussian unknown mean (GUM)

```python
def GUM(mu_0, sigma_0, sigma, y_obs):
  u = sample(Uniform(low=0, high=1))
  if u > 0.5:
    while True:
      rs_start()
      mu = sample(Normal(mean=mu_0, std=sigma_0))
      if mu > mu_0:
        rs_end ()
        break
  else:
    while True:
      rs_start()
      mu = sample(Normal(mean=mu_0, std=sigma_0))
      if mu <= mu_0:
        rs_end ()
        break
  y = observe(Normal(mean=mu, std=sigma), y_obs)
  return mu
```

4.1   Gaussian Unknown Mean

Our first *Gaussian unknown mean* (GUM) model consists of one latent variable, the mean $\mu \in \mathbb{R}$, which is given a Gaussian prior with mean $\mu_0 \in \mathbb{R}$ and variance $\sigma_0 \in \mathbb{R}^+$, and a single observation y governed by a normal observation model with fixed standard deviation

$\sigma \in \mathbb{R}^+$.

$$\mu \sim N(\mu_0, \sigma_0)$$
$$y|\mu \sim N(\mu, \sigma)$$

We introduce rejection sampling into this model by splitting the prior into two truncated normal distributions and sampling from these truncated normals via rejection sampling. This is shown in Program 1. The reason that u is a random variable will be made apparent in due course; for now suffice it to say that the full generality of our method extends to various compositions of rejection samplers, including state dependent rejection samplers that are have potentially different exit probabilities in each trace.

We fix $\mu_0 = 0, \sigma_0 = 1, \sigma = \frac{\sqrt{2}}{2}$. The observation y is 0. The proposals are fixed; for the first branch in Program 1 the proposal is $N(-2, 2)$ whereas all other proposals are fixed to be equal to the prior. In this experiment we intentionally chose the proposal so as to not be close to the true posterior, yielding a more difficult problem.

Figure 3 shows the aggregated results of running each IS inference technique 100 times. While IC converges to the ground truth, it has poor ESS. The high variance of the IC estimator can also plainly be seen in comparison to our method. On the other hand, as our method does not multiplicatively update importance sampling weights with weights from rejected samples both has has much higher ESS and converges more rapidly to the ground truth.

4.2 Gaussian Mixture Model

Our second example program is a GMM with two components and a single observation. Its implementation is shown in Program 2 where sampling from one of the mixtures is done through a rejection sampling loop. The target distribution for the rejection sampling branch is $N(\mu_1, \sigma_1)$ and is implicitly defined via a rejection-sampler with base distribution $N(\mu_0, \sigma_0)$. The implementation of `alpha(x)` is $\alpha(x) = \frac{1}{M} \frac{N(x; \mu_1, \sigma_1)}{N(x; \mu_0, \sigma_0)}$ where the value of M is computed based on the base and the target distribution analytically so as to satisfy the textbook "soft" rejection sampling requirement, i.e. $M = \arg\max_x \frac{N(x; \mu_1, \sigma_1)}{N(x; \mu_0, \sigma_0)}$.

Program 2: GMM model

```
def GMM(mixture_params, sigma, y_obs,
        mu_0, sigma_0):
    u = sample(Uniform(low=0, high=1))
    while True:
        rs_start()
        if u < mixture_params.pi_1:
            mu = sample(Normal(mean=mu_0,
                                std=sigma_0))
            u2 = sample(Uniform(low=0, high=1))
            if u2 < alpha(x):
                rs_end()
                break
        else:
            mu = sample(Normal(mean=mu_2,
                                std=sigma_2))
            break
    y = observe(Normal(mean=mu, std=sigma), y_obs)
    return mu
```

In our experiments we set the mixture probabilities to $\pi_1 = 0.5$ and $\pi_2 = 0.5$, the parameters of the first Gaussian to $\mu_1 = -1, \sigma_1 = 1$, with $\mu_2 = 2, \sigma_2 = 1$ for the second Gaussian. The observation is $y = 0$. We set $\mu_0 = 1, \sigma_0 = 2$. For inference, we use $N(-2, 2)$ as the proposal for the first Gaussian and every other proposal is the same as the prior. Figure 4a shows aggregated results of running different inference methods 100 times each.

We also run an experiment on the same GMM with a proposal for μ_1 that will be perfect if all the samples from it gets accepted, and every other proposal is equal to the prior. This corresponds to the IC variant for handling rejection sampling proposed in (Baydin et al., 2018) where the proposal is learned perfectly, as this would be the ideal proposal it learns for this particular random variable. In this experiment, we call this proposal the perfect proposal. Figure 4b shows our results.

In our next setup, besides having the aforementioned perfect proposal for μ_1, we use $N(-0.5, 0.5)$ as a proposal for u_2 (which is sampled from the acceptance distribution, $\mathrm{Uniform}(0, 1)$ in the prior). This leads to less rejection at inference time and is a better overall proposal since, ideally, samples from the perfect proposal should not be rejected as distribution of accepted samples would be imperfect otherwise. The results are shown in Figure 4c.

In this experiment IC does not converge, even when we use the perfect proposal. This is in addition to its poor sample efficiency. This is evidence of the weight variance problem with existing IC approaches to amortized IS. They may not work even when the
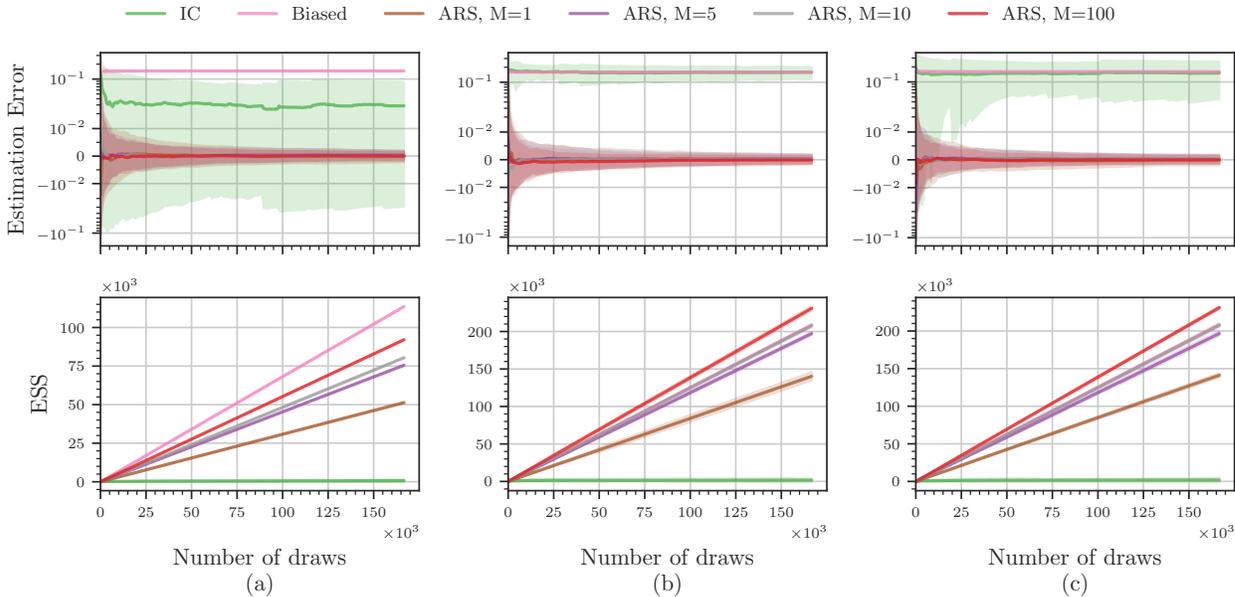
Figure 4: Results of GMM experiment. We target estimating $\mathbb{E}_{p(\mu|y)}[\mu]$ which can be computed analytically in this model. The plots show aggregation of 100 different runs of the experiment. (a) With a fixed proposal (b) With perfect proposal for the base distribution (c) With perfect proposal for the base distribution and a proposal for acceptance distribution that leads to less rejection. Our method with any M achieves zero estimation error with lower variance compared to IC. As expected, larger M leads to faster convergence, lower variance and higher ESS of the weights. In this example, IC does not converge to zero estimation error.

perfect proposal is used. Our method, as in the previous example, converges to the true expectation value with lower error, lower variance, and better sample efficiency (higher ESS).

5   IMPLEMENTATION

Program 3: Original	Program 4: Annotated
| ```
x = sample(P_x)
while True:

    z = sample(P_z(x))
    if c(x, z):


        break
observe(P_y(x,z), y)
return x, z
``` | ```
x = sample(P_x)
while True:
    rs_start()
    z = sample(P_z(x))
    if c(x, z):
        rs_end ()
        break
observe(P_y(x,z), y)
return x, z
``` |

Figure 5: An illustration of annotations required by our system. To apply our method we only require that entry and exit to each rejection sampling loop be annotated with `rs_start` and `rs_end` respectively. Our implementation then automatically handles the whole inference process, even if the annotated loops are nested.

In the previous sections we have presented and experimentally validated our method in simple programs, starting with the examplar program in Figure 1, which only has a single rejection sampling loop and a very simple structure otherwise. However, our method applies much more broadly, in fact to all probabilistic programs with arbitrary stochastic control flow structures and any number of rejection sampling loops, including nesting such loops to arbitrary degree. The only constraint is that observations can not be placed inside the loops, i.e. *no conditioning inside rejection sampling loops*. We conjecture that our method produces correct weights for all probabilistic programs satisfying this constraint, but proving that is beyond the scope of this paper and would require employing sophisticated machinery for constructing formal semantics, such as developed by Ścibior et al. (2017).

To enable practitioners to use our method in its full generality we have implementing it in PyProb (Le et al., 2017), a universal probabilistic programming library written in Python. The particulars of such a general purpose implementation pose several difficult but interesting problems, including identifying rejection sampling loops in the original program, addressing particular rejection sampling loops, and engineering solutions that allow acceptance probabilities bespoke to each loop to be estimated by repeatedly executing

the loops with different proposal distributions. In this section we describe some of these challenges and discuss our initial approach to solving them.

The first challenge is identifying rejection sampling loops in the probabilistic program itself. One mechanism for doing this is to introduce a rejection sampling primitive, macro, or syntactic sugar into the probabilistic programming language itself whose arguments are two functions, one the acceptance function, the other the body of the rejection sampling loop. While this may be feasible, our approach lies in a different part of the design space, given our choice to implement in PyProb and its applicability to performing inference in existing stochastic simulators. In this setting there are two other design choices: some kind of static analyzer that automates the labelling of rejection sampling loops by looking for rejection sampling motifs in the program (unclear how to accomplish this in a reliable and general way) or providing the probabilistic programmer functions that need to be carefully inserted into the existing probabilistic program to demarcate where rejection sampling loops start and end.

This is the initial approach we have taken. In the programs in Section 4 we nearly silently introduced the functions `rs_start` and `rs_end` to tag the beginning and end of rejection sampling loops. The purpose of these functions was hinted at then, but now is explained as the way to inform an inference engine about the scope of each rejection sampling loop, in particular so that all `sample` statements in between calls to `rs_start` and `rs_end` can be tracked. Program 5 illustrates where these primitives have to be inserted in probabilistic programs with rejection loops to invoke our ARS techniques.

The specific implementation details are beyond scope for this paper and require substantial review of PyProb internals, however, the key functionality enabled by these tags includes two critical things. First, we need to be able to execute additional iterations of every rejection sampling loop in the program to compute our estimator of $\frac{q(A|x,y)}{p(A|x)}$. For every `rs_start` we have to be able to continue the program multiple times, executing the rejection loop both proposing from p and q, terminating the continuation once the matching `rs_end` is reached. Efficient implementations of this make use of the same forking ideas that made probabilistic-c possible (Paige and Wood, 2014). Second, we have to be able to identify rejection sampler start and end pairs in a way that is more strict than the uniqueness requirement on random variable addresses in probabilistic programming. As the acceptance function of the rejection sampling loop can be an arbitrary function of program state, the identity (in the address sense)

of the rejection sampling loop and its associated exit probabilities under p and q have to be manually identified, using a manual addressing scheme like that used in Pyro for all random variables.

Our PyProb implementation addresses all of these issues and the details of our implementation will be made public as part of its continuing open source development.

6 DISCUSSION

We have addressed a significant issue in amortized and adapted importance-sampling-based inference for universal probabilistic programming languages. We have demonstrated that even simple rejection sampling loops cause major problems for existing probabilistic programming inference algorithms. In particular, we showed empirically and theoretically that sequential importance sampling will perform poorly in presence of rejection sampling loops. Our proposed method is an unbiased estimator with lower variance than naïve SIS estimators.

The cost of taking our approach is somewhat subtle but involves needing to estimate the exit probabilities of all rejections sampling loops in the program under both the prior and the proposal. In many cases this can be amortized in the sense that these rejection sampling loops can be statically determined and the relevant constants estimated at training time. Unfortunately if at test time a rejection sampling loop is encountered for which these constants have not been already estimated, the inference engine must "pause" and estimate them on the fly. Not only can this be slow but it seriously increases the implementation complexity of the probabilistic programming system.

However, when all is said and done, this "fix" of amortized and adapted importance-sampling-based inference for universal probabilistic programming systems is very significant, particularly as it pertains to uptake of this kind of probabilistic programming system. Current users of systems who wisely use rejection sampling in their generative models will experience the probabilistic programming system as confusingly but simply not working. This will be due to slowness if non-amortized inference techniques are utilized and, worse, poor sample efficiency and potentially non-diagnosable non-convergent behavior if existing fast amortized inference techniques are utilized instead. Our work makes it so that efficient, amortized inference engines work for probabilistic programs that users actually write and in so doing removes a major impediment to the uptake of universal probabilistic programming systems in general.

References

A. G. Baydin, L. Heinrich, W. Bhimji, B. Gram-Hansen, G. Louppe, L. Shao, K. Cranmer, F. Wood, et al. Efficient probabilistic inference in the quest for physics beyond the standard model. *arXiv preprint arXiv:1807.07706*, 2018.

A. G. Baydin, L. Shao, W. Bhimji, L. Heinrich, L. Meadows, J. Liu, A. Munk, S. Naderiparizi, B. Gram-Hansen, G. Louppe, et al. Etalumis: Bringing probabilistic programming to scientific simulators at scale. *arXiv preprint arXiv:1907.03382*, 2019.

X. Cai. Exact stochastic simulation of coupled chemical reactions with delays. *The Journal of chemical physics*, 126(12):124108, 2007.

S. Gershman and N. Goodman. Amortized inference in probabilistic reasoning. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 36, 2014.

N. D. Goodman, V. K. Mansinghka, D. Roy, K. Bonawitz, and J. B. Tenenbaum. Church: A language for generative models. *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence, Uai 2008*, pages 220–229, 2008.

T. A. Le, A. G. Baydin, and F. Wood. Inference compilation and universal probabilistic programming. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1338–1348, Fort Lauderdale, FL, USA, 2017. PMLR.

V. Mansinghka, D. Selsam, and Y. Perov. Venture: a higher-order probabilistic programming platform with programmable inference. *arXiv preprint arXiv:1404.0099*, 2014.

B. Paige and F. Wood. A compilation target for probabilistic programming languages. In *Proceedings of the 31st international conference on Machine learning*, volume 32, pages 1935–1943, 2014.

R. Ramaswamy and I. F. Sbalzarini. A partial-propensity variant of the composition-rejection stochastic simulation algorithm for chemical reaction networks. *The Journal of chemical physics*, 132 (4):044102, 2010.

D. Ritchie, B. Mildenhall, N. D. Goodman, and P. Hanrahan. Controlling procedural modeling programs with stochastically-ordered sequential Monte Carlo. *ACM Transactions on Graphics (TOG)*, 34 (4):105, 2015.

D. Ritchie, P. Horsfall, and N. D. Goodman. Deep amortized inference for probabilistic programs. *arXiv preprint arXiv:1610.05735*, 2016.

A. Ścibior, O. Kammar, M. Vákár, S. Staton, H. Yang, Y. Cai, K. Ostermann, S. K. Moss, C. Heunen, and Z. Ghahramani. Denotational validation of higher-order bayesian inference. *Proceedings of the ACM on Programming Languages*, 2(POPL):60, 2017.

A. M. Ścibior. *Formally justified and modular Bayesian inference for probabilistic programs*. PhD thesis, University of Cambridge, 2019.

A. Slepoy, A. P. Thompson, and S. J. Plimpton. A constant-time kinetic monte carlo algorithm for simulation of large biochemical reaction networks. *The journal of chemical physics*, 128(20):05B618, 2008.

A. Stuhlmüller and N. D. Goodman. Reasoning about reasoning by nested conditioning: Modeling theory of mind with probabilistic programs. *Cognitive Systems Research*, 28:80–99, 2014.

J.-W. van de Meent, B. Paige, H. Yang, and F. Wood. An introduction to probabilistic programming. *arXiv preprint arXiv:1809.10756*, 2018.

D. Wingate and T. Weber. Automated variational inference in probabilistic programming. *arXiv preprint arXiv:1301.1299*, 2013.

D. Wingate, A. Stuhlmüller, and N. D. Goodman. Lightweight implementations of probabilistic programming languages via transformational compilation. *Journal of Machine Learning Research*, 15: 770–778, 2011. ISSN 15337928, 15324435.

F. Wood, J. W. Meent, and V. Mansinghka. A new approach to probabilistic programming inference. In *Artificial Intelligence and Statistics*, pages 1024–1032, 2014.

A Importance weight variance

Theorem 5. *Under assumptions of Definition 2.1, if the following condition holds with positive probability under* $x \sim q(x|y)$

$$\mathbb{E}_{\boldsymbol{z} \sim q(\boldsymbol{z}|x,y)} \left[\frac{p(\boldsymbol{z}|x)^2}{q(\boldsymbol{z}|x,y)^2} (1 - p(A|x,\boldsymbol{z})) \right] \geq 1 \tag{17}$$

then the variance of w_{IC} *is infinite.*

Proof. In this proof, we carry the assumptions and definitions from Definition 2.1, with the exception that we use subscript for denoting weights and samples in iterations of the loop i.e. $\boldsymbol{z}^k \to \boldsymbol{z}_k$ and $w^k \to w_k$. Let $\mathbb{E}_q \left[\prod_{k=1}^{L-1} w_k \right]$ denote the mean value of the product of all the weights corresponding to the rejected samples, and $\mathbb{E}_q \left[\prod_{k=1}^{L-1} w_k^2 \right]$ is defined similarly. We will compute the variance of the importance weight of the rejected samples.

First, we compute the mean of weights:

$$\mathbb{E}_q \left[\prod_{k=1}^{L-1} w_k \right] = q(A|x,y) \sum_{L=1}^{\infty} q(\overline{A}|x,y)^{L-1} \prod_{k=1}^{L-1} \mathbb{E}_{\boldsymbol{z} \sim q} \left[w | \overline{A} \right] \tag{18}$$

$$\mathbb{E}_{\boldsymbol{z} \sim q} \left[w | \overline{A} \right] = \mathbb{E}_{\boldsymbol{z} \sim q(\boldsymbol{z}|x,y)} \left[\frac{p(\boldsymbol{z}|x)}{q(\boldsymbol{z}|x,y)} \frac{1 - c(x,\boldsymbol{z})}{q(\overline{A}|x,y)} \right] = \frac{1}{q(\overline{A}|x,y)} \mathbb{E}_{\boldsymbol{z} \sim p(\boldsymbol{z}|x)} \left[1 - c(x,\boldsymbol{z}) \right] = \frac{p(\overline{A}|x)}{q(\overline{A}|x,y)} \tag{19}$$

Therefore,

$$\mathbb{E}_q \left[\prod_{k=1}^{L-1} w_k \right] = q(A|x,y) \sum_{k=1}^{\infty} p(\overline{A}|x)^{k-1} = \boxed{\frac{q(A|x,y)}{p(A|x)}} \tag{20}$$

Next, we compute the expected value of squared of weights:

$$\mathbb{E}_q \left[\prod_{k=1}^{L-1} w_k^2 \right] = q(A|x,y) \sum_{L=1}^{\infty} q(\overline{A}|x,y)^{L-1} \prod_{k=1}^{L-1} \mathbb{E}_{\boldsymbol{z} \sim q(\boldsymbol{z}|x,y)} \left[w^2 | \overline{A} \right] \tag{21}$$

$$\mathbb{E}_{\boldsymbol{z} \sim q(\boldsymbol{z}|x,y)} \left[w^2 | \overline{A} \right] = \mathbb{E}_{\boldsymbol{z} \sim q} \left[\frac{p(\boldsymbol{z}|x)^2}{q(\boldsymbol{z}|x,y)^2} \frac{1 - c(x,\boldsymbol{z})}{q(\overline{A}|x,y)} \right] = \frac{1}{q(\overline{A}|x,y)} \mathbb{E}_{\boldsymbol{z} \sim q} \left[\frac{p(\boldsymbol{z}|x)^2}{q(\boldsymbol{z}|x,y)^2} p(\overline{A}|x,\boldsymbol{z}) \right] := \frac{S_{p,q}}{q(\overline{A}|x,y)} \tag{22}$$

Hence,

$$\mathbb{E}_q \left[\prod_{k=1}^{L-1} w_k^2 \right] = q(A|x,y) \sum_{L=1}^{\infty} (S_{p,q})^{L-1} \tag{23}$$

Consequently, since the mean of the weights is finite (Equation 20), if Equation 23 is not finite, the variance of the rejected weights will be infinite.

$$\text{if } S_{p,q} \geq 1 \Rightarrow Var \left(\prod_{i=1}^{k-1} w_i \right) \text{ is infinite} \tag{24}$$

Noting that if the variance of the weight of a subset of traces is infinite, the variance of the whole traces would be infinite completes the proof. □