

Towards Verified Stochastic Variational Inference for Probabilistic Programs

WONYEOL LEE, School of Computing, KAIST, South Korea

HANGYEOL YU, School of Computing, KAIST, South Korea

XAVIER RIVAL, INRIA Paris, Département d'Informatique of ENS, and CNRS/PSL University, France

HONGSEOK YANG, School of Computing, KAIST, South Korea

Probabilistic programming is the idea of writing models from statistics and machine learning using program notations and reasoning about these models using generic inference engines. Recently its combination with deep learning has been explored intensely, which led to the development of so called deep probabilistic programming languages, such as Pyro, Edward and ProbTorch. At the core of this development lie inference engines based on stochastic variational inference algorithms. When asked to find information about the posterior distribution of a model written in such a language, these algorithms convert this posterior-inference query into an optimisation problem and solve it approximately by a form of gradient ascent or descent. In this paper, we analyse one of the most fundamental and versatile variational inference algorithms, called score estimator or REINFORCE, using tools from denotational semantics and program analysis. We formally express what this algorithm does on models denoted by programs, and expose implicit assumptions made by the algorithm on the models. The violation of these assumptions may lead to an undefined optimisation objective or the loss of convergence guarantee of the optimisation process. We then describe rules for proving these assumptions, which can be automated by static program analyses. Some of our rules use nontrivial facts from continuous mathematics, and let us replace requirements about integrals in the assumptions, such as integrability of functions defined in terms of programs' denotations, by conditions involving differentiation or boundedness, which are much easier to prove automatically (and manually). Following our general methodology, we have developed a static program analysis for the Pyro programming language that aims at discharging the assumption about what we call model-guide support match. Our analysis is applied to the eight representative model-guide pairs from the Pyro webpage, which include sophisticated neural network models such as AIR. It finds a bug in one of these cases, reveals a non-standard use of an inference engine in another, and shows that the assumptions are met in the remaining six cases.

CCS Concepts: • **Mathematics of computing** → **Bayesian computation; Variational methods**; • **Theory of computation** → **Probabilistic computation; Denotational semantics**; • **Software and its engineering** → **Correctness; Automated static analysis**; • **Computing methodologies** → *Machine learning*.

Additional Key Words and Phrases: Probabilistic programming, static analysis, semantics, correctness

ACM Reference Format:

Wonyeol Lee, Hangyeol Yu, Xavier Rival, and Hongseok Yang. 2020. Towards Verified Stochastic Variational Inference for Probabilistic Programs. *Proc. ACM Program. Lang.* 4, POPL, Article 16 (January 2020), 68 pages. <https://doi.org/10.1145/3371084>

Authors' addresses: Wonyeol Lee, School of Computing, KAIST, South Korea, wonyeol@kaist.ac.kr; Hangyeol Yu, School of Computing, KAIST, South Korea, yhk1344@kaist.ac.kr; Xavier Rival, INRIA Paris, Département d'Informatique of ENS, and CNRS/PSL University, France, rival@di.ens.fr; Hongseok Yang, School of Computing, KAIST, South Korea, hongseok.yang@kaist.ac.kr.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

© 2020 Copyright held by the owner/author(s).

2475-1421/2020/1-ART16

<https://doi.org/10.1145/3371084>

1 INTRODUCTION

Probabilistic programming refers to the idea of writing models from statistics and machine learning using program notations and reasoning about these models using generic inference engines. It has been the subject of active research in machine learning and programming languages, because of its potential for enabling scientists and engineers to design and explore sophisticated models easily; when using these languages, they no longer have to worry about developing custom inference engines for their models, a highly-nontrivial task requiring expertise in statistics and machine learning. Several practical probabilistic programming languages now exist, and are used for a wide range of applications [Carpenter et al. 2017; Gehr et al. 2016; Goodman et al. 2008; Gordon et al. 2014; Mansinghka et al. 2014; Minka et al. 2014; Narayanan et al. 2016; Wood et al. 2014].

We consider inference engines that lie at the core of so called deep probabilistic programming languages, such as Pyro [Bingham et al. 2019], Edward [Tran et al. 2018, 2016] and ProbTorch [Siddharth et al. 2017]. These languages let users freely mix deep neural networks with constructs from probabilistic programming, in particular, those for writing Bayesian probabilistic models. In so doing, they facilitate the development of probabilistic deep-network models that may address the problem of measuring the uncertainty in current non-Bayesian deep-network models; a non-Bayesian model may predict that the price of energy goes up and that of a house goes down, but it cannot express, for instance, that the model is confident with the first prediction but not the second.

The primary inference engines for these deep probabilistic programming languages implement stochastic (or black-box) variational inference¹ algorithms. Converting inference problems into optimisation problems is the high-level idea of these algorithms.² When asked to find information about the posterior distribution of a model written in such a language, these algorithms convert this question to an optimisation problem and solve the problem approximately by performing a gradient descent or ascent on the optimisation objective. The algorithms work smoothly with gradient-based parameter-learning algorithms for deep neural networks, which is why they form the backbone for deep probabilistic programming languages.

In this paper, we analyse one of the most fundamental and versatile variational inference algorithms, called score estimator or REINFORCE³ [Paisley et al. 2012; Ranganath et al. 2014; Williams 1992; Wingate and Weber 2013], using tools from denotational semantics and program analysis [Cousot and Cousot 1977, 1979, 1992]. We formally express what this algorithm does on models denoted by probabilistic programs, and expose implicit assumptions made by the algorithm on the models. The violation of these assumptions can lead to undefined optimisation objective or the loss of convergence guarantee of the optimisation process. We then describe rules for proving these assumptions, which can be automated by static program analyses. Some of our rules use nontrivial facts from continuous mathematics, and let us replace requirements about integrals in the assumptions, such as integrability of functions defined in terms of programs' denotations, by the conditions involving differentiation or boundedness, which are much easier to prove automatically (and manually) than the original requirements.

Following our general methodology, we have developed a static program analysis for the Pyro programming language that can discharge one assumption of the inference algorithm about so

¹The term *stochastic* variational inference (VI) often refers to VI with data subsampling [Hoffman et al. 2013], and our usage of the term is often called black-box VI [Ranganath et al. 2014] to stress the treatment of a model as a black-box sampler.

²The inference problems in their original forms involve solving summation/integration/counting problems, which are typically more difficult than optimisation problems. The variational-inference algorithms convert the former problems to the latter ones, by looking for approximate, not exact, answers to the former.

³REINFORCE [Williams 1992] is an algorithm originally developed for reinforcement learning (RL), but it is commonly used as a synonym of the score-estimator algorithm. This is because REINFORCE and score estimator use a nearly identical method for estimating the gradient of an optimisation objective.

called model-guide pairs. In Pyro and other deep probabilistic programming languages, a program denoting a model typically comes with a companion program, called guide, decoder, or inference network. This companion, which we call guide, helps the inference algorithm to find a good approximation to what the model ultimately denotes under a given dataset (i.e., the posterior distribution of the model under the dataset); the algorithm uses the guide to fix the search space of approximations, and solves an optimisation problem defined on that space. A model and a guide should satisfy an important correspondence property, which says that they should use the same sets of random variables, and for any such random variable, if the probability of the variable having a particular value is zero in the model, it should also be zero in the guide. If the property is violated, the inference algorithm may attempt to solve an optimisation problem with undefined optimisation objective and return parameter values that do not make any sense. Our static analysis checks this correspondence property for Pyro programs. When applied to eight representative model-guide pairs from the Pyro webpage, which include sophisticated neural network models such as Attend-Infer-Repeat (AIR), the analysis found a bug in one of these cases, revealed a non-standard use of the inference algorithm in another, and proved that the property holds in the remaining six cases.

Another motivation for this paper is to demonstrate an opportunity for programming languages and verification research to have an impact on the advances of machine learning and AI technologies. One popular question is: what properties should we verify on machine-learning programs? Multiple answers have been proposed, which led to excellent research results, such as those on robustness of neural networks [Mirman et al. 2018]. But most of the existing research focuses on the final outcome of machine learning algorithms, not the process of applying these algorithms. One of our main objectives is to show that the process often relies on multiple assumptions on models and finding automatic ways for discharging these assumptions can be another way of making PL and verification techniques contribute. While our suggested solutions are not complete, they are intended to show the richness of this type of problems in terms of theory and practice.

We summarise the contributions of the paper:

- We formally express the behaviour of the most fundamental variational inference algorithm on probabilistic programs using denotational semantics, and identify requirements on program denotations that are needed for this algorithm to work correctly.
- We describe conditions that imply the identified requirements but are easier to prove. The sufficiency of the conditions relies on nontrivial results from continuous mathematics. We sketch a recipe for building program analyses for checking these conditions automatically.
- We present a static analysis for the Pyro language that checks the correspondence requirement of model-guide pairs. The analysis is based on our recipe, but extends it significantly to address challenges for dealing with features of the real-world language. Our analysis has successfully verified 6 representative Pyro model-guide examples, and found a bug in one example.

The extended version of the paper can be found in [Lee et al. 2019].

2 VARIATIONAL INFERENCE AND VERIFICATION CHALLENGES BY EXAMPLES

We start by explaining informally the idea of stochastic variational inference (in short SVI), one fundamental SVI algorithm, and the verification challenges that arise when we use this algorithm.

2.1 Stochastic Variational Inference

In a probabilistic programming language, we specify a model by a program. The program `model()` in Figure 1(a) is an example. It describes a joint probability density $p(v, obs)$ on two real-valued random variables v and obs . The value of the former is not observed, while the latter is observed to have the value 0. Finding out the value of v is the objective of writing this model. The joint

```

1  # define model and guide
2  def model():
3      v = pyro.sample("v", Normal(0., 5.))
4      if (v > 0):
5          pyro.sample("obs", Normal(1., 1.), obs=0.)
6      else:
7          pyro.sample("obs", Normal(-2., 1.), obs=0.)
8
9  def guide():
10     theta = pyro.param("theta", 3.)
11     v = pyro.sample("v", Normal(theta, 1.))
12
13 # perform stochastic variational inference
14 svi = SVI(model, guide,
15           Adam({"lr": 1.0e-2}),
16           loss=Trace_ELBO())
17 for step in range(2000):
18     svi.step()
19
20 # print result
21 print("trained theta =",
22       pyro.param("theta").item())

```

(a) Example model-guide pair for stochastic variational inference in Pyro.

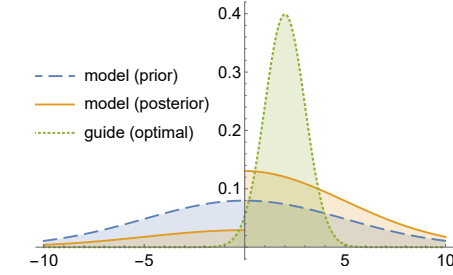
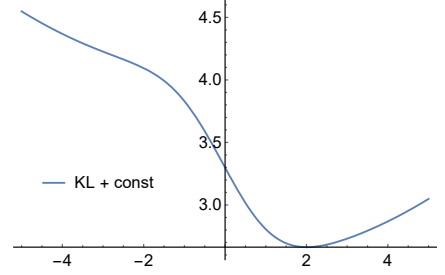
(b) Probability densities of the model and the guide as a function of $v \in \mathbb{R}$.(c) KL divergence from the guide to the model (plus $\log p(\text{obs}=0)$) as a function of $\theta \in \mathbb{R}$.

Fig. 1. Example of performing stochastic variational inference.

density $p(v, \text{obs})$ is expressed in terms of *prior* $p(v)$ and *likelihood* $p(\text{obs}|v)$ in the program. The prior $p(v)$ of v is the normal distribution with mean 0 and standard deviation 5, and it expresses the belief about the possible value of v before any observation. The likelihood $p(\text{obs}|v)$ is a normal distribution whose mean and standard deviation are either (1, 1) or (−2, 1) depending on the sign of the value of v . The purpose of most inference algorithms is to compute exactly or approximately the *posterior* density given a prior and a likelihood. In our example, the posterior $p(v|\text{obs}=0)$ is:

$$p(v|\text{obs}=0) = \frac{p(v, \text{obs}=0)}{\int dv p(v, \text{obs}=0)} = \frac{p(v) \cdot p(\text{obs}=0|v)}{p(\text{obs}=0)}.$$

Intuitively, the posterior expresses an updated belief on v upon observing $\text{obs} = 0$. The dashed blue and solid orange lines in Figure 1(b) show the prior and posterior densities, respectively. Note that the density of a positive v in the prior went up in the posterior. This is because when $v > 0$, the mean of $p(\text{obs}|v)$ is 1, a value closer to the observed value 0 than the alternative −2 for the mean.

SVI algorithms approach the posterior inference problem from the optimisation angle. They consider a collection of approximating distributions to a target posterior, formulate the problem of finding a good approximation in the collection as an optimisation problem, and solve the optimisation problem. The solution becomes the result of those algorithms. In Pyro, the users specify such a collection by a single parameterised program called *guide*; the collection can be generated by instantiating the parameters with different values. The program `guide()` in Figure 1(a) is an example. It has a real-valued parameter θ (written as `theta` in the program), and states that the probability density $q_\theta(v)$ of v is the normal distribution with unknown mean θ and standard deviation 1. The lines 13–17 in the figure show how to apply a standard SVI engine of Pyro (called `Trace_ELBO`) to find a good θ . They instruct the engine to solve the following optimisation problem:

$$\operatorname{argmin}_\theta \text{KL}(q_\theta(v) || p(v|\text{obs}=0)), \quad \text{where } \text{KL}(q_\theta(v) || p(v|\text{obs}=0)) \triangleq \mathbb{E}_{q_\theta(v)} \left[\log \frac{q_\theta(v)}{p(v|\text{obs}=0)} \right].$$

The optimisation objective $\text{KL}(q_\theta(v)||p(v|obs=0))$ is the *KL divergence* from $q_\theta(v)$ to $p(v|obs=0)$, and measures the similarity between the two densities, having a small value when the densities are similar. The KL divergence is drawn in Figure 1(c) as a function of θ , and the dotted green line in Figure 1(b) draws the density q_θ at the optimum θ . Note that the mean of this distribution is biased toward the positive side, which reflects the fact that the property $v > 0$ has a higher probability than its negation $v \leq 0$ in the posterior distribution.

One of the most fundamental and versatile algorithms for SVI is score estimator (also called RE-INFORCE). It repeatedly improves θ in two steps. First, it estimates the gradient of the optimisation objective with samples from the current q_{θ_n} :

$$\nabla_\theta \text{KL}(q_\theta(v)||p(v|obs=0)) \Big|_{\theta=\theta_n} \approx \frac{1}{N} \sum_{i=1}^N (\nabla_\theta \log q_{\theta_n}(v_i)) \cdot \log \frac{q_{\theta_n}(v_i)}{p(v_i, obs=0)}$$

where v_1, \dots, v_N are independent samples from the distribution q_{θ_n} . Then, the algorithm updates θ with the estimated gradient (the specific learning rate 0.01 is chosen to improve readability):

$$\theta_{n+1} \leftarrow \theta_n - 0.01 \times \frac{1}{N} \sum_{i=1}^N (\nabla_\theta \log q_{\theta_n}(v_i)) \cdot \log \frac{q_{\theta_n}(v_i)}{p(v_i, obs=0)}.$$

When the learning rate 0.01 is adjusted according to a known scheme, the algorithm is guaranteed to converge to a local optimum (in many cases) because its gradient estimate satisfies the following unbiasedness property (in those cases):

$$\nabla_\theta \text{KL}(q_\theta(v)||p(v|obs=0)) \Big|_{\theta=\theta_n} = \mathbb{E} \left[\frac{1}{N} \sum_{i=1}^N (\nabla_\theta \log q_{\theta_n}(v_i)) \cdot \log \frac{q_{\theta_n}(v_i)}{p(v_i, obs=0)} \right] \quad (1)$$

where the expectation is taken over the independent samples v_1, \dots, v_N from q_{θ_n} .

2.2 Verification Challenges

We now give two example model-guide pairs that illustrate verification challenges related to SVI.

The first example appears in Figure 2(a). It is the Bayesian regression example from the Pyro webpage (this example is among the benchmarks used in §8), which solves the problem of finding a line that interpolates a given set of points in \mathbb{R}^2 .

The problem with this example is that the KL divergence of its model-guide pair, the main optimisation objective in SVI, is undefined. The model and guide in the figure use the random variable `sigma`, but they use different non-zero-probability regions, called supports, for it. In the model, the support is $[0, 10]$, while that in the guide is \mathbb{R} . But the KL divergence from a guide to a model is defined only if for every random variable, its support in the guide is included in that in the model. We point out that this support mismatch was found by our static analyser explained in §8.

Figures 2(b) and 2(c) show two attempts to resolve the undefined-KL issue. To fix the issue, we change the distribution of `sigma` in the model in (b), and in the guide in (c). These revisions remove the problem about the support of `sigma`, but do not eliminate that of the undefined KL. In both (b) and (c), the KL divergence is ∞ . This happens mainly because `sigma` can be arbitrarily close to 0 in the guide in both cases, which makes integrand in the definition of the KL divergence diverge to ∞ .

An SVI-specific verification challenge related to this example is how to prove the well-definedness of the KL divergence and more generally the optimisation objective of an SVI algorithm. In §6.2, we provide a partial answer to the question. We give a condition for ensuring the well-definedness of the KL divergence. Our condition is more automation-friendly than the definition of KL, because it does not impose the difficult-to-check integrability requirement present in the definition of KL.

<pre> 1 def model(...): 2 ... 3 sigma = pyro.sample("sigma", 4 Uniform(0., 10.)) 5 ... 6 pyro.sample("obs", 7 Normal(..., sigma), obs=...) </pre>	<pre> 8 def guide(...): 9 ... 10 loc = pyro.param("sigma_loc", 1., 11 constraint=constraints.positive) 12 ... 13 sigma = pyro.sample("sigma", 14 Normal(loc, 0.05)) </pre>
---	---

(a) Bayesian regression example from the Pyro webpage.

<pre> 1 def model_r1(...): 2 ... 3 sigma = pyro.sample("sigma", 4 Normal(5., 5.)) 5 ... 6 pyro.sample("obs", 7 Normal(..., abs(sigma)), obs=...) 8 9 def guide_r1(...): 10 # same as guide() in (a) 11 ... </pre>	<pre> 1 def model_r2(...): 2 # same as model() in (a) 3 ... 4 5 def guide_r2(...): 6 ... 7 sigma = pyro.sample("sigma", 8 Uniform(0., 10.)) </pre>
---	--

(b) The example with a revised model.

(c) The example with a revised guide.

Fig. 2. Example model-guide pairs whose KL divergence is undefined.

```

1 def model():
2     v = pyro.sample("v", Normal(0., 5.))
3     if (v > 0):
4         pyro.sample("obs", Normal(1., 1.), obs=0.)
5     else:
6         pyro.sample("obs", Normal(-2., 1.), obs=0.)
7
8 def guide():
9     theta = pyro.param("theta", 3.)
10    v = pyro.sample("v",
11        Uniform(theta-1., theta+1.))

```

(a) The model from Figure 1(a), and a guide using a parameterised uniform distribution.

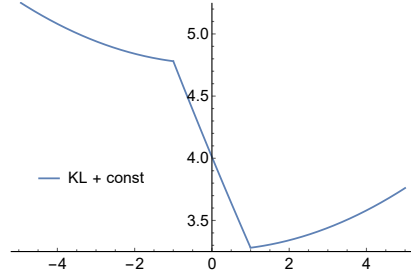
(b) KL divergence from the guide to the model (plus $\log p(\text{obs}=0)$) as a function of $\theta \in \mathbb{R}$.

Fig. 3. Example model-guide pair for which the gradient of the KL divergence is undefined, or the score estimator is biased.

The second example appears in Figure 3(a). It uses the same model as in Figure 1(a), but has a new guide that uses a uniform distribution parameterised by $\theta \in \mathbb{R}$. For this model-guide pair, the KL divergence is well-defined for all $\theta \in \mathbb{R}$, and the optimal θ^* minimising the KL is $\theta^* = 1$.

However, as shown in Figure 3(b), the gradient of the KL divergence is undefined for $\theta \in \{-1, 1\}$, because the KL divergence is not differentiable at -1 and 1 . For all the other $\theta \in \mathbb{R} \setminus \{-1, 1\}$, the KL divergence and its gradient are both defined, but the score estimator cannot estimate this gradient in an unbiased manner (i.e., in a way satisfying (1)), thereby losing the convergence guarantee to a local optimum. The precise calculation is not appropriate in this section, but we just point out that the expectation of the estimated gradient is always zero for all $\theta \in \mathbb{R} \setminus \{-1, 1\}$, but the true gradient of the KL is always non-zero for those θ , because it has the form: $\frac{\theta}{25} - \mathbb{1}_{[-1 \leq \theta \leq 1]} \cdot \frac{1}{2} \log \frac{\mathcal{N}(0;1,1)}{\mathcal{N}(0;-2,1)}$. Here $\mathcal{N}(v; \mu, \sigma)$ is the density of the normal distribution with mean μ and standard deviation σ (concretely, $\mathcal{N}(v; \mu, \sigma) = 1/(\sqrt{2\pi}\sigma) \cdot \exp(-(v-\mu)^2/(2\sigma^2))$). The mismatch comes from the invalidity of one implicit assumption about interchanging integration and gradient in the justification of the score estimator; see §5 for detail.

To sum up, the second example shows that even if the KL divergence is defined, its gradient is sometimes undefined, and also that even if both the KL divergence and its gradient are defined, the sample-based estimate of the gradient in a standard SVI algorithm may be biased—this means that the equation similar to (1) does not hold and an SVI algorithm is no longer guaranteed to converge to a local optimum. Proving that these failure cases do not arise is another SVI-specific verification challenge. In §6.3, we give another example of similar flavour, and provide an automation-friendly condition that ensures the existence of the KL divergence and its gradient as well as the unbiasedness of the gradient estimate of the score estimator.

We conclude the section by emphasising that the aforementioned issues could have a large impact on the results of SVI. For instance, running the Bayesian regression example in Figure 2(a) in Pyro after a small change (0.4 instead of 0.05 in line 14) sometimes results in a crash. Also, running its revised version in Figure 2(c) leads to the complete failure of an optimiser and produces meaningless results. These observations strengthen the importance of resolving the verification challenges presented above.⁴

3 REVIEW OF MEASURE THEORY AND NOTATIONS

A σ -algebra Σ on a set X is a collection of subsets of X such that (i) $X \in \Sigma$; (ii) $A_0 \cup A_1 \in \Sigma$ and $X \setminus A_0 \in \Sigma$ for all $A_0, A_1 \in \Sigma$; (iii) $\bigcup_n A_n \in \Sigma$ when all subsets A_n are in Σ . An equivalent but easier-to-remember characterisation is that Σ is closed under boolean operations and countable union and intersection. We call the pair (X, Σ) of a set and a σ -algebra *measurable space*, and subsets in Σ *measurable*. A function f from a measurable space (X, Σ) to another measurable space (X', Σ') is *measurable* if $f^{-1}(B) \in \Sigma$ for all $B \in \Sigma'$.

An example of measurable space is the n -dimensional Euclidean space \mathbb{R}^n with the *Borel σ -algebra* $\mathcal{B} \triangleq \sigma(\{(-\infty, r_1) \times \cdots \times (-\infty, r_n) \mid r \in \mathbb{R}^n\})$, where σ is the closure operator that converts a collection of subsets of \mathbb{R}^n into the smallest σ -algebra containing the collection. Subsets X of \mathbb{R}^n , such as $[0, \infty)^n$, form measurable spaces with the σ -algebra $\{X \cap A \mid A \in \mathcal{B}\}$. Another example is a set X with the so called *discrete σ -algebra* on X that consists of all subsets of X .

A *measure* μ on a measurable space (X, Σ) is a function from Σ to $[0, \infty]$ such that $\mu(\emptyset) = 0$ and μ satisfies the *countable additivity condition*: $\mu(\bigcup_{n=0}^{\infty} B_n) = \sum_{n=0}^{\infty} \mu(B_n)$ for any *countable* family of *disjoint* measurable subsets B_n . A well-known example is the Lebesgue measure λ^n on \mathbb{R}^n which maps each measurable subset of \mathbb{R}^n to its volume in the usual sense.⁵ When $\mu(X) \leq 1$, we call μ *subprobability measure*. If $\mu(X) = 1$, we may drop “sub”, and call μ *probability measure*.

The Lebesgue integral \int is a partial operator that maps a measure μ on (X, Σ) and a real-valued measurable function on the same space (X, Σ) to a real number. It is denoted by $\int \mu(dx) f(x)$. To follow the paper, it is enough to know that this integral generalises the usual Riemann integral from calculus.⁶ For a measure ν on (X, Σ) , if $\nu(A) = \int \mu(dx) (f(x) \cdot \mathbb{1}_{x \in A})$ for non-negative f , we say that f is the *density* of ν with respect to μ and call μ *reference measure*.

In the paper, we use a few well-known methods for building measurable spaces.

The first method applies when we are given a set X and a collection of functions $\{f_i : X \rightarrow X_i \mid i \in I\}$ to measurable spaces (X_i, Σ_i) . The method is to equip X with the smallest σ -algebra Σ making all f_i ’s measurable: $\Sigma \triangleq \sigma(\{f_i^{-1}(B) \mid i \in I, B \in \Sigma_i\})$.

⁴The aforementioned issues do not always cause problems in inference results (e.g., Figures 2(a) and 2(c) mostly give reasonable results), because the random seeds and the initial values for SVI could be set well so that the probability of SVI going wrong becomes low. We emphasise, however, that the probability is non-zero.

⁵The Lebesgue measure λ^n is the unique measure on \mathbb{R}^n that sets the volume of the unit cube $(0, 1)^n$ to 1 and is translation invariant: for all measurable subsets A and $r \in \mathbb{R}^n$, $\lambda^n(A) = \lambda^n(\{r' - r \mid r' \in A\})$.

⁶Another useful fact is that when f is non-negative, $\int \mu(dx) f(x) = \sup \sum_i (\inf_{x \in A_i} f(x)) \cdot \mu(A_i)$ where the supremum is taken with respect to all finite partitions $\{A_i\}_i$ of X into measurable subsets.

The second relies on two constructions for sets, i.e., product and disjoint union. Suppose that we are given measurable spaces (X_i, Σ_i) for all $i \in I$. We define a *product measurable space* that has $\prod_{i \in I} X_i$ as its underlying set and the following product σ -algebra $\bigotimes_{i \in I} \Sigma_i$ as its σ -algebra:

$$\bigotimes_{i \in I} \Sigma_i \triangleq \sigma\left(\left\{\prod_i A_i \mid \text{there is a finite } I_0 \subseteq I \text{ such that } (\forall j \in I \setminus I_0. A_j = X_j) \wedge (\forall i \in I_0. A_i \in \Sigma_i)\right\}\right).$$

The construction of the product σ -algebra can be viewed as a special case of the first method where we consider the smallest σ -algebra on $\prod_{i \in I} X_i$ that makes every projection map to X_i measurable. When the X_i are disjoint, they can be combined as disjoint union. The underlying set in this case is $\bigcup_{i \in I} X_i$, and the σ -algebra is

$$\bigoplus_{i \in I} \Sigma_i \triangleq \{A \mid A \cap X_i \in \Sigma_i \text{ for all } i \in I\}.$$

When $I = \{i, j\}$ with $i \neq j$, we denote the product measurable space by $(X_i \times X_j, \Sigma_i \otimes \Sigma_j)$. In addition, if X_i and X_j are disjoint, we write $(X_i \cup X_j, \Sigma_i \oplus \Sigma_j)$ for the disjoint-union measurable space.

The third method builds a measurable space out of measures or a certain type of measures, such as subprobability measures. For a measurable space (X, Σ) , we form a measurable space with measures. The underlying set $\text{Mea}(X)$ and σ -algebra $\Sigma_{\text{Mea}(X)}$ of the space are defined by

$$\text{Mea}(X) \triangleq \{\mu \mid \mu \text{ is a measure on } (X, \Sigma)\}, \quad \Sigma_{\text{Mea}(X)} \triangleq \sigma(\{\{\mu \mid \mu(A) \leq r\} \mid A \in \Sigma, r \in \mathbb{R}\}).$$

The difficult part to grasp is $\Sigma_{\text{Mea}(X)}$. Once again, a good approach for understanding it is to realise that $\Sigma_{\text{Mea}(X)}$ is the smallest σ -algebra that makes the function $\mu \mapsto \mu(A)$ from $\text{Mea}(X)$ to \mathbb{R} measurable for all measurable subsets $A \in \Sigma$. This measurable space gives rise to a variety of measurable spaces, each having a subset M of $\text{Mea}(X)$ as its underlying set and the induced σ -algebra $\Sigma_M = \{A \cap M \mid A \in \Sigma_{\text{Mea}(X)}\}$. In the paper, we use two such spaces, one induced by the set $\text{Sp}(X)$ of subprobability measures on X and the other by the set $\text{Pr}(X)$ of probability measures.

A measurable function f from (X, Σ) to $(\text{Mea}(Y), \Sigma_{\text{Mea}(Y)})$ is called *kernel*. If $f(x)$ is a subprobability measure (i.e., $f(x) \in \text{Sp}(Y)$) for all x , we say that f is a *subprobability kernel*. In addition, if $f(x)$ is a probability measure (i.e., $f(x) \in \text{Pr}(Y)$) for all x , we call f *probability kernel*. A good heuristic is to view a probability kernel as a random function and a subprobability kernel as a random partial function. We use well-known facts that a function $f : X \rightarrow \text{Mea}(Y)$ is a subprobability kernel if and only if it is a measurable map from (X, Σ) to $(\text{Sp}(Y), \Sigma_{\text{Sp}(Y)})$, and that similarly a function f is a probability kernel if and only if it is a measurable function from (X, Σ) to $(\text{Pr}(Y), \Sigma_{\text{Pr}(Y)})$.

We use a few popular operators for constructing measures throughout the paper. We say that a measure μ on a measurable space (X, Σ) is *finite* if $\mu(X) < \infty$, and *σ -finite* if there is a countable partition of X into measurable subsets X_n 's such that $\mu(X_n) < \infty$ for every n . Given a finite or countable family of σ -finite measures $\{\mu_i\}_{i \in I}$ on measurable spaces (X_i, Σ_i) 's, the product measure of μ_i 's, denoted $\bigotimes_{i \in I} \mu_i$, is the unique measure on $(\prod_{i \in I} X_i, \bigotimes_{i \in I} \Sigma_i)$ such that $(\bigotimes_{i \in I} \mu_i)(\prod_{i \in I} A_i) = \prod_{i \in I} \mu_i(A_i)$ for all measurable subsets A_i of X_i . Given a finite or countable family of measures $\{\mu_i\}_{i \in I}$ on disjoint measurable spaces (X_i, Σ_i) 's, the sum measure of μ_i 's, denoted $\bigoplus_{i \in I} \mu_i$, is the unique measure on $(\sum_{i \in I} X_i, \bigoplus_{i \in I} \Sigma_i)$ such that $(\bigoplus_{i \in I} \mu_i)(\bigcup_{i \in I} A_i) = \sum_{i \in I} \mu_i(A_i)$.

Throughout the paper, we take the convention that the set \mathbb{N} of natural numbers includes 0. For all positive integers n , we write $[n]$ to mean the set $\{1, 2, \dots, n\}$.

4 SIMPLE PROBABILISTIC PROGRAMMING LANGUAGE

In this section, we describe the syntax and semantics of a simple probabilistic programming language, which we use to present the theoretical results of the paper. The measure semantics in

<i>Real constants</i>	$c \in \mathbb{R}$	<i>Primitive real-valued functions</i>	$f ::= \dots$
<i>String constants</i>	$\alpha \in \text{Str}$	<i>Primitive string-valued functions</i>	$g ::= \dots$
<i>Real expressions</i>	$E ::= c \mid x \mid f(E, \dots, E)$		
<i>Boolean expressions</i>	$B ::= \text{true} \mid E < E \mid B \wedge B \mid \neg B$		
<i>String expressions</i>	$S ::= \alpha \mid g(S, \dots, S, E, \dots, E)$		
<i>Commands</i>	$C ::= \text{skip} \mid x := E \mid C; C \mid \text{if } B \{C\} \text{ else } \{C\} \mid \text{while } B \{C\}$ $\mid x := \text{sample}_{\text{norm}}(S, E, E) \mid \text{score}_{\text{norm}}(E, E, E)$		

Fig. 4. Grammar of our core language.

§4.2 uses results and observations from [Staton et al. 2016], but the density semantics and the other materials in §4.3 are new in this work.

4.1 Syntax

We use an extension of the standard while language with primitives for probabilistic programming. The grammar of the language is given in Figure 4. Variables in the language store real numbers, but expressions may denote reals, booleans and strings, and they are classified into E, B, S based on these denoted values. The primitive functions f for reals and g for strings may be usual arithmetic and string operations, such as multiplication and exponentiation for f and string concatenation for g .

The grammar for C includes the cases for the standard constructs of the while language, such as assignment, sequencing, conditional statement and loops. In addition, it has two constructs for probabilistic programming. The first $x := \text{sample}_{\text{norm}}(S, E_1, E_2)$ draws a sample from the *normal* distribution with mean E_1 and standard deviation E_2 and names the sample with the string S . The next $\text{score}_{\text{norm}}(E_0, E_1, E_2)$ expresses that a sample is drawn from the normal distribution with mean E_1 and standard deviation E_2 and the value of this sample is observed to be E_0 . It lets the programmers express information about observed data inside programs. Operationally, this construct can be understood as an instruction for updating a global variable that stores the so called importance score of the execution. The importance score quantitatively records how well the random choices in the current execution match the observations, and the score statement updates this score by multiplying it with the density at E_0 of the appropriate normal distribution.

Consider the following program:

$$x := \text{sample}_{\text{norm}}("a", 0.0, 5.0); \text{score}_{\text{norm}}(3.0, x, 1.0).$$

The program specifies a model with one random variable “ a ”. Using a relatively flat normal distribution, the program specifies a prior belief that the value of the random variable “ a ” is likely to be close to 0.0 and lie between -2×5.0 and 2×5.0 . The next score statement refines this belief with one data point 3.0, which is a noisy observation of the value of “ a ” (bound to x). The parameters to the normal density in the statement express that the noise is relatively small, between -2×1.0 and 2×1.0 . Getting the refined belief, called posterior distribution, is the reason that a data scientist writes a model like this program. It is done by an inference algorithm of the language.

Permitting only the normal distribution does not limit the type of models expressible in the language. Every distribution can be obtained by transforming the standard normal distribution.⁷

⁷Here we consider only Borel spaces. Although using only the normal distribution does not affect the expressiveness, it has an impact on stochastic variational inference to be discussed later, because it requires a guide to use only normal distributions. But the impact is not significant, because most well-known approaches for creating guides from the machine-learning literature (such as extensions of variational autoencoder) use normal distributions only, or can be made to do so easily.

4.2 Measure Semantics

The denotational semantics of the language just presented is mostly standard, but employs some twists to address the features for probabilistic programming [Staton et al. 2016].

Here is a short high-level overview of the measure semantics. Our semantics defines multiple measurable spaces, such as *Store* and *State*, that hold mathematical counterparts to the usual actors in computation, such as program stores (i.e., mappings from variables to values) and states (which consist of a store and further components). Then, the semantics interprets expressions E, B, S and commands C as measurable functions of the following types:

$$\llbracket E \rrbracket : \text{Store} \rightarrow \mathbb{R}, \quad \llbracket B \rrbracket : \text{Store} \rightarrow \mathbb{B}, \quad \llbracket S \rrbracket : \text{Store} \rightarrow \text{Str}, \quad \llbracket C \rrbracket : \text{State} \rightarrow \text{Sp}(\text{State} \times [0, \infty)).$$

Here \mathbb{R} is the measurable space of reals with the Borel σ -algebra, and \mathbb{B} and *Str* are discrete measurable spaces of booleans and strings. *Store* and *State* are measurable spaces for stores (i.e., maps from variables to values) and states which consist of a store and a part for recording information about sampled random variables. Note that the target measurable space of commands is built by first taking the product of measurable spaces *State* and $[0, \infty)$ and then forming a space out of subprobability measures on $\text{State} \times [0, \infty)$. This construction indicates that commands denote probabilistic computations, and the result of each such computation consists of an output state and a score which expresses how well the computation matches observations expressed with the score statements in the command C . Some of the possible outcomes of the computation may lead to non-termination or an error, and these abnormal outcomes are not accounted for by the semantics, which is why $\llbracket C \rrbracket(\sigma)$ for a state $\sigma \in \Sigma$ is a subprobability distribution. The semantics of expressions is much simpler. It just says that expressions do not involve any probabilistic computations, so that they denote deterministic measurable functions.

We now explain how this high-level idea gets implemented in our semantics. Let *Var* be a countably infinite set of variables. Our semantics uses the following sets:

$$\begin{aligned} \text{Stores} \quad s \in \text{Store} &\triangleq [\text{Var} \rightarrow \mathbb{R}] \quad (\text{which is isomorphic to } \prod_{x \in \text{Var}} \mathbb{R}) \\ \text{Random databases} \quad r \in \text{RDB} &\triangleq \bigcup_{K \subseteq_{\text{fin}} \text{Str}} [K \rightarrow \mathbb{R}] \quad (\text{which is isomorphic to } \bigcup_{K \subseteq_{\text{fin}} \text{Str}} \prod_{\alpha \in K} \mathbb{R}) \\ \text{States} \quad \sigma \in \text{State} &\triangleq \text{Store} \times \text{RDB}, \end{aligned}$$

where $[X \rightarrow Y]$ denotes the set of all functions from X to Y . A state σ consists of a store s and a random database r . The former fixes the values of variables, and the latter records the name (given as a string) and the value of each sampled random variable. The domain of r is the names of all the sampled random variables. By insisting that r should be a map, the semantics asserts that no two random variables have the same name. For each state σ , we write σ_s and σ_r for its store and random database components, respectively. Also, for a variable x , a string α and a value v , we write $\sigma[x \mapsto v]$ and $\sigma[\alpha \mapsto v]$ to mean $(\sigma_s[x \mapsto v], \sigma_r)$ and $(\sigma_s, \sigma_r[\alpha \mapsto v])$.

We equip all of these sets with σ -algebras and turn them to measurable spaces in a standard way. Note that we constructed the sets from \mathbb{R} by repeatedly applying the product and disjoint-union operators. We equip \mathbb{R} with the usual Borel σ -algebra. Then, we parallel each usage of the product and the disjoint-union operators on sets with that of the corresponding operators on σ -algebras. This gives the σ -algebras for all the sets defined above. Although absent from the above definition, the measurable spaces \mathbb{B} and *Str* equipped with discrete σ -algebras are also used in our semantics.

We interpret expressions E, B, S as measurable functions $\llbracket E \rrbracket : \text{Store} \rightarrow \mathbb{R}$, $\llbracket B \rrbracket : \text{Store} \rightarrow \mathbb{B}$, and $\llbracket S \rrbracket : \text{Store} \rightarrow \text{Str}$, under the assumption that the semantics of primitive real-valued f of

$$\begin{aligned}
\llbracket \text{skip} \rrbracket(\sigma)(A) &\triangleq \mathbb{1}_{[(\sigma, 1) \in A]} & \llbracket x := E \rrbracket(\sigma)(A) &\triangleq \mathbb{1}_{[(\sigma[x \mapsto \llbracket E \rrbracket \sigma_s, 1) \in A]} \\
\llbracket C_0; C_1 \rrbracket(\sigma)(A) &\triangleq \int \llbracket C_0 \rrbracket(\sigma)(d(\sigma', w')) \int \llbracket C_1 \rrbracket(\sigma')(d(\sigma'', w'')) \mathbb{1}_{[(\sigma'', w', w'') \in A]} \\
\llbracket \text{if } B \{C_0\} \text{ else } \{C_1\} \rrbracket(\sigma)(A) &\triangleq \mathbb{1}_{[\llbracket B \rrbracket \sigma_s = \text{true}]} \cdot \llbracket C_0 \rrbracket(\sigma)(A) + \mathbb{1}_{[\llbracket B \rrbracket \sigma_s \neq \text{true}]} \cdot \llbracket C_1 \rrbracket(\sigma)(A) \\
\llbracket \text{while } B \{C\} \rrbracket(\sigma)(A) &\triangleq (\text{fix } F)(\sigma)(A) \\
(\text{where } F(\kappa)(\sigma)(A) &\triangleq \mathbb{1}_{[\llbracket B \rrbracket \sigma_s \neq \text{true}]} \cdot \mathbb{1}_{[(\sigma, 1) \in A]} \\
&\quad + \mathbb{1}_{[\llbracket B \rrbracket \sigma_s = \text{true}]} \cdot \int \llbracket C \rrbracket(\sigma)(d(\sigma', w')) \int \kappa(\sigma')(d(\sigma'', w'')) \mathbb{1}_{[(\sigma'', w', w'') \in A]}) \\
\llbracket x := \text{sample}_{\text{norm}}(S, E_1, E_2) \rrbracket(\sigma)(A) &\triangleq \mathbb{1}_{[\llbracket S \rrbracket \sigma_s \notin \text{dom}(\sigma_r)]} \cdot \mathbb{1}_{[\llbracket E_2 \rrbracket \sigma_s \in (0, \infty)]} \\
&\quad \cdot \int d\nu \left(\mathcal{N}(\nu; \llbracket E_1 \rrbracket \sigma_s, \llbracket E_2 \rrbracket \sigma_s) \cdot \mathbb{1}_{[(\llbracket S \rrbracket \sigma_s \mapsto \nu), \sigma_r(\llbracket S \rrbracket \sigma_s \mapsto \nu), 1) \in A]} \right) \\
\llbracket \text{score}_{\text{norm}}(E_0, E_1, E_2) \rrbracket(\sigma)(A) &\triangleq \mathbb{1}_{[\llbracket E_2 \rrbracket \sigma_s \in (0, \infty)]} \cdot \mathbb{1}_{[(\sigma, \mathcal{N}(\llbracket E_0 \rrbracket \sigma_s, \llbracket E_1 \rrbracket \sigma_s, \llbracket E_2 \rrbracket \sigma_s)) \in A]}
\end{aligned}$$

Fig. 5. Measure semantics $\llbracket C \rrbracket \in \mathcal{K}$ of commands C . Here $\mathcal{N}(\nu; \mu, \sigma)$ is the density of the normal distribution with mean μ and standard deviation σ .

arity n and string-valued g of arity (m, l) are given by measurable functions $\llbracket f \rrbracket : \mathbb{R}^n \rightarrow \mathbb{R}$ and $\llbracket g \rrbracket : \text{Str}^m \times \mathbb{R}^l \rightarrow \text{Str}$. It is standard, and we describe it only for some sample cases of E and S :

$$\begin{aligned}
\llbracket x \rrbracket s &\triangleq s(x), & \llbracket f(E_0, \dots, E_{n-1}) \rrbracket s &\triangleq \llbracket f \rrbracket(\llbracket E_0 \rrbracket s, \dots, \llbracket E_{n-1} \rrbracket s), \\
\llbracket \alpha \rrbracket s &\triangleq \alpha, & \llbracket g(S_0, \dots, S_{m-1}, E_0, \dots, E_{l-1}) \rrbracket s &\triangleq \llbracket g \rrbracket(\llbracket S_0 \rrbracket s, \dots, \llbracket S_{m-1} \rrbracket s, \llbracket E_0 \rrbracket s, \dots, \llbracket E_{l-1} \rrbracket s).
\end{aligned}$$

LEMMA 4.1. *For all expressions E , B , and S , their semantics $\llbracket E \rrbracket$, $\llbracket B \rrbracket$ and $\llbracket S \rrbracket$ are measurable functions from Store to \mathbb{R} , \mathbb{B} and Str , respectively.*

We interpret commands C as measurable functions from State to $\text{Sp}(\text{State} \times [0, \infty))$, i.e., subprobability kernels from State to $\text{State} \times [0, \infty)$. Let \mathcal{K} be the set of subprobability kernels from State to $\text{State} \times [0, \infty)$, and $\Sigma_{\text{State} \times [0, \infty)}$ be the σ -algebra of the product space $\text{State} \times [0, \infty)$. We equip \mathcal{K} with the partial order \sqsubseteq : for all $\kappa, \kappa' \in \mathcal{K}$, $\kappa \sqsubseteq \kappa'$ if and only if $\kappa(\sigma)(A) \leq \kappa'(\sigma)(A)$ for all $\sigma \in \text{State}$ and $A \in \Sigma_{\text{State} \times [0, \infty)}$. The next lemma is a minor adaptation of a known result [Kozen 1981].

LEMMA 4.2. *$(\mathcal{K}, \sqsubseteq)$ is an ω -complete partial order with the least element $\perp \triangleq (\lambda \sigma. \lambda A. 0)$.*

The measure semantics of a command C is defined in Figure 5. The interpretation of the loop is the least fixed point of the function F on the ω -complete partial order \mathcal{K} . The function F is continuous, so the least fixed point is obtained by the ω -limit of the sequence $\{F^n(\perp)\}_n$. In the definition of F , the argument κ is a sub-probability kernel, and it represents the computation after the first iteration of the loop. The semantics of the sample statement uses an indicator function to exclude erroneous executions where the argument S denotes a name already used by some previous random variable, or the standard deviation $\llbracket E_2 \rrbracket \sigma_s$ is not positive. When this check passes, it distils A to a property on the value of x and computes the probability of the property using the normal distribution with mean $\llbracket E_1 \rrbracket \sigma_s$ and standard deviation $\llbracket E_2 \rrbracket \sigma_s$.

THEOREM 4.3. *For every command C , its interpretation $\llbracket C \rrbracket$ is well-defined and belongs to \mathcal{K} .*

4.3 Posterior Inference and Density Semantics

We write a probabilistic program to answer queries about the model and data that it describes. Among such queries, posterior inference is one of the most important and popular. Let $\sigma_I = (s_I, [])$ be the initial state that consists of some fixed store and the empty random database. In our setting, posterior inference amounts to finding information about the following probability measure $\text{Pr}(C, \cdot)$ for a command C . For a measurable $A \subseteq \text{RDB}$,

$$\text{Mea}(C, A) \triangleq \int \llbracket C \rrbracket(\sigma_I)(d(\sigma, w)) (w \cdot \mathbb{1}_{[\sigma \in \text{Store} \times A]}), \quad Z_C \triangleq \text{Mea}(C, \text{RDB}), \quad \text{Pr}(C, A) \triangleq \frac{\text{Mea}(C, A)}{Z_C}. \quad (2)$$

The probability measure $\text{Pr}(C, \cdot)$ is called the *posterior distribution of C* , and $\text{Mea}(C, \cdot)$ the *unnormalised posterior distribution of C* (in $\text{Mea}(C, \cdot)$ and $\text{Pr}(C, \cdot)$, we elide the dependency on s_I to avoid clutter). Finding information about the former is the goal of most inference engines of existing probabilistic programming languages. Of course, $\text{Pr}(C, \cdot)$ is not defined when the normalising constant Z_C is infinite or zero. The inference engines regard such a case as an error that a programmer should avoid, and consider only C without those errors.

Most algorithms for posterior inference use the density semantics of commands. They implicitly pick measures on some measurable spaces used in the semantics. These measures are called *reference measures*, and constructed out of Lebesgue and counting measures [Bhat et al. 2012, 2013; Hur et al. 2015]. Then, the algorithms interpret commands as density functions with respect to these measures. One outcome of this density semantics is that the unnormalised posterior distribution $\text{Mea}(C, \cdot)$ of a command C has a measurable function $f : RDB \rightarrow [0, \infty)$ such that $\text{Mea}(C, A) = \int \rho(dr) (\mathbb{1}_{[r \in A]} \cdot f(r))$, where ρ is a reference measure on RDB . Function f is called *density* of $\text{Mea}(C, \cdot)$ with respect to ρ .

In the rest of this subsection, we define the meanings of commands using density functions. To do this, we need to set up some preliminary definitions.

First, we look at a predicate and an operator for random databases, which are about the possibility and the very act of merging two databases. For $r, r' \in RDB$, define the predicate $r \# r'$ by:

$$r \# r' \iff \text{dom}(r) \cap \text{dom}(r') = \emptyset.$$

When $r \# r'$, let $r \uplus r'$ be the random database obtained by merging r and r' :

$$\text{dom}(r \uplus r') \triangleq \text{dom}(r) \cup \text{dom}(r'); \quad (r \uplus r')(\alpha) \triangleq \text{if } \alpha \in \text{dom}(r) \text{ then } r(\alpha) \text{ else } r'(\alpha).$$

LEMMA 4.4. *For every measurable $h : RDB \times RDB \times RDB \rightarrow \mathbb{R}$, the function $(r, r') \mapsto \mathbb{1}_{[r \# r']} \times h(r, r', r \uplus r')$ from $RDB \times RDB$ to \mathbb{R} is measurable.*

Second, we define a reference measure ρ on RDB :

$$\rho(R) \triangleq \sum_{K \subseteq_{fin} Str} \left(\bigotimes_{\alpha \in K} \lambda \right) (R \cap [K \rightarrow \mathbb{R}]),$$

where λ is the Lebesgue measure on \mathbb{R} . As explained in the preliminary section, the symbol \otimes here represents the operator for constructing a product measure. In particular, $\bigotimes_{\alpha \in K} \lambda$ refers to the product of the $|K|$ copies of the Lebesgue measure λ on \mathbb{R} . In the above definition, we view functions in $[K \rightarrow \mathbb{R}]$ as tuples with $|K|$ real components and measure sets of such functions using the product measure $\bigotimes_{\alpha \in K} \lambda$. When K is the empty set, $\bigotimes_{\alpha \in K} \lambda$ is the nullary-product measure on $\{\{\}\}$, which assigns 1 to $\{\{\}\}$ and 0 to the empty set.

The measure ρ computes the size of each measurable subset $R \subseteq RDB$ in three steps. It splits a given R into groups based on the domains of elements in R . Then, it computes the size of each group separately, using the product of the Lebesgue measure. Finally, it adds the computed sizes. The measure ρ is not finite, but it satisfies the σ -finiteness condition,⁸ the next best property.

Third, we define a partially-ordered set \mathcal{D} with certain measurable functions. We say that a function $g : Store \times RDB \rightarrow \{\perp\} \cup (Store \times RDB \times [0, \infty) \times [0, \infty))$ *uses random databases locally* or *is local* if for all $s, s' \in Store$, $r, r' \in RDB$, and $w', p' \in [0, \infty)$,

$$\begin{aligned} g(s, r) = (s', r', w', p') \implies & (\exists r''. r' \# r'' \wedge r = r' \uplus r'' \wedge g(s, r'') = (s', [], w', p')) \\ & \wedge (\forall r'''. r \# r''' \implies g(s, r \uplus r''') = (s', r' \uplus r''', w', p')). \end{aligned}$$

⁸The condition lets us use Fubini theorem when showing the well-formedness of the density semantics in this subsection and relating this semantics with the measure semantics in the previous subsection.

$$\begin{aligned}
\llbracket \text{skip} \rrbracket_d(s, r) &\triangleq (s, r, 1, 1) & \llbracket x := E \rrbracket_d(s, r) &\triangleq (s[x \mapsto \llbracket E \rrbracket s], r, 1, 1) \\
\llbracket C_0; C_1 \rrbracket_d(s, r) &\triangleq (\llbracket C_1 \rrbracket_d^\ddagger \circ \llbracket C_0 \rrbracket_d)(s, r) \\
\llbracket \text{if } B \{ C_0 \} \text{ else } \{ C_1 \} \rrbracket_d(s, r) &\triangleq \text{if } (\llbracket B \rrbracket s = \text{true}) \text{ then } \llbracket C_0 \rrbracket_d(s, r) \text{ else } \llbracket C_1 \rrbracket_d(s, r) \\
\llbracket \text{while } B \{ C \} \rrbracket_d(s, r) &\triangleq (\text{fix } G)(s, r) \\
&\quad (\text{where } G(g)(s, r) = \text{if } (\llbracket B \rrbracket s \neq \text{true}) \text{ then } (s, r, 1, 1) \text{ else } (g^\ddagger \circ \llbracket C \rrbracket_d)(s, r)) \\
\llbracket x := \text{sample}_{\text{norm}}(S, E_1, E_2) \rrbracket_d(s, r) &\triangleq \text{if } (\llbracket S \rrbracket s \notin \text{dom}(r) \vee \llbracket E_2 \rrbracket s \notin (0, \infty)) \text{ then } \perp \\
&\quad \text{else } (s[x \mapsto r(\llbracket S \rrbracket s)], r \setminus \llbracket S \rrbracket s, 1, \mathcal{N}(r(\llbracket S \rrbracket s); \llbracket E_1 \rrbracket s, \llbracket E_2 \rrbracket s)) \\
\llbracket \text{score}_{\text{norm}}(E_0, E_1, E_2) \rrbracket_d(s, r) &\triangleq \text{if } (\llbracket E_2 \rrbracket s \notin (0, \infty)) \text{ then } \perp \text{ else } (s, r, \mathcal{N}(\llbracket E_0 \rrbracket s; \llbracket E_1 \rrbracket s, \llbracket E_2 \rrbracket s), 1)
\end{aligned}$$

Fig. 6. Density semantics $\llbracket C \rrbracket_d \in \mathcal{D}$ of commands C

The condition describes the way that g uses a given random database r , which plays the role of a bank of random seeds (that g may partially consume as it needs random values). Some part r'' of r may be consumed by g , but the unconsumed part r' of r does not change and is returned in the output. Also, the behaviour of g does not depend on the unconsumed r' . We define the set \mathcal{D} by

$$\mathcal{D} \triangleq \{g : \text{Store} \times \text{RDB} \rightarrow \{\perp\} \cup (\text{Store} \times \text{RDB} \times [0, \infty) \times [0, \infty)) \mid g \text{ is measurable and local}\}. \quad (3)$$

Here we view $\{\perp\}$ and $[0, \infty)$ as measurable spaces equipped with discrete and Borel σ -algebras. Also, we regard $\text{Store} \times \text{RDB}$ and $\{\perp\} \cup (\text{Store} \times \text{RDB} \times [0, \infty) \times [0, \infty))$ as measurable spaces constructed by the product and disjoint-union operators on measurable spaces, as explained in §3.

The locality in the definition of \mathcal{D} formalises expected behaviours of commands. In fact, as we will show shortly, it is satisfied by all commands in our density semantics. This property plays an important role when we establish the connection between the density semantics in this subsection and the standard measure semantics in §4.2.

The functions in \mathcal{D} are ordered pointwise: for all $g, g' \in \mathcal{D}$,

$$g \sqsubseteq g' \iff \forall (s, r) \in \text{Store} \times \text{RDB}. (g(s, r) = \perp \vee g(s, r) = g'(s, r)).$$

LEMMA 4.5. $(\mathcal{D}, \sqsubseteq)$ is an ω -complete partial order and has the least element $a \mapsto \perp$. Thus, every continuous function G on \mathcal{D} has a least fixed point (and this least fixed point is unique).

For each $g \in \mathcal{D}$, let g^\ddagger be the following lifting to a function on $\{\perp\} \cup (\text{Store} \times \text{RDB} \times [0, \infty) \times [0, \infty))$:

$$g^\ddagger(\perp) \triangleq \perp, \quad g^\ddagger(s, r, w, p) \triangleq \begin{cases} \perp & \text{if } g(s, r) = \perp, \\ (s', r', w \times w', p \times p') & \text{if } g(s, r) = (s', r', w', p'). \end{cases}$$

This lifting lets us compose two functions in \mathcal{D} .

Using these preliminary definitions, we define a density semantics in Figure 6, where a command C means a function $\llbracket C \rrbracket_d \in \mathcal{D}$. The notation $r \setminus v$ in the figure means the removal of the entry v from the finite map r if $v \in \text{dom}(r)$; otherwise, $r \setminus v$ is just r . The set membership $\llbracket C \rrbracket_d \in \mathcal{D}$ says that $\llbracket C \rrbracket_d$ is a local measurable function from $\text{Store} \times \text{RDB}$ to $\{\perp\} \cup \text{Store} \times \text{RDB} \times [0, \infty) \times [0, \infty)$. Thus, the function $\llbracket C \rrbracket_d$ takes a store s and a random database r as inputs, where the former fixes the values of variables at the start of C and the latter specifies random seeds some of which C may consume to sample random variables. Given such inputs, the function outputs an updated store s' , the part r' of r not consumed by C , the total score w' expressing how well the execution of C matches observations, and the probability density p' of C at the consumed part of r . If r does not contain enough random seeds, or the execution of C encounters some runtime error, or it falls into an infinite loop, then the function returns \perp .

LEMMA 4.6. For every command C , its semantics $\llbracket C \rrbracket_d$ is well-defined and belongs to \mathcal{D} .

The density semantics $\llbracket C \rrbracket_d$ is closely related to the measure semantics $\llbracket C \rrbracket$ defined in §4.2. Both interpretations of C describe the computation of C but from slightly different perspectives. To state this relationship formally, we need a few notations. For $g \in \mathcal{D}$ and $s \in \text{Store}$, define

$$\begin{aligned} \text{dens}(g, s) : RDB \rightarrow [0, \infty), \quad \text{dens}(g, s)(r) &\triangleq \begin{cases} w' \times p' & \text{if } \exists s', w', p'. (g(s, r) = (s', [], w', p')), \\ 0 & \text{otherwise,} \end{cases} \\ \text{get}(g, s) : RDB \rightarrow \text{Store} \cup \{\perp\}, \quad \text{get}(g, s)(r) &\triangleq \begin{cases} s' & \text{if } \exists s', w', p'. (g(s, r) = (s', [], w', p')), \\ \perp & \text{otherwise.} \end{cases} \end{aligned}$$

Both $\text{dens}(g, s)$ and $\text{get}(g, s)$ are concerned with random databases r that precisely describe the randomness needed by the execution of g from s . This is formalised by the use of $[]$ in the definitions. The function $\text{dens}(g, s)$ assigns a score to such an r , and in so doing, it defines a probability density on RDB with respect to the reference measure ρ . The function $\text{get}(g, s)$ computes a store that the computation of g would result in when started with such an r . We often write $\text{dens}(C, s)$ and $\text{get}(C, s)$ to mean $\text{dens}(\llbracket C \rrbracket_d, s)$ and $\text{get}(\llbracket C \rrbracket_d, s)$, respectively.

LEMMA 4.7. *For all $g \in \mathcal{D}$, the following functions from $\text{Store} \times RDB$ to \mathbb{R} and $\{\perp\} \cup \text{Store}$ are measurable: $(s, r) \mapsto \text{dens}(g, s)(r)$ and $(s, r) \mapsto \text{get}(g, s)(r)$.*

The next lemma is the main reason that we considered the locality property. It plays a crucial role in proving Theorem 4.9, the key result of this subsection.

LEMMA 4.8. *For all non-negative bounded measurable functions $h : (\{\perp\} \cup \text{Store}) \times RDB \rightarrow \mathbb{R}$, stores s , and functions $g_1, g_2 \in \mathcal{D}$, we have that*

$$\begin{aligned} &\int \rho(dr) \left(\text{dens}(g_2^\ddagger \circ g_1, s)(r) \cdot h\left(\text{get}(g_2^\ddagger \circ g_1, s)(r), r\right) \right) \\ &= \int \rho(dr_1) \left(\text{dens}(g_1, s)(r_1) \cdot \mathbb{1}_{[\text{get}(g_1, s)(r_1) \neq \perp]} \right. \\ &\quad \cdot \left. \int \rho(dr_2) \left(\text{dens}(g_2, \text{get}(g_1, s)(r_1))(r_2) \cdot \mathbb{1}_{[r_1 \# r_2]} \cdot h\left(\text{get}(g_2, \text{get}(g_1, s)(r_1))(r_2), r_1 \uplus r_2\right) \right) \right). \end{aligned}$$

Assume that $(\{\perp\} \cup \text{Store}) \times RDB$ is ordered as follows: for all $(a, r), (a', r') \in (\{\perp\} \cup \text{Store}) \times RDB$,

$$(a, r) \sqsubseteq (a', r') \iff (a = \perp \vee a = a') \wedge r = r'.$$

THEOREM 4.9. *For all non-negative bounded measurable monotone functions $h : (\{\perp\} \cup \text{Store}) \times RDB \rightarrow \mathbb{R}$ and states σ ,*

$$\int \llbracket C \rrbracket(\sigma)(d(\sigma', w')) (w' \cdot h(\sigma'_s, \sigma'_r)) = \int \rho(dr') \left(\text{dens}(C, \sigma_s)(r') \cdot \mathbb{1}_{[r' \# \sigma_r]} \cdot h(\text{get}(C, \sigma_s)(r'), r' \uplus \sigma_r) \right).$$

COROLLARY 4.10. $\text{Mea}(C, A) = \int \rho(dr) (\mathbb{1}_{[r \in A]} \cdot \text{dens}(C, s_I)(r))$ for all C and all measurable A .

This corollary says that $\text{dens}(C, s_I)$ is the density of the measure $\text{Mea}(C, \cdot)$ with respect to ρ , and supports our informal claim that $\llbracket C \rrbracket_d$ computes the density of the measure $\llbracket C \rrbracket$.

5 STOCHASTIC VARIATIONAL INFERENCE

In this section, we explain stochastic variational inference (SVI) algorithms using the semantics that we have developed so far. In particular, we describe the requirements made implicitly by one fundamental SVI algorithm, which is regarded most permissive by the ML researchers because the algorithm does not require the differentiability of the density of a given probabilistic model.

We call a command C *model* if it has a finite nonzero normalising constant:

$$Z_C = \text{Mea}(C, RDB) = \left(\int \llbracket C \rrbracket(\sigma_I)(d(\sigma, w)) w \right) \in (0, \infty),$$

```

model C  $\equiv$  (s := samplenorm("slope", 0.0, 5.0); i := samplenorm("intercept", 0.0, 5.0);
           x1 := 1.0; y1 := 2.3; x2 := 2.0; y2 := 4.2; x3 := 3.0; y3 := 6.9;
           scorenorm(y1, s · x1 + i, 1.0); scorenorm(y2, s · x2 + i, 1.0); scorenorm(y3, s · x3 + i, 1.0))
guide Dθ  $\equiv$  (s := samplenorm("slope", θ1, exp(θ2)); i := samplenorm("intercept", θ3, exp(θ4)))

```

Fig. 7. Example model-guide pair for simple Bayesian linear regression.

where σ_I is the initial state. Given a model C , the SVI algorithms attempt to infer a good approximation of C 's posterior distribution $\Pr(C, \cdot)$ defined in (2). They tackle this posterior-inference problem in two steps.

First, the SVI algorithms fix a collection of approximate distributions. They usually do so by asking the developer of C to provide a command D_θ parameterised by $\theta \in \mathbb{R}^p$, which can serve as a template for approximation distributions. The command D_θ typically has a control-flow structure similar to that of C , but it is simpler than C : it does not use any score statements, and may replace complex computation steps of C by simpler ones. In fact, D_θ should satisfy two formal requirements, which enforce this simplicity. The first is

$$\text{Mea}(D_\theta, RDB) = 1 \quad \text{for all } \theta \in \mathbb{R}^p,$$

which means that the normalising constant of D_θ is 1. The second is that D_θ should keep the score (i.e., the w component) to be 1, i.e.,

$$\llbracket D_\theta \rrbracket(\sigma_I)(\text{State} \times ([0, \infty) \setminus \{1\})) = 0.$$

Meeting these requirements is often not too difficult. A common technique is to ensure that D_θ does not use the score statement and always terminates. Figure 7 gives an example of (C, D_θ) for simple Bayesian linear regression with three data points. Note that in this case, D_θ is obtained from C by deleting the score statements and replacing the arguments 0.0 and 5.0 of normal distributions by parameter $\theta = (\theta_1, \theta_2, \theta_3, \theta_4)$. Following the terminology of Pyro, we call a parameterised command D_θ *guide* if it satisfies the two requirements just mentioned.

Second, the SVI algorithms search for a good parameter θ that makes the distribution described by D_θ close to the posterior of C . Concretely, they formulate an optimisation problem where the optimisation objective expresses that some form of distance from D_θ 's distribution to C 's posterior should be minimised. Then, they solve the problem by a version of gradient descent.

The KL divergence is a standard choice for distance. Let μ, μ' be measures on RDB that have densities g and g' with respect to the measure ρ . The KL divergence from g to g' is defined by

$$\text{KL}(g||g') \triangleq \int \rho(\text{dr}) \left(g(r) \cdot \log \frac{g(r)}{g'(r)} \right). \quad (4)$$

In words, it is the ratio of densities g and g' averaged according to g . If $g = g'$, the ratio is always 1, so that the KL becomes 0. The KL divergence is defined only if the following conditions are met:

- *Absolute continuity*: $g'(r) = 0 \implies g(r) = 0$ for all $r \in RDB$,⁹ which ensures that the integrand in (4) is well-defined even when the denominator $g'(r)$ in (4) takes the value 0;
- *Integrability*: the integral in (4) has a finite value.

Using our semantics, we can express the KL objective as follows:

$$\text{argmin}_{\theta \in \mathbb{R}^p} \text{KL} \left(\text{dens}(D_\theta, s_I) \parallel \frac{\text{dens}(C, s_I)}{Z_C} \right). \quad (5)$$

⁹This condition can be relaxed in a more general formulation of the KL divergence stated in terms of the so called Radon-Nikodym derivative. We do not use the relaxed condition to reduce the amount of materials on measure theory in the paper.

Recall that $\text{dens}(D_\theta, s_I)$ and $\text{dens}(C, s_I)/Z_C$ are densities of the probability measures of the command D_θ and the posterior of C (Corollary 4.10), and they are defined by means of our density semantics in §4.3. Most SVI engines solve this optimisation problem by a version of gradient descent.

In the paper, we consider one of the most fundamental and versatile SVI algorithms. The algorithm is called score estimator or REINFORCE, and it works by estimating the gradient of the objective in (5) using samples and performing the gradient descent with this estimated gradient. More concretely, the algorithm starts by initialising θ with some value (usually chosen randomly) and updating it repeatedly by the following procedure:

(i) Sample r_1, \dots, r_N independently from $\text{dens}(D_\theta, s_I)$

(ii) $\theta \leftarrow \theta - \eta \times \left(\frac{1}{N} \sum_{i=1}^N \left(\nabla_\theta \log \text{dens}(D_\theta, s_I)(r_i) \right) \cdot \log \frac{\text{dens}(D_\theta, s_I)(r_i)}{\text{dens}(C, s_I)(r_i)} \right)$

Here N and η are hyperparameters to this algorithm, the former determining the number of samples used to estimate the gradient and the latter, called learning rate, deciding how much the algorithm should follow the direction of the estimated gradient. Although we do not explain here, sampling r_1, \dots, r_N and computing all of $\text{dens}(D_\theta, s_I)(r_i)$, $\text{dens}(C, s_I)(r_i)$ and $\nabla_\theta(\log \text{dens}(D_\theta, s_I)(r_i))$ can be done by executing D_θ and C multiple times under slightly unusual operational semantics [Yang 2019]. The SVI engines of Pyro and Anglican implement such operational semantics.

The average over the N terms in the θ -update step is the core of the algorithm. It approximates the gradient of the optimisation objective in (5):

$$\nabla_\theta \text{KL} \left(\text{dens}(D_\theta, s_I) \parallel \frac{\text{dens}(C, s_I)}{Z_C} \right) \approx \frac{1}{N} \sum_{i=1}^N \left(\nabla_\theta \log \text{dens}(D_\theta, s_I)(r_i) \right) \cdot \log \frac{\text{dens}(D_\theta, s_I)(r_i)}{\text{dens}(C, s_I)(r_i)}.$$

The average satisfies an important property called *unbiasedness*, summarised by Theorem 5.1.

THEOREM 5.1. *Let C be a model, D_θ be a guide, and $N \neq 0 \in \mathbb{N}$. Define $\text{KL}_{(-)} : \mathbb{R}^P \rightarrow \mathbb{R}_{\geq 0}$ as $\text{KL}_\theta \triangleq \text{KL}(\text{dens}(D_\theta, s_I) \parallel \text{dens}(C, s_I)/Z_C)$. Then, $\text{KL}_{(-)}$ is well-defined and continuously differentiable with*

$$\nabla_\theta \text{KL}_\theta = \mathbb{E}_{\prod_i \text{dens}(D_\theta, s_I)(r_i)} \left[\frac{1}{N} \sum_{i=1}^N \left(\nabla_\theta \log \text{dens}(D_\theta, s_I)(r_i) \right) \log \frac{\text{dens}(D_\theta, s_I)(r_i)}{\text{dens}(C, s_I)(r_i)} \right] \quad (6)$$

if

(R1) $\text{dens}(C, s_I)(r) = 0 \implies \text{dens}(D_\theta, s_I)(r) = 0$, for all $r \in \text{RDB}$ and $\theta \in \mathbb{R}^P$;

(R2) for all $(r, \theta, j) \in \text{RDB} \times \mathbb{R}^P \times [p]$, the function $v \mapsto \text{dens}(D_{\theta[j:v]}, s_I)(r)$ on \mathbb{R} is differentiable;

(R3) for all $\theta \in \mathbb{R}^P$,

$$\int \rho(\text{dr}) \left(\text{dens}(D_\theta, s_I)(r) \cdot \log \frac{\text{dens}(D_\theta, s_I)(r)}{\text{dens}(C, s_I)(r)} \right) < \infty;$$

(R4) for all $(\theta, j) \in \mathbb{R}^P \times [p]$, the function

$$v \mapsto \int \rho(\text{dr}) \left(\text{dens}(D_{\theta[j:v]}, s_I)(r) \cdot \log \frac{\text{dens}(D_{\theta[j:v]}, s_I)(r)}{\text{dens}(C, s_I)(r)} \right)$$

on \mathbb{R} is continuously differentiable;

(R5) for all $\theta \in \mathbb{R}^P$,

$$\nabla_\theta \int \rho(\text{dr}) \left(\text{dens}(D_\theta, s_I)(r) \cdot \log \frac{\text{dens}(D_\theta, s_I)(r)}{\text{dens}(C, s_I)(r)} \right) = \int \rho(\text{dr}) \nabla_\theta \left(\text{dens}(D_\theta, s_I)(r) \cdot \log \frac{\text{dens}(D_\theta, s_I)(r)}{\text{dens}(C, s_I)(r)} \right);$$

(R6) for all $\theta \in \mathbb{R}^P$,

$$\int \rho(\text{dr}) \nabla_\theta \text{dens}(D_\theta, s_I)(r) = \nabla_\theta \int \rho(\text{dr}) \text{dens}(D_\theta, s_I)(r).$$

Here $\theta[j : v]$ denotes a vector in \mathbb{R}^P that is the same as θ except that its j -th component is v .

The conclusion of this theorem (Equation (6)) and its proof are well-known [Ranganath et al. 2014], but the requirements in the theorem (and the continuous differentiability of KL_θ in the conclusion) are rarely stated explicitly in the literature.

The correctness of the algorithm crucially relies on the unbiasedness property in Theorem 5.1. The property ensures that the algorithm converges to a local minimum with probability 1. Thus, it is important that the requirements in the theorem are met. In fact, some of the requirements there are needed even to state the optimisation objective in (5), because without them, the objective does not exist. In the next section, we describe conditions that imply those requirements and can serve as target properties of program analysis for probabilistic programs. The latter point is worked out in detail in §7 and §8 where we discuss program analysis for probabilistic programs and SVI.

6 CONDITIONS FOR STOCHASTIC VARIATIONAL INFERENCE

Ideally we want to have program analysers that discharge the six requirements R1-R6 in Theorem 5.1. However, except R1 and R2, the requirements are not ready for serving as the targets of static analysis algorithms. Automatically discharging them based on the first principles (such as the definition of integrability with respect to a measure) may be possible, but seems less immediate than doing so using powerful theorems from continuous mathematics.

In this section, we explain conditions that imply the requirements R3-R6 and are more friendly to program analysis than the requirements themselves. The conditions are given in two boxes (9) and (10). Throughout the section, we fix a model C and a guide D_θ .

6.1 Assumption

Throughout the section, we assume that the densities of D_θ and C have the following form:

$$\begin{aligned} dens(D_\theta, s_I)(r) &= \sum_{i=1}^M \mathbb{1}_{[r \in A_i]} \prod_{\alpha \in K_i} \mathcal{N}(r(\alpha); \mu_{(i,\alpha)}(\theta), \sigma_{(i,\alpha)}(\theta)), \\ dens(C, s_I)(r) &= \sum_{i=1}^M \mathbb{1}_{[r \in A_i]} \left(\prod_{\alpha \in K_i} \mathcal{N}(r(\alpha); \mu'_{(i,\alpha)}(r), \sigma'_{(i,\alpha)}(r)) \right) \left(\prod_{j \in [N_i]} \mathcal{N}(c_{(i,j)}; \mu''_{(i,j)}(r), \sigma''_{(i,j)}(r)) \right), \end{aligned} \quad (7)$$

where

- $M, N_i \in \mathbb{N} \setminus \{0\}$;
- A_1, \dots, A_M are disjoint measurable subsets of RDB ;
- K_i 's are finite sets of strings such that $\text{dom}(r) = K_i$ for all $r \in A_i$;
- $\mu_{(i,\alpha)}$ and $\sigma_{(i,\alpha)}$ are measurable functions from \mathbb{R}^p to \mathbb{R} and $(0, \infty)$, respectively;
- $\mu'_{(i,\alpha)}$ and $\mu''_{(i,j)}$ are measurable functions from $[K_i \rightarrow \mathbb{R}]$ to \mathbb{R} ;
- $\sigma'_{(i,\alpha)}$ and $\sigma''_{(i,j)}$ are measurable functions from $[K_i \rightarrow \mathbb{R}]$ to $(0, \infty)$;
- $c_{(i,j)}$ is a real number.

In programming terms, our assumption first implies that both C and D_θ use at most a fixed number of random variables. That is, the number of random variables they generate must be finite not only within a single execution but also across all possible executions since the names of all random variables are found in a finite set $\bigcup_{i=1}^M K_i$. This property is met if the number of steps in every execution of C and D_θ from σ_I is bounded by some T and the executions over the same program path use the same set of random variables. Note that the bound may depend on s_I . Such a bound exists for most probabilistic programs.¹⁰ Also, the assumption says that the parameters of normal distributions in sample statements in D_θ may depend only on θ , but not on other sampled random

¹⁰Notable exceptions are models using probabilistic grammars or those from Bayesian nonparametrics.

variables. This is closely related to a common approach for designing approximate distributions in variational inference, called mean-field approximation, where the approximate distribution consists of independent normal random variables.

We use the term “assumption” here, instead of “condition” in the following subsections because the assumed properties are rather conventional and they are not directly related to the requirements in Theorem 5.1, at least not as much as the conditions that we will describe next.

6.2 Condition for Requirement R3

Note that the integral in the requirement R3 can be written as the sum of two expectations:

$$\int \rho(dr) \left(\text{dens}(D_\theta, s_I)(r) \cdot \log \frac{\text{dens}(D_\theta, s_I)(r)}{\text{dens}(C, s_I)(r)} \right) \\ = \mathbb{E}_{\text{dens}(D_\theta, s_I)(r)} [\log \text{dens}(D_\theta, s_I)(r)] - \mathbb{E}_{\text{dens}(D_\theta, s_I)(r)} [\log \text{dens}(C, s_I)(r)]. \quad (8)$$

The minus of the first term (i.e., $-\mathbb{E}_{\text{dens}(D_\theta, s_I)(r)} [\log \text{dens}(D_\theta, s_I)(r)]$) is called the *differential entropy* of the density $\text{dens}(D_\theta, s_I)$. Intuitively, it is large when the density on \mathbb{R}^n is close to the Lebesgue measure, which is regarded to represent the absence of information. The differential entropy is sometimes undefined [Ghouchian et al. 2017]. Fortunately, a large class of probability densities (containing many commonly used probability distributions) have well-defined entropies [Ghouchian et al. 2017; Nair et al. 2006]. Our $\text{dens}(D_\theta, s_I)$ is one of such fortunate cases.

THEOREM 6.1. $\mathbb{E}_{\text{dens}(D_\theta, s_I)(r)} [|\log \text{dens}(D_\theta, s_I)(r)|] < \infty$ under our assumption in §6.1.

We remark that a violation of the assumption (7) for D_θ can result in an undefined entropy, as illustrated by the following examples.

Example 6.2. Consider guides $D_{(i, \theta)}$ defined as follows ($i = 1, 2$):

$$D_{(i, \theta)} \equiv (x_1 := \text{sample}_{\text{norm}}("a_1", \theta_1, 1); x_2 := \text{sample}_{\text{norm}}("a_2", \theta_2, E_i[x_1]))$$

where for some $n \geq 1$ and $c \neq 0 \in \mathbb{R}$,¹¹

$$E_1[x_1] \equiv \text{if } (x_1=0) \text{ then } 1 \text{ else } \exp(-1/|x_1|^n), \quad E_2[x_1] \equiv \exp(\exp(c \cdot x_1^3)).$$

None of $\text{dens}(D_{(i, \theta)}, s_I)$'s satisfies the assumption (7) because the standard deviation of the normal distribution for x_2 depends on the value of x_1 . The entropies of $\text{dens}(D_{(i, \theta)}, s_I)$'s are all undefined: $\mathbb{E}_{\text{dens}(D_{(i, \theta)}, s_I)(r)} [|\log \text{dens}(D_{(i, \theta)}, s_I)(r)|] = \infty$ for all $i = 1, 2$. \square

Since the first term of (8) is always finite by Theorem 6.1, it is enough to ensure the finiteness of the second term of (8). For $i \in [M]$, define the set of (absolute) affine functions on $[K_i \rightarrow \mathbb{R}]$ as:

$$\mathcal{A}_i \triangleq \left\{ f \in [[K_i \rightarrow \mathbb{R}] \rightarrow \mathbb{R}] \mid f(r) = c + \sum_{\alpha \in K_i} (c_\alpha \cdot |r(\alpha)|) \text{ for some } c, c_\alpha \in \mathbb{R} \right\}.$$

Our condition for ensuring the finiteness of the second term is as follows:

For all $i \in [M]$, there are $f', f'', l', u', l'', u'' \in \mathcal{A}_i$ such that

$$\begin{aligned} |\mu'_{(i, \alpha)}(r)| &\leq \exp(f'(r)), & \exp(l'(r)) &\leq \sigma'_{(i, \alpha)}(r) \leq \exp(u'(r)), \\ |\mu''_{(i, j)}(r)| &\leq \exp(f''(r)), & \exp(l''(r)) &\leq \sigma''_{(i, j)}(r) \leq \exp(u''(r)), \\ \text{all four hold for every } (\alpha, j, r) &\in K_i \times [N_i] \times A_i. \end{aligned}$$

(9)

THEOREM 6.3. *The condition (9) implies $\mathbb{E}_{\text{dens}(D_\theta, s_I)(r)} [|\log \text{dens}(C, s_I)(r)|] < \infty$ under our assumption in §6.1. Thus, in that case, it entails the requirement R3 (i.e., the objective in (5) is well-defined).*

¹¹Formally, we should implement $E_1[x_1]$ as an application of a primitive function f_1 to x_1 that has the semantics described by the if-then-else statement here.

Our condition in (9) is sufficient but not necessary for the objective in (5) to be defined. However, its violation is a good warning, as illustrated by our next examples.

Example 6.4. Consider models C_1, \dots, C_4 and a guide D_θ defined as follows:

$$\begin{aligned} C_i &\equiv (x_1 := \text{sample}_{\text{norm}}("a_1", 0, 1); x_2 := \text{sample}_{\text{norm}}("a_2", E_i[x_1], 1)) \quad \text{for } i = 1, 2 \\ C_i &\equiv (x_1 := \text{sample}_{\text{norm}}("a_1", 0, 1); x_2 := \text{sample}_{\text{norm}}("a_2", 0, E_i[x_1])) \quad \text{for } i = 3, 4 \\ D_\theta &\equiv (x_1 := \text{sample}_{\text{norm}}("a_1", \theta_1, 1); x_2 := \text{sample}_{\text{norm}}("a_2", \theta_2, 1)) \end{aligned}$$

where for some $n \geq 1$ and $c \neq 0 \in \mathbb{R}$,

$$E_1[x_1] \equiv \text{if } (x_1=0) \text{ then } 0 \text{ else } \frac{1}{x_1^n}, \quad E_2[x_1] \equiv E_4[x_1] \equiv \exp(c \cdot x_1^3), \quad E_3[x_1] \equiv \text{if } (x_1=0) \text{ then } 1 \text{ else } |x_1|^n.$$

Let $A = [{"a_1"}, {"a_2"}] \rightarrow \mathbb{R}$. For $r \in A$, define

$$\begin{aligned} \mu'_1(r) &\triangleq \text{if } r("a_1") = 0 \text{ then } 0 \text{ else } 1/r("a_1")^n, & \mu'_2(r) &\triangleq \exp(c \cdot r("a_1")^3), \\ \sigma'_3(r) &\triangleq \text{if } r("a_1") = 0 \text{ then } 1 \text{ else } |r("a_1")|^n, & \sigma'_4(r) &\triangleq \exp(c \cdot r("a_1")^3). \end{aligned}$$

Then, we have that

$$\begin{aligned} \text{dens}(C_i, s_I)(r) &= \mathbb{1}_{[r \in A]} \cdot \mathcal{N}(r("a_1"); 0, 1) \cdot \mathcal{N}(r("a_2"); \mu'_i(r), 1) \quad \text{for } i = 1, 2 \\ \text{dens}(C_i, s_I)(r) &= \mathbb{1}_{[r \in A]} \cdot \mathcal{N}(r("a_1"); 0, 1) \cdot \mathcal{N}(r("a_2"); 0, \sigma'_i(r)) \quad \text{for } i = 3, 4 \\ \text{dens}(D_\theta, s_I)(r) &= \mathbb{1}_{[r \in A]} \cdot \mathcal{N}(r("a_1"); \theta_1, 1) \cdot \mathcal{N}(r("a_2"); \theta_2, 1). \end{aligned}$$

None of $\mu'_1, \mu'_2, \sigma'_3$, and σ'_4 satisfies the condition in (9). The function μ'_1 is not bounded in $\{r \in A \mid -1 \leq r("a_1") \leq 1 \wedge -1 \leq r("a_2") \leq 1\}$, but every μ' satisfying the condition in (9) should be bounded. Also, the cubic exponential growth of μ'_2 cannot be bounded by any linear exponential function on r . The violation of the condition by σ'_3 and σ'_4 can be shown similarly.

In fact, the objective in (5) is not defined for all of the four cases. This is because for all $i = 1, \dots, 4$, $\mathbb{E}_{\text{dens}(D_\theta, s_I)(r)} [\log \text{dens}(C_i, s_I)(r)] = \infty$. \square

We now show that the condition in (9) is satisfied by a large class of functions in machine learning applications, including functions using neural networks. We call a function $nn : \mathbb{R}^n \rightarrow \mathbb{R}$ an *affine-bounded neural network* if there exist functions $f_j : \mathbb{R}^{n_j} \rightarrow \mathbb{R}^{n_{j+1}}$ and affine functions $l_j : \mathbb{R}^{n_j} \rightarrow \mathbb{R}$ for all $1 \leq j \leq d$ such that (i) $n_1 = n$ and $n_{d+1} = 1$; (ii) $nn = f_d \circ \dots \circ f_1$; and (iii) $\|f_j(v)\|_1 \leq l_j(|v_1|, \dots, |v_{n_j}|)$ for all $1 \leq j \leq d$ and $v \in \mathbb{R}^{n_j}$, where $\|\cdot\|_1$ denotes the ℓ_1 -norm. Note that each component of f_j can be, for instance, an affine function, one of commonly used activation functions (e.g., relu, tanh, sigmoid, softplus), or one of min/max functions (min and max). Therefore, most of neural networks used in machine learning applications are indeed affine-bounded. Lemma 6.5 indicates that a wide range of functions satisfy the condition (9).

LEMMA 6.5. *Pick $i \in [M]$ and $\alpha \in K_i$. Let $(\alpha_1, \dots, \alpha_j)$ be an enumeration of the elements in K_i , and $\bar{r} \triangleq (r(\alpha_1), \dots, r(\alpha_j)) \in \mathbb{R}^J$ be an enumeration of the values of $r \in A_i$. Consider any affine-bounded neural network $nn : \mathbb{R}^J \rightarrow \mathbb{R}$, polynomial $poly : \mathbb{R}^J \rightarrow \mathbb{R}$, and $c \in (0, \infty)$. Then, the below list of functions $\mu'_{(i,\alpha)}$ and $\sigma'_{(i,\alpha)}$ on A_i satisfy the condition in (9):*

$$\begin{aligned} \mu'_{(i,\alpha)}(r) &= nn(\bar{r}), & \mu'_{(i,\alpha)}(r) &= poly(\bar{r}), & \mu'_{(i,\alpha)}(r) &= \exp(nn(\bar{r})), \\ \sigma'_{(i,\alpha)}(r) &= |nn(\bar{r})| + c, & \sigma'_{(i,\alpha)}(r) &= |poly(\bar{r})| + c, & \sigma'_{(i,\alpha)}(r) &= \exp(nn(\bar{r})), \\ \sigma'_{(i,\alpha)}(r) &= (|nn(\bar{r})| + c)^{-1}, & \sigma'_{(i,\alpha)}(r) &= (|poly(\bar{r})| + c)^{-1}, & \sigma'_{(i,\alpha)}(r) &= \text{softplus}(nn(\bar{r})), \end{aligned}$$

where $\text{softplus}(v) \triangleq \log(1 + \exp(v))$. Moreover, the same holds for $\mu''_{(i,j)}$ and $\sigma''_{(i,j)}$ as well.

6.3 Condition for Requirements R4-R6

Assume that the model C and the guide D_θ satisfy our assumption and condition in the previous two subsections. Our condition for the requirements R4-R6 is given below:

$$\boxed{\begin{array}{l} \text{For all } i \in [M], \alpha \in K_i, \text{ and } (\theta, j) \in \mathbb{R}^p \times [p], \\ \text{the function } v \in \mathbb{R} \mapsto \mu_{(i,\alpha)}(\theta[j : v]) \text{ is continuously differentiable;} \\ \text{the function } v \in \mathbb{R} \mapsto \sigma_{(i,\alpha)}(\theta[j : v]) \text{ is continuously differentiable.} \end{array}} \quad (10)$$

THEOREM 6.6. *If both our assumption in §6.1 and the condition (9) hold, then the condition (10) implies the requirements R4-R6.*

The proof of the theorem uses the following nontrivial result [Klenke 2014, Theorem 6.28] about exchanging differentiation and integration, a consequence of the dominated convergence theorem.

THEOREM 6.7. *Let $V \subset \mathbb{R}$ be an open interval, and (X, Σ, μ) be a measure space. Suppose that a measurable function $f : V \times X \rightarrow \mathbb{R}$ satisfies the following conditions: (a) for all $v \in V$, the integral $\int \mu(dx) f_v(x)$ is well-defined; (b) for almost all $x \in X$ (w.r.t. μ) and all $v \in V$, the partial derivative $\nabla_v f_v(x)$ with respect to v is well-defined;¹² (c) there is a measurable function $h : X \rightarrow \mathbb{R}$ such that $\int \mu(dx) h(x)$ is well-defined and $|\nabla_v f_v(x)| \leq h(x)$ for all $v \in V$ and almost all $x \in X$ (w.r.t. μ). Then, for all $v \in V$, both sides of the below equation are well-defined, and the equality holds:*

$$\nabla_v \int \mu(dx) f_v(x) = \int \mu(dx) \nabla_v f_v(x).$$

Note that the theorem ensures not only the validity of interchanging differentiation and integration, but also the differentiability of $\int \mu(dx) f_v(x)$ (w.r.t. v) and the integrability of $\nabla_v f_v(x)$ over $x \in X$.

Our condition in (10) is sufficient but not necessary for the requirements R4-R6 to hold, in particular for the objective in (5) to have well-defined partial derivatives in θ . However, its violation is a good indication of a potential problem. The following example illustrates this point.

Example 6.8. Consider a model C and a guide D_θ defined as follows:

$$C \equiv x := \text{sample}_{\text{norm}}("a", 0, 1) \quad D_\theta \equiv x := \text{sample}_{\text{norm}}("a", 0, E[\theta])$$

where $E[\theta] \equiv \text{relu}(\theta) + 2$ and $\text{relu}(v) \triangleq \mathbb{1}_{[v \geq 0]} \cdot v$. Such $E[\theta]$ can definitely appear in machine learning applications, once a guide starts to use neural networks with parameters θ . Let $A = [\{ "a" \} \rightarrow \mathbb{R}]$ and $\sigma(\theta) \triangleq \text{relu}(\theta) + 2$. Then, $\text{dens}(C, s_I)(r) = \mathbb{1}_{[r \in A]} \cdot \mathcal{N}(r("a"); 0, 1)$ and $\text{dens}(D_\theta, s_I)(r) = \mathbb{1}_{[r \in A]} \cdot \mathcal{N}(r("a"); 0, \sigma(\theta))$. Note (10) is violated: σ is non-differentiable at $\theta = 0$. A simple calculation shows:

$$\nabla_\theta \int \rho(dr) \left(\text{dens}(D_\theta, s_I)(r) \cdot \log \frac{\text{dens}(D_\theta, s_I)(r)}{\text{dens}(C, s_I)(r)} \right) = \begin{cases} 0 & \text{if } \theta \in (-\infty, 0) \\ ((2 + \theta)^2 - 1)/(2 + \theta) & \text{if } \theta \in (0, \infty) \\ \text{undefined} & \text{if } \theta = 0. \end{cases}$$

Hence, the objective in (5) does not have a well-defined partial derivative in θ at $\theta = 0$. \square

7 ANALYSIS

In this section, we describe a recipe for building a static analysis that automatically discharges some of the assumptions and conditions given in §6. The recipe ensures that the constructed static analyses are sound with respect to the density semantics in §4.3. We illustrate it by describing four static analyses for verifying the model-guide support match (the requirement R1), the guide-parameter differentiability (the requirement R2), the condition (9), and the condition (10). The analysis for the model-guide support match has been developed significantly more for the Pyro

¹²A more popular notation is $(\partial f_v(x))/(\partial v)$, but we opt for $\nabla_v f_v(x)$ to avoid clutter.

$$\begin{aligned}
\llbracket \text{skip} \rrbracket^\# &\triangleq \text{skip}^\# & \llbracket \text{if } E \{C_0\} \text{ else } \{C_1\} \rrbracket^\# &\triangleq \text{cond}(E)^\#(\llbracket C_0 \rrbracket^\#, \llbracket C_1 \rrbracket^\#) \\
\llbracket x := E \rrbracket^\# &\triangleq \text{update}(x, E)^\# & \llbracket x := \text{sample}_{\text{norm}}(S, E_1, E_2) \rrbracket^\# &\triangleq \text{sample}(x, S, E_1, E_2)^\# \\
\llbracket C_0; C_1 \rrbracket^\# &\triangleq \llbracket C_1 \rrbracket^\# \circ^\# \llbracket C_0 \rrbracket^\# & \llbracket \text{score}_{\text{norm}}(E_0, E_1, E_2) \rrbracket^\# &\triangleq \text{score}(E_0, E_1, E_2)^\# \\
\llbracket \text{while } E \{C\} \rrbracket^\# &\triangleq (\text{wfix } T) \text{ (where } T(t') \triangleq \text{cond}(E)^\#(t' \circ^\# \llbracket C \rrbracket^\#, \text{skip}^\#))
\end{aligned}$$

Fig. 8. Abstract semantics $\llbracket C \rrbracket^\# \in \mathcal{T}^\#$ of commands C

programming language, and applied to analyse realistic examples of the language. This fully-blown analysis and our experiments will be described in §8.

Throughout this section, we assume that the parameters θ of a guide D_θ are included in Var , and are only read by D_θ and not accessed by a model C . What we used to call s_I will be the part of the store for variables in $\text{Var} \setminus \theta$, and what we used to write θ will correspond to the other part for θ .

7.1 Generic Program Analysis Framework

Our recipe is for building a static analysis that infers information about the state transformation of a given command. It is similar to the conventional methodology for building a so called *relational* static analysis, which also attempts to find information about the relationship between input and output states of a given command. However, our recipe diverges from the convention in one important point: while the abstract states of conventional relational analyses represent relations on states, we let abstract states directly express sets of concrete state transformers. This departure from the convention is due to the difficulty of using relations for expressing properties of state transformers that we desire. For instance, we could not express a set of functions with a certain type of differentiability using relations.

Recall the domain \mathcal{D} in (3), and the notion of *admissible* subset from domain theory: $D_0 \subseteq \mathcal{D}$ is *admissible* if it contains \perp and is closed under taking the limits of ω -chains in D_0 .

Our recipe assumes an abstraction instance defined by the following items:

- An *abstract domain*, i.e., a set $\mathcal{T}^\#$ with a designated element $\perp^\#$.
- A *concretisation function*, i.e., a function $\gamma : \mathcal{T}^\# \rightarrow \mathcal{P}(\mathcal{D})$ such that for every $t \in \mathcal{T}^\#$, $\gamma(t)$ is an admissible subset of \mathcal{D} . Note that the concretisation interprets each abstract element t as a set of concrete transformers in \mathcal{D} . The admissibility is imposed to enable the sound analysis of loops.
- A *widening operator* $\text{widen} : \mathcal{T}^\# \times \mathcal{T}^\# \rightarrow \mathcal{T}^\#$, such that for all $t_1, t_2 \in \mathcal{T}^\#$ and $i \in [2]$, $\gamma(t_i) \subseteq \gamma(\text{widen}(t_1, t_2))$ and for every sequence $\{t_n\}_{n \geq 1}$ in $\mathcal{T}^\#$, its widened sequence $\{t'_n\}_{n \geq 1}$, defined by $t'_1 \triangleq t_1$ and $t'_{n+1} \triangleq \text{widen}(t'_n, t_{n+1})$ for $n \geq 1$, has an index m such that $t'_m = t'_{m+1}$.
- An *abstract conditional operator* for every expression E , that is, a function $\text{cond}(E)^\# : \mathcal{T}^\# \times \mathcal{T}^\# \rightarrow \mathcal{T}^\#$ such that for all $t_1, t_2 \in \mathcal{T}^\#$ and $g_1, g_2 \in \mathcal{D}$, if $g_1 \in \gamma(t_1)$ and $g_2 \in \gamma(t_2)$, then $(\lambda(s, r).(\text{if } (\llbracket E \rrbracket s = \text{true}) \text{ then } g_1(s, r) \text{ else } g_2(s, r))) \in \gamma(\text{cond}(E)^\#(t_1, t_2))$.
- An *abstract composition operator* $\circ^\# : \mathcal{T}^\# \times \mathcal{T}^\# \rightarrow \mathcal{T}^\#$ such that for all $t_1, t_2 \in \mathcal{T}^\#$ and $g_1, g_2 \in \mathcal{D}$, if $g_1 \in \gamma(t_1)$ and $g_2 \in \gamma(t_2)$, then $g_2^\ddagger \circ g_1 \in \gamma(t_2 \circ^\# t_1)$.
- For all expressions E_0, E_1, E_2 and for all variables x , the abstract elements $\text{skip}^\#$, $\text{update}(x, E_0)^\#$, $\text{sample}(x, S, E_1, E_2)^\#$, and $\text{score}(E_0, E_1, E_2)^\# \in \mathcal{T}^\#$ such that

$$\begin{aligned}
\llbracket \text{skip} \rrbracket_d &\in \gamma(\text{skip}^\#), & \llbracket x := \text{sample}_{\text{norm}}(S, E_1, E_2) \rrbracket_d &\in \gamma(\text{sample}(x, S, E_1, E_2)^\#), \\
\llbracket x := E_0 \rrbracket_d &\in \gamma(\text{update}(x, E_0)^\#), & \llbracket \text{score}_{\text{norm}}(E_0, E_1, E_2) \rrbracket_d &\in \gamma(\text{score}(E_0, E_1, E_2)^\#).
\end{aligned}$$

Given these data, we define the static analysis $\llbracket C \rrbracket^\# \in \mathcal{T}^\#$ of a command C in Figure 8. Here the $(\text{wfix } T)$ is the usual widened fixed point of T , which is defined as the first element t_m with $t_m = t_{m+1}$ in the widened sequence $(t_n)_{n \geq 1}$ where $t_1 \triangleq \perp^\#$ and $t_{n+1} \triangleq \text{widen}(t_n, T(t_n))$.

THEOREM 7.1 (SOUNDNESS). *For all commands C , we have $\llbracket C \rrbracket_d \in \gamma(\llbracket C \rrbracket^\#)$.*

In the rest of this section, we instantiate this framework into four static analysis instances. In each case, we describe the abstract domain, the abstract bottom element, and the concretisation function. Moreover, in the first two cases, we detail the transfer functions. In the following, for a tuple $(s', r', w', p') \in \text{Store} \times \text{RDB} \times [0, \infty) \times [0, \infty)$, we use the subscripts $-_s$, $-_r$, $-_w$, and $-_p$ to denote its components. For instance, $(s', r', w', p')_s = s'$ and $(s', r', w', p')_r = r'$.

7.2 Analysis for Model-Guide Match

The first instance analysis finds information about the names of sampled random variables. Such information can be used for discharging the requirement **R1**, the correspondence between the support of a model and that of a guide. The analysis is based on the below abstraction:

$$\begin{aligned} \mathcal{T}^\# &\triangleq \{\perp^\#, \top^\#\} \cup \mathcal{P}(\text{Str}), & \perp^\# &\triangleq \perp^\#, & \gamma(\perp^\#) &\triangleq \{\lambda(s, r). \perp\}, \\ \gamma(\top^\#) &\triangleq \mathcal{D}, & \gamma(K) &\triangleq \{g \in \mathcal{D} \mid \forall s, r. g(s, r) \neq \perp \wedge (g(s, r))_r = [] \implies \text{dom}(r) = K\}, \end{aligned}$$

where $[]$ denotes the empty random database. A typical abstract element in $\mathcal{T}^\#$ is a set of names K , which represents concrete commands sampling random variables in K . The domain $\mathcal{T}^\#$ contains $\perp^\#$ and $\top^\#$ to express two extreme cases, the set containing only one command that always returns \perp , and the set of all commands.

Sound abstract operations can be derived from the density semantics and from the abstraction following the abstract interpretation methodology [Cousot and Cousot 1977]:

$$\begin{aligned} \text{widen}(\perp^\#, S) &= \text{widen}(S, \perp^\#) = S; & \text{cond}(E)^\# &= \text{widen}; & \text{skip}^\# &= \text{update}(x, E_0)^\# = \emptyset; \\ \text{widen}(\top^\#, S) &= \text{widen}(S, \top^\#) = \top^\#; & \circ^\# &= \text{widen}; & \text{sample}(x, S, E_1, E_2)^\# &= \{x\}; \\ \text{widen}(S_0, S_1) &= S_0 \text{ if } S_0 = S_1, \top^\# \text{ otherwise} & \text{for } S_0, S_1 \in \mathcal{P}(\text{Str}); & \text{score}(E_0, E_1, E_2)^\# &= \emptyset. \end{aligned}$$

We can use the resulting analysis to discharge the requirement **R1**. We just need to run it on both C and D_θ , and check whether $\llbracket C \rrbracket^\# = \llbracket D_\theta \rrbracket^\# = K$ for some $K \in \mathcal{P}(\text{Str})$. The positive answer implies the requirement **R1**, because all the random variables are drawn from the normal distribution. Our extension of this analysis for Pyro (§8) does not rely on this exclusive use of the normal distribution, and tracks information about the type of distribution of each random variable and state properties, so as to prove the model-guide support match for realistic Pyro programs.

7.3 Analysis for Guide Parameter Differentiability

The second instance analysis aims at proving the differentiability of the density of a guide D_θ with respect to its parameters θ . It infers the continuous partial differentiability of multiple functions with respect to variables in the input state. The analysis is defined by the below abstraction:

$$\begin{aligned} \mathcal{T}^\# &\triangleq \mathcal{P}(\text{Var}) \times \mathcal{P}(\text{Var}) \times \mathcal{P}(\text{Var} \times \text{Var}), & \perp^\# &\triangleq (\text{Var}, \text{Var}, \text{Var} \times \text{Var}), \\ \gamma(X, Y, R) &\triangleq \{g \in \mathcal{D} \mid (\forall x \in X. \forall s, r. g(s, r) \neq \perp \implies s(x) = (g(s, r))_s(x)) \\ &\quad \wedge (\forall y \in Y. \forall s, r. g(s, r) \neq \perp \implies \lambda v \in \mathbb{R}. \text{dens}(g, s[y \mapsto v])(r)) \text{ is } C^1) \\ &\quad \wedge (\forall (z, u) \in R. \forall s, r. g(s, r) \neq \perp \implies \lambda v \in \mathbb{R}. (g(s[z \mapsto v], r))_s(u) \text{ is } \mathbb{R}\text{-valued and } C^1)\}. \end{aligned}$$

By “ C^1 ”, we mean that the relevant function is continuously differentiable. Being a \mathbb{R} -valued function in the last part requires that $g(s[z \mapsto v], r)$ be never \perp . An (X, Y, R) in $\mathcal{T}^\#$ expresses a property of a transformer $g \in \mathcal{D}$ (which can be viewed as semantic command) that g does not change variables in X , its density is C^1 with respect to each variable in Y in the input state, and for each $(z, u) \in R$, it assigns a real value to u in the output state in a C^1 manner with respect to z in the input state.

We now define the abstract operations induced by this abstraction. Given an expression E , we let $\mathcal{V}(E)$ denote the set of variables that occur in E , and we write $C^1(E)$ for the set of variables with

respect to which $\llbracket E \rrbracket$ is C^1 (based on classical differentiability rules). The definitions below follow from general principles such as the multivariate chain rule and account for discontinuities induced by conditions which break differentiability.

$$\begin{aligned}
\text{widen}((X_0, Y_0, R_0), (X_1, Y_1, R_1)) &= (X_0 \cap X_1, Y_0 \cap Y_1, R_0 \cap R_1) \\
\text{cond}(E)^\#((X_0, Y_0, R_0), (X_1, Y_1, R_1)) &= (X_0 \cap X_1, (Y_0 \cap Y_1) \setminus \mathcal{V}(E), \\
&\quad \{(z, u) \in R_0 \cap R_1 \mid z \notin \mathcal{V}(E) \vee u \in X_0 \cap X_1\}) \\
\circ^\#((X_0, Y_0, R_0), (X_1, Y_1, R_1)) &= (X_0 \cap X_1, \{x \in Y_1 \mid \forall y \in \text{Var}. (x, y) \in R_1 \wedge y \in Y_0\}, \\
&\quad \{(z, v) \mid \forall u \in \text{Var}. (z, u) \in R_1 \wedge (u, v) \in R_0\}) \\
\text{skip}^\# &= (\text{Var}, \text{Var}, \text{Var} \times \text{Var}) \\
\text{update}(x, E)^\# &= (\text{Var} \setminus \{x\}, \text{Var}, \text{Var} \times (\text{Var} \setminus \{x\}) \cup \{(y, x) \mid y \in C^1(E)\}) \\
\text{sample}(x, S, E_1, E_2)^\# &= (\text{Var} \setminus \{x\}, C^1(E_1) \setminus (\mathcal{V}(S) \cup \mathcal{V}(E_2)), \\
&\quad (\text{Var} \setminus (\mathcal{V}(S) \cup \mathcal{V}(E_2))) \times \text{Var}) \\
\text{score}(E_0, E_1, E_2)^\# &= (\text{Var}, (C^1(E_0) \cap C^1(E_1)) \setminus \mathcal{V}(E_2), (\text{Var} \setminus \mathcal{V}(E_2)) \times \text{Var})
\end{aligned}$$

To discharge the differentiability requirement **R2**, we need to run this analysis on a guide D_θ . If the Y component of the analysis result contains all the parameters θ (i.e., there exists (X, Y, R) such that $\llbracket D_\theta \rrbracket^\# = (X, Y, R)$ and $\theta \subseteq Y$), then the requirement **R2** is met.

7.4 Analysis for Condition (10)

The third analysis extends the second by tracking and checking more properties. Its aim is to prove the condition (10). Just like the second analysis, it infers information about the continuous partial differentiability of multiple functions involving the output state and the density. Also, it checks whether the density of a given command C_0 has the form

$$\text{dens}(\llbracket C_0 \rrbracket_d, s)(r) = \text{dens}(g, s)(r) = \sum_{i=1}^M \mathbb{1}_{[r \in A_i]} \prod_{\alpha \in K_i} \mathcal{N}(r(\alpha); \mu_{(i, \alpha)}(s), \sigma_{(i, \alpha)}(s)) \quad (11)$$

for some finite M , and some A_i , K_i , $\mu_{(i, \alpha)}$ and $\sigma_{(i, \alpha)}$, and if so, it tracks properties of the $\mu_{(i, \alpha)}$ and $\sigma_{(i, \alpha)}$. Here is the abstraction for the analysis:

$$\begin{aligned}
\mathcal{T}^\# &\triangleq \{\top^\#\} \cup (\mathcal{P}(\text{Var}) \times \mathcal{P}(\text{Var}) \times \mathcal{P}(\text{Var} \times \text{Var})), \quad \perp^\# \triangleq (\text{Var}, \text{Var}, \text{Var} \times \text{Var}), \quad \gamma(\top^\#) \triangleq \mathcal{D}, \\
\gamma(X, Y, R) &\triangleq \{g \in \mathcal{D} \mid g \text{ has the form (11)} \wedge (\forall x \in X. \forall s, r. g(s, r) \neq \perp \implies s(x) = (g(s, r))_s(x)) \\
&\quad \wedge (\forall s, r. g(s, r) \neq \perp \implies \lambda v \in \mathbb{R}. \mu_{(i, \alpha)}(s[\theta_j \mapsto v]) \text{ is } C^1 \text{ for all } i, j, \alpha) \\
&\quad \wedge (\forall s, r. g(s, r) \neq \perp \implies \lambda v \in \mathbb{R}. \sigma_{(i, \alpha)}(s[\theta_j \mapsto v]) \text{ is } C^1 \text{ for all } i, j, \alpha) \\
&\quad \wedge (\forall y \in Y. \forall s, r. g(s, r) \neq \perp \implies \lambda v \in \mathbb{R}. \mu_{(i, \alpha)}(s[y \mapsto v]) \text{ is } C^1 \text{ for all } i, \alpha) \\
&\quad \wedge (\forall y \in Y. \forall s, r. g(s, r) \neq \perp \implies \lambda v \in \mathbb{R}. \sigma_{(i, \alpha)}(s[y \mapsto v]) \text{ is } C^1 \text{ for all } i, \alpha) \\
&\quad \wedge (\forall (z, u) \in R. \forall s, r. g(s, r) \neq \perp \implies \lambda v \in \mathbb{R}. (g(s[z \mapsto v], r))_s(u) \text{ is } \mathbb{R}\text{-valued and } C^1)\}.
\end{aligned}$$

The abstract operations are similar to those for the differentiability analysis thus we omit their definitions. We can use the analysis to prove the condition (10). We just need to run it on a guide D_θ and check whether $\llbracket D_\theta \rrbracket^\#$ is not $\top^\#$. If so, the condition holds.

7.5 Analysis for Condition (9)

The last instance is a static analysis that aims at proving the condition (9). The analysis checks whether a given command C_0 has a density of the following form:

$$\text{dens}(C_0, s)(r) = \text{dens}(g, s)(r) = \sum_{i=1}^M \mathbb{1}_{[r \in A_i]} \left(\prod_{\alpha \in K_i} \mathcal{N}(r(\alpha); \mu'_{(i, \alpha)}(s, r), \sigma'_{(i, \alpha)}(s, r)) \right) \left(\prod_{j \in [N_i]} \mathcal{N}(c_{(i, j)}; \mu''_{(i, j)}(s, r), \sigma''_{(i, j)}(s, r)) \right) \quad (12)$$

for some *finite* M , and some $A_i, K_i, N_i, \mu'_{(i,\alpha)}, \sigma'_{(i,\alpha)}, \mu''_{(i,j)}$, and $\sigma''_{(i,j)}$. If so, it tracks whether the $\mu'_{(i,\alpha)}, \sigma'_{(i,\alpha)}, \mu''_{(i,j)}$, and $\sigma''_{(i,j)}$ and some other functions can be bounded by affine exponentials on the input variables. The abstraction for the analysis is as follows:

$$\begin{aligned} \mathcal{T}^\# &\triangleq \{\tau^\#\} \cup \mathcal{P}(\text{Var}) \times \mathcal{P}(\text{Var}) \times (\text{Var} \rightarrow \mathcal{P}(\text{Var})), \quad \perp^\# \triangleq (\text{Var}, \text{Var}, \lambda x \in \text{Var}. \emptyset), \quad \gamma(\tau^\#) \triangleq \mathcal{D}, \\ \gamma(X, Y, R) &\triangleq \{g \in \mathcal{D} \mid g \text{ has the form (12)} \wedge (\forall x \in X. \forall s, r. g(s, r) \neq \perp \implies s(x) = (g(s, r))_s(x)) \\ &\wedge (\forall i \in [M], \alpha \in K_i. \exists \text{ an affine function } l \text{ from } [Y \cup K_i \rightarrow \mathbb{R}] \text{ to } \mathbb{R} \text{ such that for all } s, r, \\ &\quad g(s, r) \neq \perp \implies \max(|\mu'_{(i,\alpha)}(s, r)|, \sigma'_{(i,\alpha)}(s, r), \sigma'_{(i,\alpha)}(s, r)^{-1}) \leq \exp(l(\{|s(x)|\}_{x \in Y}, \{|r(\beta)|\}_{\beta \in K_i}))) \\ &\wedge (\forall i \in [M], j \in [N_i]. \exists \text{ an affine function } l \text{ from } [Y \cup K_i \rightarrow \mathbb{R}] \text{ to } \mathbb{R} \text{ such that for all } s, r, \\ &\quad g(s, r) \neq \perp \implies \max(|\mu''_{(i,j)}(s, r)|, \sigma''_{(i,j)}(s, r), \sigma''_{(i,j)}(s, r)^{-1}) \leq \exp(l(\{|s(x)|\}_{x \in Y}, \{|r(\beta)|\}_{\beta \in K_i}))) \\ &\wedge (\forall y \in \text{dom}(R), i \in [M]. \exists \text{ an affine function } l \text{ from } [R(y) \cup K_i \rightarrow \mathbb{R}] \text{ to } \mathbb{R} \text{ such that for all } s, r, \\ &\quad g(s, r) \neq \perp \implies (g(s, r))_s(y) \leq \exp(l(\{|s(x)|\}_{x \in R(y)}, \{|r(\beta)|\}_{\beta \in K_i})))\}. \end{aligned}$$

Here we use the notation $\{|s(x)|\}_{x \in Y}$ to mean a partial map from variables x in Y to values $|s(x)|$. The abstract operations that derive from this abstraction are quite similar to those of the differentiability analysis, therefore we do not detail them. To verify the condition (9), we run the analysis on a model C and check whether $\llbracket C \rrbracket^\# \neq \tau^\#$. If the check succeeds, the condition holds.

8 STATIC ANALYSIS FOR MODEL-GUIDE PAIRS IN PYRO AND ITS EVALUATION

We present a static analysis that can verify the support correspondence for Pyro model-guide pairs. The analysis extends the first instance of the framework presented in §7. Our presentation focuses on the most salient aspects of this analysis and experimental results.

8.1 Some Features of Pyro Programs

Pyro is a probabilistic programming framework based on Python and PyTorch. It supports a wide spectrum of distributions and neural networks, and features commands for sampling and scoring. It comes with multiple inference engines, including SVI. We chose Pyro over other probabilistic programming languages (e.g., Anglican) because unlike most of other languages, in Pyro, SVI algorithms are considered as main inference engines and neural networks can be used together with probabilistic programming, which leads to more interesting examples.

In Pyro programs, random-variable names are often created at runtime, for instance by concatenating a fixed string with a sequence of dynamically-generated indices.

Example 8.1 (Use of indexed random-variable names). The code excerpt (of a model or guide program in Pyro) below samples $N \times M$ instances of independent random variables, and names them with “ x_1_1 ”, ..., “ x_N_M ”.

```
for i in range(1, N+1):
    for j in range(1, M+1):
        val = pyro.sample("x_{}-{}".format(i, j), Normal(m, d))
```

□

Since Pyro is based on PyTorch and is adapted to implement data-science applications, Pyro programs heavily use multidimensional arrays, called tensors, and operations over them from PyTorch, in particular, element-wise operations and broadcasting. More precisely, when a binary operation (such as addition) is applied to two tensors of identical size, it outputs a new tensor of the same size where each output element is computed separately from the corresponding input elements. This principle often makes it possible to parallelise computations. Broadcasting occurs when a binary operation is applied to two tensors of different dimensions that can somehow be unified. Intuitively, it introduces and duplicates dimensions to produce tensors of identical dimensions, so that binary operations can be performed element-wise.

Tensors are also heavily used for sampling, which has several consequences. First, it means that an analysis targeted at Pyro programs should be able to track information about at least the dimensions of sampled tensors. Second, the dimensions of sampled tensors are grouped such that each of these groups has a different property with respect to probabilistic independence of tensor components. Pyro inference engines exploit this property for optimisation, for instance, via subsampling, but for this optimisation to be valid, the grouping of dimensions in a model should match that of a guide. Our analysis tracks information about dimension grouping of sampled tensors and the use of the Pyro construct called plate, which enables the optimisation just mentioned.

8.2 Abstract Domain

We extend our analysis from §7 so that it tracks not just how each part of a given program transforms states, but also which states can reach at each program point. We need the latter to get information about random variables with dynamically generated names that is precise enough for our verification task. To describe states, we rely on an abstract domain that consists of multiple subdomains (combined by product). Among them, the key part is $RDB^\# \triangleq [Str \rightarrow_{fin} \{\top\} \cup (\mathcal{P}_{fin}(Zone^\#) \times Dist^\#)]$, where \rightarrow_{fin} denotes a finite partial map. $Dist^\#$ is an abstract domain whose element represents a set of elementary probability distributions, such as the standard normal distribution. The other $Zone^\#$ is our custom domain for zones, which express higher-dimensional rectangles in \mathbb{N}^n .

An element $r^\# \in RDB^\#$ means a set of concrete random databases r for each concrete store s . The domain of r consists of names that are obtained by concatenating baseline strings α in $\text{dom}(r^\#)$ with index sequences. If $r^\#$ maps a baseline string α to \top , it does not give any information about index sequence and a probability distribution used for the random variable. Otherwise, the second component of $r^\#$ specifies the information about the distribution, and the first component a region of \mathbb{N}^n that contains the index sequences used in the names. The region is described by a finite subset $\{Z_1^\#, \dots, Z_n^\#\}$ of $Zone^\#$, which means the disjoint union of the rectangles represented by the $Z_i^\#$. To emphasise this union interpretation, we write $Z_1^\# \cup \dots \cup Z_n^\#$ for the subset. The following table summarises the definitions of our abstract domain, based on $Zone^\#$:

$$\begin{aligned} RDB^\# &\triangleq [Str \rightarrow_{fin} \{\top\} \cup (\mathcal{P}_{fin}(Zone^\#) \times Dist^\#)]; \\ Z^\# &\triangleq I_1^\# \times \dots \times I_m^\#, \text{ product of intervals in } \mathbb{N}; \\ I^\# &\triangleq [B_l^\#, B_r^\#], \text{ closed interval specified by bounds;} \\ B^\# &\triangleq c \mid x + c \mid x + c = c', \text{ equality to constant, variable plus constant, or both.} \end{aligned}$$

A higher-dimensional rectangular zone $Z^\#$ is described by a finite sequence of intervals $I^\#$, each of which is made of two bounds. A bound $B^\#$ may be defined as one or two constraints which express that this bound is equal to a constant, or to a variable plus a constant, or both. This intuitive denotation defines a concretisation function $\gamma_r : RDB^\# \rightarrow \mathcal{P}(State)$.

Example 8.2. Consider the code of Example 8.1. After $i - 1$ iterations in the main loop and j iterations in the last execution of the inner loop, we expect the analysis to come up with the invariant, $["x"] \mapsto ([1, i-1] \times [1, M] \cup [i, i] \times [1, j], \text{normal}(m, d))$. In turn, at the exit of the main loop, we expect the analysis to infer the invariant, $["x"] \mapsto ([1, i=N] \times [1, j=M], \text{normal}(m, d))$. \square

In addition to these constraints over random databases, our analyser also uses an abstraction that maintains typing information and numerical constraints over variables. We do not fully formalise these constraints as the overall structure of the domain relies on a classical reduced product.

Finally, we describe the combination of the above state abstraction with the abstraction of §7.2. More precisely, we start with the abstract domain exposed in §7.2 (which we denote by $\mathcal{T}^\#$) and

build a novel abstract domain $\mathcal{T}_s^\#$ that also satisfies the requirements of §7.1. We let the set of abstract elements be $\mathcal{T}_s^\# \triangleq [RDB^\# \rightarrow \mathcal{T}^\# \times RDB^\#]$. Intuitively, such an element maps an input abstract random database into a set of functions together with an over-approximation of their output, when applied to this abstract input. The concretisation below formalises this: for all $t_s \in \mathcal{T}_s^\#$,

$$\gamma_s(t_s) \triangleq \{g \in \mathcal{D} \mid \forall r_i^\# \in RDB^\#. t_s(r_i^\#) = (t, r_o^\#) \implies \\ \exists g' \in \gamma(t). \forall (s, r) \in \gamma_r(r_i^\#). g(s, r) = g'(s, r) \wedge (g(s, r) = \perp \vee (g(s, r)_s, g(s, r)_r) \in \gamma_r(r_o^\#))\}.$$

8.3 Computation of Loop Invariants

Although our static analysis for Pyro requires a state abstraction, its principles and structure are similar to those of the general analysis shown in §7. In the following, we first describe the integration of the state abstraction in the analysis of §7.2.

The abstract operations in $\mathcal{T}_s^\#$ can all be derived by lifting those in $\mathcal{T}^\#$ into functions. We illustrate this for the abstract composition $\circ_s^\#$ for $\mathcal{T}_s^\#$. Recall the operator $\circ^\#$ for $\mathcal{T}^\#$ in §7.2. Given $t_s, t'_s \in \mathcal{T}_s^\#$,

$$t_s \circ_s^\# t'_s \triangleq \lambda r_0^\#. (t \circ^\# t', r_2^\#) \quad \text{where } (t', r_1^\#) = t'_s(r_0^\#) \text{ and } (t, r_2^\#) = t_s(r_1^\#).$$

The other abstract operators over $\mathcal{T}_s^\#$ are defined in a similar manner.

As in most static analysis problems, the computation of precise loop invariants requires a carefully designed widening operator. In the case of zones, the analysis needs to generalise information about the bounds. We assume $r_0^\#, r_1^\# \in RDB^\#$ are abstract random databases, and present the main steps in the computation of the widening $\text{widen}(r_0^\#, r_1^\#)$.

- (i) For each $r_i^\#$, we fuse zones $Z_0^\#, Z_1^\#$ in $r_i^\#$ into a single zone when $Z_0^\#$ and $Z_1^\#$ are the same except for one component and they represent adjacent high-dimensional rectangles. The following rewriting illustrates this step.

$$\begin{aligned} ("x" \mapsto ([1, i-1] \times [1, j=M] \cup [i, i] \times [1, j=M], \text{normal}(m, d))) \\ \rightsquigarrow ("x" \mapsto ([1, i] \times [1, j=M], \text{normal}(m, d))) \end{aligned}$$

- (ii) We generalise the intervals of corresponding zones in $r_0^\#$ and $r_1^\#$ using a weak version of unification on their bounds. The bounds of corresponding intervals in two such zones survive this step in a weakened form if they share at least one syntactically equal expression. Otherwise, the bounds are dropped, which causes the introduction of \top into abstract random databases. The following rewriting instances illustrate this bound generalisation.

$$\begin{aligned} \text{widen}((("x" \mapsto ([1, i] \times [1, j-1=M], \text{normal}(m, d))), ("x" \mapsto ([1, i] \times [1, j=M], \text{normal}(m, d)))) \\ = ("x" \mapsto ([1, i] \times [1, M], \text{normal}(m, d))) \\ \text{widen}((("x" \mapsto ([1, i] \times [1, j-1], \text{normal}(m, d))), ("x" \mapsto ([1, i] \times [1, j=M], \text{normal}(m, d)))) \\ = ("x" \mapsto \top) \end{aligned}$$

The analysis also applies standard widening techniques to typing information and numerical constraints mentioned in §8.2. Finally, a model-guide pair can be verified if and only if their analyses return equivalent abstract random databases, without any name mapped into \top .

8.4 Experimental Evaluation

We have implemented a prototype analyser and carried out experiments so as to assess the effectiveness of our analysis to verify the support correspondence for Pyro model-guide pairs.¹³ More precisely, we evaluated the following three research questions:

¹³Code is available at <https://github.com/wonyeol/static-analysis-for-support-match>.

- (i) Can our analysis discover incorrect model-guide pairs in realistic probabilistic programs?
- (ii) Can the analysis verify correct model-guide pairs despite the complexity of the definition?
- (iii) Is the analysis efficient enough so that it can cope with realistic probabilistic programs?

Benchmark programs. We took example programs from the Pyro webpage that can be handled by standard SVI engines or can be naturally rewritten to be so. Those engines do not use Pyro's recent vectorised enumeration-based optimisation (enabled by the option `TraceEnum_ELBO`), and are run with the option `Trace_ELBO`. We made this choice because the optimisation is newly introduced and whether it is used or not changes the set of random variables used in a Pyro program.

We have applied our analysis to two groups of Pyro programs. The first is the *Pyro regression test suite* [Uber AI Labs 2019b], which comprises 66 examples exercising the basic programming patterns expected in Pyro programs. While each of these is small, they cover many standard patterns of both correct and incorrect model-guide pairs. Among these, 33 test cases use only `TraceEnum_ELBO` and fall outside the scope of our analysis, and 6 of the remaining 33 test cases come with two subsampling options. Hence, we can consider 39 experiments based on this regression suite.

The second is a collection of *examples* from the Pyro webpage [Uber AI Labs 2019a]. The webpage features 18 *Pyro examples*, among which 9 involve model-guide pairs (other examples use automatic guide generation, or do not perform SVI at all). Out of these 9 examples, 5 use `Trace_ELBO`, and three can be converted naturally into `Trace_ELBO` versions although they do not use `Trace_ELBO`. Thus, 8 Pyro examples fall within the scope of our analysis. These examples correspond to advanced probabilistic models¹⁴ from the machine-learning literature, most of which use sophisticated neural networks and probabilistic modelling. Table 1 describes the structure of these 8 examples in detail.

Prototype analyser and results. Our analyser is implemented in OCaml, and supports the main data-structures and operations defined by Python, PyTorch, and Pyro. In particular, it precisely abstracts the shape of PyTorch tensor objects, the shape transformation information of PyTorch neural-network-related objects, the automatic broadcasting information of Pyro plate objects, and the shape of allocated indices for sample names, using the zone abstraction described above. It also supports common Pyro probability distributions, and can precisely cope with standard Pyro and PyTorch operations manipulating the Pyro distribution objects and PyTorch tensor objects. While our prototype supports a wide range of Python, PyTorch, and Pyro features, we point out that we did not implement a static analysis for the full Python (plus PyTorch and Pyro) language (e.g., no support for classes, dictionaries, named tuples, and user-defined functions).

The analysis results are summarised in Table 2 and are discussed in detail in the following. Run-times were measured on an Intel Core i7-7700 machine running Linux Ubuntu 16.04.

Discovery of invalid model-guide pairs. The analysis rejected two Pyro examples, `br` and `lda`, as incorrect due to an invalid model-guide pair. `br` is the Bayesian regression example discussed in §2.

For `br`, the analysis discovers that a random variable `sigma` is sampled from `Uniform(0., 10.)` in the model C , but from `Normal(. .)` in the guide D_θ (Figure 2(a)). Since the support of `sigma` in D_θ is not a subset of that in C (i.e., $\mathbb{R} \not\subseteq [0, 10]$), the requirement R1 is violated. Thus, the SVI objective, $KL(D_\theta \| C)$, is undefined, and `br` has an invalid model-guide pair.

For `lda`, the analysis discovers that a random variable `doc_topics` is sampled from `Dirichlet(. .)` in the model C , but from `Delta(. .)` in the guide D_θ . Since the reference measures of `doc_topics` in C and D_θ are different (the Lebesgue measure vs. the counting measure), $KL(D_\theta \| C)$ cannot be computed by (4). For this reason, `lda` has an invalid model-guide pair. Our analyser tracks the

¹⁴The models include variational autoencoder (VAE) [Kingma and Welling 2014], semi-supervised VAE [Kingma et al. 2014], deep exponential family [Ranganath et al. 2015], attend-infer-repeat [Eslami et al. 2016], deep Markov model [Krishnan et al. 2017], inference compilation [Le et al. 2017], and amortised latent Dirichlet allocation [Srivastava and Sutton 2017].

Table 1. Key features of the model-guide pairs from Pyro examples. LoC denotes the lines of code of model and guide. The columns “Total #” show the number of objects/commands of each type used in model and guide, and the columns “Total dimension” show the total dimension of tensors in model and guide, either sampled from sample or used inside score, as well as the dimension of θ in guide.

Name	Corresponding probabilistic model	LoC	Total #				Total dimension		
			for	plate	sample	score	sample	score	θ
br	Bayesian regression	27	0	1	10	1	10	170	9
csis	Inference compilation	31	0	0	2	2	2	2	480
lda	Amortised latent Dirichlet allocation	76	0	5	8	1	21008	64000	121400
vae	Variational autoencoder (VAE)	91	0	2	2	1	25600	200704	353600
sgdef	Deep exponential family	94	0	8	12	1	231280	1310720	231280
dmm	Deep Markov model	246	3	2	2	1	640000	281600	594000
ssvae	Semi-supervised VAE	349	0	2	4	1	24000	156800	844000
air	Attend-infer-repeat	410	2	2	6	1	20736	160000	6040859

Table 2. Analysis results on two benchmark sets: Pyro test suite (Left) and Pyro examples (Right). The model-guide pairs from Pyro test suite are grouped into 7 categories, based on which type of plate objects are used. #Same (or #Diff) denotes the number of model-guide pairs for which the output of our analyser, valid or invalid, is the same as (or different from) the documented output. #Crash denotes the number of pairs for which our analyser crashes. The column “Time” shows the analysis time in seconds; on Left, it is averaged over those model-guide pairs (in each category) for which our analyser does not crash. The column “Valid?” shows the output of our analysis, valid or invalid.

Category	#Same	#Diff	#Crash	Time	Name	Valid?	Time
No plates	9	0	0	0.001	br	x	0.006
Single for-plate	4	0	3	0.004	csis	o	0.007
Nested for-plates	2	0	2	0.026	lda	x	0.014
Single with-plate	5	0	0	0.001	vae	o	0.005
Nested with-plates	7	2	0	0.002	sgdef	o	0.070
Non-nested with-plates	2	0	0	0.002	dmm	o	0.536
Nested for-plate & with-plate	0	0	3	N/A	ssvae	o	0.013
Total	29	2	8	0.003	air	o	4.093

reference measure implicitly by regarding the support of any distribution with Lebesgue measure, as disjoint from that of any distribution with counting measure (which is sound due to the aforementioned reason), and this allowed us to discover the issue of `lda`.

In both cases, the found correctness issues are unknown before and subtle. In particular, it turned out that `lda`, though incorrect when viewed as an example of SVI, is a valid implementation, because it performs not SVI but variational expectation-maximisation (EM) [Neal and Hinton 1998].¹⁵ The `lda` uses an SVI engine just to solve an optimisation problem in variational EM (not to do SVI), and uses the Delta distribution to perform a particular realisation of the M-step in variational EM.

Verification of probabilistic programs relying on model-guide pairs. Among the Pyro test suite, the analysis successfully verifies 31 examples among 39. Interestingly, two of these 31 successful validations, highlighted in Table 2(a), correspond to cases that were flagged as “invalid model-guide pairs” in the Pyro git repository. Upon inspection, these two examples turn out to be correct.

¹⁵We thank an anonymous reviewer and Eli Bingham for pointing this out and explaining it in detail.

On the other hand, 8 examples from the Pyro test suite could not be verified due to the crashes of the analyser. One of these failures is due to the need to reason more precisely about the content of a `for` loop (e.g., using some partitioning techniques), and seven are due to the use of plates with subsampling, as ranges for `for` loops. Therefore, these failures could be resolved using existing static analysis techniques and a more precise handling of the semantics of Python constructions.

Moreover, 6 Pyro examples (among the 8 that we considered) were verified successfully, which means all correct Pyro examples were verified. Finally, we corrected the two examples that were rejected due to invalid model-guide pairs, and these two examples were also successfully verified.

Analysis efficiency. The analysis returned within a second on each program in the Pyro test suite, and on most of the Pyro examples. The slowest analysis was observed on `air`, which was analysed within 5 seconds. Most of the Pyro examples sample from and score with distributions of very high dimension arranged in complex tensors, using nested `for` and `plate`'s. While they are not large, they present a high degree of logical complexity, that is representative of realistic probabilistic programs. The fact that such programs get analysed within seconds shows that the analysis and the underlying abstract domain to describe zones, sampled dimensions, and distributions can generalise predicates quickly so that precise loop invariants can be computed.

9 RELATED WORK, CONCLUSION AND LIMITATION

Related work. As far as we know, the idea of using SVI for probabilistic programs first appeared in [Wingate and Weber 2013]. When further insights into how to create generic (sometimes also called black-box) SVI engines were found [Kucukelbir et al. 2015, 2017; Ranganath et al. 2014], the idea was tried for realistic probabilistic programming languages, such as Stan [Kucukelbir et al. 2015] and Anglican [van de Meent et al. 2016], resulting in impressive case studies [Kucukelbir et al. 2015]. The major inference engines for deep probabilistic programming languages [Bingham et al. 2019; Siddharth et al. 2017; Tran et al. 2018, 2016] are all based on SVI nowadays. However, we do not know of any prior work that attempts to reveal implicit assumptions made by these SVI engines and to discharge these assumptions manually or automatically, as we did in this paper.

The correctness of a different type of inference engines based on Monte-Carlo methods has been the subject of study in the PL community. Such engines have clearly formulated correctness conditions from Markov chain theory [Geyer 2011; Green 1995; Hastings 1970; Metropolis et al. 1953], such as ergodicity and correct stationarity. Tools from formal semantics have been employed to show that the inference engines satisfy these conditions [Borgström et al. 2016; Hur et al. 2015; Kiselyov 2016; Scibior et al. 2018]. While looking at different algorithms, some of these works consider more expressive languages than what we used in the paper, in particular, those supporting higher-order functions. One interesting direction is to extend our results to such expressive languages using the ideas from these works, especially the operational technique in [Borgström et al. 2016] and the denotational monad-based technique in [Scibior et al. 2018].

The consequence of having random choice in a programming language has been actively investigated by the semantics researchers from the early days [Borgström et al. 2016; Ehrhard et al. 2014; Heunen et al. 2017; Jones and Plotkin 1989; Kozen 1981; Smolka et al. 2017; Staton 2017; Staton et al. 2016; Toronto et al. 2015; Vákár et al. 2019]. Our work uses the technique developed in this endeavour, such as Giry monad and denotational formulation of idealised importance sampling [Staton et al. 2016]. Also, just as we connected the measure semantics with the density semantics, [Kozen 1981] and [Borgström et al. 2016] related two semantics with similar flavours, although the considered languages (with or without continuous distribution and score statements) and the style of semantics (operational vs denotational) are different. Our density semantics uses a reference measure built out of Lebesgue measure as in [Bhat et al. 2012, 2013; Hur et al. 2015]. This choice

is found to cause an issue when the score statement is treated not as a scoring mechanism but in terms of conditional expectation [Wu et al. 2018]. How to resolve this matter is still open.

Static analyses for probabilistic programming languages or languages with probabilistic choice typically attempt to check probabilistic properties [Chakarov and Sankaranarayanan 2013; Cousot and Monerau 2012; Monniaux 2000; Wang et al. 2018], or perform posterior inference [Chaganty et al. 2013; Wang et al. 2018], or find information useful for posterior inference [Nori et al. 2014]. For instance, [Monniaux 2000] defines a probabilistic abstract interpretation framework, which is applied to estimate the failure probability [Monniaux 2001]. More recently, [Cousot and Monerau 2012] sets up a general framework to design probabilistic abstract interpretations, which lift conventional static analysis tools to a probabilistic setup, and [Wang et al. 2018] proposes a framework for analysing probabilistic programs based on hyper-graphs with probabilistic actions on edges. Our program analyses aim at a different type of verification tasks, namely, proving the safety requirements imposed by the SVI engines. Static analyses for checking the continuity properties of programs are proposed in [Chaudhuri et al. 2010]. Some techniques used in those analyses may help track the kind of smoothness properties considered in this paper precisely.

Conclusion. In this paper, we have set up a semantic model of probabilistic programming languages that allows to reason about the correctness of subtle optimisation algorithms such as SVI. In particular, we have identified a set of conditions that guarantee that these algorithms will produce meaningful results. These conditions simplify the verification of correctness, and open the way towards the automation of this verification. To evaluate the effectiveness of our framework, we have completed the design of a static analysis to validate the correspondence assumptions for the model-guide pairs. Our analysis could verify nontrivial implementations as well as uncover nontrivial bugs. Such bugs would lead to incorrect results, which would be very difficult to diagnose.

Limitation. We described four static analyses (§7.2–§7.5) for discharging the requirements R1–R6, and developed one of the four analyses further (§8.2–§8.4) to build a prototype analyser for actual Pyro programs. Here we clarify the underlying assumptions and limitations of these analyses.

The static analysis for model-guide support match (§7.2) was implemented into our static analyser for Pyro (§8.4) but with additional extensions (§8.2–§8.3), so that it does not make the assumption in §6.1; the assumption was introduced mainly to develop analyses for the requirements R3–R6. Hence, our static analyser handles both continuous and discrete random variables.

On the other hand, other analyses (§7.3–§7.5) are unimplemented and make the assumption in §6.1. We expect that the assumption can be relaxed, without much difficulty, to permit continuous or discrete distributions having finite entropy and moments of all degrees, because our proofs of theorems and lemmas bound various expectations mostly by entropies and moments. It would be more challenging, however, to relax the assumption to allow distributions not having finite entropy and moments of all degrees, or models having unboundedly many random variables (of any kind). In particular, addressing the later limitation might require techniques developed for proving that probabilistic systems has finite expected execution time.

This paper focuses on a particular optimisation objective (5) for SVI, but we point out that other objectives have been proposed for different inference or learning algorithms, such as variational EM (e.g., `lda` in §8.4) and importance weighted autoencoder [Burda et al. 2016]. One interesting research direction is to develop techniques for verifying the well-definedness of these objectives.

ACKNOWLEDGMENTS

We thank Fritz Obermeyer and Eli Bingham for explaining the subtleties of Pyro and suggesting us to try the Pyro regression test suite. Sam Staton, Ohad Kammar and Matthijs Vákár helped us to understand the semantics of recursion in the probabilistic programming languages better. Lee, Yang

and Yu were supported by the Engineering Research Center Program through the National Research Foundation of Korea (NRF) funded by the Korean Government MSIT (NRF-2018R1A5A1059921), and also by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT (2017M3C4A7068177).

REFERENCES

- Sooraj Bhat, Ashish Agarwal, Richard W. Vuduc, and Alexander G. Gray. 2012. A type theory for probability density functions. In *Principles of Programming Languages (POPL)*. 545–556.
- Sooraj Bhat, Johannes Borgström, Andrew D. Gordon, and Claudio V. Russo. 2013. Deriving Probability Density Functions from Probabilistic Functional Programs. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. 508–522.
- Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul A. Szerlip, Paul Horsfall, and Noah D. Goodman. 2019. Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research* 20, 28 (2019), 1–6.
- Johannes Borgström, Ugo Dal Lago, Andrew D. Gordon, and Marcin Szymczak. 2016. A lambda-calculus foundation for universal probabilistic programming. In *International Conference on Functional Programming (ICFP)*. 33–46.
- Yuri Burda, Roger B. Grosse, and Ruslan Salakhutdinov. 2016. Importance Weighted Autoencoders. In *International Conference on Learning Representations (ICLR)*.
- Bob Carpenter, Andrew Gelman, Matthew Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. 2017. Stan: A Probabilistic Programming Language. *Journal of Statistical Software, Articles* 76, 1 (2017), 1–32.
- Arun Tejasvi Chaganty, Aditya V. Nori, and Sriram K. Rajamani. 2013. Efficiently Sampling Probabilistic Programs via Program Analysis. In *Artificial Intelligence and Statistics (AISTATS)*. 153–160.
- Aleksandar Chakarov and Sriram Sankaranarayanan. 2013. Probabilistic Program Analysis with Martingales. In *Computer Aided Verification (CAV)*. 511–526.
- Swarat Chaudhuri, Sumit Gulwani, and Roberto Lubliner. 2010. Continuity analysis of programs. In *Principles of Programming Languages (POPL)*. 57–70.
- Patrick Cousot and Radhia Cousot. 1977. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Principles of Programming Languages (POPL)*. 238–252.
- Patrick Cousot and Radhia Cousot. 1979. Systematic design of program analysis frameworks. In *Principles of Programming Languages (POPL)*. 269–282.
- Patrick Cousot and Radhia Cousot. 1992. Abstract Interpretation Frameworks. *Journal of Logic and Computation* 2, 4 (1992), 511–547.
- Patrick Cousot and Michael Monerau. 2012. Probabilistic Abstract Interpretation. In *European Symposium on Programming (ESOP)*. 169–193.
- Thomas Ehrhard, Christine Tasson, and Michele Pagani. 2014. Probabilistic coherence spaces are fully abstract for probabilistic PCF. In *Principles of Programming Languages (POPL)*. 309–320.
- S. M. Ali Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, David Szepesvari, Koray Kavukcuoglu, and Geoffrey E. Hinton. 2016. Attend, Infer, Repeat: Fast Scene Understanding with Generative Models. In *Neural Information Processing Systems (NIPS)*. 3233–3241.
- Timon Gehr, Sasa Misailovic, and Martin T. Vechev. 2016. PSI: Exact Symbolic Inference for Probabilistic Programs. In *Computer Aided Verification (CAV)*. 62–83.
- Charles J. Geyer. 2011. Introduction to Markov Chain Monte Carlo. In *Handbook of Markov Chain Monte Carlo*, Steve Brooks, Andrew Gelman, Galin L. Jones, and Xiao-Li Meng (Eds.). Chapman and Hall/CRC, Chapter 1, 3–48.
- Hamid Ghouchian, Amin Gohari, and Arash Amini. 2017. Existence and Continuity of Differential Entropy for a Class of Distributions. *IEEE Communications Letters* 21, 7 (2017), 1469–1472.
- Noah Goodman, Vikash Mansinghka, Daniel M Roy, Keith Bonawitz, and Joshua B Tenenbaum. 2008. Church: a language for generative models. In *Uncertainty in Artificial Intelligence (UAI)*. 220–229.
- Andrew D. Gordon, Thore Graepel, Nicolas Rolland, Claudio Russo, Johannes Borgstrom, and John Guiver. 2014. Tabular: A Schema-driven Probabilistic Programming Language. In *Principles of Programming Languages (POPL)*. 321–334.
- Peter J. Green. 1995. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika* 82, 4 (1995), 711–732.
- Wilfred Keith Hastings. 1970. Monte Carlo Sampling Methods Using Markov Chains and Their Applications. *Biometrika* 57, 1 (1970), 97–109.
- Chris Heunen, Ohad Kammar, Sam Staton, and Hongseok Yang. 2017. A convenient category for higher-order probability theory. In *Logic in Computer Science (LICS)*. 1–12.

- Matthew D. Hoffman, David M. Blei, Chong Wang, and John Paisley. 2013. Stochastic Variational Inference. *Journal of Machine Learning Research* 14 (2013), 1303–1347.
- Chung-Kil Hur, Aditya V. Nori, Sriram K. Rajamani, and Selva Samuel. 2015. A Provably Correct Sampler for Probabilistic Programs. In *Foundation of Software Technology and Theoretical Computer Science (FSTTCS)*. 475–488.
- C. Jones and Gordon D. Plotkin. 1989. A Probabilistic Powerdomain of Evaluations. In *Logic in Computer Science (LICS)*. 186–195.
- Diederik P. Kingma, Danilo J. Rezende, Shakir Mohamed, and Max Welling. 2014. Semi-supervised Learning with Deep Generative Models. In *Neural Information Processing Systems (NIPS)*. 3581–3589.
- Diederik P. Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes. In *International Conference on Learning Representations (ICLR)*.
- Oleg Kiselyov. 2016. Probabilistic Programming Language and its Incremental Evaluation. In *Asian Symposium on Programming Languages and Systems (APLAS)*. 357–376.
- Achim Klenke. 2014. *Probability Theory: A Comprehensive Course* (second ed.). Springer-Verlag London.
- Dexter Kozen. 1981. Semantics of Probabilistic Programs. *J. Comput. System Sci.* 22, 3 (1981), 328–350.
- Rahul G. Krishnan, Uri Shalit, and David Sontag. 2017. Structured Inference Networks for Nonlinear State Space Models. In *AAAI Conference on Artificial Intelligence (AAAI)*. 2101–2109.
- Alp Kucukelbir, Rajesh Ranganath, Andrew Gelman, and David M. Blei. 2015. Automatic Variational Inference in Stan. In *Neural Information Processing Systems (NIPS)*. 568–576.
- Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M. Blei. 2017. Automatic Differentiation Variational Inference. *Journal of Machine Learning Research* 18 (2017), 14:1–14:45.
- Tuan Anh Le, Atılım Güneş Baydin, and Frank Wood. 2017. Inference Compilation and Universal Probabilistic Programming. In *Artificial Intelligence and Statistics (AISTATS)*. 1338–1348.
- Wonyeol Lee, Hangyeol Yu, Xavier Rival, and Hongseok Yang. 2019. Towards Verified Stochastic Variational Inference for Probabilistic Programs. *arXiv:1907.08827* (2019).
- Vikash K. Mansinghka, Daniel Selsam, and Yura N. Perov. 2014. Venture: a higher-order probabilistic programming platform with programmable inference. *arXiv:1404.0099* (2014).
- Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. 1953. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics* 21, 6 (1953), 1087–1092.
- T. Minka, J.M. Winn, J.P. Guiver, S. Webster, Y. Zaykov, B. Yangel, A. Spengler, and J. Bronskill. 2014. Infer.NET 2.6. Microsoft Research Cambridge. <http://research.microsoft.com/infernet>.
- Matthew Mirman, Timon Gehr, and Martin T. Vechev. 2018. Differentiable Abstract Interpretation for Provably Robust Neural Networks. In *International Conference on Machine Learning (ICML)*. 3575–3583.
- David Monniaux. 2000. Abstract Interpretation of Probabilistic Semantics. In *Static Analysis Symposium (SAS)*. 322–339.
- David Monniaux. 2001. Backwards Abstract Interpretation of Probabilistic Programs. In *European Symposium on Programming (ESOP)*. 367–382.
- Chandra Nair, Balaji Prabhakar, and Devavrat Shah. 2006. On Entropy for Mixtures of Discrete and Continuous Variables. *arXiv:cs/0607075* (2006).
- Praveen Narayanan, Jacques Carette, Wren Romano, Chung-chieh Shan, and Robert Zinkov. 2016. Probabilistic inference by program transformation in Hakaru (system description). In *Functional and Logic Programming (FLOPS)*. 62–79.
- Radford M. Neal and Geoffrey E. Hinton. 1998. A View of the Em Algorithm that Justifies Incremental, Sparse, and other Variants. In *Learning in Graphical Models*. 355–368.
- Aditya V. Nori, Chung-Kil Hur, Sriram K. Rajamani, and Selva Samuel. 2014. R2: An Efficient MCMC Sampler for Probabilistic Programs. In *AAAI Conference on Artificial Intelligence (AAAI)*. 2476–2482.
- John William Paisley, David M. Blei, and Michael I. Jordan. 2012. Variational Bayesian Inference with Stochastic Search. In *International Conference on Machine Learning (ICML)*. 1363–1370.
- Rajesh Ranganath, Sean Gerrish, and David M. Blei. 2014. Black Box Variational Inference. In *Artificial Intelligence and Statistics (AISTATS)*. 814–822.
- Rajesh Ranganath, Linpeng Tang, Laurent Charlin, and David Blei. 2015. Deep Exponential Families. In *Artificial Intelligence and Statistics (AISTATS)*. 762–771.
- Adam Scibior, 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. *PACMPL* 2, POPL (2018), 60:1–60:29.
- N. Siddharth, Brooks Paige, Jan-Willem van de Meent, Alban Desmaison, Noah D. Goodman, Pushmeet Kohli, Frank Wood, and Philip Torr. 2017. Learning Disentangled Representations with Semi-Supervised Deep Generative Models. In *Neural Information Processing Systems (NIPS)*. 5927–5937.
- Steffen Smolka, Praveen Kumar, Nate Foster, Dexter Kozen, and Alexandra Silva. 2017. Cantor meets scott: semantic foundations for probabilistic networks. In *Principles of Programming Languages (POPL)*. 557–571.

- Akash Srivastava and Charles A. Sutton. 2017. Autoencoding Variational Inference For Topic Models. In *International Conference on Learning Representations (ICLR)*.
- Sam Staton. 2017. Commutative Semantics for Probabilistic Programming. In *European Symposium on Programming (ESOP)*. 855–879.
- Sam Staton, Hongseok Yang, Frank D. Wood, Chris Heunen, and Ohad Kammar. 2016. Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. In *Logic in Computer Science (LICS)*. 525–534.
- Neil Toronto, Jay McCarthy, and David Van Horn. 2015. Running Probabilistic Programs Backwards. In *European Symposium on Programming (ESOP)*. 53–79.
- Dustin Tran, Matthew D. Hoffman, Dave Moore, Christopher Suter, Srinivas Vasudevan, and Alexey Radul. 2018. Simple, Distributed, and Accelerated Probabilistic Programming. In *Neural Information Processing Systems (NeurIPS)*. 7609–7620.
- Dustin Tran, Alp Kucukelbir, Adji B. Dieng, Maja R. Rudolph, Dawen Liang, and David M. Blei. 2016. Edward: A library for probabilistic modeling, inference, and criticism. *arXiv:1610.09787* (2016).
- Uber AI Labs. 2019a. Pyro examples. <http://pyro.ai/examples/>. Version used: April 1, 2019.
- Uber AI Labs. 2019b. Pyro regression test suite. https://github.com/pyro-ppl/pyro/blob/dev/tests/infer/test_valid_models.py. Version used: March 1, 2019.
- Matthijs Vákár, Ohad Kammar, and Sam Staton. 2019. A domain theory for statistical probabilistic programming. *PACMPL* 3, POPL (2019), 36:1–36:29.
- Jan-Willem van de Meent, Brooks Paige, David Tolpin, and Frank D. Wood. 2016. Black-Box Policy Search with Probabilistic Programs. In *Artificial Intelligence and Statistics (AISTATS)*. 1195–1204.
- Di Wang, Jan Hoffmann, and Thomas W. Reps. 2018. PMAF: an algebraic framework for static analysis of probabilistic programs. In *Programming Language Design and Implementation (PLDI)*. 513–528.
- Ronald J. Williams. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning* 8, 3-4 (1992), 229–256.
- David Wingate and Theophane Weber. 2013. Automated Variational Inference in Probabilistic Programming. *arXiv:1301.1299* (2013).
- Frank Wood, Jan Willem van de Meent, and Vikash Mansinghka. 2014. A New Approach to Probabilistic Programming Inference. In *Artificial Intelligence and Statistics (AISTATS)*. 1024–1032.
- Yi Wu, Siddharth Srivastava, Nicholas Hay, Simon Du, and Stuart J. Russell. 2018. Discrete-Continuous Mixtures in Probabilistic Programming: Generalized Semantics and Inference Algorithms. In *International Conference on Machine Learning (ICML)*. 5339–5348.
- Hongseok Yang. 2019. Implementing Inference Algorithms for Probabilistic Programs. <https://github.com/hongseok-yang/probprog19/blob/master/Lectures/Lecture6/Note6.pdf>. Lecture Note of the 2019 Course on Probabilistic Programming at KAIST.

A PROOFS OF LEMMAS AND THEOREMS IN §4

LEMMA 4.1. *For all expressions E , B , and S , their semantics $\llbracket E \rrbracket$, $\llbracket B \rrbracket$ and $\llbracket S \rrbracket$ are measurable functions from $Store$ to \mathbb{R} , \mathbb{B} and Str , respectively.*

PROOF. We prove the lemma by structural induction. We present the proof for E only, because those for B and S are similar. When E is a constant, $\llbracket E \rrbracket$ is a constant function and so it is measurable. When E is a variable, $\llbracket x \rrbracket$ is a projection of the x component of the given store s . Since $Store$ is given the product σ -algebra, the projection $\llbracket x \rrbracket$ is measurable. Finally, assume that E is $f(E_0, \dots, E_{n-1})$. Then, by induction hypothesis, the $\llbracket E_i \rrbracket$ are measurable functions from $Store$ to \mathbb{R} . Thus,

$$\lambda s. (\llbracket E_0 \rrbracket s, \dots, \llbracket E_{n-1} \rrbracket s) : Store \rightarrow \mathbb{R}^n$$

is measurable. Furthermore, $\llbracket f \rrbracket : \mathbb{R}^n \rightarrow \mathbb{R}$ is measurable. Thus, their composition is measurable as well. This implies the measurability of $\llbracket E \rrbracket$ because

$$\llbracket E \rrbracket = \llbracket f \rrbracket \circ (\lambda s. (\llbracket E_0 \rrbracket s, \dots, \llbracket E_{n-1} \rrbracket s)).$$

□

LEMMA 4.2. $(\mathcal{K}, \sqsubseteq)$ is an ω -complete partial order with the least element \perp .

PROOF. The fact that \sqsubseteq is a partial order immediately follows from its definition. The least element is the function $\lambda \sigma. \lambda A. 0$. It remains to show the ω -completeness property. Consider an ω -chain

$$\kappa_0 \sqsubseteq \kappa_1 \sqsubseteq \dots$$

Let $\kappa = \sup_n \kappa_n$. We will show that κ is the desired upper bound.

Pick a state σ . Then, $\kappa(\sigma)(A)$ is non-negative for all measurable subsets A , and $\kappa(\sigma)(\emptyset) = 0$. Furthermore, for every countable collection $\{A_j\}_j$ of disjoint measurable subsets of $State \times [0, \infty)$,

$$\begin{aligned} \kappa(\sigma)\left(\bigcup_j A_j\right) &= \sup_n \kappa_n(\sigma)\left(\bigcup_j A_j\right) = \lim_n \kappa_n(\sigma)\left(\bigcup_j A_j\right) \\ &= \lim_n \sum_j \kappa_n(\sigma)(A_j) = \sum_j \lim_n \kappa_n(\sigma)(A_j) \\ &= \sum_j \sup_n \kappa_n(\sigma)(A_j) = \sum_j \kappa(\sigma)(A_j). \end{aligned}$$

The exchange between \sup and \lim in the derivation uses the fact that the sequence involved is increasing. The third equality follows from the countable additivity of $\kappa_n(\sigma)$, and the fourth from the monotone convergence theorem applied to the counting measure on the set of natural numbers. Finally,

$$\kappa(\sigma)(State \times [0, \infty)) = \sup_n \kappa_n(\sigma)(State \times [0, \infty)) \leq 1.$$

We have just shown that $\kappa(\sigma)$ is a subprobability measure.

To show the measurability of κ , we use the fact that a map f from a measurable space (X, Σ) to $\text{Sp}(State \times [0, \infty))$ is measurable if and only if for all measurable subsets A of $State \times [0, \infty)$ and reals $r \in \mathbb{R}$,

$$f^{-1}(\{\mu \mid \mu(A) \leq r\}) \in \Sigma.$$

Pick a measurable subset A of $State \times [0, \infty)$ and a real $r \in \mathbb{R}$. Then,

$$\kappa^{-1}(\{\mu \mid \mu(A) \leq r\}) = \{\sigma \mid \kappa(\sigma)(A) \leq r\}$$

$$\begin{aligned}
&= \{\sigma \mid \sup_n \kappa_n(\sigma)(A) \leq r\} \\
&= \{\sigma \mid \kappa_n(\sigma)(A) \leq r \text{ for all } n\} \\
&= \bigcap_n \{\sigma \mid \kappa_n(\sigma)(A)\}.
\end{aligned}$$

But the intersection at the end of this derivation gives a measurable set because each $\{\sigma \mid \kappa_n(\sigma)(A)\}$ is measurable and the countable intersection preserves measurability. \square

LEMMA A.1. *Let $\{\kappa_n\}_n$ be an ω -chain in \mathcal{K} . For all non-negative measurable functions $h : \text{State} \times [0, \infty) \rightarrow [0, \infty)$ and all states σ ,*

$$\int \left(\bigcup_n \kappa_n \right) (\sigma)(d(\sigma', w')) h(\sigma', w') = \lim_n \int \kappa_n(\sigma)(d(\sigma', w')) h(\sigma', w').$$

PROOF. We prove it by a simpler version of the so called monotone class theorem. This version says that in order to show the claimed equality for all non-negative measurable h , it suffices to show the following properties:

- The claimed equality holds if h is the indicator function $\lambda(\sigma, w). \mathbb{1}_{[(\sigma, w) \in A]}$ for a measurable subset A of $\text{State} \times [0, \infty)$.
- If non-negative measurable h_1 and h_2 satisfy the claimed equality and r_1, r_2 are non-negative real numbers, the linear combination $r_1 \cdot h_1 + r_2 \cdot h_2$ also satisfies the equality.
- If $\{h_m\}_m$ is a countable family of non-negative measurable functions such that each h_m satisfies the claimed equality and $h_m \leq h_{m+1}$ for every m , then $\sup_m h_m$ also satisfies the claimed equality.

We will show these properties one by one.

Let us start with the proof of the first property. Pick a measurable subset A of $\text{State} \times [0, \infty)$. Then,

$$\begin{aligned}
\int \left(\bigcup_n \kappa_n \right) (\sigma)(d(\sigma', w')) \mathbb{1}_{[(\sigma', w') \in A]} &= \left(\bigcup_n \kappa_n \right) (\sigma)(A) = \sup_n (\kappa_n(\sigma)(A)) = \lim_n (\kappa_n(\sigma)(A)) \\
&= \lim_n \int \kappa_n(\sigma)(d(\sigma', w')) \mathbb{1}_{[(\sigma', w') \in A]}.
\end{aligned}$$

The first and the last equalities use the definition of integration, the second the characterisation of the least upper bound in \mathcal{K} , and the third the fact that $\{\kappa_n(\sigma)(A)\}_n$ is increasing.

For the second property, assume that the claimed equality holds for non-negative measurable functions h_1 and h_2 . Let r_1 and r_2 be non-negative real numbers. Then,

$$\begin{aligned}
&\int \left(\bigcup_n \kappa_n \right) (\sigma)(d(\sigma', w')) (r_1 h_1(\sigma', w') + r_2 h_2(\sigma', w')) \\
&= r_1 \int \left(\bigcup_n \kappa_n \right) (\sigma)(d(\sigma', w')) h_1(\sigma', w') + r_2 \int \left(\bigcup_n \kappa_n \right) (\sigma)(d(\sigma', w')) h_2(\sigma', w') \\
&= r_1 \lim_n \int \kappa_n(\sigma)(d(\sigma', w')) h_1(\sigma', w') + r_2 \lim_n \int \kappa_n(\sigma)(d(\sigma', w')) h_2(\sigma', w') \\
&= \lim_n \left(r_1 \int \kappa_n(\sigma)(d(\sigma', w')) h_1(\sigma', w') + r_2 \int \kappa_n(\sigma)(d(\sigma', w')) h_2(\sigma', w') \right) \\
&= \lim_n \int \kappa_n(\sigma)(d(\sigma', w')) (r_1 h_1(\sigma', w') + r_2 h_2(\sigma', w')).
\end{aligned}$$

The first and the last equalities use the linearity of integration, the second the assumption on h_1 and h_2 , and the third the continuity of constant multiplication and addition.

For the final third property, consider an increasing sequence $\{h_m\}_m$ in the property. Then,

$$\begin{aligned}
 \int \left(\bigsqcup_n \kappa_n \right) (\sigma) (d(\sigma', w')) \left(\sup_m h_m (\sigma', w') \right) &= \int \left(\bigsqcup_n \kappa_n \right) (\sigma) (d(\sigma', w')) \sup_m (h_m (\sigma', w')) \\
 &= \int \left(\bigsqcup_n \kappa_n \right) (\sigma) (d(\sigma', w')) \lim_m (h_m (\sigma', w')) \\
 &= \lim_m \int \left(\bigsqcup_n \kappa_n \right) (\sigma) (d(\sigma', w')) h_m (\sigma', w') \\
 &= \lim_m \lim_n \int \kappa_n (\sigma) (d(\sigma', w')) h_m (\sigma', w') \\
 &= \lim_n \lim_m \int \kappa_n (\sigma) (d(\sigma', w')) h_m (\sigma', w').
 \end{aligned}$$

The third equality uses the monotone convergence theorem, and the fourth the assumption on the h_m . \square

THEOREM 4.3. *For every command C , its interpretation $\llbracket C \rrbracket$ is well-defined and it belongs to \mathcal{K} .*

PROOF. We prove the theorem by induction on the structure of C . In the proof, we use concepts and results from the measure theory that go beyond what we reviewed in §3.

We start with the case $C \equiv (x := E)$. For every $\sigma \in \text{State}$, $\llbracket x := E \rrbracket (\sigma)$ is the Dirac measure at $(\sigma[x \mapsto \llbracket E \rrbracket \sigma_s], 1)$. Thus, it suffices to show the measurability of the following function F for every measurable subset A :

$$F(\sigma) = \mathbb{1}_{[(\sigma[x \mapsto \llbracket E \rrbracket \sigma_s], 1) \in A]}.$$

This holds because the functions

$$(\sigma, w) \mapsto \mathbb{1}_{[(\sigma, w) \in A]} : \text{State} \times [0, \infty) \rightarrow [0, 1], \quad \sigma \mapsto (\sigma[x \mapsto \llbracket E \rrbracket \sigma_s], 1) : \text{State} \rightarrow \text{State} \times [0, \infty)$$

are both measurable.

The case $C \equiv \text{skip}$ follows from our argument for the previous case $C \equiv (x := E)$, because $\llbracket \text{skip} \rrbracket = \llbracket x := x \rrbracket$.

Next we consider sequential composition. Assume that $C \equiv (C_0; C_1)$. By induction hypothesis, we know that $\llbracket C_0 \rrbracket$ and $\llbracket C_1 \rrbracket$ are subprobability kernels. Then, for all σ and A ,

$$\begin{aligned}
 \llbracket C_0; C_1 \rrbracket (\sigma)(A) &= \int \llbracket C_0 \rrbracket (\sigma)(d(\sigma', w')) \int \llbracket C_1 \rrbracket (\sigma')(d(\sigma'', w'')) \mathbb{1}_{[(\sigma'', w' \cdot w'') \in A]} \\
 &\leq \int \llbracket C_0 \rrbracket (\sigma)(d(\sigma', w')) \leq 1.
 \end{aligned}$$

Furthermore, it is not difficult to show that $\llbracket C_0; C_1 \rrbracket (\sigma)$ is a measure. The measurability of $\llbracket C_0; C_1 \rrbracket$ follows from three facts. First, it suffices to show that for all measurable subsets A , the function

$$\sigma \mapsto \llbracket C_0; C_1 \rrbracket (\sigma)(d(\sigma', w'))(A)$$

is measurable. Second, for every measurable f and every subprobability kernel κ , the function $x \mapsto \int \kappa(x)(dy)f(y)$ is measurable. Second, the functions $(\sigma, w) \mapsto \llbracket C_1 \rrbracket (\sigma) \otimes \delta_w$ and $((\sigma'', w''), w) \mapsto (\sigma'', w' \cdot w'')$ are measurable. Here δ_w is the Dirac measure at w and \otimes is the operator for product measure.

We move on to the case of $C \equiv (\text{if } B \{C_0\} \text{ else } \{C_1\})$. We first show that the function $\llbracket C \rrbracket$ is a well-defined function from State to $\text{Sp}(\text{State} \times [0, \infty))$, that is, $\llbracket C \rrbracket (\sigma)$ is a subprobability measure

for every $\sigma \in \text{State}$. The non-negativity and countable additivity of $\llbracket C \rrbracket(\sigma)$ follow from the same properties of $\llbracket C_0 \rrbracket(\sigma)$ and $\llbracket C_1 \rrbracket(\sigma)$. Also,

$$\begin{aligned} \llbracket C \rrbracket(\sigma)(A) &= \llbracket \text{if } B \{C_0\} \text{ else } \{C_1\} \rrbracket(\sigma)(A) \\ &= \mathbb{1}_{\llbracket B \rrbracket \sigma_s = \text{true}} \cdot \llbracket C_0 \rrbracket(\sigma)(A) + \mathbb{1}_{\llbracket B \rrbracket \sigma_s \neq \text{true}} \cdot \llbracket C_1 \rrbracket(\sigma)(A) \\ &\leq \mathbb{1}_{\llbracket B \rrbracket \sigma_s = \text{true}} \cdot 1 + \mathbb{1}_{\llbracket B \rrbracket \sigma_s \neq \text{true}} \cdot 1 = 1. \end{aligned}$$

It remains to show that $\llbracket C \rrbracket$ is measurable. This boils down to showing that

$$\{\sigma \mid \mathbb{1}_{\llbracket B \rrbracket \sigma_s = \text{true}} \cdot \llbracket C_0 \rrbracket(\sigma)(A) + \mathbb{1}_{\llbracket B \rrbracket \sigma_s \neq \text{true}} \cdot \llbracket C_1 \rrbracket(\sigma)(A) \leq r\}$$

is measurable for all $r \in \mathbb{R}$ and measurable subset A of $\text{State} \times [0, \infty)$. But this follows easily from the induction hypothesis and the measurability of $\llbracket B \rrbracket$.

The next case is $C \equiv (\text{while } B \{C_0\})$. Recall Lemma 4.2, which says that the set of subprobability kernels from State to $\text{State} \times [0, \infty)$, denoted by \mathcal{K} , is an ω -cpo when given the pointwise order \sqsubseteq . Let F be the function on \mathcal{K} used in the semantics of C . That is, for all $\kappa \in \mathcal{K}$, $\sigma \in \text{State}$ and measurable $A \subseteq \text{Store} \times [0, \infty)$,

$$\begin{aligned} F(\kappa)(\sigma)(A) &\triangleq \mathbb{1}_{\llbracket B \rrbracket \sigma_s \neq \text{true}} \cdot \mathbb{1}_{[(\sigma, 1) \in A]} \\ &\quad + \mathbb{1}_{\llbracket B \rrbracket \sigma_s = \text{true}} \cdot \int \llbracket C_0 \rrbracket(\sigma)(d(\sigma', w')) \int \kappa(\sigma')(d(\sigma'', w'')) \mathbb{1}_{[(\sigma'', w' \cdot w'') \in A]}. \end{aligned}$$

It suffices to show that F is a continuous function on \mathcal{K} . The arguments used to handle the if and sequencing cases show that $F(\kappa)$ is a subprobability kernel. The monotonicity of F follows from the fact that the integration is monotone with respect to the measure and its integrand. It remains to show that F preserves the limits of ω -chains. Consider an ω -chain $\{\kappa_n\}_n$ of subprobability kernels in \mathcal{K} . Then, the least upper bound of this ω -chain is $\kappa = \bigsqcup_n \kappa_n$. The following calculation shows that F preserves this limit.

$$\begin{aligned} F(\kappa)(\sigma)(A) &= \mathbb{1}_{\llbracket B \rrbracket \sigma_s \neq \text{true}} \cdot \mathbb{1}_{[(\sigma, 1) \in A]} \\ &\quad + \mathbb{1}_{\llbracket B \rrbracket \sigma_s = \text{true}} \cdot \int \llbracket C_0 \rrbracket(\sigma)(d(\sigma', w')) \int \kappa(\sigma')(d(\sigma'', w'')) \mathbb{1}_{[(\sigma'', w' \cdot w'') \in A]} \\ &= \mathbb{1}_{\llbracket B \rrbracket \sigma_s \neq \text{true}} \cdot \mathbb{1}_{[(\sigma, 1) \in A]} \\ &\quad + \mathbb{1}_{\llbracket B \rrbracket \sigma_s = \text{true}} \cdot \int \llbracket C_0 \rrbracket(\sigma)(d(\sigma', w')) \int \bigsqcup_n \kappa_n(\sigma')(d(\sigma'', w'')) \mathbb{1}_{[(\sigma'', w' \cdot w'') \in A]} \\ &= \mathbb{1}_{\llbracket B \rrbracket \sigma_s \neq \text{true}} \cdot \mathbb{1}_{[(\sigma, 1) \in A]} \\ &\quad + \mathbb{1}_{\llbracket B \rrbracket \sigma_s = \text{true}} \cdot \int \llbracket C_0 \rrbracket(\sigma)(d(\sigma', w')) \lim_n \int \kappa_n(\sigma')(d(\sigma'', w'')) \mathbb{1}_{[(\sigma'', w' \cdot w'') \in A]} \\ &= \mathbb{1}_{\llbracket B \rrbracket \sigma_s \neq \text{true}} \cdot \mathbb{1}_{[(\sigma, 1) \in A]} \\ &\quad + \mathbb{1}_{\llbracket B \rrbracket \sigma_s = \text{true}} \cdot \lim_n \int \llbracket C_0 \rrbracket(\sigma)(d(\sigma', w')) \int \kappa_n(\sigma')(d(\sigma'', w'')) \mathbb{1}_{[(\sigma'', w' \cdot w'') \in A]} \\ &= \lim_n \left(\mathbb{1}_{\llbracket B \rrbracket \sigma_s \neq \text{true}} \cdot \mathbb{1}_{[(\sigma, 1) \in A]} \right. \\ &\quad \left. + \mathbb{1}_{\llbracket B \rrbracket \sigma_s = \text{true}} \cdot \int \llbracket C_0 \rrbracket(\sigma)(d(\sigma', w')) \int \kappa_n(\sigma')(d(\sigma'', w'')) \mathbb{1}_{[(\sigma'', w' \cdot w'') \in A]} \right) \\ &= \lim_n F(\kappa_n)(\sigma)(A) = \left(\bigsqcup_n F(\kappa_n) \right)(\sigma)(A). \end{aligned}$$

The third equality uses the continuity of integration with respect to the integrating measure and the non-negative integrand (Lemma A.1), and the fourth equality uses the monotone convergence theorem.

The next case is $C \equiv (x := \text{sample}_{\text{norm}}(S, E_1, E_2))$. The requirement that

$$\llbracket x := \text{sample}_{\text{norm}}(S, E_1, E_2) \rrbracket(\sigma)$$

is a measure for every σ follows immediately from the semantic equation for this sample statement, which says that $\llbracket x := \text{sample}_{\text{norm}}(S, E_1, E_2) \rrbracket(\sigma)$ is either the constant-0 measure or the pushforward of a normal distribution with mean $\llbracket E_1 \rrbracket_{\sigma_s}$ and standard deviation $\llbracket E_2 \rrbracket_{\sigma_s}$ by some measurable function. In fact, the following calculation shows that $\llbracket x := \text{sample}_{\text{norm}}(S, E_1, E_2) \rrbracket$ is a subprobability measure:

$$\begin{aligned} & \llbracket x := \text{sample}_{\text{norm}}(S, E_1, E_2) \rrbracket(\sigma)(A) \\ &= \mathbb{1}[\llbracket S \rrbracket_{\sigma_s} \notin \text{dom}(\sigma_r)] \cdot \mathbb{1}[\llbracket E_2 \rrbracket_{\sigma_s} \notin (0, \infty)] \cdot \int dv \left(\mathcal{N}(v; \llbracket E_1 \rrbracket_{\sigma_s}, \llbracket E_2 \rrbracket_{\sigma_s}) \cdot \mathbb{1}_{[(\sigma_s[x \mapsto v], \sigma_r[\llbracket E_2 \rrbracket_{\sigma_s} \mapsto v]), 1) \in A]} \right) \\ &\leq \mathbb{1}[\llbracket S \rrbracket_{\sigma_s} \notin \text{dom}(\sigma_r)] \cdot \mathbb{1}[\llbracket E_2 \rrbracket_{\sigma_s} \notin (0, \infty)] \cdot 1 \\ &\leq 1. \end{aligned}$$

Also, since all of \mathcal{N} , $\llbracket S \rrbracket$, $\llbracket E_0 \rrbracket$ and $\llbracket E_1 \rrbracket$ are measurable,

$$\{\sigma \mid \llbracket x := \text{sample}_{\text{norm}}(S, E_1, E_2) \rrbracket(\sigma)(A) \leq r\}$$

is a measurable subset of *State* for all A and r . This means that $\llbracket x := \text{sample}_{\text{norm}}(S, E_1, E_2) \rrbracket$ is a measurable function. From what we have proved, it follows that $\llbracket x := \text{sample}_{\text{norm}}(S, E_1, E_2) \rrbracket$ is a subprobability kernel as required.

The last case is $C \equiv \text{score}_{\text{norm}}(E_0, E_1, E_2)$. Because of the semantic equation for this score statement, $\llbracket \text{score}_{\text{norm}}(E_0, E_1, E_2) \rrbracket(\sigma)$ is a measure for all σ . Furthermore, $\llbracket \text{score}_{\text{norm}}(E_0, E_1, E_2) \rrbracket(\sigma)(A)$ is the product of two indicator functions, and so it is at most 1. That is, $\llbracket \text{score}_{\text{norm}}(E_0, E_1, E_2) \rrbracket(\sigma)$ is a subprobability measure. By the semantic equation for the score statement again, the measurability of the normal density and the measurability of E_0, E_1 and E_2 , the set $\{\sigma \mid \llbracket \text{score}_{\text{norm}}(E_0, E_1, E_2) \rrbracket(\sigma)(A) \leq r\}$ is measurable for all A and r . This measurability implies that $\llbracket \text{score}_{\text{norm}}(E_0, E_1, E_2) \rrbracket$ is a measurable function. By putting all these together, we can derive the required fact that $\llbracket \text{score}_{\text{norm}}(E_0, E_1, E_2) \rrbracket$ is a subprobability kernel. \square

LEMMA 4.4. *For every measurable $h : \text{RDB} \times \text{RDB} \times \text{RDB} \rightarrow \mathbb{R}$, the function $(r, r') \mapsto \mathbb{1}_{[r \# r']} \times h(r, r', r \uplus r')$ from $\text{RDB} \times \text{RDB}$ to \mathbb{R} is measurable.*

PROOF. Let $*$: $\text{RDB} \times \text{RDB} \rightarrow \text{RDB}$ be the following extension of the partial function \uplus : $\text{RDB} \times \text{RDB} \rightarrow_{\text{partial}} \text{RDB}$ to a total function:

$$r * r' \triangleq \begin{cases} r \uplus r' & \text{if } r \# r', \\ [] & \text{otherwise.} \end{cases}$$

Then, for all r, r' ,

$$\mathbb{1}_{[r \# r']} \cdot h(r, r', r \uplus r') = \mathbb{1}_{[r \# r']} \cdot h(r, r', r * r').$$

We will show that $(r, r') \mapsto \mathbb{1}_{[r \# r']}$ and $(r, r') \mapsto r * r'$ are measurable. This is enough because the function on the RHS of the above equation (namely, $(r, r') \mapsto \mathbb{1}_{[r \# r']} \cdot h(r, r', r * r')$) is constructed by applying pairing, composition and multiplication operators on measurable functions and so it is measurable.

Pick a real number $v \in \mathbb{R}$. We should show that $B = \{(r, r') \mid \mathbb{1}_{[r \# r']} \leq v\}$ is measurable. If $v \geq 1$, the set B is $RDB \times RDB$, and so it is measurable. If $v < 0$, then B is the empty set, so that it is measurable as well. If $0 \leq v < 1$, then

$$\{(r, r') \mid \mathbb{1}_{[r \# r']} \leq v\} = \bigcup_{K \subseteq_{fin} Str} \bigcup_{\substack{K' \subseteq_{fin} Str \\ K \cap K' \neq \emptyset}} [K \rightarrow \mathbb{R}] \times [K' \rightarrow \mathbb{R}],$$

which is measurable because the RHS of the above equation is the countable union of measurable sets.

Pick a measurable subset A of RDB . We have to prove that $B = \{(r, r') \mid r * r' \in A\}$ is measurable. If $A = \{\square\}$, then $B = \{(r, r') \mid \mathbb{1}_{[r \# r']} \leq 0.5\} \cup \{(\square, \square)\}$, so that it is measurable. If $\square \notin A$, then

$$\begin{aligned} \{(r, r') \mid r * r' \in A\} = \\ \bigcup_{K \subseteq_{fin} Str} \bigcup_{\substack{K' \subseteq_{fin} Str \\ K \cap K' = \emptyset}} \{(r, r') \in [K \rightarrow \mathbb{R}] \times [K' \rightarrow \mathbb{R}] \mid r \uplus r' \in \{r'' \in A \mid \text{dom}(r'') = K \cup K'\}\}. \end{aligned}$$

But $\{r'' \in A \mid \text{dom}(r'') = K \cup K'\}$ is measurable, and the set on the RHS of the above equation is the countable union of measurable sets. Thus, B is measurable in this case as well. The remaining case is that A contains \square and some other element. We can prove this case by splitting A into $\{\square\}$ and $A \setminus \{\square\}$ and using what we proved in the previous two cases. \square

LEMMA 4.5. *(\mathcal{D}, \sqsubseteq) is an ω -complete partial order and has the least element $a \mapsto \perp$. Thus, every continuous function G on \mathcal{D} has the least fixed point.*

PROOF. All constant functions from a measurable space to another are measurable. Thus, the constant function $a \mapsto \perp$ is measurable. It is immediate that this constant function is local and smaller than any other functions in \mathcal{D} .

Let $g_0 \subseteq g_1 \subseteq g_2 \subseteq \dots$ be an ω -chain. We show that the following function is the least upper bound of the sequence in \mathcal{D} : for all s, r ,

$$\left(\bigsqcup_i g_i\right)(s, r) = \begin{cases} \perp & \text{if } g_i(s, r) = \perp \text{ for all } i, \\ c \neq \perp & \text{if } g_i(s, r) = c \neq \perp \text{ for some } i. \end{cases}$$

By definition, this function is the least upper bound in \mathcal{D} that we are looking for, if it is measurable and local. To check the locality, suppose that $(\bigsqcup_i g_i)(s, r) = (s', r', w', p')$. Then, there exists some i such that $g_i(s, r) = (s', r', w', p')$. By the locality of g_i , we have that

$$\begin{aligned} (\exists r''. r' \# r'' \wedge r = r' \uplus r'' \wedge g_i(s, r'') = (s', \square, w', p')) \\ \wedge (\forall r''. r \# r'' \implies g_i(s, r \uplus r'') = (s', r' \uplus r'', w', p')). \end{aligned}$$

Thus, the definition of $\bigsqcup_i g_i$ implies that the above property holds when we replace g_i by $\bigsqcup_i g_i$. We have just shown that $\bigsqcup_i g_i$ is local.

It remains to show that $\bigsqcup_i g_i$ is measurable. Let $A \subseteq \{\perp\} \cup (\text{Store} \times RDB \times [0, \infty) \times [0, \infty))$ be a measurable set. We prove that $(\bigsqcup_i g_i)^{-1}(A)$ is a measurable set.

- Case $A = \{\perp\}$:

$$\begin{aligned} (s, r) \in \left(\bigsqcup_i g_i\right)^{-1}(\{\perp\}) &\iff g_i(s, r) = \perp \text{ for all } i \iff (s, r) \in g_i^{-1}(\{\perp\}) \text{ for all } i \\ &\iff (s, r) \in \bigcap_i g_i^{-1}(\{\perp\}). \end{aligned}$$

The intersection of countably many measurable sets are measurable, and g_0, g_1, \dots are measurable functions. Hence we get the measurability of $(\bigsqcup_i g_i)^{-1}(\{\perp\})$.

- Case $\perp \notin A$:

$$\begin{aligned} (s, r) \in \left(\bigsqcup_i g_i \right)^{-1}(A) &\iff g_i(s, r) \in A \text{ for some } i \iff (s, r) \in g_i^{-1}(A) \text{ for some } i \\ &\iff (s, r) \in \bigcup_i g_i^{-1}(A). \end{aligned}$$

The first equivalence follows from the definition of the order \sqsubseteq . The union of measurable sets are measurable, and g_0, g_1, \dots are measurable functions. Thus, the desired result follows.

- Case $A = \{\perp\} \cup A'$ with $\perp \notin A'$:

Then A' is also measurable as $\{\perp\}$ is and the set subtraction preserves measurability. Moreover $(\bigsqcup_i g_i)^{-1}(A) = (\bigsqcup_i g_i)^{-1}(\{\perp\}) \cup (\bigsqcup_i g_i)^{-1}(A')$, and the union of countably many measurable sets is measurable, and so we can apply the two cases above. \square

The ω -cpo \mathcal{D} permits an operator for composing functions in it. We say that a function

$$g : \left(\{\perp\} \cup (Store \times RDB \times [0, \infty) \times [0, \infty)) \right) \rightarrow \left(\{\perp\} \cup (Store \times RDB \times [0, \infty) \times [0, \infty)) \right)$$

is *local* if for all $(s, r, w, p), (s', r', w', p') \in Store \times RDB \times [0, \infty) \times [0, \infty)$,

$$\begin{aligned} g(s, r, w, p) = (s', r', w', p') &\implies (\exists r''. r' \# r'' \wedge r = r' \uplus r'' \wedge g(s, r'', w, p) = (s', [], w', p')) \\ &\wedge (\forall r''. r \# r'' \implies g(s, r \uplus r'', w, p) = (s', r' \uplus r'', w', p')). \end{aligned}$$

We say that g is *strict* if $g(\perp) = \perp$. Let

$$\mathcal{D}^\ddagger \triangleq \left\{ g : \left(\{\perp\} \cup (Store \times RDB \times [0, \infty) \times [0, \infty)) \right) \rightarrow \left(\{\perp\} \cup (Store \times RDB \times [0, \infty) \times [0, \infty)) \right) \mid g \text{ is measurable, local and strict} \right\}.$$

Define a partial order \sqsubseteq on \mathcal{D}^\ddagger by

$$g \sqsubseteq g' \iff \forall a \in \{\perp\} \cup (Store \times RDB \times [0, \infty) \times [0, \infty)). (g(a) = \perp \vee g(a) = g'(a)).$$

LEMMA A.2. $(\mathcal{D}^\ddagger, \sqsubseteq)$ is an ω -complete partial order and has the least element.

PROOF. Let

$$\mathcal{E} \triangleq \left\{ g : \left(Store \times RDB \times [0, \infty) \times [0, \infty) \right) \rightarrow \left(\{\perp\} \cup (Store \times RDB \times [0, \infty) \times [0, \infty)) \right) \mid g \text{ is measurable and local} \right\}.$$

Essentially by the same argument as the one used in the proof of Lemma 4.5, the set \mathcal{E} with the pointwise order \sqsubseteq is an ω -cpo and has the least element. But $(\mathcal{D}^\ddagger, \sqsubseteq)$ and $(\mathcal{E}, \sqsubseteq)$ are isomorphic as partially order sets. Thus, $(\mathcal{D}^\ddagger, \sqsubseteq)$ has the property claimed by the lemma. \square

The following lemma says that the lifting g^\ddagger of $g \in \mathcal{D}$ in §4.3 is a well-defined function from \mathcal{D} to this new domain.

LEMMA A.3. The function $g \mapsto g^\ddagger$ from \mathcal{D} to \mathcal{D}^\ddagger is well-defined and continuous.

PROOF. We first show that the function is well-defined. Let $g \in \mathcal{D}$. Then, $g^\ddagger(\perp) = \perp$ by the definition of g^\ddagger . The measurability of g^\ddagger follows from the measurability of g , projection, and multiplication and the fact that both the pairing of measurable functions (into a function to a product measurable space) and the function composition preserve measurability. The locality of g^\ddagger follows from that of g .

Next, we show the claimed continuity. The monotonicity is an immediate consequence of the definitions of the $-\ddagger$ operator and the orders on \mathcal{D} and \mathcal{D}^\ddagger . It remains to show the limit preservation. Let $g_0 \sqsubseteq g_1 \sqsubseteq g_2 \sqsubseteq \dots$ be an ω -chain in \mathcal{D} . Then, by the monotonicity of $-\ddagger$, $\{g_n^\ddagger\}_n$ is also an ω -chain in \mathcal{D}^\ddagger . We should show that

$$\left(\bigsqcup_n g_n\right)^\ddagger(a) = \left(\bigsqcup_n g_n^\ddagger\right)(a)$$

for all a from $(\{\perp\} \cup \text{Store} \times \text{RDB} \times [0, \infty) \times [0, \infty))$. Pick such an a . If $a = \perp$, the required holds because both sides become \perp . If $a \neq \perp$ but $g_n(a) = \perp$ for all n , then again both sides of the equality are \perp , and so the equality holds. If neither of these two cases hold, there exists n such that $g_n(a) \neq \perp$ and $g_n(a) = g_m(a)$ for all $m \geq n$. Let $(s', r', w', p') = g_n(a)$, and $(s, r) = a$. Then,

$$\left(\bigsqcup_n g_n\right)^\ddagger(a) = (s', r', w \times w', p \times p') = \left(\bigsqcup_n g_n^\ddagger\right)(a).$$

□

LEMMA 4.6. *For every command C , its semantics $\llbracket C \rrbracket_d$ is well-defined and belongs to \mathcal{D} .*

PROOF. The proof that $\llbracket C \rrbracket_d$ is well-defined and measurable proceeds by induction on the structure of commands C .

- Case of $\llbracket \text{skip} \rrbracket_d$:

The semantics is clearly well-defined in this case. Note that $\llbracket \text{skip} \rrbracket_d$ is essentially the pairing of the identity function and a constant function. Since both of these functions are measurable and the pairing preserves measurability, $\llbracket \text{skip} \rrbracket_d$ is measurable. Also, since $\llbracket \text{skip} \rrbracket_d$ neither changes nor depends on the r component, it is local.

- Case of $\llbracket x := E \rrbracket_d$:

The proof is identical to the previous case except that instead of the identity function, we should use the function $(s, r) \mapsto (s[x \mapsto \llbracket E \rrbracket s], r)$, which is measurable because it can be constructed by composing and pairing three measurable functions, $\llbracket E \rrbracket$, $(s, v) \mapsto s[x \mapsto v]$, and projection.

- Case of $\llbracket \text{if } B \{C_0\} \text{ else } \{C_1\} \rrbracket_d$:

The well-definedness and the locality are immediate consequences of the induction hypothesis on $\llbracket C_0 \rrbracket_d$ and $\llbracket C_1 \rrbracket_d$. Let $A \subseteq \{\perp\} \cup \text{Store} \times \text{RDB} \times [0, \infty) \times [0, \infty)$ be a measurable set. Then,

$$\begin{aligned} & \llbracket \text{if } B \{C_0\} \text{ else } \{C_1\} \rrbracket_d^{-1}(A) \\ &= \left(\{(s, r) \mid \llbracket B \rrbracket s = \text{true}\} \cap \llbracket C_0 \rrbracket_d^{-1}(A) \right) \cup \left(\{(s, r) \mid \llbracket B \rrbracket s \neq \text{true}\} \cap \llbracket C_1 \rrbracket_d^{-1}(A) \right). \end{aligned}$$

Moreover, $\{\text{true}\}$ is measurable, and so is its complement. Since $\llbracket B \rrbracket$ is a measurable function, $\{(s, r) \mid \llbracket B \rrbracket s = \text{true}\}$ and $\{(s, r) \mid \llbracket B \rrbracket s \neq \text{true}\}$ are measurable. As a result the above set is measurable. So, $\llbracket \text{if } B \{C_0\} \text{ else } \{C_1\} \rrbracket_d$ is well-defined, local and measurable as claimed.

- Case of $\llbracket C_0; C_1 \rrbracket_d$:

By induction hypothesis, both $\llbracket C_0 \rrbracket_d$ and $\llbracket C_1 \rrbracket_d$ are well-defined, local and measurable. This and Lemma A.3 imply that $\llbracket C_1 \rrbracket_d^\ddagger$ is also local and measurable. The function $\llbracket C_0; C_1 \rrbracket_d$ is nothing but the composition of two well-defined measurable functions $\llbracket C_1 \rrbracket_d^\ddagger$ and $\llbracket C_0 \rrbracket_d$, and

so it satisfies the claimed well-definedness and measurability. To see the locality of $\llbracket C_0; C_1 \rrbracket_d$, suppose that

$$(\llbracket C_1 \rrbracket_d^\ddagger \circ \llbracket C_0 \rrbracket_d)(s, r) = (s'', r'', w'', p'').$$

Then, there exist s', r', w', p' such that

$$\llbracket C_0 \rrbracket_d(s, r) = (s', r', w', p') \wedge \llbracket C_1 \rrbracket_d^\ddagger(s', r', w', p') = (s'', r'', w'', p'').$$

By the locality of $\llbracket C_0 \rrbracket_d$ and $\llbracket C_1 \rrbracket_d^\ddagger$, there exist r'_0 and r''_0 such that

$$\begin{aligned} r'_0 \# r' \wedge r'_0 \uplus r' &= r \wedge \llbracket C_0 \rrbracket_d(s, r'_0) = (s', [], w', p') \\ \wedge r''_0 \# r'' \wedge r''_0 \uplus r'' &= r' \wedge \llbracket C_1 \rrbracket_d^\ddagger(s', r''_0, w', p') = (s'', [], w'', p''). \end{aligned}$$

Note that $r'_0 \# r''_0$. Let $r_0 \triangleq r'_0 \uplus r''_0$. Then,

$$r_0 \# r'' \wedge r_0 \uplus r'' = r \wedge (\llbracket C_1 \rrbracket_d^\ddagger \circ \llbracket C_0 \rrbracket_d)(s, r_0) = \llbracket C_1 \rrbracket_d^\ddagger(s', r''_0, w', p') = (s'', [], w'', p'').$$

The first equality from above uses the locality of $\llbracket C_0 \rrbracket_d$. Thus, the first conjunct in the definition of locality holds if we use r_0 as a witness in the conjunct. To prove the second in the definition, consider r_1 such that $r_1 \# r$. Then, $r_1 \# r'$. Thus, by the locality of $\llbracket C_0 \rrbracket_d$ and $\llbracket C_1 \rrbracket_d^\ddagger$, we have

$$(\llbracket C_1 \rrbracket_d^\ddagger \circ \llbracket C_0 \rrbracket_d)(s, r \uplus r_1) = \llbracket C_1 \rrbracket_d^\ddagger(s', r' \uplus r_1, w', p') = (s'', r'' \uplus r_1, w'', p'').$$

- Case of $\llbracket \text{while } B \{C\} \rrbracket_d$:
Since \mathcal{D} is an ω -cpo and has the least element, we simply need to prove that G in the definition of $\llbracket \text{while } B \{C\} \rrbracket_d$ is well-defined and continuous so that Kleene's fixed point theorem applies (Lemma 4.5). By induction hypothesis, $\llbracket C \rrbracket_d$ is well-defined, local, and measurable. Thus, G is a well-defined function from \mathcal{D} to \mathcal{D} ; the proof that the image of G consists of local measurable functions is similar to the case of the if statement extended with the argument with the case of sequencing. Furthermore, G is monotone because both the lifting $-\ddagger$ and the function composition are monotone. Thus, G transforms an ω -chain into an ω -chain. Finally, G preserves the least upper bound of such an ω -chain. This is because both the lifting $-\ddagger$ and the function composition preserve the least upper bound. We have just shown that G is well-defined and continuous.

- Case of $\llbracket x := \text{sample}_{\text{norm}}(S, E_1, E_2) \rrbracket_d$:
The well-definedness and the locality are immediate. To prove the measurability, pick a measurable subset A of $\{\perp\} \cup (\text{Store} \times \text{RDB} \times [0, \infty) \times [0, \infty)$. Then,

$$\{(s, r) \mid \llbracket x := \text{sample}_{\text{norm}}(S, E_1, E_2) \rrbracket_d(s, r) \in A\} = A_0 \cup A_1 \cup A_2 \cap A_3$$

where

$$\begin{aligned} A_0 &= \{(s, r) \mid \perp \in A \wedge \llbracket S \rrbracket s \notin \text{dom}(r)\}, \\ A_1 &= \{(s, r) \mid \perp \in A \wedge \llbracket E_2 \rrbracket s \notin (0, \infty)\}, \\ A_2 &= \{(s, r) \mid \llbracket S \rrbracket s \in \text{dom}(r) \wedge \llbracket E_2 \rrbracket s \in (0, \infty)\}, \\ A_3 &= A_2 \cap \{(s, r) \mid (s[x \mapsto r(\llbracket S \rrbracket s)], r \setminus \llbracket S \rrbracket s, 1, \mathcal{N}(r(\llbracket S \rrbracket s); \llbracket E_1 \rrbracket s, \llbracket E_2 \rrbracket s)) \in A\}. \end{aligned}$$

We show that all of A_0, A_1, A_2 and A_3 are measurable. The measurability of the first three sets follow from the measurability of the following two sets:

$$A'_0 \triangleq \{(s, r) \mid \llbracket E_2 \rrbracket s \in (0, \infty)\} \quad \text{and} \quad A'_1 \triangleq \{(s, r) \mid \llbracket S \rrbracket s \in \text{dom}(r)\}.$$

The set A'_0 is measurable because it is the inverse image of the composition of two measurable functions (namely, $\llbracket E_1 \rrbracket$ and projection) to the measurable set $(0, \infty)$. For the measurability of A'_1 , note that

$$\{(s, r) \mid \llbracket S \rrbracket s \in \text{dom}(r)\} = \bigcup_{K \subseteq_{\text{fin}} \text{Str}} \{(s, r) \mid \text{dom}(r) = K\} \cap \{(s, r) \mid \llbracket S \rrbracket s \in K\}.$$

The set on the RHS is measurable, because it is a countable union of the intersection of two measurable subsets; the measurability of $\{(s, r) \mid \text{dom}(r) = K\}$ comes from the fact that RDB is built by the disjoint-union construction over $[K \rightarrow \mathbb{R}]$ for all finite K , and the measurability of $\{(s, r) \mid \llbracket S \rrbracket s \in K\}$ follows from the measurability of the function $\llbracket S \rrbracket$ and the set K . It remains to show the measurability of A_3 . To do so, we note that A_3 is the inverse image of a function constructed by applying measurability-preserving operators on $\llbracket S \rrbracket$, $\llbracket E_1 \rrbracket$, $\llbracket E_2 \rrbracket$ and \mathcal{N} , and the application and restriction functions on random databases. Since the first four are measurable, it is sufficient to show the measurability of the last two, which we spell out below:

$$\begin{aligned} f_1 : RDB \times \text{Str} &\rightarrow \mathbb{R}, & f_1(r, \alpha) &\triangleq \begin{cases} r(\alpha) & \text{if } \alpha \in \text{dom}(r), \\ 0 & \text{otherwise,} \end{cases} \\ f_2 : RDB \times \text{Str} &\rightarrow RDB, & f_2(r, \alpha)(\beta) &\triangleq \begin{cases} \text{undefined} & \text{if } \alpha = \beta, \\ r(\beta) & \text{otherwise.} \end{cases} \end{aligned}$$

For any measurable subset A of \mathbb{R} ,

$$\begin{aligned} f_1^{-1}(A) = \bigcup_{\alpha \in \text{Str}} \bigcup_{K \subseteq_{\text{fin}} \text{Str}} &\left(\{(r, \beta) \mid \text{dom}(r) = K \wedge \beta = \alpha \wedge \alpha \notin K \wedge 0 \in A\} \right. \\ &\left. \cup \{(r, \beta) \mid \text{dom}(r) = K \wedge \beta = \alpha \wedge \alpha \in K \wedge r(\alpha) \in A\} \right). \end{aligned}$$

All the explicitly-defined sets on the RHS of this equation are measurable, and the RHS only uses finite or countable union. Thus, $f_1^{-1}(A)$ is measurable. Now consider a measurable subset A' of RDB . Then,

$$\begin{aligned} f_2^{-1}(A') = \bigcup_{\alpha \in \text{Str}} \bigcup_{K \subseteq_{\text{fin}} \text{Str}} &\left(\{(r, \beta) \mid \beta = \alpha \wedge \text{dom}(r) = K \wedge \alpha \notin K \wedge r \in A'\} \right. \\ &\left. \cup \{(r, \beta) \mid \beta = \alpha \wedge \text{dom}(r) = K \wedge \alpha \in K \wedge r \setminus \alpha \in A'\} \right). \end{aligned}$$

Again all the explicitly-defined sets on the RHS of this equation are measurable, and the RHS only uses finite or countable union. Thus, $f_2^{-1}(A')$ is measurable.

- Case of $\llbracket \text{score}_{\text{norm}}(E_0, E_1, E_2) \rrbracket_d$:

The well-definedness and the locality are immediate in this case. Pick a measurable set

$$A \subseteq \{\perp\} \cup (\text{Store} \times RDB \times [0, \infty) \times [0, \infty)).$$

Then,

$$\begin{aligned} \llbracket \text{score}_{\text{norm}}(E_0, E_1, E_2) \rrbracket_d^{-1}(A) = \{(s, r) \mid (s, r, \mathcal{N}(\llbracket E_0 \rrbracket s; \llbracket E_1 \rrbracket s, \llbracket E_2 \rrbracket s), 1) \in A\} \\ \cup \{(s, r) \mid \perp \in A \wedge \llbracket E_2 \rrbracket s \notin (0, \infty)\} \end{aligned}$$

Both sets in the union here are measurable because all of \mathcal{N} , $\llbracket E_0 \rrbracket$, $\llbracket E_1 \rrbracket$ and $\llbracket E_2 \rrbracket$ are measurable and the composition and the pairing operators on functions preserve measurability. Thus, the union itself is measurable. \square

LEMMA 4.7. For all $g \in \mathcal{D}$, the following functions from $\text{Store} \times \text{RDB}$ to \mathbb{R} and $\{\perp\} \cup \text{Store}$ are measurable: $(s, r) \mapsto \text{dens}(g, s)(r)$ and $(s, r) \mapsto \text{get}(g, s)(r)$.

PROOF. We have to show that for every real $v \in \mathbb{R}$ and measurable subset $A \subseteq \{\perp\} \cup \text{Store}$,

$$B_0 \triangleq \{(s, r) \mid \text{dens}(g, s)(r) \leq v\} \quad \text{and} \quad B_1 \triangleq \{(s, r) \mid \text{get}(g, s)(r) \in A\}$$

are measurable. The set B_0 is measurable because

$$\begin{aligned} & \{(s, r) \mid \text{dens}(g, s)(r) \leq v\} \\ &= \{(s, r) \mid g(s, r) = \perp \wedge 0 \leq v\} \\ & \quad \cup \{(s, r) \mid \exists s', r', w', p'. g(s, r) = (s', r', w', p') \wedge r' \neq \perp \wedge 0 \leq v\} \\ & \quad \cup \{(s, r) \mid \exists s', r', w', p'. g(s, r) = (s', r', w', p') \wedge r' = \perp \wedge w' \cdot p' \leq v\} \end{aligned}$$

and the three sets on the RHS of the equation are all measurable. The measurability of the other B_1 follows from a similar argument based on case split. That is,

$$\begin{aligned} & \{(s, r) \mid \text{get}(g, s)(r) \in A\} \\ &= \{(s, r) \mid g(s, r) = \perp \wedge \perp \in A\} \\ & \quad \cup \{(s, r) \mid \exists s', r', w', p'. g(s, r) = (s', r', w', p') \wedge r' \neq \perp \wedge \perp \in A\} \\ & \quad \cup \{(s, r) \mid \exists s', r', w', p'. g(s, r) = (s', r', w', p') \wedge r' = \perp \wedge s' \in A\} \end{aligned}$$

and all three sets on the RHS of the equation are measurable. Thus, B_1 is a measurable subset. \square

In the next lemma, we regard $\{\perp\} \cup \text{Store}$ as a partially-ordered set with the expected order: $a \sqsubseteq b \iff a = \perp \vee a = b$, for all $a, b \in \{\perp\} \cup \text{Store}$.

LEMMA A.4. Let $\{g_n\}_n$ be an ω -chain in \mathcal{D} . Then, for all stores s and random databases r , the sequences $\{\text{dens}(g_n, s)(r)\}_n$ and $\{\text{get}(g_n, s)(r)\}_n$ are increasing,

$$\lim_n \text{dens}(g_n, s)(r) = \text{dens}\left(\bigsqcup_n g_n, s\right)(r), \quad \text{and} \quad \lim_n \text{get}(g_n, s)(r) = \text{get}\left(\bigsqcup_n g_n, s\right)(r).$$

PROOF. Let s be a store, r be a random database, and $\{g_n\}_n$ be an ω -chain in \mathcal{D} .

We first show that the sequences $\{\text{dens}(g_n, s)(r)\}_n$ and $\{\text{get}(g_n, s)(r)\}_n$ are increasing. We do so by showing the monotonicity of $\text{dens}(-, s)(r)$ and $\text{get}(-, s)(r)$. Consider $g, g' \in \mathcal{D}$ such that $g \sqsubseteq g'$. Then, $g(s, r) = \perp$ or $g(s, r) = g'(s, r)$. In the former case, $\text{dens}(g, s)(r) = 0 \leq \text{dens}(g', s)(r)$ and $\text{get}(g, s)(r) = \perp \sqsubseteq \text{get}(g', s)(r)$. In the latter case, $\text{dens}(g, s)(r) = \text{dens}(g', s)(r)$ and $\text{get}(g, s)(r) = \text{get}(g', s)(r)$. Thus, $\text{dens}(-, s)(r)$ and $\text{get}(-, s)(r)$ are monotone.

Next we show that $\text{dens}(-, s)(r)$ and $\text{get}(-, s)(r)$ preserve the limit. By what we have already shown, the sequence $\{g_n(s, r)\}_n$ is monotone. But this sequence is ultimately stable. The stable value becomes the least upper bound $\bigsqcup_n g_n(s, r)$, which is also equal to $(\bigsqcup_n g_n)(s, r)$. Thus, $\{\text{dens}(g_n, s)(r)\}_n$ is ultimately stable as well, and its limit is

$$\lim_n \text{dens}(g_n, s)(r) = \text{dens}\left(\bigsqcup_n g_n, s\right)(r).$$

By the same reason, $\{\text{get}(g_n, s)(r)\}_n$ is ultimately stable, and has the limit

$$\lim_n \text{get}(g_n, s)(r) = \text{get}\left(\bigsqcup_n g_n, s\right)(r).$$

\square

LEMMA 4.8. For all non-negative bounded measurable functions $h : (\{\perp\} \cup \text{Store}) \times \text{RDB} \rightarrow \mathbb{R}$, stores s , and functions $g_1, g_2 \in \mathcal{D}$, we have that

$$\begin{aligned} & \int \rho(dr) \left(\text{dens}(g_2^\ddagger \circ g_1, s)(r) \cdot h(\text{get}(g_2^\ddagger \circ g_1, s)(r), r) \right) \\ &= \int \rho(dr_1) \left(\text{dens}(g_1, s)(r_1) \cdot \mathbb{1}_{[\text{get}(g_1, s)(r_1) \neq \perp]} \right. \\ & \quad \cdot \left. \int \rho(dr_2) \left(\text{dens}(g_2, \text{get}(g_1, s)(r_1))(r_2) \cdot \mathbb{1}_{[r_1 \# r_2]} \cdot h(\text{get}(g_2, \text{get}(g_1, s)(r_1))(r_2), r_1 \uplus r_2) \right) \right). \end{aligned}$$

PROOF. For $g \in \mathcal{D}$ and $s \in \text{Store}$, let

$$\begin{aligned} R_1(g, s) &: \text{RDB} \rightarrow \text{RDB}, \\ R_1(g, s)(r) &\triangleq \begin{cases} r \setminus \text{dom}(r') & \text{if } g(s, r) = (s', r', w', p') \text{ for some } (s', r', w', p'), \\ [] & \text{otherwise,} \end{cases} \\ R_2(g, s) &: \text{RDB} \rightarrow \text{RDB}, \\ R_2(g, s)(r) &\triangleq \begin{cases} r' & \text{if } g(s, r) = (s', r', w', p') \text{ for some } (s', r', w', p'), \\ [] & \text{otherwise.} \end{cases} \end{aligned}$$

Here $r \setminus K$ means the restriction of r to $\text{dom}(r) \setminus K$. Both $R_1(g, s)$ and $R_2(g, s)$ are measurable. The measurability of the latter is an immediate consequence of the measurability of g . For the former, we note that for every measurable subset A of RDB with $[] \notin A$,

$$R_1(g, s)^{-1}(A) = \bigcup_{\substack{K_1, K_2 \subseteq \text{finStr} \\ K_1 \cap K_2 = \emptyset}} \{r_1 \uplus r_2 \mid r_1 \in (A \cap [K_1 \rightarrow \mathbb{R}]) \wedge r_2 \in [K_2 \rightarrow \mathbb{R}]\} \cap g^{-1}(s, -)(\text{Store} \times [K_2 \rightarrow \mathbb{R}] \times [0, \infty) \times [0, \infty)).$$

The partially applied function $g(s, -)$ is measurable, so that the RHS of this equation is the countable union of measurable subsets. If $A = \{[]\}$, then

$$R_1(g, s)^{-1}(A) = g^{-1}(s, -)(\{\perp\}) \cup \left(A \cap g^{-1}(s, -)(\text{Store} \times \{[]\} \times [0, \infty) \times [0, \infty)) \right).$$

Thus, $R_1(g, s)^{-1}(A)$ is measurable. The remaining case is that $[] \in A$. We can handle this case by splitting A into $\{[]\}$ and $A \setminus \{[]\}$, and dealing with the slit cases separately.

For $g \in \mathcal{D}$, $s \in \text{Store}$, and $r \in \text{RDB}$, let $\text{consumed}(g, s, r)$ be the predicate defined by

$$\text{consumed}(g, s, r) \iff \exists s', w', p'. (g(s, r) = (s', [], w', p')).$$

Then, $r \mapsto \mathbb{1}_{[\text{consumed}(g, s, r)]}$ is a measurable function.

Now we note a few useful equalities about the entities h, s, g_1, g_2 assumed in the lemma.

First, for all r , if

$$\text{consumed}(g_2^\ddagger \circ g_1, s, r),$$

then

$$r = R_1(g_1, s)(r) \uplus R_2(g_1, s)(r),$$

$$\text{get}(g_2^\ddagger \circ g_1, s)(r) = \text{get}(g_2, \text{get}(g_1, s)(R_1(g_1, s)(r)))(R_2(g_1, s)(r)),$$

$$\text{dens}(g_2^\ddagger \circ g_1, s)(r) = \text{dens}(g_1, s)(R_1(g_1, s)(r)) \cdot \text{dens}(g_2, \text{get}(g_1, s)(R_1(g_1, s)(r)))(R_2(g_1, s)(r)).$$

The first equality is just an immediate consequence of the definitions of R_1 and R_2 , and the other two equalities hold mainly because g_1 and g_2 are local.

Next, consider the density functions f_1 on RDB and f_2 on $RDB \times RDB$:

$$\begin{aligned} f_1 : RDB &\rightarrow [0, \infty), & f_1(r) &\triangleq \mathbb{1}_{[\text{consumed}(g_2^\dagger \circ g_1, s, r)]}, \\ f_2 : RDB \times RDB &\rightarrow [0, \infty) & f_2(r_1, r_2) &\triangleq \mathbb{1}_{[\text{consumed}(g_1, s, r_1)]} \cdot \mathbb{1}_{[\text{consumed}(g_2, \text{get}(g_1, s)(r_1), r_2)]} \\ & & &\cdot \mathbb{1}_{[r_1 \# r_2]}. \end{aligned}$$

The densities are taken with respect to ρ and $\rho \otimes \rho$, respectively. There is a measurable bijection β between the supports of f_1 and f_2 , and the pushforward of the measure of f_1 by this bijection is the measure of f_2 . Concretely, the function β is:

$$\beta : RDB \rightarrow RDB \times RDB, \quad \beta(r) = \begin{cases} (R_1(g_1, s)(r), R_2(g_1, s)(r)) & \text{if } \text{consumed}(g_2^\dagger \circ g_1, s, r) \\ (r_d, r_d) & \text{otherwise,} \end{cases}$$

where $r_d \in RDB$ is $[“s” \mapsto \text{nil}]$. Note that β is not a bijection. But the restriction of β to $A_1 = \{r \mid f_1(r) > 0\}$ is injective, and its image is $A_2 = \{(r_1, r_2) \mid f_2(r_1, r_2) > 0\}$. These properties follow from the definitions of *consumed*, R_1 and R_2 and the locality of g . The function β is measurable because $R_1(g_1, s)$ and $R_2(g_2, s)$ are measurable functions and $\{r \mid \text{consumed}(g_2^\dagger \circ g_1, s, r)\}$ is a measurable set. It remains to show the measure preservation of β . Pick a measurable subset A' of A_2 . We have to show that

$$(\rho \otimes \rho)(A') = \rho(\beta^{-1}(A')).$$

We calculate the equation as follows:

$$\begin{aligned} (\rho \otimes \rho)(A') &= (\rho \otimes \rho) \left(\bigcup_{\substack{K_1, K_2 \subseteq \text{finStr} \\ K_1 \cap K_2 = \emptyset}} A' \cap ([K_1 \rightarrow \mathbb{R}] \times [K_2 \rightarrow \mathbb{R}]) \right) \\ &= \bigcup_{\substack{K_1, K_2 \subseteq \text{finStr} \\ K_1 \cap K_2 = \emptyset}} \left(\left(\bigotimes_{\alpha_1 \in K_1} \rho_v \right) \otimes \left(\bigotimes_{\alpha_2 \in K_2} \rho_v \right) (A' \cap ([K_1 \rightarrow \mathbb{R}] \times [K_2 \rightarrow \mathbb{R}]) \right) \\ &= \bigcup_{\substack{K_1, K_2 \subseteq \text{finStr} \\ K_1 \cap K_2 = \emptyset}} \left(\bigotimes_{\alpha \in K_1 \cup K_2} \rho_v \right) (\beta^{-1}(A' \cap ([K_1 \rightarrow \mathbb{R}] \times [K_2 \rightarrow \mathbb{R}]) \big) \\ &= \bigcup_{\substack{K_1, K_2 \subseteq \text{finStr} \\ K_1 \cap K_2 = \emptyset}} \rho(\beta^{-1}(A' \cap ([K_1 \rightarrow \mathbb{R}] \times [K_2 \rightarrow \mathbb{R}]) \big) \\ &= \rho(\beta^{-1} \left(\bigcup_{\substack{K_1, K_2 \subseteq \text{finStr} \\ K_1 \cap K_2 = \emptyset}} (A' \cap ([K_1 \rightarrow \mathbb{R}] \times [K_2 \rightarrow \mathbb{R}]) \right) \big) \\ &= \rho(\beta^{-1}(A')). \end{aligned}$$

Using what we have shown so far, we calculate the claim of the lemma. In the calculation we often omit the g_1 and s parameters from $R_1(g_1, s)(r)$ and $R_2(g_1, s)(r)$, and just write $R_1(r)$ and $R_2(r)$.

$$\begin{aligned} &\int \rho(dr) \left(\text{dens}(g_2^\dagger \circ g_1, s)(r) \cdot h(\text{get}(g_2^\dagger \circ g_1, s)(r), r) \right) \\ &= \int \rho(dr) \left(\mathbb{1}_{[\text{consumed}(g_2^\dagger \circ g_1, s, r)]} \cdot \text{dens}(g_2^\dagger \circ g_1, s)(r) \cdot h(\text{get}(g_2^\dagger \circ g_1, s)(r), r) \right) \\ &= \int \rho(dr) \left(\mathbb{1}_{[\text{consumed}(g_2^\dagger \circ g_1, s, r)]} \cdot \text{dens}(g_2^\dagger \circ g_1, s)(r) \cdot \mathbb{1}_{[\text{get}(g_1, s)(R_1(r_1)) \neq \perp]} \cdot h(\text{get}(g_2^\dagger \circ g_1, s)(r), r) \right) \end{aligned}$$

$$\begin{aligned}
&= \int \rho(dr) \left(\mathbb{1}_{[\text{consumed}(g_2^* \circ g_1, s, r)]} \cdot \text{dens}(g_1, s)(R_1(r)) \cdot \mathbb{1}_{[\text{get}(g_1, s)(R_1(r)) \neq \perp]} \right. \\
&\quad \left. \cdot \text{dens}(g_2, \text{get}(g_1, s)(R_1(r)))(R_2(r)) \cdot h(\text{get}(g_2, \text{get}(g_1, s)(R_1(r)))(R_2(r)), R_1(r) \uplus R_2(r)) \right) \\
&= \int \rho(dr_1) \int \rho(dr_2) \left(\mathbb{1}_{[\text{consumed}(g_1, s, r_1)]} \cdot \mathbb{1}_{[\text{consumed}(g_2, \text{get}(g_1, s)(r_1), r_2)]} \cdot \mathbb{1}_{[r_1 \# r_2]} \right. \\
&\quad \cdot \text{dens}(g_1, s)(r_1) \cdot \mathbb{1}_{[\text{get}(g_1, s)(r_1) \neq \perp]} \cdot \text{dens}(g_2, \text{get}(g_1, s)(r_1))(r_2) \\
&\quad \left. \cdot h(\text{get}(g_2, \text{get}(g_1, s)(r_1))(r_2), r_1 \uplus r_2) \right) \\
&= \int \rho(dr_1) \int \rho(dr_2) \left(\mathbb{1}_{[r_1 \# r_2]} \cdot \text{dens}(g_1, s)(r_1) \cdot \mathbb{1}_{[\text{get}(g_1, s)(r_1) \neq \perp]} \cdot \text{dens}(g_2, \text{get}(g_1, s)(r_1))(r_2) \right. \\
&\quad \left. \cdot h(\text{get}(g_2, \text{get}(g_1, s)(r_1))(r_2), r_1 \uplus r_2) \right) \\
&= \int \rho(dr_1) \left(\text{dens}(g_1, s)(r_1) \cdot \mathbb{1}_{[\text{get}(g_1, s)(r_1) \neq \perp]} \right. \\
&\quad \left. \cdot \int \rho(dr_2) \left(\text{dens}(g_2, \text{get}(g_1, s)(r_1))(r_2) \cdot \mathbb{1}_{[r_1 \# r_2]} \cdot h(\text{get}(g_2, \text{get}(g_1, s)(r_1))(r_2), r_1 \uplus r_2) \right) \right).
\end{aligned}$$

□

THEOREM 4.9. *For all non-negative bounded measurable monotone functions $h : (\{\perp\} \cup \text{Store}) \times \text{RDB} \rightarrow \mathbb{R}$ and states σ ,*

$$\int \llbracket C \rrbracket(\sigma)(d(\sigma', w'))(w' \cdot h(\sigma'_s, \sigma'_r)) = \int \rho(dr') \left(\text{dens}(C, \sigma_s)(r') \cdot \mathbb{1}_{[r' \# \sigma_r]} \cdot h(\text{get}(C, \sigma_s)(r'), r' \uplus \sigma_r) \right).$$

PROOF. We prove the theorem by induction on the structure of C .

When $C \equiv \text{skip}$, we derive the claimed equality as follows:

$$\begin{aligned}
&\int \llbracket \text{skip} \rrbracket(\sigma)(d(\sigma', w'))(w' \cdot h(\sigma'_s, \sigma'_r)) \\
&= h(\sigma_s, \sigma_r) \\
&= \int \rho(dr') \left(\mathbb{1}_{[r' = []]} \cdot \mathbb{1}_{[[] \# \sigma_r]} \cdot h(\sigma_s, [] \uplus \sigma_r) \right) \\
&= \int \rho(dr') \left(\mathbb{1}_{[r' = []]} \cdot \mathbb{1}_{[r' \# \sigma_r]} \cdot h(\sigma_s, r' \uplus \sigma_r) \right) \\
&= \int \rho(dr') \left(\text{dens}(\text{skip}, \sigma_s)(r') \cdot \mathbb{1}_{[r' \# \sigma_r]} \cdot h(\text{get}(\text{skip}, \sigma_s)(r'), r' \uplus \sigma_r) \right).
\end{aligned}$$

Next we handle the case that $C \equiv (x := E)$.

$$\begin{aligned}
&\int \llbracket x := E \rrbracket(\sigma)(d(\sigma', w'))(w' \cdot h(\sigma'_s, \sigma'_r)) \\
&= h(\sigma_s[x \mapsto \llbracket E \rrbracket \sigma_s], \sigma_r) \\
&= \int \rho(dr') \left(\mathbb{1}_{[r' = []]} \cdot \mathbb{1}_{[[] \# \sigma_r]} \cdot h(\sigma_s[x \mapsto \llbracket E \rrbracket \sigma_s], [] \uplus \sigma_r) \right) \\
&= \int \rho(dr') \left(\mathbb{1}_{[r' = []]} \cdot \mathbb{1}_{[r' \# \sigma_r]} \cdot h(\sigma_s[x \mapsto \llbracket E \rrbracket \sigma_s], r' \uplus \sigma_r) \right) \\
&= \int \rho(dr') \left(\text{dens}(x := E, \sigma_s)(r') \cdot \mathbb{1}_{[r' \# \sigma_r]} \cdot h(\text{get}(x := E, \sigma_s)(r'), r' \uplus \sigma_r) \right).
\end{aligned}$$

We move on to the case that $C \equiv (\text{if } B \{C_0\} \text{ else } \{C_1\})$, and prove the desired equality:

$$\begin{aligned}
& \int \llbracket \text{if } B \{C_0\} \text{ else } \{C_1\} \rrbracket(\sigma)(d(\sigma', w'))(w' \cdot h(\sigma'_s, \sigma'_r)) \\
&= \mathbb{1}_{[\llbracket B \rrbracket \sigma_s = \text{true}]} \cdot \int \llbracket C_0 \rrbracket(\sigma)(d(\sigma', w'))(w' \cdot h(\sigma'_s, \sigma'_r)) \\
&\quad + \mathbb{1}_{[\llbracket B \rrbracket \sigma_s \neq \text{true}]} \cdot \int \llbracket C_1 \rrbracket(\sigma)(d(\sigma', w'))(w' \cdot h(\sigma'_s, \sigma'_r)) \\
&= \mathbb{1}_{[\llbracket B \rrbracket \sigma_s = \text{true}]} \cdot \int \rho(dr') \left(\text{dens}(C_0, \sigma_s)(r') \cdot \mathbb{1}_{[r' \# \sigma_r]} \cdot h(\text{get}(C_0, \sigma_s)(r'), r' \uplus \sigma_r) \right) \\
&\quad + \mathbb{1}_{[\llbracket B \rrbracket \sigma_s \neq \text{true}]} \cdot \int \rho(dr') \left(\text{dens}(C_1, \sigma_s)(r') \cdot \mathbb{1}_{[r' \# \sigma_r]} \cdot h(\text{get}(C_1, \sigma_s)(r'), r' \uplus \sigma_r) \right) \\
&= \int \rho(dr') \left(\text{dens}(\text{if } B \{C_0\} \text{ else } \{C_1\}, \sigma_s)(r') \cdot \mathbb{1}_{[r' \# \sigma_r]} \cdot h(\text{get}(\text{if } B \{C_0\} \text{ else } \{C_1\}, \sigma_s)(r'), r' \uplus \sigma_r) \right).
\end{aligned}$$

Now we prove the claimed equality for $C \equiv (C_0; C_1)$.

$$\begin{aligned}
& \int \llbracket C_0; C_1 \rrbracket(\sigma)(d(\sigma', w'))(w' \cdot h(\sigma'_s, \sigma'_r)) \\
&= \int \llbracket C_0 \rrbracket(\sigma)(d(\sigma', w')) \int \llbracket C_1 \rrbracket(\sigma')(d(\sigma'', w''))(w' \cdot w'' \cdot h(\sigma''_s, \sigma''_r)) \\
&= \int \llbracket C_0 \rrbracket(\sigma)(d(\sigma', w')) \left(w' \cdot \int \llbracket C_1 \rrbracket(\sigma')(d(\sigma'', w''))(w'' \cdot h(\sigma''_s, \sigma''_r)) \right) \\
&= \int \llbracket C_0 \rrbracket(\sigma)(d(\sigma', w')) \left(w' \cdot \int \rho(dr_2) (\text{dens}(C_1, \sigma'_s)(r_2) \cdot \mathbb{1}_{[r_2 \# \sigma'_r]} \cdot h(\text{get}(C_1, \sigma'_s)(r_2), r_2 \uplus \sigma'_r)) \right) \\
&= \int \llbracket C_0 \rrbracket(\sigma)(d(\sigma', w')) \\
&\quad \left(w' \cdot \mathbb{1}_{[\sigma'_s \neq \perp]} \cdot \int \rho(dr_2) (\text{dens}(C_1, \sigma'_s)(r_2) \cdot \mathbb{1}_{[r_2 \# \sigma'_r]} \cdot h(\text{get}(C_1, \sigma'_s)(r_2), r_2 \uplus \sigma'_r)) \right) \\
&= \int \rho(dr_1) \left(\text{dens}(C_0, \sigma_s)(r_1) \cdot \mathbb{1}_{[r_1 \# \sigma_r]} \right. \\
&\quad \cdot \mathbb{1}_{[\text{get}(C_0, \sigma_s)(r_1) \neq \perp]} \cdot \int \rho(dr_2) \left(\text{dens}(C_1, \text{get}(C_0, \sigma_s)(r_1))(r_2) \cdot \mathbb{1}_{[r_2 \# (r_1 \uplus \sigma_r)]} \right. \\
&\quad \left. \left. \cdot h(\text{get}(C_1, \text{get}(C_0, \sigma_s)(r_1))(r_2), r_2 \uplus (r_1 \uplus \sigma_r)) \right) \right) \\
&= \int \rho(dr_1) \left(\text{dens}(C_0, \sigma_s)(r_1) \cdot \mathbb{1}_{[\text{get}(C_0, \sigma_s)(r_1) \neq \perp]} \right. \\
&\quad \cdot \int \rho(dr_2) \left(\text{dens}(C_1, \text{get}(C_0, \sigma_s)(r_1))(r_2) \cdot \mathbb{1}_{[r_1 \# r_2]} \cdot \mathbb{1}_{[(r_1 \uplus r_2) \# \sigma_r]} \right. \\
&\quad \left. \left. \cdot h(\text{get}(C_1, \text{get}(C_0, \sigma_s)(r_1))(r_2), (r_1 \uplus r_2) \uplus \sigma_r) \right) \right) \\
&= \int \rho(dr) \left(\text{dens}(C_0; C_1)(r) \cdot \mathbb{1}_{[r \# \sigma_r]} \cdot h(\text{get}(C_0; C_1)(r), r \uplus \sigma_r) \right).
\end{aligned}$$

The third and fifth equalities follow from the induction hypothesis, and the last equality from Lemma 4.8.

The next case is $C \equiv (\text{while } B \{C_0\})$. Let $\mathcal{R} \subseteq \mathcal{K} \times \mathcal{D}$ be the relation defined by

$$\kappa[\mathcal{R}]g \iff \text{for all states } \sigma \text{ and non-negative bounded measurable monotone functions } h,$$

$$\begin{aligned}
& \int \kappa(\sigma)(d(\sigma', w')) (w' \cdot h(\sigma'_s, \sigma'_r)) \\
&= \int \rho(dr') \left(\text{dens}(g, \sigma_s)(r') \cdot \mathbb{1}_{[r' \# \sigma_r]} \cdot h(\text{get}(g, \sigma_s)(r'), r' \uplus \sigma_r) \right).
\end{aligned}$$

Note that the least elements from \mathcal{K} and \mathcal{D} are related by \mathcal{R} because they make both sides of the equality in the definition of \mathcal{R} be zero. Furthermore, for all ω -chains $\{\kappa_n\}_n$ and $\{g_n\}_n$ in \mathcal{K} and \mathcal{D} , if $\kappa_n [\mathcal{R}] g_n$ for all n , then

$$\bigsqcup_n \kappa_n [\mathcal{R}] \bigsqcup_n g_n.$$

This is because for all stores σ and non-negative bounded measurable functions h ,

$$\begin{aligned}
& \int \left(\bigsqcup_n \kappa_n \right) (\sigma)(d(\sigma', w')) (w' \cdot h(\sigma'_s, \sigma'_r)) \\
&= \lim_n \int \kappa_n(\sigma)(d(\sigma', w')) (w' \cdot f(\sigma'_r)) \\
&= \lim_n \int \rho(dr') \left(\text{dens}(g_n, \sigma_s)(r') \cdot \mathbb{1}_{[r' \# \sigma_r]} \cdot h(\text{get}(g_n, \sigma_s)(r'), r' \uplus \sigma_r) \right) \\
&= \int \rho(dr') \lim_n \left(\text{dens}(g_n, \sigma_s)(r') \cdot \mathbb{1}_{[r' \# \sigma_r]} \cdot h(\text{get}(g_n, \sigma_s)(r'), r' \uplus \sigma_r) \right) \\
&= \int \rho(dr') \left(\text{dens}\left(\bigsqcup_n g_n, \sigma_s\right)(r') \cdot \mathbb{1}_{[r' \# \sigma_r]} \cdot h\left(\text{get}\left(\bigsqcup_n g_n, \sigma_s\right)(r'), r' \uplus \sigma_r\right) \right).
\end{aligned}$$

The first equation follows from Lemma A.1, and the third from the monotone convergence theorem and Lemma A.4. The last equality holds because of Lemma A.4 again.

Thus, we can complete the proof of this case if we show one more thing. Let K and G be the functions on \mathcal{K} and \mathcal{D} used in the interpretation of the loop (namely, while $B \{C_0\}$) in the two semantics. What we have to show is:

$$\kappa [\mathcal{R}] g \implies K(\kappa) [\mathcal{R}] G(g).$$

Pick κ and g with $\kappa [\mathcal{R}] g$. Then, for all σ and non-negative bounded measurable h ,

$$\begin{aligned}
& \int (K(\kappa))(\sigma)(d(\sigma', w')) (w' \cdot h(\sigma'_s, \sigma'_r)) \\
&= \mathbb{1}_{[\llbracket B \rrbracket \sigma_s \neq \text{true}]} \cdot h(\sigma_s, \sigma_r) \\
&\quad + \mathbb{1}_{[\llbracket B \rrbracket \sigma_s = \text{true}]} \cdot \int \llbracket C \rrbracket (\sigma)(d(\sigma', w')) \int \kappa(\sigma')(d(\sigma'', w'')) (w' \cdot w'' \cdot h(\sigma''_s, \sigma''_r)) \\
&= \mathbb{1}_{[\llbracket B \rrbracket \sigma_s \neq \text{true}]} \cdot h(\sigma_s, \sigma_r) \\
&\quad + \mathbb{1}_{[\llbracket B \rrbracket \sigma_s = \text{true}]} \cdot \int \llbracket C \rrbracket (\sigma)(d(\sigma', w')) \left(w' \cdot \int \kappa(\sigma')(d(\sigma'', w'')) (w'' \cdot h(\sigma''_s, \sigma''_r)) \right).
\end{aligned}$$

But

$$\begin{aligned}
& \int \llbracket C \rrbracket (\sigma)(d(\sigma', w')) \left(w' \cdot \int \kappa(\sigma')(d(\sigma'', w'')) (w'' \cdot h(\sigma''_s, \sigma''_r)) \right) \\
&= \int \llbracket C \rrbracket (\sigma)(d(\sigma', w')) \left(w' \cdot \int \rho(dr'') \left(\text{dens}(g, \sigma'_s)(r'') \cdot \mathbb{1}_{[r'' \# \sigma'_r]} \cdot h(\text{get}(g, \sigma'_s)(r''), r'' \uplus \sigma'_r) \right) \right) \\
&= \int \llbracket C \rrbracket (\sigma)(d(\sigma', w'))
\end{aligned}$$

$$\begin{aligned}
& \left(w' \cdot \mathbb{1}_{[\sigma'_s \neq \perp]} \cdot \int \rho(dr'') \left(\text{dens}(g, \sigma'_s)(r'') \cdot \mathbb{1}_{[r'' \# \sigma'_r]} \cdot h(\text{get}(g, \sigma'_s)(r''), r'' \uplus \sigma'_r) \right) \right) \\
&= \int \rho(dr') \left(\text{dens}(C, \sigma_s)(r') \cdot \mathbb{1}_{[r' \# \sigma_r]} \right. \\
&\quad \cdot \mathbb{1}_{[\text{get}(C, \sigma_s)(r') \neq \perp]} \cdot \int \rho(dr'') \left(\text{dens}(g, \text{get}(C, \sigma_s)(r'))(r'') \cdot \mathbb{1}_{[r'' \# (r' \uplus \sigma_r)]} \right. \\
&\quad \left. \left. \cdot h(\text{get}(g, \text{get}(C, \sigma_s))(r''), r'' \uplus (r' \uplus \sigma_r)) \right) \right) \\
&= \int \rho(dr') \left(\text{dens}(C, \sigma_s)(r') \cdot \int \rho(dr'') \left(\text{dens}(g, \text{get}(C, \sigma_s)(r'))(r'') \cdot \mathbb{1}_{[r'' \# r']} \cdot \mathbb{1}_{[(r' \uplus r') \# \sigma_r]} \right. \right. \\
&\quad \left. \left. \cdot h(\text{get}(g, \text{get}(C, \sigma_s))(r''), (r' \uplus r') \uplus \sigma_r) \right) \right) \\
&= \int \rho(dr) \left(\text{dens}(g^\ddagger \circ \llbracket C \rrbracket_d, \sigma_s)(r) \cdot \mathbb{1}_{[r \# \sigma_r]} \cdot h(\text{get}(g^\ddagger \circ \llbracket C \rrbracket_d, \sigma_s)(r), r \uplus \sigma_r) \right).
\end{aligned}$$

The last equality uses Lemma 4.8. Thus, if we continue our calculation that we paused momentarily, we get

$$\begin{aligned}
& \mathbb{1}_{[\llbracket B \rrbracket \sigma_s \neq \text{true}]} \cdot h(\sigma_s, \sigma_r) \\
&+ \mathbb{1}_{[\llbracket B \rrbracket \sigma_s = \text{true}]} \cdot \int \llbracket C \rrbracket(\sigma)(d(\sigma', w')) \left(w' \cdot \int \kappa(\sigma')(d(\sigma'', w'')) (w'' \cdot h(\sigma'_s, \sigma'_r)) \right) \\
&= \mathbb{1}_{[\llbracket B \rrbracket \sigma_s \neq \text{true}]} \cdot h(\sigma_s, \sigma_r) \\
&+ \mathbb{1}_{[\llbracket B \rrbracket \sigma_s = \text{true}]} \cdot \int \rho(dr) \left(\text{dens}(g^\ddagger \circ \llbracket C \rrbracket_d, \sigma_s)(r) \cdot \mathbb{1}_{[r \# \sigma_r]} \cdot h(\text{get}(g^\ddagger \circ \llbracket C \rrbracket_d, \sigma_s)(r), r \uplus \sigma_r) \right) \\
&= \mathbb{1}_{[\llbracket B \rrbracket \sigma_s \neq \text{true}]} \cdot \int \rho(dr') \left(\mathbb{1}_{[r' = \square]} \cdot \mathbb{1}_{[\square \# \sigma_r]} \cdot h(\sigma_s, \square \uplus \sigma_r) \right) \\
&+ \mathbb{1}_{[\llbracket B \rrbracket \sigma_s = \text{true}]} \cdot \int \rho(dr) \left(\text{dens}(g^\ddagger \circ \llbracket C \rrbracket_d, \sigma_s)(r) \cdot \mathbb{1}_{[r \# \sigma_r]} \cdot h(\text{get}(g^\ddagger \circ \llbracket C \rrbracket_d, \sigma_s)(r), r \uplus \sigma_r) \right) \\
&= \mathbb{1}_{[\llbracket B \rrbracket \sigma_s \neq \text{true}]} \cdot \int \rho(dr') \left(\mathbb{1}_{[r' = \square]} \cdot \mathbb{1}_{[r' \# \sigma_r]} \cdot h(\sigma_s, r' \uplus \sigma_r) \right) \\
&+ \mathbb{1}_{[\llbracket B \rrbracket \sigma_s = \text{true}]} \cdot \int \rho(dr) \left(\text{dens}(g^\ddagger \circ \llbracket C \rrbracket_d, \sigma_s)(r) \cdot \mathbb{1}_{[r \# \sigma_r]} \cdot h(\text{get}(g^\ddagger \circ \llbracket C \rrbracket_d, \sigma_s)(r), r \uplus \sigma_r) \right) \\
&= \int \rho(dr') \left(\text{dens}(G(g), \sigma_s)(r') \cdot \mathbb{1}_{[r' \# \sigma_r]} \cdot h(\text{get}(G(g), \sigma_s)(r'), r' \uplus \sigma_r) \right).
\end{aligned}$$

Next we handle the case that $C \equiv (x := \text{sample}_{\text{norm}}(S, E_1, E_2))$.

$$\begin{aligned}
& \int \llbracket x := \text{sample}_{\text{norm}}(S, E_1, E_2) \rrbracket(\sigma)(d(\sigma', w')) (w' \cdot h(\sigma'_s, \sigma'_r)) \\
&= \mathbb{1}_{[\llbracket S \rrbracket \sigma_s \notin \text{dom}(\sigma_r)]} \cdot \mathbb{1}_{[\llbracket E_2 \rrbracket \sigma_s \in (0, \infty)]} \\
&\quad \cdot \int dv \left(\mathcal{N}(v; \llbracket E_1 \rrbracket \sigma_s, \llbracket E_2 \rrbracket \sigma_s) \cdot h(\sigma_s[x \mapsto v], \sigma_r[\llbracket S \rrbracket \sigma_s \mapsto v]) \right) \\
&= \mathbb{1}_{[\llbracket E_2 \rrbracket \sigma_s \in (0, \infty)]} \\
&\quad \cdot \int dv \left(\mathcal{N}(v; \llbracket E_1 \rrbracket \sigma_s, \llbracket E_2 \rrbracket \sigma_s) \cdot \mathbb{1}_{[\llbracket S \rrbracket \sigma_s \mapsto v \# \sigma_r]} \cdot h(\sigma_s[x \mapsto v], [\llbracket S \rrbracket \sigma_s \mapsto v] \uplus \sigma_r) \right)
\end{aligned}$$

$$\begin{aligned}
&= \mathbb{1}_{[\llbracket E_2 \rrbracket \sigma_s \in (0, \infty)]} \cdot \int \rho(dr') \left(\mathbb{1}_{[\text{dom}(r') = \{\llbracket S \rrbracket \sigma_s\}]} \cdot \mathcal{N}(r'(\llbracket S \rrbracket \sigma_s); \llbracket E_1 \rrbracket \sigma_s, \llbracket E_2 \rrbracket \sigma_s) \right. \\
&\quad \left. \cdot \mathbb{1}_{[r' \# \sigma_r]} \cdot h(\sigma_s[x \mapsto r'(\llbracket E_0 \rrbracket \sigma_s)], r' \uplus \sigma_r) \right) \\
&= \int \rho(dr') \left(\mathbb{1}_{[\llbracket E_2 \rrbracket \sigma_s \in (0, \infty)]} \cdot \mathbb{1}_{[\text{dom}(r') = \{\llbracket S \rrbracket \sigma_s\}]} \cdot \mathcal{N}(r'(\llbracket S \rrbracket \sigma_s); \llbracket E_1 \rrbracket \sigma_s, \llbracket E_2 \rrbracket \sigma_s) \right. \\
&\quad \left. \cdot \mathbb{1}_{[r' \# \sigma_r]} \cdot h(\sigma_s[x \mapsto r'(\llbracket E_0 \rrbracket \sigma_s)], r' \uplus \sigma_r) \right) \\
&= \int \rho(dr') \left(\text{dens}(x := \text{sample}_{\text{norm}}(S, E_1, E_2), \sigma_s)(r') \right. \\
&\quad \left. \cdot \mathbb{1}_{[r' \# \sigma_r]} \cdot h(\text{get}(x := \text{sample}_{\text{norm}}(S, E_1, E_2), \sigma_s)(r'), r' \uplus \sigma_r) \right).
\end{aligned}$$

It remains to prove the case that $C \equiv \text{score}_{\text{norm}}(E_0, E_1, E_2)$. Here is our proof for this last case:

$$\begin{aligned}
&\int \llbracket \text{score}_{\text{norm}}(E_0, E_1, E_2) \rrbracket(\sigma)(d(\sigma', w'))(w' \cdot h(\sigma'_s, \sigma'_r)) \\
&= \mathbb{1}_{[\llbracket E_2 \rrbracket \sigma_s \in (0, \infty)]} \cdot \mathcal{N}(\llbracket E_0 \rrbracket \sigma_s; \llbracket E_1 \rrbracket \sigma_s, \llbracket E_2 \rrbracket \sigma_s) \cdot h(\sigma_s, \sigma_r) \\
&= \mathbb{1}_{[\llbracket E_2 \rrbracket \sigma_s \in (0, \infty)]} \cdot \int \rho(dr') \left(\mathbb{1}_{[r' = []]} \cdot \mathcal{N}(\llbracket E_0 \rrbracket \sigma_s; \llbracket E_1 \rrbracket \sigma_s, \llbracket E_2 \rrbracket \sigma_s) \cdot \mathbb{1}_{[[] \# \sigma_r]} \cdot h(\sigma_s, [] \uplus \sigma_r) \right) \\
&= \mathbb{1}_{[\llbracket E_2 \rrbracket \sigma_s \in (0, \infty)]} \cdot \int \rho(dr') \left(\mathbb{1}_{[r' = []]} \cdot \mathcal{N}(\llbracket E_0 \rrbracket \sigma_s; \llbracket E_1 \rrbracket \sigma_s, \llbracket E_2 \rrbracket \sigma_s) \cdot \mathbb{1}_{[r' \# \sigma_r]} \cdot h(\sigma_s, r' \uplus \sigma_r) \right) \\
&= \int \rho(dr') \left(\text{dens}(\text{score}_{\text{norm}}(E_0, E_1, E_2), \sigma_s)(r') \right. \\
&\quad \left. \cdot \mathbb{1}_{[r' \# \sigma_r]} \cdot h(\text{get}(\text{score}_{\text{norm}}(E_0, E_1, E_2), \sigma_s)(r'), r' \uplus \sigma_r) \right).
\end{aligned}$$

□

COROLLARY 4.10. $\text{Mea}(C, A) = \int \rho(dr) (\mathbb{1}_{[r \in A]} \cdot \text{dens}(C, s_I)(r))$ for all C and all measurable A .

PROOF. We instantiate Theorem 4.9 with $h(a, r) = \mathbb{1}_{[r \in A]}$ and $\sigma = \sigma_I$. Recall that $(\sigma_I)_s = s_I$ and $(\sigma_I)_r = []$. Thus, the conclusion of Theorem 4.9 in this case says that

$$\begin{aligned}
&\int \llbracket C \rrbracket(\sigma_I)(d(\sigma', w'))(w' \cdot \mathbb{1}_{[\sigma'_r \in A]}) = \int \rho(dr') \left(\text{dens}(C, s_I)(r') \cdot \mathbb{1}_{[r' \# (\sigma_I)_r]} \cdot \mathbb{1}_{[r' \uplus (\sigma_I)_r \in A]} \right) \\
&= \int \rho(dr') \left(\text{dens}(C, s_I)(r') \cdot \mathbb{1}_{[r' \in A]} \right).
\end{aligned}$$

This gives the equation claimed by the corollary. □

B PROOF OF THEOREM IN §5

THEOREM 5.1. Let C be a model, D_θ be a guide, and $N \neq 0 \in \mathbb{N}$. Define $\text{KL}_{(-)} : \mathbb{R}^P \rightarrow \mathbb{R}_{\geq 0}$ as $\text{KL}_\theta \triangleq \text{KL}(\text{dens}(D_\theta, s_I) \parallel \text{dens}(C, s_I) / Z_C)$. Then, $\text{KL}_{(-)}$ is well-defined and continuously differentiable with

$$\nabla_\theta \text{KL}_\theta = \mathbb{E}_{\prod_i \text{dens}(D_\theta, s_I)(r_i)} \left[\frac{1}{N} \sum_{i=1}^N \left(\nabla_\theta \log \text{dens}(D_\theta, s_I)(r_i) \right) \log \frac{\text{dens}(D_\theta, s_I)(r_i)}{\text{dens}(C, s_I)(r_i)} \right] \quad (6)$$

if

(R1) $\text{dens}(C, s_I)(r) = 0 \implies \text{dens}(D_\theta, s_I)(r) = 0$, for all $r \in \text{RDB}$ and $\theta \in \mathbb{R}^P$;

(R2) for all $(r, \theta, j) \in \text{RDB} \times \mathbb{R}^P \times [p]$, the function $v \mapsto \text{dens}(D_{\theta[j:v]}, s_I)(r)$ on \mathbb{R} is differentiable;

(R3) for all $\theta \in \mathbb{R}^p$,

$$\int \rho(dr) \left(\text{dens}(D_\theta, s_I)(r) \cdot \log \frac{\text{dens}(D_\theta, s_I)(r)}{\text{dens}(C, s_I)(r)} \right) < \infty;$$

(R4) for all $(\theta, j) \in \mathbb{R}^p \times [p]$, the function

$$v \mapsto \int \rho(dr) \left(\text{dens}(D_{\theta[j:v]}, s_I)(r) \cdot \log \frac{\text{dens}(D_{\theta[j:v]}, s_I)(r)}{\text{dens}(C, s_I)(r)} \right)$$

on \mathbb{R} is continuously differentiable;

(R5) for all $\theta \in \mathbb{R}^p$,

$$\begin{aligned} \nabla_\theta \int \rho(dr) \left(\text{dens}(D_\theta, s_I)(r) \cdot \log \frac{\text{dens}(D_\theta, s_I)(r)}{\text{dens}(C, s_I)(r)} \right) \\ = \int \rho(dr) \nabla_\theta \left(\text{dens}(D_\theta, s_I)(r) \cdot \log \frac{\text{dens}(D_\theta, s_I)(r)}{\text{dens}(C, s_I)(r)} \right); \end{aligned}$$

(R6) for all $\theta \in \mathbb{R}^p$,

$$\int \rho(dr) \nabla_\theta \text{dens}(D_\theta, s_I)(r) = \nabla_\theta \int \rho(dr) \text{dens}(D_\theta, s_I)(r).$$

As we explained in the main text already, the theorem (except the continuous differentiability of KL_θ) is well-known with a well-known proof. We include its proof for completeness and also to help the reader to see why each of the requirements in the theorem is needed.

PROOF. KL_θ is well-defined because of the following derivation with R1 and R3:

$$\begin{aligned} \text{KL}_\theta &= \text{KL} \left(\text{dens}(D_\theta, s_I) \parallel \frac{\text{dens}(C, s_I)}{Z_C} \right) \\ &= \int \rho(dr) \left(\text{dens}(D_\theta, s_I)(r) \cdot \log \left(\frac{\text{dens}(D_\theta, s_I)(r)}{\text{dens}(C, s_I)(r)/Z_C} \right) \right) \\ &= \int \rho(dr) \left(\text{dens}(D_\theta, s_I)(r) \cdot \left(\log Z_C + \log \frac{\text{dens}(D_\theta, s_I)(r)}{\text{dens}(C, s_I)(r)} \right) \right) \\ &= \log Z_C + \int \rho(dr) \left(\text{dens}(D_\theta, s_I)(r) \cdot \log \frac{\text{dens}(D_\theta, s_I)(r)}{\text{dens}(C, s_I)(r)} \right). \end{aligned}$$

It remains to show the continuous differentiability of KL_θ and Equation (6). Recall the fact that for any function $f : \mathbb{R}^N \rightarrow \mathbb{R}^M$, if all partial derivatives of f exist and are continuous, f is continuously differentiable. Because of the fact and the linearity of expectation, it suffices to show that all partial derivatives of KL_θ are well-defined and continuous, and

$$\nabla_\theta \text{KL}_\theta = \mathbb{E}_{\text{dens}(D_\theta, s_I)(r)} \left[\left(\nabla_\theta \log \text{dens}(D_\theta, s_I)(r) \right) \log \frac{\text{dens}(D_\theta, s_I)(r)}{\text{dens}(C, s_I)(r)} \right]. \quad (13)$$

Here is a well-known derivation of (13):

$$\begin{aligned} \nabla_\theta \text{KL}_\theta &= \nabla_\theta \left(\log Z_C + \int \rho(dr) \left(\text{dens}(D_\theta, s_I)(r) \cdot \log \frac{\text{dens}(D_\theta, s_I)(r)}{\text{dens}(C, s_I)(r)} \right) \right) \\ &= \nabla_\theta \int \rho(dr) \left(\text{dens}(D_\theta, s_I)(r) \cdot \log \frac{\text{dens}(D_\theta, s_I)(r)}{\text{dens}(C, s_I)(r)} \right) \end{aligned} \quad (14)$$

$$= \int \rho(dr) \nabla_{\theta} \left(\text{dens}(D_{\theta}, s_I)(r) \cdot \log \frac{\text{dens}(D_{\theta}, s_I)(r)}{\text{dens}(C, s_I)(r)} \right) \quad (15)$$

$$\begin{aligned} &= \int \rho(dr) \left(\left(\nabla_{\theta} \text{dens}(D_{\theta}, s_I)(r) \right) \cdot \log \frac{\text{dens}(D_{\theta}, s_I)(r)}{\text{dens}(C, s_I)(r)} \right) + \int \rho(dr) \nabla_{\theta} \text{dens}(D_{\theta}, s_I)(r) \\ &= \int \rho(dr) \left(\left(\nabla_{\theta} \text{dens}(D_{\theta}, s_I)(r) \right) \cdot \log \frac{\text{dens}(D_{\theta}, s_I)(r)}{\text{dens}(C, s_I)(r)} \right) + \nabla_{\theta} \int \rho(dr) \text{dens}(D_{\theta}, s_I)(r) \quad (16) \\ &= \int \rho(dr) \left(\text{dens}(D_{\theta}, s_I)(r) \cdot \left(\nabla_{\theta} \log \text{dens}(D_{\theta}, s_I)(r) \right) \cdot \log \frac{\text{dens}(D_{\theta}, s_I)(r)}{\text{dens}(C, s_I)(r)} \right) + \nabla_{\theta} 1 \\ &= \mathbb{E}_{\text{dens}(D_{\theta}, s_I)(r)} \left[\left(\nabla_{\theta} \log \text{dens}(D_{\theta}, s_I)(r) \right) \cdot \log \frac{\text{dens}(D_{\theta}, s_I)(r)}{\text{dens}(C, s_I)(r)} \right]. \end{aligned}$$

The well-definedness and the continuity of $(\nabla_{\theta} \text{KL}_{\theta})_j$ for all $j \in [p]$ follow from (14) and R4, which concludes the proof.

Note that R2 is used to guarantee the well-definedness of $\nabla_{\theta} \log \text{dens}(D_{\theta}, s_I)(r)$ in (13); R5 and R6 are used in (15) and (16), respectively. \square

C PROOFS OF EXAMPLES, LEMMA, AND THEOREMS IN §6

THEOREM 6.1. *Under our assumption in §6.1,*

$$\mathbb{E}_{\text{dens}(D_{\theta}, s_I)(r)} [|\log \text{dens}(D_{\theta}, s_I)(r)|] < \infty.$$

PROOF. For $i \in [M]$ and $r \in A_i$, define

$$g_{(i, \theta)}(r) \triangleq \prod_{\alpha \in K_i} \mathcal{N}(r(\alpha); \mu_{(i, \alpha)}(\theta), \sigma_{(i, \alpha)}(\theta)).$$

Then,

$$\begin{aligned} &\mathbb{E}_{\text{dens}(D_{\theta}, s_I)(r)} [|\log \text{dens}(D_{\theta}, s_I)(r)|] \\ &= \int \rho(dr) \left(\left(\sum_{i=1}^M \mathbb{1}_{[r \in A_i]} \cdot g_{(i, \theta)}(r) \right) \cdot \left| \log \sum_{i=1}^M (\mathbb{1}_{[r \in A_i]} \cdot g_{(i, \theta)}(r)) \right| \right) \\ &= \int \rho(dr) \left(\sum_{i=1}^M \mathbb{1}_{[r \in A_i]} \cdot g_{(i, \theta)}(r) \cdot \left| \log g_{(i, \theta)}(r) \right| \right) \\ &= \sum_{i=1}^M \int \rho(dr) \left(\mathbb{1}_{[r \in A_i]} \cdot g_{(i, \theta)}(r) \cdot \left| \log g_{(i, \theta)}(r) \right| \right) \\ &\leq \sum_{i=1}^M \int \rho(dr) \left(\mathbb{1}_{[r \in [K_i \rightarrow \mathbb{R}]]} \cdot g_{(i, \theta)}(r) \cdot \left| \log g_{(i, \theta)}(r) \right| \right) \\ &= \sum_{i=1}^M \mathbb{E}_{g_{(i, \theta)}(r)} [|\log g_{(i, \theta)}(r)|] \\ &< \infty. \end{aligned}$$

The last inequality uses a well-known result that the differential entropy of any multivariate normal distribution is finite. \square

EXAMPLE 6.2. Consider guides $D_{(i,\theta)}$ defined as follows ($i = 1, 2$):

$$D_{(i,\theta)} \equiv (x_1 := \text{sample}_{\text{norm}}("a_1", \theta_1, 1); x_2 := \text{sample}_{\text{norm}}("a_2", \theta_2, E_i[x_1]))$$

where for some $n \geq 1$ and $c \neq 0 \in \mathbb{R}$,

$$E_1[x_1] \equiv \text{if } (x_1=0) \text{ then } 1 \text{ else } \exp(-1/|x_1|^n), \quad E_2[x_1] \equiv \exp(\exp(c \cdot x_1^3)).$$

Then, the entropies of $\text{dens}(D_{(i,\theta)}, s_I)$'s are all undefined.

PROOF. We prove that

$$\mathbb{E}_{\text{dens}(D_{(i,\theta)}, s_I)(r)}[|\log \text{dens}(D_{(i,\theta)}, s_I)(r)|] = \infty \quad \text{for all } i = 1, 2$$

by bounding the the expectation as follows.

$$\begin{aligned} & \mathbb{E}_{\text{dens}(D_{(i,\theta)}, s_I)(r)}[|\log \text{dens}(D_{(i,\theta)}, s_I)(r)|] \\ &= \mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1) \cdot \mathcal{N}(x_2; \theta_2, E_i[x_1])} [|\log \mathcal{N}(x_1; \theta_1, 1) + \log \mathcal{N}(x_2; \theta_2, E_i[x_1])|] \\ &\geq -\mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1) \cdot \mathcal{N}(x_2; \theta_2, E_i[x_1])} [|\log \mathcal{N}(x_1; \theta_1, 1)|] \\ &\quad + \mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1) \cdot \mathcal{N}(x_2; \theta_2, E_i[x_1])} [|\log \mathcal{N}(x_2; \theta_2, E_i[x_1])|] \end{aligned}$$

Here the two terms in the last equation are bounded as follows.

$$\begin{aligned} & -\mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1) \cdot \mathcal{N}(x_2; \theta_2, E_i[x_1])} [|\log \mathcal{N}(x_1; \theta_1, 1)|] \\ &= -\mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1)} \left[\left| \log \sqrt{2\pi} - \frac{1}{2}(x_1 - \theta_1)^2 \right| \right] \\ &\geq -\mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1)} \left[\log \sqrt{2\pi} \right] - \frac{1}{2} \mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1)} [(x_1 - \theta_1)^2] \\ &= -\log \sqrt{2\pi} - \frac{1}{2} \cdot 1^2 \\ & \mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1) \cdot \mathcal{N}(x_2; \theta_2, E_i[x_1])} [|\log \mathcal{N}(x_2; \theta_2, E_i[x_1])|] \\ &= \mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1) \cdot \mathcal{N}(x_2; \theta_2, E_i[x_1])} \left[\left| -\log \sqrt{2\pi} - \frac{(x_2 - \theta_2)^2}{2E_i[x_1]^2} - \log E_i[x_1] \right| \right] \\ &\geq -\mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1)} \left[\log \sqrt{2\pi} \right] - \frac{1}{2} \mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1)} \left[\underbrace{\mathbb{E}_{\mathcal{N}(x_2; \theta_2, E_i[x_1])} [(x_2 - \theta_2)^2]}_{E_i[x_1]^2} \cdot \frac{1}{E_i[x_1]^2} \right] \\ &\quad + \mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1)} [|\log E_i[x_1]|] \\ &= -\log \sqrt{2\pi} - \frac{1}{2} + \mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1)} [|\log E_i[x_1]|] \end{aligned}$$

Note that

$$\mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1)} [|\log E_i[x_1]|] = \begin{cases} \mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1)} [\mathbb{1}_{[x_1 \neq 0]} \cdot 1/|x_1|^n] & \text{for } i = 1 \\ \mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1)} [\exp(c \cdot x_1^3)] & \text{for } i = 2 \end{cases} = \infty$$

because $\int_{[0, \epsilon]} 1/v^n dv = \infty$ for any $\epsilon > 0$ and $n \geq 1$, and $\mathbb{E}_{\mathcal{N}(v; \mu, \sigma)} [\exp(c \cdot v^3)] = \infty$ for any $\mu \in \mathbb{R}$ and $\sigma \in \mathbb{R}_{>0}$. Hence, we obtain the desired result. \square

THEOREM 6.3. Under our assumption in §6.1, the condition (9) implies

$$\mathbb{E}_{\text{dens}(D_{\theta}, s_I)(r)} [|\log \text{dens}(C, s_I)(r)|] < \infty,$$

and thus the requirement R3 (i.e., the objective in (5) is well-defined).

PROOF. For $i \in [M]$, $j \in [N_i]$, $\alpha \in K_i$, and $r \in A_i$, let

$$g_{(i,\theta)}(r) \triangleq \prod_{\beta \in K_i} \mathcal{N}(r(\beta); \mu_{(i,\beta)}(\theta), \sigma_{(i,\beta)}(\theta)),$$

$$h_{(i,\alpha)}(r) \triangleq \mathcal{N}(r(\alpha); \mu'_{(i,\alpha)}(r), \sigma'_{(i,\alpha)}(r)), \quad k_{(i,j)}(r) \triangleq \mathcal{N}(c_{(i,j)}; \mu''_{(i,j)}(r), \sigma''_{(i,j)}(r)).$$

We note that

$$\begin{aligned} \mathbb{E}_{g_{(i,\theta)}(r)} [\log h_{(i,\alpha)}(r)] &= \mathbb{E}_{g_{(i,\theta)}(r)} \left[-\log \sqrt{2\pi} - \log \sigma'_{(i,\alpha)}(r) - \frac{(r(\alpha) - \mu'_{(i,\alpha)}(r))^2}{2 (\sigma'_{(i,\alpha)}(r))^2} \right] \\ &\leq \log \sqrt{2\pi} + \mathbb{E}_{g_{(i,\theta)}(r)} [\log \sigma'_{(i,\alpha)}(r)] + \mathbb{E}_{g_{(i,\theta)}(r)} \left[\frac{(r(\alpha) - \mu'_{(i,\alpha)}(r))^2}{2 (\sigma'_{(i,\alpha)}(r))^2} \right]. \end{aligned}$$

Thus, we can have

$$\mathbb{E}_{g_{(i,\theta)}(r)} [\log h_{(i,\alpha)}(r)] < \infty \quad (17)$$

if the following two inequalities hold:

$$\mathbb{E}_{g_{(i,\theta)}(r)} [\log \sigma'_{(i,\alpha)}(r)] < \infty \quad \text{and} \quad \mathbb{E}_{g_{(i,\theta)}(r)} \left[\frac{(r(\alpha) - \mu'_{(i,\alpha)}(r))^2}{(\sigma'_{(i,\alpha)}(r))^2} \right] < \infty. \quad (18)$$

We show that the inequalities are indeed true. By the condition (9) and the parts (a) and (b) of Lemma C.1 there exist functions $f_i, u_i \in \mathcal{A}_i$ such that for every $r \in A_i$,

$$|\log \sigma'_{(i,\alpha)}(r)| \leq \exp(f_i(r)) \quad \text{and} \quad \frac{(r(\alpha) - \mu'_{(i,\alpha)}(r))^2}{(\sigma'_{(i,\alpha)}(r))^2} \leq \exp(u_i(r)). \quad (19)$$

For every $f \in \mathcal{A}_i$, by the part (e) of Lemma C.1 we have that

$$\mathbb{E}_{g_{(i,\theta)}(r)} [\exp(f(r))] < \infty. \quad (20)$$

The inequalities in (18) follow from the properties in (19) and (20).

By an argument similar to what we have just given, we also have that

$$\mathbb{E}_{g_{(i,\theta)}(r)} [\log k_{(i,j)}(r)] < \infty. \quad (21)$$

Using (17) and (21), we calculate the claim of the theorem:

$$\begin{aligned} &\mathbb{E}_{dens(D_\theta, s_I)(r)} [\log dens(C, s_I)(r)] \\ &= \int \rho(dr) \left(\left(\sum_{i=1}^M \mathbb{1}_{[r \in A_i]} \cdot g_{(i,\theta)}(r) \right) \cdot \left| \log \sum_{i=1}^M \left(\mathbb{1}_{[r \in A_i]} \cdot \prod_{\beta \in K_i} h_{(i,\beta)}(r) \cdot \prod_{j \in [N_i]} k_{(i,j)}(r) \right) \right| \right) \\ &= \int \rho(dr) \left(\sum_{i=1}^M \mathbb{1}_{[r \in A_i]} \cdot g_{(i,\theta)}(r) \cdot \left| \log \left(\prod_{\beta \in K_i} h_{(i,\beta)}(r) \cdot \prod_{j \in [N_i]} k_{(i,j)}(r) \right) \right| \right) \\ &= \sum_{i=1}^M \int \rho(dr) \left(\mathbb{1}_{[r \in A_i]} \cdot g_{(i,\theta)}(r) \cdot \left| \sum_{\beta \in K_i} \log h_{(i,\beta)}(r) + \sum_{j \in [N_i]} \log k_{(i,j)}(r) \right| \right) \end{aligned}$$

$$\begin{aligned}
&\leq \sum_{i=1}^M \sum_{\beta \in K_i} \int \rho(dr) (\mathbb{1}_{[r \in A_i]} \cdot g_{(i,\theta)}(r) \cdot |\log h_{(i,\beta)}(r)|) \\
&\quad + \sum_{i=1}^M \sum_{j \in [N_i]} \int \rho(dr) (\mathbb{1}_{[r \in A_i]} \cdot g_{(i,\theta)}(r) \cdot |\log k_{(i,j)}(r)|) \\
&\leq \sum_{i=1}^M \sum_{\beta \in K_i} \int \rho(dr) (\mathbb{1}_{[r \in [K_i \rightarrow \mathbb{R}]]} \cdot g_{(i,\theta)}(r) \cdot |\log h_{(i,\beta)}(r)|) \\
&\quad + \sum_{i=1}^M \sum_{j \in [N_i]} \int \rho(dr) (\mathbb{1}_{[r \in [K_i \rightarrow \mathbb{R}]]} \cdot g_{(i,\theta)}(r) \cdot |\log k_{(i,j)}(r)|) \\
&= \sum_{i=1}^M \sum_{\beta \in K_i} \mathbb{E}_{g_{(i,\theta)}(r)} [|\log h_{(i,\beta)}(r)|] + \sum_{i=1}^M \sum_{j \in [N_i]} \mathbb{E}_{g_{(i,\theta)}(r)} [|\log k_{(i,j)}(r)|] \\
&< \infty.
\end{aligned}$$

The last inequality follows from (17) and (21). Finally, the requirement R3 follows from the above and Theorem 6.1:

$$\begin{aligned}
&\int \rho(dr) \left| \text{dens}(D_\theta, s_I)(r) \cdot \log \frac{\text{dens}(D_\theta, s_I)(r)}{\text{dens}(C, s_I)(r)} \right| \\
&\leq \mathbb{E}_{\text{dens}(D_\theta, s_I)(r)} [|\log \text{dens}(D_\theta, s_I)(r)|] + \mathbb{E}_{\text{dens}(D_\theta, s_I)(r)} [|\log \text{dens}(C, s_I)(r)|] \\
&< \infty.
\end{aligned}$$

□

EXAMPLE 6.4. Consider models C_1, \dots, C_4 and a guide D_θ defined as follows:

$$C_i \equiv (x_1 := \text{sample}_{\text{norm}}("a_1", 0, 1); x_2 := \text{sample}_{\text{norm}}("a_2", E_i[x_1], 1)) \quad \text{for } i = 1, 2$$

$$C_i \equiv (x_1 := \text{sample}_{\text{norm}}("a_1", 0, 1); x_2 := \text{sample}_{\text{norm}}("a_2", 0, E_i[x_1])) \quad \text{for } i = 3, 4$$

$$D_\theta \equiv (x_1 := \text{sample}_{\text{norm}}("a_1", \theta_1, 1); x_2 := \text{sample}_{\text{norm}}("a_2", \theta_2, 1))$$

where for some $n \geq 1$ and $c \neq 0 \in \mathbb{R}$,

$$E_1[x_1] \equiv \text{if } (x_1=0) \text{ then } 0 \text{ else } 1/x_1^n,$$

$$E_2[x_1] \equiv E_4[x_1] \equiv \exp(c \cdot x_1^3),$$

$$E_3[x_1] \equiv \text{if } (x_1=0) \text{ then } 1 \text{ else } |x_1|^n.$$

Then, the requirement R3 does not hold (i.e., the objective in (5) is undefined) for all $i = 1, \dots, 4$.

PROOF. Since we have

$$\begin{aligned}
\int \rho(dr) \left| \text{dens}(D_\theta, s_I)(r) \cdot \log \frac{\text{dens}(D_\theta, s_I)(r)}{\text{dens}(C_i, s_I)(r)} \right| &\geq -\mathbb{E}_{\text{dens}(D_\theta, s_I)(r)} [|\log \text{dens}(D_\theta, s_I)(r)|] \\
&\quad + \mathbb{E}_{\text{dens}(D_\theta, s_I)(r)} [|\log \text{dens}(C_i, s_I)(r)|] \\
&\mathbb{E}_{\text{dens}(D_\theta, s_I)(r)} [|\log \text{dens}(D_\theta, s_I)(r)|] < \infty,
\end{aligned}$$

it suffices to show that

$$\mathbb{E}_{\text{dens}(D_\theta, s_I)(r)} [|\log \text{dens}(C_i, s_I)(r)|] = \infty \quad \text{for all } i = 1, \dots, 4. \quad (22)$$

Case of $i = 1, 2$: The quantity in (22) is bounded as follows.

$$\begin{aligned}
&\mathbb{E}_{\text{dens}(D_\theta, s_I)(r)} [|\log \text{dens}(C_i, s_I)(r)|] \\
&= \mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1) \cdot \mathcal{N}(x_2; \theta_2, 1)} [|\log \mathcal{N}(x_1; 0, 1) + \log \mathcal{N}(x_2; E_i[x_1], 1)|]
\end{aligned}$$

$$\begin{aligned} &\geq -\mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1) \cdot \mathcal{N}(x_2; \theta_2, 1)} [|\log \mathcal{N}(x_1; 0, 1)|] \\ &\quad + \mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1) \cdot \mathcal{N}(x_2; \theta_2, 1)} [|\log \mathcal{N}(x_2; E_i[x_1], 1)|] \end{aligned}$$

Here the two terms in the last equation are bounded as follows.

$$\begin{aligned} &-\mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1) \cdot \mathcal{N}(x_2; \theta_2, 1)} [|\log \mathcal{N}(x_1; 0, 1)|] \\ &= -\mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1)} \left[\left| \log \sqrt{2\pi} - \frac{1}{2}x_1^2 \right| \right] \\ &\geq -\mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1)} \left[\log \sqrt{2\pi} \right] - \frac{1}{2} \mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1)} [x_1^2] \\ &= -\log \sqrt{2\pi} - \frac{1}{2}(1 + \theta^2) \\ &\mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1) \cdot \mathcal{N}(x_2; \theta_2, 1)} [|\log \mathcal{N}(x_2; E_i[x_1], 1)|] \\ &= \mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1) \cdot \mathcal{N}(x_2; \theta_2, 1)} \left[\left| -\log \sqrt{2\pi} - \frac{1}{2}(x_2 - E_i[x_1])^2 \right| \right] \\ &\geq -\mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1)} \left[\log \sqrt{2\pi} \right] + \frac{1}{2} \mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1) \cdot \mathcal{N}(x_2; \theta_2, 1)} [x_2^2 - 2x_2 \cdot E_i[x_1] + E_i[x_1]^2] \\ &\geq -\log \sqrt{2\pi} - \frac{1}{2} \mathbb{E}_{\mathcal{N}(x_2; \theta_2, 1)} [x_2^2] + \frac{1}{2} \mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1) \cdot \mathcal{N}(x_2; \theta_2, 1)} [E_i[x_1]^2 - 2x_2 \cdot E_i[x_1]] \\ &\geq -\log \sqrt{2\pi} - \frac{1}{2}(1 + \theta_2^2) + \frac{1}{2} \mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1) \cdot \mathcal{N}(x_2; \theta_2, 1)} [E_i[x_1]^2 - 2x_2 \cdot E_i[x_1]] \end{aligned}$$

We bound the last term in the last equation as follows, assuming $c > 0$.

$$\begin{aligned} &\mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1) \cdot \mathcal{N}(x_2; \theta_2, 1)} [E_i[x_1]^2 - 2x_2 \cdot E_i[x_1]] \\ &\geq \mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1) \cdot \mathcal{N}(x_2; \theta_2, 1)} \left[\mathbb{1}_{[x_1 \geq 0 \wedge x_2 \leq 0]} \cdot \underbrace{E_i[x_1]^2}_{\geq 0} + \underbrace{(-2x_2) \cdot E_i[x_1]}_{\geq 0} \right] \\ &\geq \mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1) \cdot \mathcal{N}(x_2; \theta_2, 1)} \left[\mathbb{1}_{[x_1 \geq 0 \wedge x_2 \leq 0]} \cdot (E_i[x_1]^2 + (-2x_2) \cdot E_i[x_1]) \right] \\ &\geq \mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1)} \left[\mathbb{1}_{[x_1 \geq 0]} \cdot E_i[x_1]^2 \right] \cdot \mathbb{E}_{\mathcal{N}(x_2; \theta_2, 1)} \left[\mathbb{1}_{[x_2 \leq 0]} \right] \\ &\quad + \mathbb{E}_{\mathcal{N}(x_1; \theta_1, 1)} \left[\mathbb{1}_{[x_1 \geq 0]} \cdot E_i[x_1] \right] \cdot \mathbb{E}_{\mathcal{N}(x_2; \theta_2, 1)} \left[\mathbb{1}_{[x_2 \leq 0]} \cdot (-2x_2) \right] \\ &\geq \infty \quad \text{for all } i = 1, 2 \end{aligned}$$

Here the last inequality comes from that the two expectations over x_2 are finite while the other two over x_1 are both ∞ , because $\int_{[0, \epsilon]} 1/v^n dv = \infty$ for any $\epsilon > 0$ and $n \geq 1$, and $\mathbb{E}_{\mathcal{N}(v; \mu, \sigma)}[\exp(c \cdot v^3)] = \infty$ for any $\mu \in \mathbb{R}$ and $\sigma \in \mathbb{R}_{>0}$. Note that we can use a similar calculation for the case of $c < 0$. This proves the equality in (22).

Case of $i = 3, 4$: The proof is similar to the above case, so we omit it. Remark that in the proof, we additionally use the following fact: $\int_{[0, 1]} |\log v| dv < \infty$. \square

LEMMA C.1. *Pick $i \in [M]$. Let $(\alpha_1, \dots, \alpha_J)$ be an enumeration of the elements in K_i , and $\bar{r} \triangleq (r(\alpha_1), \dots, r(\alpha_J)) \in \mathbb{R}^J$ for $r \in A_i$.*

(a) *For every $l_1, l_2 \in \mathcal{A}_i$ and every $\odot \in \{+, -, \times, /, \max\}$, there exists $l \in \mathcal{A}_i$ such that*

$$\exp(l_1(r)) \odot \exp(l_2(r)) \leq \exp(l(r)) \text{ for all } r \in A_i.$$

(b) *For every polynomial $\text{poly} : \mathbb{R}^J \rightarrow \mathbb{R}$, there exists $l \in \mathcal{A}_i$ such that*

$$|\text{poly}(\bar{r})| \leq \exp(l(r)) \text{ for all } r \in A_i.$$

In particular, for every $l \in \mathcal{A}_i$, there exists $l' \in \mathcal{A}_i$ such that

$$|l(r)| \leq \exp(l'(r)) \text{ for all } r \in A_i.$$

(c) For every affine-bounded neural network $nn : \mathbb{R}^J \rightarrow \mathbb{R}$, there exists $l \in \mathcal{A}_i$ such that

$$|nn(\bar{r})| \leq l(r) \text{ for all } r \in A_i.$$

(d) There exist affine functions $l_1, l_2 : \mathbb{R} \rightarrow \mathbb{R}$ such that

$$\exp(l_1(|v|)) \leq \text{softplus}(v) \leq \exp(l_2(|v|)) \text{ for all } v \in \mathbb{R}.$$

(e) For every $l \in \mathcal{A}_i$ and every J -dimensional normal distribution q ,

$$\mathbb{E}_{q(r(\alpha_1), \dots, r(\alpha_J))} [\exp(l(r))] < \infty.$$

Proof of (a). We split the cases.

- Case $\odot \in \{\times, /\}$: Take $l = l_1 \pm l_2$.
- Case $\odot = \max$: For $m = 1, 2$, let

$$l_m(r) = c_{m,0} + \sum_{j=1}^J c_{m,j} \cdot |r(\alpha_j)|$$

where $c_{m,j} \in \mathbb{R}$. For all $r \in A_i$, we have

$$\max(l_1(r), l_2(r)) \leq \max\left(c_{1,0} + \sum_{j=1}^J |c_{1,j}| \cdot |r(\alpha_j)|, c_{2,0} + \sum_{j=1}^J |c_{2,j}| \cdot |r(\alpha_j)|\right) \leq l(r),$$

where

$$l(r) \triangleq \max(c_{1,0}, c_{2,0}) + \sum_{j=1}^J \max(|c_{1,j}|, |c_{2,j}|) \cdot |r(\alpha_j)| \in \mathcal{A}_i.$$

Hence, for every $r \in A_i$,

$$\max(\exp(l_1(r)), \exp(l_2(r))) = \exp(\max(l_1(r), l_2(r))) \leq \exp(l(r)).$$

- Case $\odot \in \{+, -\}$: Using the case $\odot = \max$, we have the following for some $l \in \mathcal{A}_i$:

$$\begin{aligned} \exp(l_1(r)) + \exp(l_2(r)) &\leq 2 \cdot \max(\exp(l_1(r)), \exp(l_2(r))) \\ &\leq \exp(\log 2) \cdot \exp(l(r)) \\ &= \exp(l(r) + \log 2). \end{aligned}$$

Proof of (b). Let n be an index in $\{1, \dots, J\}$ such that

$$\text{poly}(r(\alpha_1), \dots, r(\alpha_J)) = \text{poly}'(r(\alpha_1), \dots, r(\alpha_n))$$

for some polynomial $\text{poly}' : \mathbb{R}^n \rightarrow \mathbb{R}$. We show that there exists $l \in \mathcal{A}_i$ satisfying

$$\text{poly}'(r(\alpha_1), \dots, r(\alpha_n)) \leq \exp(l(r)).$$

Note that the claim of the lemma follows from this. Our proof is by induction on n .

- Case $n = 1$: Let $v \in \mathbb{R}$ and $\text{poly}'(v) = \sum_{d=0}^D c_d \cdot v^d$ where $c_d \in \mathbb{R}$. We use the following fact.
FACT: For every $d \in \mathbb{N}$, there exists an affine function $l_d : \mathbb{R} \rightarrow \mathbb{R}$ such that

$$|v|^d \leq \exp(l_d(|v|)) \text{ for all } v \in \mathbb{R}.$$

PROOF. Since $v^d = O(\exp(v))$ for positive $v \in \mathbb{R}$, there exist $C_1, C_2 > 0$ such that $|v| > C_1$ implies $|v|^d < C_2 \exp(|v|)$. Since $|v|^d$ is continuous, $C_3 \triangleq \sup\{|v|^d \mid |v| \leq C_1\}$ satisfies $0 \leq C_3 < \infty$. Putting these together, for every $v \in \mathbb{R}$,

$$\begin{aligned} |v|^d &\leq \max(C_2 \exp(|v|), C_3) \\ &\leq \max(\max(C_2, 1) \cdot \exp(|v|), C_3 + 1) \\ &\leq C_4 \exp(|v|), \end{aligned}$$

where $C_4 \triangleq \max(C_2, 1) \cdot (C_3 + 1) \geq 1$. Thus, $l_d(v) \triangleq (v + \log C_4)$ is a desired affine function. \square

Using the fact, we have

$$\begin{aligned} |\text{poly}'(r(\alpha_1))| &\leq \sum_{d=0}^D |c_d| \cdot |r(\alpha_1)|^d \\ &\leq \sum_{d=0}^D |c_d| \cdot \exp(l_d(|r(\alpha_1)|)) = \sum_{d=0, c_d \neq 0}^D \exp(l'_d(r)) \\ &\leq \exp(l(r)) \end{aligned}$$

for some $l'_d, l \in \mathcal{A}_i$. The existence of l satisfying the last inequality follows from (a).

- Case $n > 1$: Let $v \in \mathbb{R}^n$, and define $v_{1:n-1}$ to be the projection of v to its first $n-1$ components. Also, let

$$\text{poly}'(v) = \sum_{d=0}^D \text{poly}'_d(v_{1:n-1}) \cdot v_n^d$$

where $\text{poly}'_d : \mathbb{R}^{n-1} \rightarrow \mathbb{R}$ is a polynomial. Using the induction hypothesis, we have

$$\begin{aligned} |\text{poly}'(r(\alpha_1), \dots, r(\alpha_n))| &\leq \sum_{d=0}^D |\text{poly}'_d(r(\alpha_1), \dots, r(\alpha_{n-1}))| \cdot |r(\alpha_n)|^d \\ &\leq \sum_{d=0}^D \exp(l'_d(r)) \cdot \exp(l_d(|r(\alpha_n)|)) = \sum_{d=0}^D \exp(l''_d(r)) \\ &\leq \exp(l(r)) \end{aligned}$$

for some affine functions $l_d : \mathbb{R} \rightarrow \mathbb{R}$, and functions $l'_d, l''_d, l \in \mathcal{A}_i$. The last inequality follows from (a).

Proof of (c). By the definition of affine-bounded neural network, there exist functions

$$f_1 : \mathbb{R}^{n_1} \rightarrow \mathbb{R}^{n_2}, \quad f_2 : \mathbb{R}^{n_2} \rightarrow \mathbb{R}^{n_3}, \quad \dots, \quad f_d : \mathbb{R}^{n_d} \rightarrow \mathbb{R}^{n_{d+1}}$$

and affine functions

$$l'_j : \mathbb{R}^{n_j} \rightarrow \mathbb{R} \quad \text{for all } 1 \leq j \leq d$$

such that (i) $n_1 = J$ and $n_{d+1} = 1$; (ii) $nn = f_d \circ \dots \circ f_1$; and (iii) for all $1 \leq j \leq d$ and $v \in \mathbb{R}^{n_j}$,

$$\sum_{k=1}^{n_{j+1}} |f_j(v)_k| \leq l'_j(|v_1|, \dots, |v_{n_j}|).$$

We prove the claim of the lemma by induction on the depth d . For $d = 1$, the claim follows immediately from the defining property (iii) of the affine-bounded neural network nn . Assume

$d > 1$. By induction hypothesis, there exist affine functions $l_1, \dots, l_{n_d} : \mathbb{R}^J \rightarrow \mathbb{R}$ such that for all $k \in \{1, \dots, n_d\}$ and $v \in \mathbb{R}^J$,

$$|(f_{d-1} \circ \dots \circ f_1)(v)_k| \leq l_k(|v_1|, \dots, |v_J|)$$

This is because the $(f_{d-1} \circ \dots \circ f_1)(v)_k$ are affine-bounded neural networks of depth $d - 1$. Let

$$l'_d(v_1, \dots, v_{n_d}) = c_0 + \sum_{k=1}^{n_d} c_k \cdot |v_k|.$$

Using what we have proved so far, we calculate:

$$\begin{aligned} |nn(v)| &= |f_d((f_{d-1} \circ \dots \circ f_1)(v))| \\ &\leq l'_d(|(f_{d-1} \circ \dots \circ f_1)(v)_1|, \dots, |(f_{d-1} \circ \dots \circ f_1)(v)_{n_d}|) \\ &= c_0 + \sum_{k=1}^{n_d} c_k \cdot |(f_{d-1} \circ \dots \circ f_1)(v)_k| \\ &\leq c_0 + \sum_{k=1}^{n_d} |c_k| \cdot |(f_{d-1} \circ \dots \circ f_1)(v)_k| \\ &\leq c_0 + \sum_{k=1}^{n_d} |c_k| \cdot l_k(|v_1|, \dots, |v_J|). \end{aligned}$$

Let $l(r) = c_0 + \sum_{k=1}^{n_d} |c_k| \cdot l_k(|r(\alpha_1)|, \dots, |r(\alpha_J)|)$. The function l is essentially the composition of two affine functions, and so it belongs to \mathcal{A}_i . This and the conclusion of the above calculation show that l is the function in \mathcal{A}_i that we look for.

Proof of (d). Since $\log(1 + v) \geq v/2$ for all $v \in [0, 1]$, we have

$$\begin{aligned} \log(1 + \exp(v)) &\geq \exp(v)/2 = \exp(-|v| - \log 2) && \text{for all } v \in (-\infty, 0], \quad \text{and} \\ \log(1 + \exp(v)) &\leq \log(2 \cdot \exp(v)) = |v| + \log 2 && \text{for all } v \in [0, \infty). \end{aligned}$$

Let $l_1(v) = -v - \log 2$ and $l_2(v) = v + \log 2$. As $\text{softplus}(-)$ is increasing, we have

$$\exp(l_1(|v|)) \leq \text{softplus}(v) \leq \exp(l_2(|v|)) \quad \text{for all } v \in \mathbb{R}.$$

Proof of (e). Let $l(r) = c_0 + \sum_{j=1}^J c_j \cdot |r(\alpha_j)|$, where $c_j \in \mathbb{R}$. The moment generating function $G_p : \mathbb{R}^J \rightarrow \mathbb{R}$ of a random variable of density $p(v)$ on \mathbb{R}^J is defined by $G_p(t) \triangleq \mathbb{E}_{p(v)} [\exp(t \cdot v)]$ where $t \cdot v$ is the inner product of t and v . It is well-known that the moment generating functions are well defined for multivariate normal distributions $q(v)$ on \mathbb{R}^J in the lemma, i.e., $G_q(t) < \infty$ for all $t \in \mathbb{R}^J$. From this fact and that $\exp(|v|) \leq \exp(v) + \exp(-v)$, it follows that

$$\begin{aligned} \mathbb{E}_{q(r(\alpha_1), \dots, r(\alpha_J))} [\exp(l(r))] &= \exp(c_0) \cdot \mathbb{E}_{q(r(\alpha_1), \dots, r(\alpha_J))} \left[\exp \left(\sum_{j=1}^J c_j \cdot |r(\alpha_j)| \right) \right] \\ &\leq \exp(c_0) \cdot \sum_{k_1, \dots, k_J \in \{0, 1\}} G_q \left((-1)^{k_1} c_1, \dots, (-1)^{k_J} c_J \right) \\ &< \infty. \end{aligned}$$

□

The proof of Theorem 6.6 uses two more ingredients: Theorem C.2 which, compared to Theorem 6.7, additionally guarantees the continuity of $\nabla_v \int f_v \, d\mu$ by assuming the continuity of $\nabla_v f_v$;

and Lemma C.3 which bounds the densities of multiple normal distributions using the density of a single, fixed normal distribution.

THEOREM C.2. *Let $V \subset \mathbb{R}$ be an open interval, and (X, Σ, μ) be a measure space. Suppose that a measurable function $f : V \times X \rightarrow \mathbb{R}$ satisfies the following conditions:*

- (a) *The function $v \in V \mapsto \int \mu(dx) f_v(x) \in \mathbb{R}$ is well-defined.*
- (b) *For almost all $x \in X$ (w.r.t. μ), the function $v \in V \mapsto \nabla_v f_v(x) \in \mathbb{R}$ is well-defined, continuous.*
- (c) *There is a measurable function $h : X \rightarrow \mathbb{R}$ such that $\int \mu(dx) h(x)$ is well-defined and $|\nabla_v f_v(x)| \leq h(x)$ for all $v \in V$ and almost all $x \in X$ (w.r.t. μ).*

Then, for all $v \in V$, both sides of the below equation are well-defined, and the equality holds:

$$\nabla_v \int \mu(dx) f_v(x) = \int \mu(dx) \nabla_v f_v(x). \quad (23)$$

Moreover, the function $v \in V \mapsto \nabla_v \int \mu(dx) f_v(x)$ is continuous.

PROOF. Theorem 6.7 implies the former part of the conclusion: for all $v \in V$, each side of (23) is well-defined and the equation (23) holds. To prove the latter part of the conclusion, define $G : V \rightarrow \mathbb{R}$ and $g : V \times X \rightarrow \mathbb{R}$ by

$$G(v) \triangleq \nabla_v \int \mu(dx) f_v(x) = \int \mu(dx) g(v, x), \quad g(v, x) \triangleq \nabla_v f_v(x),$$

and let $(v_n)_{n \in \mathbb{N}}$ be any sequence in V such that $\lim_{n \rightarrow \infty} v_n = v^* \in V$. Then, it suffices to show that

$$\lim_{n \rightarrow \infty} G(v_n) = G(v^*).$$

To apply the dominated convergence theorem, define $g_n, g^* : X \rightarrow \mathbb{R}$ by $g_n(x) \triangleq g(v_n, x)$ and $g^*(x) \triangleq g(v^*, x)$. By (b), the function $g(\cdot, x) : V \rightarrow \mathbb{R}$ is continuous for almost all $x \in X$ (w.r.t. μ), so

$$\lim_{n \rightarrow \infty} g_n(x) = \lim_{n \rightarrow \infty} g(v_n, x) = g(v^*, x) = g^*(x) \quad \text{for almost all } x \in X \text{ (w.r.t. } \mu).$$

By (c), for all $n \in \mathbb{N}$, $|g_n(x)| \leq h(x)$ for almost all $x \in X$ (w.r.t. μ), and $h(x)$ is integrable (w.r.t. μ). Therefore, we can apply the dominated convergence theorem to $(g_n)_{n \in \mathbb{N}}$ and g^* with h , which gives:

$$\lim_{n \rightarrow \infty} G(v_n) = \lim_{n \rightarrow \infty} \int \mu(dx) g_n(x) = \int \mu(dx) g^*(x) = G(v^*).$$

□

LEMMA C.3. *Suppose $[\mu_1, \mu_2] \subset \mathbb{R}$ and $[\sigma_1, \sigma_2] \subset (0, \infty)$. Then, there exists $c \in \mathbb{R}$ such that*

$$\mathcal{N}(v; \mu, \sigma) \leq c \cdot \mathcal{N}(v; 0, 2\sigma_2)$$

for all $v \in \mathbb{R}$, $\mu \in [\mu_1, \mu_2]$, and $\sigma \in [\sigma_1, \sigma_2]$.

PROOF. We claim that $c = \frac{2\sigma_2}{\sigma_1} \exp(\frac{1}{6\sigma_1^2} \max(\mu_1^2, \mu_2^2))$ is the desired coefficient. For the constant c , the claim we need show is that

$$\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(v-\mu)^2}{2\sigma^2}\right) \leq \frac{1}{\sqrt{2\pi}\sigma_1} \exp\left(-\frac{v^2}{8\sigma_2^2} + \frac{1}{6\sigma_1^2} \max(\mu_1^2, \mu_2^2)\right)$$

for any $v \in \mathbb{R}$, $\mu \in [\mu_1, \mu_2]$, and $\sigma \in [\sigma_1, \sigma_2]$. Since $\frac{1}{\sqrt{2\pi}\sigma_1} \geq \frac{1}{\sqrt{2\pi}\sigma}$, it suffices to show

$$\begin{aligned} & -\frac{v^2}{8\sigma_2^2} + \frac{1}{6\sigma_1^2} \max(\mu_1^2, \mu_2^2) \geq -\frac{(v-\mu)^2}{2\sigma^2} \\ \iff & \frac{1}{2\sigma^2} \left(av^2 - 2\mu v + \mu^2 + \frac{\sigma^2}{3\sigma_1^2} \max(\mu_1^2, \mu_2^2) \right) \geq 0, \quad \text{where } a = 1 - \frac{\sigma^2}{4\sigma_2^2} \end{aligned}$$

$$\begin{aligned}
&\Longleftrightarrow a \left(v - \frac{\mu}{a}\right)^2 + \left(1 - \frac{1}{a}\right) \mu^2 + \frac{\sigma^2}{3\sigma_1^2} \max(\mu_1^2, \mu_2^2) \geq 0 \\
&\Longleftarrow \left(1 - \frac{1}{a}\right) \mu^2 + \frac{\sigma^2}{3\sigma_1^2} \max(\mu_1^2, \mu_2^2) \geq 0, \quad \text{since } a = 1 - \frac{\sigma^2}{4\sigma_2^2} \geq \frac{3}{4}.
\end{aligned}$$

The last inequality holds because

$$\left(1 - \frac{1}{a}\right) \mu^2 + \frac{\sigma^2}{3\sigma_1^2} \max(\mu_1^2, \mu_2^2) \geq -\frac{1}{3} \mu^2 + \frac{1}{3} \max(\mu_1^2, \mu_2^2) \geq 0.$$

□

THEOREM 6.6. *If both our assumption in §6.1 and the condition (9) hold, then the condition (10) implies the requirements R4-R6.*

PROOF. Before starting the proof, we note that by the condition (10), there is an open interval $V_{(i,\alpha,j,\theta)} \subseteq \mathbb{R}$ for all $i \in [M]$, $\alpha \in K_i$, $j \in [p]$ and $\theta \in \mathbb{R}^p$, such that

- $V_{(i,\alpha,j,\theta)}$ contains θ_j ;
- the function $v \in \mathbb{R} \mapsto \mu_{(i,\alpha)}(\theta[j : v])$ is continuously differentiable in $V_{(i,\alpha,j,\theta)}$;
- the function $v \in \mathbb{R} \mapsto \sigma_{(i,\alpha)}(\theta[j : v])$ is continuously differentiable in $V_{(i,\alpha,j,\theta)}$, and $\inf\{\sigma_{(i,\alpha)}(\theta[j : v]) \mid v \in V_{(i,\alpha,j,\theta)}\} > 0$.

Let F_1 and F_2 be functions from $\mathbb{R}^p \times RDB$ to \mathbb{R} defined as follows: for all $\theta \in \mathbb{R}^p$ and $r \in RDB$,

$$F_1(\theta, r) \triangleq \text{dens}(D_\theta, s_I)(r) \cdot \log \frac{\text{dens}(D_\theta, s_I)(r)}{\text{dens}(C, s_I)(r)}, \quad F_2(\theta, r) \triangleq \text{dens}(D_\theta, s_I)(r).$$

We have to show that for every $\theta \in \mathbb{R}^p$ and $j \in [p]$,

- the partial derivatives

$$\nabla_{\theta_j} \int \rho(dr) F_1(\theta, r) \quad \text{and} \quad \nabla_{\theta_j} \int \rho(dr) F_2(\theta, r)$$

are well-defined and are continuous functions of $\theta_j \in \mathbb{R}$;

- these derivatives commute with integration:

$$\nabla_{\theta_j} \int \rho(dr) F_1(\theta, r) = \int \rho(dr) \nabla_{\theta_j} F_1(\theta, r) \quad \text{and} \quad \nabla_{\theta_j} \int \rho(dr) F_2(\theta, r) = \int \rho(dr) \nabla_{\theta_j} F_2(\theta, r).$$

We discharge these proof obligations using Theorem C.2. Pick $\theta^* \in \mathbb{R}^p$ and $j \in [p]$. We instantiate Theorem C.2 twice for F_1 and F_2 as follows.

- For both cases of F_1 and F_2 , we set V to be an open interval (a, b) such that both a and b are finite, the closure $[a, b]$ is contained in $V_{(j,\theta^*)} \triangleq \bigcap_{i \in [M]} \bigcap_{\alpha \in K_i} V_{(i,\alpha,j,\theta^*)}$, and (a, b) contains θ_j^* . Note that such an open interval exists because $V_{(j,\theta^*)}$ is open and contains θ_j^* .
- The measurable space (X, Σ) in the theorem is RDB with its σ -algebra in both cases.
- The measure μ is the reference measure ρ on RDB in both cases.
- The measurable function f is instantiated twice for $(v, r) \mapsto F_i(\theta^*[j : v], r)$ with $i = 1, 2$.

Under this instantiation, the conclusion of Theorem C.2 implies all of the requirements R4-R6.

It remains to show that both instantiations satisfy the three provisos of Theorem C.2 marked (a), (b) and (c). We tackle the provisos one-by-one. We use the notations in §6.1 without recalling them explicitly. For $i \in [M]$, $l \in [N_i]$, $\alpha \in K_i$, $v \in \mathbb{R}$, $r \in A_i$ and $\theta \in \mathbb{R}^p$, let

$$g_{(i,\alpha,\theta)}(v) \triangleq \mathcal{N}(v; \mu_{(i,\alpha)}(\theta), \sigma_{(i,\alpha)}(\theta)), \quad g_{(i,\theta)}(r) \triangleq \prod_{\beta \in K_i} g_{(i,\beta,\theta)}(r(\beta)),$$

$$\begin{aligned}
h_{(i,\alpha)}(r) &\triangleq \mathcal{N}(r(\alpha); \mu'_{(i,\alpha)}(r), \sigma'_{(i,\alpha)}(r)), & h_i(r) &\triangleq \prod_{\beta \in K_i} h_{(i,\beta)}(r), \\
k_{(i,l)}(r) &\triangleq \mathcal{N}(c_{(i,l)}; \mu''_{(i,l)}(r), \sigma''_{(i,l)}(r)), & k_i(r) &\triangleq \prod_{m=1}^{N_i} k_{(i,m)}(r).
\end{aligned}$$

Proof of (a). Since we assume the condition 9, Theorem 6.3 ensures that the integral $\int \mu(dr) F_1(\theta^*[j : v], r)$ is well-defined for all $v \in V$. The other integral involving F_2 is the integral of the density of $D_{\theta^*[j:v]}$. Since $D_{\theta^*[j:v]}$ is a guide, the integral is well-defined and has the value 1.

Proof of (b). Note that

$$\begin{aligned}
F_1(\theta^*[j : v], r) &= \text{dens}(D_{\theta^*[j:v]}, s_I)(r) \cdot \log \frac{\text{dens}(D_{\theta^*[j:v]}, s_I)(r)}{\text{dens}(C, s_I)(r)} \\
&= \sum_{i=1}^M \mathbb{1}_{[r \in A_i]} \cdot g_{(i, \theta^*[j:v])}(r) \cdot \log \frac{g_{(i, \theta^*[j:v])}(r)}{h_i(r) \cdot k_i(r)} \\
&= \sum_{i=1}^M \mathbb{1}_{[r \in A_i]} \cdot g_{(i, \theta^*[j:v])}(r) \cdot (-\log(h_i(r) \cdot k_i(r)) + \log(g_{(i, \theta^*[j:v])}(r))) \\
&= \sum_{i=1}^M \mathbb{1}_{[r \in A_i]} \cdot \prod_{\alpha \in K_i} \mathcal{N}(r(\alpha); \mu_{(i,\alpha)}(\theta^*[j : v]), \sigma_{(i,\alpha)}(\theta^*[j : v])) \\
&\quad \cdot \left(-\log(h_i(r) \cdot k_i(r)) + \sum_{\alpha \in K_i} \log \mathcal{N}(r(\alpha); \mu_{(i,\alpha)}(\theta^*[j : v]), \sigma_{(i,\alpha)}(\theta^*[j : v])) \right),
\end{aligned}$$

$$\begin{aligned}
F_2(\theta^*[j : v], r) &= \text{dens}(D_{\theta^*[j:v]}, s_I)(r) \\
&= \sum_{i=1}^M \mathbb{1}_{[r \in A_i]} \cdot \prod_{\alpha \in K_i} \mathcal{N}(r(\alpha); \mu_{(i,\alpha)}(\theta^*[j : v]), \sigma_{(i,\alpha)}(\theta^*[j : v])).
\end{aligned}$$

Recall that every $r \in A_i$ is a map from K_i to \mathbb{R} . The normal density $\mathcal{N}(v'; \mu', \sigma')$ and its log are smooth functions on $(v', \mu', \sigma') \in \mathbb{R} \times \mathbb{R} \times (0, \infty)$. Also, by the condition (10), the functions

$$v \in V_{(i,\alpha,j,\theta^*)} \mapsto \mu_{(i,\alpha)}(\theta^*[j : v]) \quad \text{and} \quad v \in V_{(i,\alpha,j,\theta^*)} \mapsto \sigma_{(i,\alpha)}(\theta^*[j : v])$$

are continuously differentiable for all $i \in [M]$ and $\alpha \in K_i$. Thus, they are also continuously differentiable when we restrict their domains to V . One more thing to notice is that neither $\mathbb{1}_{[r \in A_i]}$ nor $-\log(h_i(r) \cdot k_i(r))$ depends on v , so that they can be viewed as constant functions on v , which are obviously smooth. Now note that both $v \in V \mapsto F_1(\theta^*[j : v], r)$ and $v \in V \mapsto F_2(\theta^*[j : v], r)$ are obtained from these smooth or continuously differentiable functions by function composition, addition and multiplication. Thus, they are continuously differentiable.

Proof of (c). For all $i \in [M]$, we will construct measurable function $H_i, H'_i : A_i \rightarrow [0, \infty)$ such that

- both $\int \rho(dr) (\mathbb{1}_{[r \in A_i]} \cdot H_i(r))$ and $\int \rho(dr) (\mathbb{1}_{[r \in A_i]} \cdot H'_i(r))$ are well-defined;
- for all $r \in A_i$ and $v \in V$,

$$\left| \nabla_v \left(g_{(i, \theta^*[j:v])}(r) \cdot \log \frac{g_{(i, \theta^*[j:v])}(r)}{h_i(r) \cdot k_i(r)} \right) \right| \leq H_i(r);$$

- for all $r \in A_i$ and $v \in V$,

$$|\nabla_v g_{(i, \theta^*[j:v])}(r)| \leq H'_i(r).$$

Once we have such H_i 's, we can use them to define desired measurable functions $H, H' : RDB \rightarrow \mathbb{R}$ as follows:

$$H(r) \triangleq \sum_{i=1}^M \mathbb{1}_{[r \in A_i]} \cdot H_i(r) \quad \text{and} \quad H'(r) \triangleq \sum_{i=1}^M \mathbb{1}_{[r \in A_i]} \cdot H'_i(r).$$

Since both H_i and H'_i are integrable for all i , so are H and H' . We prove this only for H below. The case for H' is similar.

$$\int \rho(dr) |H(r)| \leq \int \rho(dr) \sum_{i=1}^M |\mathbb{1}_{[r \in A_i]} \cdot H_i(r)| = \sum_{i=1}^M \int \rho(dr) |\mathbb{1}_{[r \in A_i]} \cdot H_i(r)| < \infty.$$

Both H_i and H'_i bound the magnitudes of the partial derivatives involving $g_{(i, \theta^*[j:v])}$ for every i . Using this, we show that H and H' meet the condition about bounding the magnitude of a derivative:

$$\begin{aligned} |\nabla_v F_1(\theta^*[j:v], r)| &= \left| \nabla_v \sum_{i=1}^M \mathbb{1}_{[r \in A_i]} \cdot \left(g_{(i, \theta^*[j:v])}(r) \cdot \left(-\log(h_i(r) \cdot k_i(r)) + \log(g_{(i, \theta^*[j:v])}(r)) \right) \right) \right| \\ &\leq \sum_{i=1}^M \mathbb{1}_{[r \in A_i]} \cdot \left| \nabla_v \left(g_{(i, \theta^*[j:v])}(r) \cdot \left(-\log(h_i(r) \cdot k_i(r)) + \log(g_{(i, \theta^*[j:v])}(r)) \right) \right) \right| \\ &\leq \sum_{i=1}^M \mathbb{1}_{[r \in A_i]} \cdot H_i(r) = H(r), \\ |\nabla_v F_2(\theta^*[j:v], r)| &= \left| \nabla_v \sum_{i=1}^M \mathbb{1}_{[r \in A_i]} \cdot g_{(i, \theta^*[j:v])}(r) \right| \\ &\leq \sum_{i=1}^M \mathbb{1}_{[r \in A_i]} \cdot |\nabla_v g_{(i, \theta^*[j:v])}(r)| \\ &\leq \sum_{i=1}^M \mathbb{1}_{[r \in A_i]} \cdot H'_i(r) = H'(r). \end{aligned}$$

It remains to construct measurable $H_i, H'_i : A_i \rightarrow \mathbb{R}$ with the required properties for every $i \in [M]$. Pick $i \in [M]$. Note that

$$\begin{aligned} \nabla_v \left(g_{(i, \theta^*[j:v])}(r) \cdot \log \frac{g_{(i, \theta^*[j:v])}(r)}{h_i(r) \cdot k_i(r)} \right) &= (\nabla_v g_{(i, \theta^*[j:v])}(r)) \cdot \log \frac{g_{(i, \theta^*[j:v])}(r)}{h_i(r) \cdot k_i(r)} + (g_{(i, \theta^*[j:v])}(r) \cdot \nabla_v \log(g_{(i, \theta^*[j:v])}(r))) \\ &= (\nabla_v g_{(i, \theta^*[j:v])}(r)) \cdot \left(\log \frac{g_{(i, \theta^*[j:v])}(r)}{h_i(r) \cdot k_i(r)} + 1 \right). \end{aligned}$$

We will build H_i and H'_i by finding good bounds for $|\nabla_v g_{(i, \theta^*[j:v])}(r)|$, $|\log g_{(i, \theta^*[j:v])}(r)|$ and $|\log h_i(r) \cdot k_i(r)|$ and then combining these bounds. We do so in three steps.

First, we build functions that bound $|\nabla_v g_{(i, \theta^*[j:v])}(r)|$ and $|\log g_{(i, \theta^*[j:v])}(r)|$ over $v \in V$. For every $\alpha \in K_i$,

$$\begin{aligned} \nabla_v \log g_{(i, \alpha, \theta^*[j:v])}(w) &= \nabla_v \left(-\log \sqrt{2\pi} - \log \sigma_{(i, \alpha)}(\theta^*[j:v]) - \frac{(w - \mu_{(i, \alpha)}(\theta^*[j:v]))^2}{2\sigma_{(i, \alpha)}(\theta^*[j:v])^2} \right) \\ &= -\frac{\nabla_v \sigma_{(i, \alpha)}(\theta^*[j:v])}{\sigma_{(i, \alpha)}(\theta^*[j:v])} - \frac{(\mu_{(i, \alpha)}(\theta^*[j:v]) - w) \nabla_v \mu_{(i, \alpha)}(\theta^*[j:v])}{\sigma_{(i, \alpha)}(\theta^*[j:v])^2} \end{aligned}$$

$$\begin{aligned}
& + \frac{(\mathbf{w} - \mu_{(i,\alpha)}(\theta^*[j:v]))^2 \nabla_v \sigma_{(i,\alpha)}(\theta^*[j:v])}{\sigma_{(i,\alpha)}(\theta^*[j:v])^3} \\
& = \nabla_v \sigma_{(i,\alpha)}(\theta^*[j:v]) \cdot \frac{(\mathbf{w} - \mu_{(i,\alpha)}(\theta^*[j:v]))^2 - \sigma_{(i,\alpha)}(\theta^*[j:v])^2}{\sigma_{(i,\alpha)}(\theta^*[j:v])^3} \\
& \quad + \nabla_v \mu_{(i,\alpha)}(\theta^*[j:v]) \cdot \frac{\mathbf{w} - \mu_{(i,\alpha)}(\theta^*[j:v])}{\sigma_{(i,\alpha)}(\theta^*[j:v])^2}.
\end{aligned}$$

Thus, we have

$$\begin{aligned}
\nabla_v g_{(i,\theta^*[j:v])}(r) &= \nabla_v \prod_{\alpha \in K_i} g_{(i,\alpha,\theta^*[j:v])}(r(\alpha)) \\
&= \sum_{\alpha \in K_i} \nabla_v g_{(i,\alpha,\theta^*[j:v])}(r(\alpha)) \cdot \prod_{\substack{\beta \in K_i \\ \beta \neq \alpha}} g_{(i,\beta,\theta^*[j:v])}(r(\beta)) \\
&= \sum_{\alpha \in K_i} g_{(i,\alpha,\theta^*[j:v])}(r(\alpha)) \cdot \nabla_v \log g_{(i,\alpha,\theta^*[j:v])}(r(\alpha)) \cdot \prod_{\substack{\beta \in K_i \\ \beta \neq \alpha}} g_{(i,\beta,\theta^*[j:v])}(r(\beta)) \\
&= g_{(i,\theta^*[j:v])}(r) \cdot \sum_{\alpha \in K_i} \nabla_v \log g_{(i,\alpha,\theta^*[j:v])}(r(\alpha)) \\
&= g_{(i,\theta^*[j:v])}(r) \cdot \sum_{\alpha \in K_i} \left(\nabla_v \sigma_{(i,\alpha)}(\theta^*[j:v]) \cdot \frac{(r(\alpha) - \mu_{(i,\alpha)}(\theta^*[j:v]))^2 - \sigma_{(i,\alpha)}(\theta^*[j:v])^2}{\sigma_{(i,\alpha)}(\theta^*[j:v])^3} \right. \\
& \quad \left. + \nabla_v \mu_{(i,\alpha)}(\theta^*[j:v]) \cdot \frac{r(\alpha) - \mu_{(i,\alpha)}(\theta^*[j:v])}{\sigma_{(i,\alpha)}(\theta^*[j:v])^2} \right) \\
&= g_{(i,\theta^*[j:v])}(r) \cdot \sum_{\alpha \in K_i} \sum_{n=0}^2 G_{(1,\alpha,n)}(v) \cdot r(\alpha)^n \tag{24}
\end{aligned}$$

for some continuous functions $G_{(1,\alpha,n)} : V_{(j,\theta^*)} \rightarrow \mathbb{R}$ that do not depend on r . The continuity of $G_{(1,\alpha,n)}$ follows from our condition (10). Also,

$$\begin{aligned}
\log g_{(i,\theta^*[j:v])}(r) &= \sum_{\alpha \in K_i} \log g_{(i,\alpha,\theta^*[j:v])}(r(\alpha)) \\
&= \sum_{\alpha \in K_i} \left(-\log \sqrt{2\pi} - \log \sigma_{(i,\alpha)}(\theta^*[j:v]) - \frac{(r(\alpha) - \mu_{(i,\alpha)}(\theta^*[j:v]))^2}{2\sigma_{(i,\alpha)}(\theta^*[j:v])^2} \right) \\
&= \sum_{\alpha \in K_i} \sum_{n=0}^2 G_{(2,\alpha,n)}(v) \cdot r(\alpha)^n
\end{aligned}$$

for some continuous functions $G_{(2,\alpha,n)} : V_{(j,\theta^*)} \rightarrow \mathbb{R}$ that do not depend on r . The continuity of $G_{(2,\alpha,n)}$ follows from the condition (10). Since $\bar{V} \subset V_{(j,\theta^*)}$, the functions $G_{(m,\alpha,n)}$ are all continuous on the closed (bounded) interval \bar{V} , where \bar{V} is the closure of V . Thus, by the extreme value theorem,

$$C_{\max} \triangleq \max_{(m,\alpha,n)} \sup_{v \in \bar{V}} |G_{(m,\alpha,n)}(v)| < \infty.$$

Since the term $g_{(i,\theta^*[j:v])}(r)$ in (24) still depends on v , we bound it over $v \in V$ using Lemma C.3 which is instantiated with

$$\begin{aligned}\mu_1 &\triangleq \min_{\alpha \in K_i} \inf_{v \in \bar{V}} \mu_{(i,\alpha)}(\theta^*[j:v]) & \mu_2 &\triangleq \max_{\alpha \in K_i} \sup_{v \in \bar{V}} \mu_{(i,\alpha)}(\theta^*[j:v]), \\ \sigma_1 &\triangleq \min_{\alpha \in K_i} \inf_{v \in \bar{V}} \sigma_{(i,\alpha)}(\theta^*[j:v]) & \sigma_2 &\triangleq \max_{\alpha \in K_i} \sup_{v \in \bar{V}} \sigma_{(i,\alpha)}(\theta^*[j:v]).\end{aligned}$$

We check that the assumptions of the lemma are satisfied: since $\bar{V} \subset V_{(j,\theta^*)}$ and the functions

$$v \in V_{(j,\theta^*)} \mapsto \mu_{(i,\alpha)}(\theta^*[j:v]) \quad \text{and} \quad v \in V_{(j,\theta^*)} \mapsto \sigma_{(i,\alpha)}(\theta^*[j:v])$$

are continuous by the condition (10), μ_m and σ_m ($m = 1, 2$) are all finite by the extreme value theorem; moreover, σ_1 is positive by the choice of $V_{(i,\alpha,j,\theta^*)}$. Thus, by Lemma C.3, there exists $c \in \mathbb{R}$ such that

$$\begin{aligned}g_{(i,\theta^*[j:v])}(r) &= \prod_{\alpha \in K_i} \mathcal{N}(r(\alpha); \mu_{(i,\alpha)}(\theta^*[j:v]), \sigma_{(i,\alpha)}(\theta^*[j:v])) \\ &\leq \prod_{\alpha \in K_i} c \cdot \mathcal{N}(r(\alpha); 0, 2\sigma_2) = c' \cdot g'(r)\end{aligned}$$

for all $v \in V$, where $c' \triangleq c^{|K_i|}$ and $g'(r) \triangleq \prod_{\alpha \in K_i} \mathcal{N}(r(\alpha); 0, 2\sigma_2)$.

Combining what we have proved, we obtain the following bounds: for all $v \in V$,

$$\begin{aligned}|\nabla_v g_{(i,\theta^*[j:v])}(r)| &= g_{(i,\theta^*[j:v])}(r) \cdot \left| \sum_{\alpha \in K_i} \sum_{n=0}^2 G_{(1,\alpha,n)}(v) \cdot r(\alpha)^n \right| \\ &\leq c' \cdot g'(r) \cdot \sum_{\alpha \in K_i} \sum_{n=0}^2 C_{\max} \cdot |r(\alpha)^n| \\ &\leq c' \cdot g'(r) \cdot \exp(l_1(r)) \quad \text{for some } l_1 \in \mathcal{A}_i \text{ (Lemma C.1 (a) and (b)),} \\ |\log g_{(i,\theta^*[j:v])}(r)| &= \left| \sum_{\alpha \in K_i} \sum_{n=0}^2 G_{(2,\alpha,n)}(v) \cdot r(\alpha)^n \right| \\ &\leq \sum_{\alpha \in K_i} \sum_{n=0}^2 C_{\max} \cdot |r(\alpha)^n| \leq \exp(l_1(r)).\end{aligned}$$

Second, we bound $|\log h_i(r) \cdot k_i(r)|$ by $\exp(l_3(r))$ for some $l_3 \in \mathcal{A}_i$.

$$\begin{aligned}|\log h_i(r) \cdot k_i(r)| &\leq \sum_{\alpha \in K_i} |\log h_{(i,\alpha)}(r)| + \sum_{l=1}^{N_i} |\log k_{(i,l)}(r)| \\ &= \sum_{\alpha \in K_i} \left| -\log \sqrt{2\pi} - \log \sigma'_{(i,\alpha)}(r) - \frac{(r(\alpha) - \mu'_{(i,\alpha)}(r))^2}{2\sigma'_{(i,\alpha)}(r)^2} \right| \\ &\quad + \sum_{l=1}^{N_i} \left| -\log \sqrt{2\pi} - \log \sigma''_{(i,l)}(r) - \frac{(c_{(i,l)} - \mu''_{(i,l)}(r))^2}{2\sigma''_{(i,l)}(r)^2} \right|\end{aligned}$$

$$\begin{aligned}
&\leq (|K_i| + N_i) \cdot \left| \log \sqrt{2\pi} \right| + \sum_{\alpha \in K_i} \left(\left| \log \sigma'_{(i,\alpha)}(r) \right| + \frac{(r(\alpha) - \mu'_{(i,\alpha)}(r))^2}{2\sigma'_{(i,\alpha)}(r)^2} \right) \\
&\quad + \sum_{l=1}^{N_i} \left(\left| \log \sigma''_{(i,l)}(r) \right| + \frac{(c_{(i,l)} - \mu''_{(i,l)}(r))^2}{2\sigma''_{(i,l)}(r)^2} \right) \\
&\leq \exp(l_2(r)) \quad \text{for some } l_2 \in \mathcal{A}_i \text{ (Lemma C.1 and Condition 9).}
\end{aligned}$$

Finally, we define H_i and H'_i by combining the bounds that we got so far. Here are our definitions for them:

$$\begin{aligned}
H_i(r) &\triangleq c' \cdot g'(r) \cdot \exp(l_1(r)) \cdot (\exp(l_1(r)) + \exp(l_2(r)) + 1), \\
H'_i(r) &\triangleq c' \cdot g'(r) \cdot \exp(l_1(r)).
\end{aligned}$$

Then, by the choice of l_1 and l_2 , for all $r \in A_i$ and $v \in V$,

$$\left| \nabla_v \left(g_{(i,\theta^*[j:v])}(r) \cdot \log \frac{g_{(i,\theta^*[j:v])}(r)}{h_i(r) \cdot k_i(r)} \right) \right| \leq H_i(r) \quad \text{and} \quad \left| \nabla_v g_{(i,\theta^*[j:v])}(r) \right| \leq H'_i(r).$$

Also, we have

$$\begin{aligned}
&\int \rho(dr) \left| \mathbb{1}_{[r \in A_i]} \cdot H_i(r) \right| \\
&= \int \rho(dr) \left(\mathbb{1}_{[r \in A_i]} \cdot g'(r) \cdot c' \cdot \exp(l_1(r)) \cdot (\exp(l_1(r)) + \exp(l_2(r)) + 1) \right) \\
&\leq \int \rho(dr) \left(\mathbb{1}_{[r \in [K_i \rightarrow \mathbb{R}]]} \cdot g'(r) \cdot \exp(l_3(r)) \right) \quad \text{for some } l_3 \in \mathcal{A}_i \text{ (Lemma C.1 (a))} \\
&= \mathbb{E}_{\prod_{\alpha \in K_i} \mathcal{N}(r(\alpha); 0, 2\sigma_2)} [\exp(l_3(r))] < \infty \quad \text{(Lemma C.1 (e))},
\end{aligned}$$

and

$$\begin{aligned}
&\int \rho(dr) \left| \mathbb{1}_{[r \in A_i]} \cdot H'_i(r) \right| \\
&= \int \rho(dr) \left(\mathbb{1}_{[r \in A_i]} \cdot g'(r) \cdot c' \cdot \exp(l_1(r)) \right) \\
&\leq \int \rho(dr) \left(\mathbb{1}_{[r \in [K_i \rightarrow A]]} \cdot g'(r) \cdot \exp(l_4(r)) \right) \quad \text{for some } l_4 \in \mathcal{A}_i \text{ (Lemma C.1 (a))} \\
&= \mathbb{E}_{\prod_{\alpha \in K_i} \mathcal{N}(r(\alpha); 0, 2\sigma_2)} [\exp(l_4(r))] < \infty \quad \text{(Lemma C.1 (e))},
\end{aligned}$$

We have just shown that H_i and H'_i are the functions that we are looking for. \square

D PROOF OF THEOREM IN §7

THEOREM 7.1 (SOUNDNESS). *For all commands C , we have $\llbracket C \rrbracket_d \in \gamma(\llbracket C \rrbracket^\#)$.*

PROOF. We prove the theorem by structural induction on C . Every case except the one for the loop follows from the assumption made for an abstract element in $\mathcal{T}^\#$ or an operator on it that corresponds to C . The proof of the case of $C \equiv (\text{while } E \{C_0\})$ goes as follows. Let $t_{\text{fix}} \triangleq (\text{wfix } T)$. Then, by the assumption on the concretisation, $\gamma(t_{\text{fix}})$ is an admissible subset of \mathcal{D} . Thus, it suffices to prove that when G is the function in the density semantics of $\llbracket \text{while } E \{C_0\} \rrbracket_d$, the image of G

on $\gamma(t_{fix})$ is included in $\gamma(t_{fix})$. To do so, pick $g \in \gamma(t_{fix})$. We have to show that $G(g) \in \gamma(t_{fix})$. By the induction hypothesis and the assumption on the abstract composition,

$$(g^{\ddagger} \circ \llbracket C_0 \rrbracket_d) \in \gamma(t_{fix} \circ^{\#} \llbracket C_0 \rrbracket^{\#}).$$

Also, $\llbracket \text{skip} \rrbracket_d \in \gamma(\text{skip}^{\#})$. Thus,

$$\begin{aligned} G(g) &= \lambda(s, r). \text{ if } (\llbracket E \rrbracket s = \text{true}) \text{ then } \llbracket \text{skip} \rrbracket_d(s, r) \text{ else } (g^{\ddagger} \circ \llbracket C_0 \rrbracket_d)(s, r) \\ &\in \gamma(\text{cond}(E)^{\#}(t_{fix} \circ^{\#} \llbracket C_0 \rrbracket^{\#}, \text{skip}^{\#})) = \gamma(T(t_{fix})) = \gamma(t_{fix}). \end{aligned}$$

The set membership holds because of the assumption on the abstract conditional operator. \square