Smoothness Analysis for Probabilistic Programs with Application to Optimised Variational Inference

Wonyeol Lee, Xavier Rival, Hongseok Yang

Smoothness Analysis for Probabilistic Programs with Application to Optimised Variational Inference

WONYEOL LEE, Stanford University, USA
XAVIER RIVAL, INRIA Paris, France and ENS, CNRS, and PSL University, Paris, France
HONGSEOK YANG, KAIST, South Korea and Institute for Basic Science (IBS), South Korea

We present a static analysis for discovering differentiable or more generally smooth parts of a given probabilistic program, and show how the analysis can be used to improve the pathwise gradient estimator, one of the most popular methods for posterior inference and model learning. Our improvement increases the scope of the estimator from differentiable models to non-differentiable ones without requiring manual intervention of the user; the improved estimator automatically identifies differentiable parts of a given probabilistic program using our static analysis, and applies the pathwise gradient estimator to the identified parts while using a more general but less efficient estimator, called score estimator, for the rest of the program. Our analysis has a surprisingly subtle soundness argument, partly due to the misbehaviours of some target smoothness properties when viewed from the perspective of program analysis designers. For instance, some smoothness properties, such as partial differentiability and partial continuity, are not preserved by function composition, and this makes it difficult to analyse sequential composition soundly without heavily sacrificing precision. We formulate five assumptions on a target smoothness property, prove the soundness of our analysis under those assumptions, and show that our leading examples satisfy these assumptions. We also show that by using information from our analysis instantiated for differentiability, our improved gradient estimator satisfies an important differentiability requirement and thus computes the correct estimate on average (i.e., returns an unbiased estimate) under a regularity condition. Our experiments with representative probabilistic programs in the Pyro language show that our static analysis is capable of identifying smooth parts of those programs accurately, and making our improved pathwise gradient estimator exploit all the opportunities for high performance in those programs.

CCS Concepts: • **Software and its engineering → Correctness**; **Automated static analysis**; • **Mathematics of computing → Bayesian computation**; **Variational methods**.

Additional Key Words and Phrases: smoothness, static analysis, probabilistic programming, variational inference

ACM Reference Format:
Wonyeol Lee, Xavier Rival, and Hongseok Yang. 2023. Smoothness Analysis for Probabilistic Programs with Application to Optimised Variational Inference. *Proc. ACM Program. Lang.* 7, POPL, Article 12 (January 2023), 32 pages. https://doi.org/10.1145/3571205

1 INTRODUCTION

Probabilistic programs define models from machine learning and statistics, and are used to analyse datasets from a wide range of applications [Bingham et al. 2019; Carpenter et al. 2017; Ge et al. 2018; Gehr et al. 2016; Goodman et al. 2008; Gordon et al. 2014; Mansinghka et al. 2014; Minka et al. 2014; Narayanan et al. 2016; Salvatier et al. 2016; Siddharth et al. 2017; Tolpin et al. 2016; Tran et al. 2018, 2016;

Authors' addresses: Wonyeol Lee, Computer Science, Stanford University, USA, wonyeol@cs.stanford.edu; Xavier Rival, INRIA Paris, France , Département d'Informatique, ENS, CNRS, and PSL University, Paris, France, rival@di.ens.fr; Hongseok Yang, School of Computing and Kim Jaechul Graduate School of AI, KAIST, South Korea , Discrete Mathematics Group, Institute for Basic Science (IBS), South Korea, hongseok.yang@kaist.ac.kr.

2475-1421/2023/1-ART12
https://doi.org/10.1145/3571205

Wood et al. 2014]. These programs are written in languages with special runtimes, called inference engines, which can be used to answer probabilistic queries, such as posterior inference and marginal likelihood estimation, or to learn model parameters in those programs, such as weights of neural networks. Whether a probabilistic program is useful for, for instance, discovering a hidden pattern in a given dataset or making an accurate prediction largely lies in these inference engines. These engines should compute accurate approximations or good model parameters within a fixed time budget. It is, thus, not surprising that substantial research efforts have been made to develop efficient inference algorithms and their implementations (as inference engines) [Chaganty et al. 2013; Holtzen et al. 2020; Kucukelbir et al. 2015; Mansinghka et al. 2014; Nori et al. 2014; Ritchie et al. 2016a; Schulman et al. 2015; van de Meent et al. 2018; Wingate et al. 2011a,b; Wingate and Weber 2013; Zhou et al. 2020].

We are concerned with smoothness[1] properties of probabilistic programs, which have been exploited by performant posterior-inference and model-learning algorithms and engines. For instance, when probabilistic programs are differentiable (in the sense that they define differentiable unnormalised densities), their posteriors can be inferred by Hamiltonian Monte Carlo [Neal 2011], one of the best performing MCMC algorithms. Also, in that case, their posteriors and model parameters can be inferred or learnt using the pathwise gradient estimator [Kingma and Welling 2014; Rezende et al. 2014], a popular technique for estimating the gradient of a function using samples. We also point out that the need for smoothness arises in a broader context of machine learning and computer science; Lipschitz continuity is one of the desired or at least recommended properties for neural networks [Arjovsky et al. 2017; Kim et al. 2021], and also differentiability commonly features as a requirement for pieces of code inside simulation software and cyber physical systems, where differential equations are used to specify the environment [Platzer 2018].

We present a static analysis that enables optimised posterior inference and model learning for probabilistic programs. We develop a static analysis that discovers differentiable or more generally smooth parts of given probabilistic programs, and show how the analysis can be used to improve the pathwise gradient estimator. Our improvement increases the scope of the estimator from differentiable to non-differentiable models, without requiring any intervention from the user; the improved estimator automatically identifies differentiable parts of probabilistic programs using our static analysis, and applies the pathwise gradient estimator to the identified parts while using a more general but less efficient estimator, called score estimator [Ranganath et al. 2014; Williams 1992], for the rest of the programs.

Our static analysis for smoothness has a surprisingly subtle soundness argument, partly due to the misbehaviours of some target smoothness properties when viewed from the perspective of program analysis designers. For instance, some smoothness properties, such as partial differentiability and partial continuity, are not preserved by function composition, and this makes it difficult to analyse sequential composition soundly without heavily sacrificing precision. In fact, overlooking such misbehaviours has been a source of errors in published static analyses for continuity [Chaudhuri et al. 2010, 2012].[2] We formulate five assumptions that clearly identify what a smoothness property should satisfy in order to avoid unsound analysis. Interestingly, these assumptions also determine what the property is allowed to violate. For instance, they reveal that the smoothness property does not

[1] In mathematics, "smoothness" typically refers to the *specific* property of functions: being infinitely differentiable. In this paper, we override the term to denote a *set* of properties of functions describing well-behavedness (e.g., differentiability).

[2] The analysis in [Chaudhuri et al. 2010] infers the continuity property for multivariate programs, but it incorrectly joins two input-variable sets if a program is continuous with respect to each set *jointly*. Such a rule would hold if separate per-input-variable continuity were considered, but it does not hold for multivariate joint continuity. Conversely, the analysis in [Chaudhuri et al. 2012] considers a *per-input-variable* definition of continuity, but incorrectly assumes that this per-input-variable continuity is preserved by function composition. These (and other) issues make the two analyses unsound in several aspects (see §A for details). We do not claim that these unsoundness issues are hard to fix. Instead, our point is that a similar issue may be introduced easily and remain undetected due to the subtlety in the soundness of a static analysis.

have to be closed under the limits of chains of smooth (partial) functions, although the closure under such limits, called admissibility, has often been used to justify proof rules about or static analysis of loops. Dispensing with the admissibility requirement broadens the scope of our program analysis non-trivially; some useful smoothness properties from mathematics fail to meet the requirement.

Our variant of the pathwise gradient estimator works by program transformation and non-standard execution. It first transforms given probabilistic programs based on the results of our static analysis. Then, our estimator executes the transformed programs according to a standard (sampling) semantics, and collects sampled values during the execution. Finally, using the collected values, the estimator executes the original programs according to a non-standard (density) semantics this time. During the execution, our estimator computes a quantity involving differentiation, which becomes the estimate of the target gradient. We prove that our estimator satisfies an important differentiability requirement and thus, under a regularity condition, it is correct: the computed estimate is unbiased, i.e., it is the target gradient when averaged over random choices made during execution.

Our static analysis and variant of the pathwise gradient estimator have been implemented for a subset of the Pyro probabilistic programming language [Bingham et al. 2019]. They have been successfully applied to the 13 representative Pyro examples, which include advanced models with deep neural networks, such as attend-infer-repeat [Eslami et al. 2016] and single-cell annotation using variational inference [Xu et al. 2021]. For each of these examples, Pyro provides a (default) selective use of the pathwise gradient estimator but without any correctness guarantee. Our analysis and improved estimator automatically reproduced those uses, and proved that in those use cases, the estimator satisfies an important differentiability requirement and it is, thus, correct (i.e., unbiased) under a regularity condition (which needs to be discharged separately).

We summarise the main contributions of the paper:

- We present a program analysis for smoothness properties such as differentiability, and explain a subtle soundness argument for the analysis. Our argument identifies five assumptions for target smoothness properties, which are violated by some well-known smoothness properties and can help detect and prevent soundness errors in static analyses for smoothness properties (§5).
- We present a gradient estimator for probabilistic programs that improves the well-known pathwise gradient estimator using our program analysis. We also prove that our estimator satisfies an important differentiability requirement and it is, thus, correct (namely unbiased) under a regularity condition (§4 and §6).
- We show that our program analysis and gradient estimator can be successfully applied to representative probabilistic programs in Pyro, and can prove that existing unproved optimisations for these programs satisfy the differentiability requirement (§7).

The appendix (i.e., §A–§I) includes omitted proofs and details, and can be found in [Lee et al. 2022b].

2 INFORMAL DESCRIPTION OF BASIC CONCEPTS AND OUR APPROACH

We start by describing informally basic concepts and the goal of our approach, which we hope helps the reader to see the big picture of our technical contributions. To simplify presentation, we use toy examples in the section. But we emphasise that our approach has been applied to representative Pyro programs that describe advanced machine-learning models with deep neural networks.

Probabilistic programming and variational inference. In a probabilistic programming language (PPL), a program expresses a probabilistic model. As an example, consider the program c_m in Fig. 1, which describes a probabilistic model of the random variables z_1 and z_2 in $\mathbb{R}$ by specifying their *unnormalised* density

$$p_{c_m}(z_1, z_2) = \mathcal{N}(z_1; 0, 5) \cdot \mathcal{N}(z_2; z_1, 3) \cdot (\mathbf{1}_{[z_2>0]} \cdot \mathcal{N}(0; 1, 1) + \mathbf{1}_{[z_2\leq 0]} \cdot \mathcal{N}(0; -2, 1)),$$

$$c_m = \begin{pmatrix} x_1 := \texttt{sam}(\texttt{"z}_1\texttt{"}, \texttt{dist}_\texttt{N}(0,5), \lambda y.y); \\ x_2 := \texttt{sam}(\texttt{"z}_2\texttt{"}, \texttt{dist}_\texttt{N}(x_1,3), \lambda y.y); \\ \texttt{if}\ (x_2 > 0)\ \{\texttt{obs}(\texttt{dist}_\texttt{N}(1,1), 0)\} \\ \texttt{else}\qquad\ \{\texttt{obs}(\texttt{dist}_\texttt{N}(-2,1), 0)\} \end{pmatrix}, \qquad c_g = \begin{pmatrix} x_1 := \texttt{sam}(\texttt{"z}_1\texttt{"}, \texttt{dist}_\texttt{N}(\theta_1,1), \lambda y.y); \\ x_2 := \texttt{sam}(\texttt{"z}_2\texttt{"}, \texttt{dist}_\texttt{N}(\theta_2,1), \lambda y.y) \end{pmatrix}.$$

Fig. 1. A model c_m and a guide c_g in a PPL. Here $\texttt{dist}_\texttt{N}(a,b)$ is the distribution expression, and denotes the normal distribution with mean a and variance b.

$$c'_g = \begin{pmatrix} x_1 := \texttt{sam}(\texttt{"z}_1\texttt{"}, \texttt{dist}_\texttt{N}(0,1), \lambda y.y + \theta_1); \\ x_2 := \texttt{sam}(\texttt{"z}_2\texttt{"}, \texttt{dist}_\texttt{N}(0,1), \lambda y.y + \theta_2) \end{pmatrix}, \quad c''_g = \begin{pmatrix} x_1 := \texttt{sam}(\texttt{"z}_1\texttt{"}, \texttt{dist}_\texttt{N}(0,1), \lambda y.y + \theta_1); \\ x_2 := \texttt{sam}(\texttt{"z}_2\texttt{"}, \texttt{dist}_\texttt{N}(\theta_2,1), \lambda y.y) \end{pmatrix}.$$

Fig. 2. A fully (or selectively) reparameterised guide c'_g (or c''_g).

where $\mathcal{N}(x; a, b)$ is the probability density of a normal distribution with mean a and variance b, and $\mathbf{1}_{[\varphi]}$ is the indicator function that returns 1 if φ holds and 0 otherwise. The first two $\mathcal{N}$ factors in the equation come from the sample commands (sam) in c_m. They are called prior distributions, and describe prior knowledge on two random variables named z_1 and z_2. The last factor comes from the if and observe commands (if and obs), which express that an unnamed random variable is sampled and observed to be 0 and its distribution is $\texttt{dist}_\texttt{N}(1,1)$ or $\texttt{dist}_\texttt{N}(-2,1)$ depending on whether z_2 is positive or not. This factor is called likelihood, and it states information about z_1 and z_2 that comes from an observed data point 0. Ignore the third arguments of the sample commands of c_m for now, which have no effect on p_{c_m}; they will be explained later.

The purpose of writing c_m in a PPL, called *model*, is to infer its *normalised* probability density

$$\overline{p}_{c_m}(z_1, z_2) \triangleq p_{c_m}(z_1, z_2) / \int p_{c_m}(z_1, z_2)\, dz_1 dz_2,$$

also called normalised posterior density. Intuitively, this normalised density brings together two types of information about z_1 and z_2, the first from their prior distributions (expressed in the first and second lines of c_m), and the second from the observed data point 0 that depends on z_1 and z_2 (the third and fourth lines of c_m). This inference task is called *posterior inference* problem. Among a wide range of approaches to the problem, we focus on the approach called *variational inference*, which forms the core of the recent combination of PPLs and deep learning.

In variational inference, we posit another program c_g, called *guide*, that is simpler than c_m and parameterised by θ. Then, we approximate the normalised density of c_m by c_g with an optimal choice of θ. For instance, consider the program c_g in Fig. 1. The program specifies the following already-normalised probability density

$$p_{c_g,\theta}(z_1, z_2) = \mathcal{N}(z_1; \theta_1, 1) \cdot \mathcal{N}(z_2; \theta_2, 1).$$

It can serve as a guide program for $\overline{p}_{c_m}$. To best approximate $\overline{p}_{c_m}$ by $p_{c_g,\theta}$, variational inference aims at finding θ that minimises some notion of the discrepancy (called KL divergence) between $p_{c_g,\theta}$ and $\overline{p}_{c_m}$, or equivalently that maximises the objective function $\mathcal{L}$ (called evidence lower bound):

$$\arg\max_\theta \mathcal{L}(\theta) \quad \text{for } \mathcal{L}(\theta) \triangleq \mathbb{E}_{p_{c_g,\theta}(z_1,z_2)}\left[f_\theta(z_1, z_2)\right] \text{ with } f_\theta(z_1, z_2) \triangleq \log(p_{c_m}(z_1, z_2)/p_{c_g,\theta}(z_1, z_2)).$$

A standard way to solve this optimisation problem is to apply the gradient-ascent algorithm: starting from an initial value $\theta^{(0)}$ of θ, compute $\theta^{(t)}$ iteratively by $\theta^{(t+1)} \triangleq \theta^{(t)} + \eta \cdot \nabla_\theta \mathcal{L}(\theta^{(t)})$, and return $\theta^{(T)}$ for a sufficiently large $T \in \mathbb{N}$. Here $\eta \in \mathbb{R}_{>0}$ denotes a learning rate.

A challenging part in the algorithm is to compute $\nabla_\theta \mathcal{L}(\theta)$. An exact computation of the gradient is mostly intractable due to the expectation inside $\mathcal{L}$, which hinders the gradient from having a closed-form formula. Hence, in practice, we rather *estimate* (not exactly compute) the gradient via a Monte Carlo method: draw a random sample $(\hat{z}_1, \hat{z}_2)$ from some distribution q_θ, apply some function g_θ to the sample, and use the result as an estimate to the gradient, i.e.,

$$g_\theta(\hat{z}_1, \hat{z}_2) \approx \nabla_\theta \mathcal{L}(\theta) \quad \text{for a sample } (\hat{z}_1, \hat{z}_2) \text{ drawn from } q_\theta(\hat{z}_1, \hat{z}_2). \tag{1}$$

An important desired property of such a gradient estimator is *unbiasedness*, which states that the estimate is accurate in expectation: $\mathbb{E}_{q_\theta(z_1,z_2)}[g_\theta(z_1,z_2)] = \nabla_\theta \mathcal{L}(\theta)$. The property is necessary for the algorithm to converge to a local optimum, and is, thus, desired.

Gradient estimators for variational inference: SCE, PGE, and SPGE. A standard estimator for $\nabla_\theta \mathcal{L}(\theta)$ is the *score estimator* (SCE) [Ranganath et al. 2014; Williams 1992], which is unbiased under mild conditions. It estimates $\nabla_\theta \mathcal{L}(\theta)$ by using the recipe in Eq. (1) with $q_\theta(z_1,z_2) = p_{c_g,\theta}(z_1,z_2)$ and

$$g_\theta(z_1,z_2) = f_\theta(z_1,z_2) \cdot \nabla_\theta \log q_\theta(z_1,z_2).$$

That is, the estimator draws a sample from the guide distribution $p_{c_g,\theta}$ and applies the above g_θ to obtain a gradient estimate.[3] It is applicable to a wide range of model-guide pairs while remaining unbiased, but it is known to have a large approximation error (i.e., have a large variance).

The *pathwise gradient estimator* (PGE) [Kingma and Welling 2014; Rezende et al. 2014] is another standard gradient estimator, which is known to have a smaller approximation error than the SCE and thus has been a preferred option against the SCE. The PGE requires an additional program c'_g that is a θ-independent reparameterisation of the guide c_g. A program c' is said to be *θ-independent* if the probability densities of the sampled random variables in c' are θ-independent. It is called a *reparameterisation* of c if c and c' sample the same set of random variables and they have the same semantics on those variables in the following sense: when there are n random variables, for any measurable $h : \mathbb{R}^n \to \mathbb{R}$, we have $\mathbb{E}_{p_c(z)}[h(v_c(z))] = \mathbb{E}_{p_{c'}(z)}[h(v_{c'}(z))]$, where $p_c : \mathbb{R}^n \to \mathbb{R}$ is the probability density of all n random variables in c, and $v_c : \mathbb{R}^n \to \mathbb{R}^n$ is the so called value function of c, which applies the lambda functions in the third arguments of c's sample commands to the corresponding random variables. For example, c'_g in Fig. 2 is a θ-independent reparameterisation of c_g for $\theta = (\theta_1, \theta_2)$. It has the probability density $p_{c'_g}(z_1,z_2) = \mathcal{N}(z_1;0,1) \cdot \mathcal{N}(z_2;0,1)$, and the value function $v_{c'_g,\theta}(z_1,z_2) = (z_1+\theta_1, z_2+\theta_2)$. Note that $p_{c'_g}$ does not depend on θ, as required by the θ-independence of c_g. We can show this c'_g is a reparameterisation of c_g in Fig. 1 by using the fact that v_{c_g} is the identity function and $y = x + a$ for x drawn from $\mathcal{N}(x;0,1)$ follows the distribution $\mathcal{N}(y;a,1)$.

Given a reparameterised guide c'_g, the PGE estimates $\nabla_\theta \mathcal{L}(\theta)$ by again following the recipe in Eq. (1) this time with $q'(z_1,z_2) = p_{c'_g}(z_1,z_2)$ and

$$g'_\theta(z_1,z_2) = \nabla_\theta f_\theta(z'_1,z'_2) \quad \text{for } (z'_1,z'_2) = v_{c'_g,\theta}(z_1,z_2).$$

This estimator differs from the SCE in two aspects. First, a random sample is drawn from a reparameterised-guide distribution $p_{c'_g}$, not from $p_{c_g,\theta}$. Next, the estimation function g'_θ computes the derivative of $f_\theta(z'_1,z'_2)$ with respect to θ and (z'_1,z'_2) (not with respect to θ only), since the argument of $f_\theta(-)$ in g'_θ depends on θ via $v_{c'_g,\theta}$.[4]

While having a small approximation error, the PGE requires more than the SCE to ensure the unbiasedness. An important additional requirement for the PGE is that (i) $p_{c_m}(z_1,z_2)$ and $p_{c_g,\theta}(z_1,z_2)$ should be differentiable in θ and z_1, z_2 and (ii) $v_{c'_g,\theta}(z_1,z_2)$ be differentiable in θ for all z_1, z_2. The requirement is imposed partly to ensure that no differentiation error arises in computing g'_θ. This differentiability requirement, however, can be easily violated if a model or a guide starts to use branches or loops. For instance, it is violated by our example in Figs. 1 and 2 as $p_{c_m}(z_1,z_2)$ is not differentiable in z_2. This violation makes the PGE biased for the example, i.e.,

$$\mathbb{E}_{q'_\theta(z_1,z_2)}[g'_\theta(z_1,z_2)] = (\cdots, \tfrac{1}{3}(\theta_1-\theta_2)) \neq (\cdots, \tfrac{1}{3}(\theta_1-\theta_2) + \tfrac{3}{2}\mathcal{N}(-\theta_2;0,1)) = \nabla_\theta \mathcal{L}(\theta),$$

[3] The log term in g_θ comes from the well-known log-derivative trick: $\nabla_\theta q_\theta(z_1,z_2) = q_\theta(z_1,z_2) \cdot \nabla_\theta \log q_\theta(z_1,z_2)$.

[4] By the chain rule, we have the following for each $i \in \{1,2\}$:

$$\frac{\partial f_\theta(z'_1,z'_2)}{\partial \theta_i} = \left.\frac{\partial f_\theta(x_1,x_2)}{\partial \theta_i}\right|_{\substack{(x_1,x_2,\theta)\\=(z'_1,z'_2,\theta)}} + \left\langle \left.\frac{\partial f_\theta(x_1,x_2)}{\partial x_1}\right|_{\substack{(x_1,x_2,\theta)\\=(z'_1,z'_2,\theta)}}, \left.\frac{\partial f_\theta(x_1,x_2)}{\partial x_2}\right|_{\substack{(x_1,x_2,\theta)\\=(z'_1,z'_2,\theta)}} \right\rangle, \left\langle \left.\frac{\partial v_{c'_g,\theta}(y_1,y_2)}{\partial \theta_i}\right|_{\substack{(y_1,y_2,\theta)\\=(z_1,z_2,\theta)}} \right\rangle.$$

and thus causes the gradient-ascent algorithm to converge to a suboptimal θ: applying the PGE to the example produces a suboptimal solution $\theta = (0, 0)$, whereas the optimal solution is $\theta \approx (0.95, 1.52)$.

The *selective pathwise gradient estimator* (SPGE) [Schulman et al. 2015] combines the two previous gradient estimators to alleviate their limitations: one has a large approximation error, and the other imposes a strong requirement for unbiasedness. The SPGE requires an additional program c_g'' that is a reparameterisation of the guide c_g but needs not be θ-independent (unlike the PGE). An instance of c_g'' for our example is given in Fig. 2, which changes the sample command for z_1 in c_g but keeps the one for z_2. Note that the changed sample command for z_1 in c_g'' uses a θ-independent probability distribution. Typically, c_g'' is obtained by *selecting* a subset of the random variables in c_g and changing the sample commands for the selected variables such that their probability distributions become θ-independent; the sample commands for the unselected remain as they are. Given c_g'', the SPGE estimates $\nabla_\theta \mathcal{L}(\theta)$ by following the recipe in Eq. (1) with $q_\theta''(z_1, z_2) = p_{c_g'',\theta}(z_1, z_2)$ and

$$g_\theta''(z_1, z_2) = \nabla_\theta f_\theta(z_1'', z_2'') + f_\theta(z_1'', z_2'') \cdot \nabla_\theta \log q_\theta''(z_1, z_2) \quad \text{for } (z_1'', z_2'') = v_{c_g'',\theta}(z_1, z_2). \tag{2}$$

Note that the estimation function g_θ'' consists of two terms, which come from that of the PGE and the SCE. The second term adjusts the PGE to correctly account for unchanged random variables (e.g., z_2 in the example of Fig. 2).

By allowing a guide that makes only some selected (not all) random variables θ-independent, the SPGE offers two advantages at the same time: it achieves a smaller approximation error than the SCE, and imposes a weaker requirement for unbiasedness than the PGE. In particular, the differentiability requirement for the SPGE is weaker than that for the PGE, which is as follows for our example in Figs. 1 and 2: (i) $p_{c_m}(z_1, z_2)$ and $p_{c_g,\theta}(z_1, z_2)$ be differentiable in θ and z_1 (but they may be non-differentiable in z_2); and (ii) $v_{c_g'',\theta}(z_1, z_2)$ and $p_{c_g'',\theta}(z_1, z_2)$ be differentiable in θ for all z_1, z_2. This requirement holds, and as a result, the SPGE with this c_g'' is unbiased (whereas the PGE with the given c_g' is biased as seen before). You can find in §B a table summarising and comparing SCE, PGE, and SPGE.

Variable-selection problem for SPGE. To maximize the advantages offered by the SPGE, we consider the following algorithmic problem:

Definition 2.1 (SPGE Variable-Selection Problem; Informal). Assume that we are given a model c_m, a guide c_g, and a *reparameterisation plan* π, i.e., a map from sample commands to sample commands. Then, find automatically a large subset S of random variables such that if we let $\overline{c_g}^{\pi,S}$ be the result of π-transforming every sample command in c_g that defines a random variable in S, then $\overline{c_g}^{\pi,S}$ is a reparameterisation of c_g and $(c_m, c_g, \overline{c_g}^{\pi,S})$ satisfies the differentiability requirement for the SPGE. □

An instantiation of the problem for our example is that c_m and c_g are programs in Fig. 1 and π transforms commands of the form $y := \mathtt{sam}(n, \mathtt{dist}_{\mathsf{N}}(e', 1), \lambda y.y)$ to $y := \mathtt{sam}(n, \mathtt{dist}_{\mathsf{N}}(0, 1), \lambda y.y + e')$, while leaving all the other sample commands as they are. In this instantiation, the condition in the problem is met by $S = \emptyset$ and $S = \{z_1\}$, and the latter option is preferred due to its size. Note that the solution $S = \{z_1\}$ yields the guide c_g'' in Fig. 2, that is, $\overline{c_g}^{\pi,S} = c_g''$.

Existing PPLs, when applying the SPGE, choose an S without checking the differentiability requirement, and this can make the requirement easily violated. For instance, given a model-guide pair, in one of its standard settings, Pyro automatically applies the SPGE with S being the set of all continuous random variables in the guide. This choice of S, however, does not guarantee the requirement is met. For our example in Fig. 1, Pyro chooses $S = \{z_1, z_2\}$, but this S violates the requirement; due to this, the SPGE becomes biased and Pyro returns a suboptimal $\theta = (0, 0)$.

In the rest of the paper, we will present our solution to the SPGE variable-selection problem. A core component of our solution is a general static analysis framework for smoothness properties such as differentiability (§5), which our solution uses to discharge the differentiability requirement

for the SPGE correctly and automatically. As we briefly mentioned in the introduction, automatically analysing the smoothness properties of a program in a sound manner is surprisingly subtle. Our analysis framework identifies five assumptions for smoothness properties, and prove that the analysis is sound if a target smoothness property satisfies these assumptions.

Our solution for the SPGE variable-selection problem (§6) runs the static analysis on given c_m and c_g, and computes a maximal set S' of random variables in which p_{c_m} and $p_{c_g,\theta}$ are differentiable. Then, it heuristically searches for a subset of S' starting from S' itself such that $\overline{c_g}^{\pi,S'}$ satisfies the differentiability requirement. For instance, for our example in Fig. 1, our differentiability analysis infers that p_{c_m} and $p_{c_g,\theta}$ are differentiable in $\{z_1\}$ and $\{z_1, z_2\}$, respectively. From this, we set $S' = \{z_1\}$, run our analysis again on $\overline{c_g}^{\pi,S'}$, and get confirmation that $\overline{c_g}^{\pi,S'}$ meets the requirement, i.e., $p_{\overline{c_g}^{\pi,S'},\theta}$ and $v_{\overline{c_g}^{\pi,S'},\theta}$ are differentiable in θ. Thus, this S' becomes the final result. In fact, this first-round success appeared in our experiments (§7): our implementation shows on all tested examples that the initial choice of S' is indeed valid in the above sense so that no subset search is necessary.

We point out that to mathematically develop and analyse our solution for the SPGE variable-selection problem, we formalise the SPGE in the PPL setting and formally derive a sufficient condition for its unbiasedness, which includes the differentiability requirement (§4).

3 SETUP

We use a simple imperative probabilistic programming language, which models the core of popular imperative PPLs, such as Pyro. Programs in the language describe densities, which are sometimes unnormalised (i.e., they do not integrate to 1). In this section, we describe the syntax and semantics of the language, and also variational inference for the language.

Syntax of a simple imperative PPL. Let PVar be a finite set of program variables, Str be a finite set of strings, and Fn be a set of function symbols that represent measurable maps of type $\mathbb{R}^k \to \mathbb{R}$. The language has the following syntax:

$$\text{Real Expr. } e ::= x \mid r \mid op(e_1, \ldots, e_k) \qquad \text{Boolean Expr. } b ::= \mathtt{true} \mid e_1 < e_2 \mid b_1 \wedge b_2 \mid \neg b$$
$$\text{Name Expr. } n ::= \mathtt{name}(\alpha, e) \qquad \text{Distribution Expr. } d ::= \mathtt{dist_N}(e, e')$$
$$\text{Command } c ::= \mathtt{skip} \mid x := e \mid c; c' \mid \mathtt{if}\, b\, \{c\}\, \mathtt{else}\, \{c'\} \mid \mathtt{while}\, b\, \{c\} \mid x := \mathtt{sam}(n, d, \lambda y.e) \mid \mathtt{obs}(d, r)$$

Here x, r, op, and α stand for a program variable in PVar, a real number, a function symbol in Fn, and a string in Str, respectively.

The language has four kinds of expressions, which denote maps from states to values of appropriate types. All the real and boolean expressions are standard. The name expressions n denote the identifiers of drawn samples (i.e., random variables). They are built by appending an integer (obtained by the floor of a real) to a string in Str; e.g., $\mathtt{name}(\mathtt{"z"}, 3.2)$ denotes the name $(\mathtt{"z"}, 3)$.[5] The distribution expression $\mathtt{dist_N}(e, e')$ denotes the normal distribution with mean e and variance e'. The language supports standard commands for imperative computation, and additionally has sample and observe for probabilistic programming. The sample command $x := \mathtt{sam}(n, d, \lambda y.e)$ creates a random variable named n by drawing a sample r from d; then, it transforms r to $e[r/y]$ and stores the result in the program variable x. In the programs written by the user of the language, only the identity function $\lambda y.y$ appears as the third argument of the sample commands. But as we explain later, when a program is constructed from another by a gradient estimator, such as the SPGE, it may contain sample commands with non-identity function arguments. The observe command $\mathtt{obs}(d, r)$ describes that an unnamed random variable is drawn from d and is immediately observed to have the value r. Computationally, $\mathtt{obs}(d, r)$ calculates the probability density of d at r and updates a variable that tracks the product of these probabilities from all the observations, by multiplying the variable with the calculated density.

[5] The name construct has the second argument to easily support the sampling of (conditionally) i.i.d. random variables.

Density semantics of the PPL.[6] We use a semantics of our language where commands are interpreted as calculators for densities, which may be unnormalised. Commands transform states, but in so doing, they compute densities of sampled random variables. More precisely, in the semantics, a command starts with an initial state that fixes not just the values of program variables but also those of all the random variables that are to be sampled during execution. When the command runs, it calculates the densities of those random variables at their given initial values, and also computes the probability density of all the observations, called *likelihood*. The product of all these densities and the likelihood becomes the so called *unnormalised posterior density*.

Let $\mathbb{N}$ be the set of natural numbers. Fix $N \in \mathbb{N}$ with $N \geq 1$. Formally, the semantics uses the states of the following form:

$$\begin{aligned} \mu \in \mathsf{Name} &\triangleq \{(\alpha, i) \mid \alpha \in \mathsf{Str}, i \in \mathbb{N} \cap [0, N)\}, \\ a \in \mathsf{AVar} &\triangleq \{\mathit{like}\} \cup \{\mathit{pr}_\mu, \mathit{val}_\mu, \mathit{cnt}_\mu \mid \mu \in \mathsf{Name}\}, \\ u, v \in \mathsf{Var} &\triangleq \mathsf{Name} \uplus \mathsf{PVar} \uplus \mathsf{AVar}, \\ \sigma \in \mathsf{St} &\triangleq [\mathsf{Var} \to \mathbb{R}], \quad \mathsf{St}[K] \triangleq [K \to \mathbb{R}] \text{ for } K \subseteq \mathsf{Var}. \end{aligned}$$

Here $\sigma(\mu)$ for $\mu \in \mathsf{Name}$ is the initial value of the random variable μ, which is used by the sample command and does not change during execution. For technical simplicity, the set Name has the restriction that the integer part of a name must be in $[0, N)$.[7] The set AVar consists of four types of auxiliary variables. The auxiliary variable *like* stores the likelihood (i.e., the probability density of all the observations), and its value is initialised to 1 and changes whenever the observe command $\mathtt{obs}(d, r)$ runs; the new value becomes the old times the density of the probability distribution d at r. The other auxiliary variables pr_μ, val_μ, and cnt_μ are associated with a random variable μ, standing for the "prior", "value", and "counter" of μ. They are initialised with $\mathcal{N}(\sigma(\mu); 0, 1)$ (i.e., the density of the standard normal distribution at $\sigma(\mu)$), $\sigma(\mu)$, and 0, respectively, and get updated by the sample command $x := \mathtt{sam}(n, d, \lambda y.e)$ where n denotes μ. The command increases cnt_μ by 1, so as to record the occurrence of a sampling event for μ. Then, it looks up the given value $\sigma(\mu)$ of the random variable μ, transforms the value to $e[\sigma(\mu)/y]$, and stores the result in x and val_μ. Finally, the command computes the density of the distribution d at the looked-up value $\sigma(\mu)$, and updates pr_μ with this density. The unnormalised posterior density (i.e., the joint density of all the random variables and observations) is then obtained by multiplying at the end of program execution the values of *like* and pr_μ for all $\mu \in \mathsf{Name}$.

The formal semantics of expressions is standard, and has the following types:

$$[\![e]\!] : \mathsf{St} \to \mathbb{R}, \qquad [\![b]\!] : \mathsf{St} \to \mathbb{B}, \qquad [\![n]\!] : \mathsf{St} \to \mathsf{Name}, \qquad [\![d]\!] : \mathsf{St} \to \mathbb{D}.$$

Here $\mathbb{B}$ is the set of booleans, i.e., *true* and *false*, and $\mathbb{D}$ is the set of positive probability-density functions on $\mathbb{R}$, i.e., a subset of $[\mathbb{R} \to (0, \infty)]$ whose elements are measurable functions that integrate to 1. The semantics is defined for a minor extension of the set of expressions where non-program variables are allowed to appear, such as $(\mu + x)$. The interpretation of expressions is mostly standard. We

[6] Our semantics is an instance (or variant) of existing density semantics (e.g., [Lee et al. 2020]), and is different from sampling semantics (e.g., [Staton et al. 2016]). Although the density semantics and the sampling semantics have different presentations, they are closely related and equivalent in a formal sense (see, e.g., [Lee et al. 2020]). We use the density semantics instead of the sampling semantics, because the gradient estimator (§4) of our interest performs computation on (unnormalised) densities and it is easier for a program analysis (§5) to work with the density semantics than the sampling semantics.

[7] This restriction is often respected by probabilistic programs in practice, since they commonly sample random variables whose number is uniformly bounded over all traces; note, however, that it is not always respected (e.g., by programs from Bayesian nonparametrics). The uniform bound N can often be found by a simple static analysis. This restriction along with the finiteness of PVar and Str implies the finiteness of Var, and this makes our technical development simpler since $\sigma \in \mathsf{St}$ becomes a function on a finite-dimensional space. Lifting the restriction would make the technical development more complicated, since this would require St to be isomorphic to $[\mathbb{R}^\infty \to \mathbb{R}]$ or $\biguplus_k [\mathbb{R}^k \to \mathbb{R}]$ and the former (or latter) choice of St makes defining differentiability (or formalising our program analysis) nontrivial; we leave it as future work.

show only the case for the name expressions $n \equiv \mathtt{name}(\alpha, e)$: $[\![\mathtt{name}(\alpha, e)]\!]\sigma \triangleq \mathit{create\_name}(\alpha, [\![e]\!]\sigma)$, where $\mathit{create\_name} : \mathrm{Str} \times \mathbb{R} \to \mathrm{Name}$ is an operator converting a string-real pair to a name, defined by $\mathit{create\_name}(\alpha, r) = (\alpha, \min\{\max\{\lfloor r \rfloor, 0\}, N-1\})$.[8]

Note that the types of the semantics of expressions imply that the evaluation of an expression always produces a value of the right type. In particular, it never generates an error. For instance, when an argument of an operator *op* or a distribution constructor $\mathtt{dist}_{\mathsf{N}}$ is outside its intended domain as in $\log(-3)$ and $\mathtt{dist}_{\mathsf{N}}(0, -2)$, or when the integer part of a name expression is outside $[0, N)$ as in $\mathtt{name}(\mathtt{"z"}, -1)$, our semantics does not generate an error. Instead, it returns some pre-chosen default value of the right type. This slightly unusual way of handling errors is also adopted in our semantics of commands to be presented shortly, and it lets us avoid the complexity caused by error handling when we formalise variational inference and develop our program analysis for smoothness properties.

The formal semantics of commands is also mostly standard with the handling of errors via default values, although its interpretation of sample and observe commands deserves special attention. Let $\bot$ be an element not in St, and define $\mathrm{St}_\bot$ to be the usual lifting of St with $\bot$. That is, $\mathrm{St}_\bot$ is a partially-ordered set $\mathrm{St} \uplus \{\bot\}$ with the following order: for $\xi, \xi' \in \mathrm{St}_\bot$, we have $\xi \sqsubseteq \xi'$ if and only if $\xi = \bot$ or $\xi = \xi'$. We write the standard lifting of a function $f : \mathrm{St} \to \mathrm{St}_\bot$ by $f^\dagger : \mathrm{St}_\bot \to \mathrm{St}_\bot$ (i.e., $f^\dagger(\xi) \triangleq$ if $(\xi = \bot)$ then ξ else $f(\xi)$). The semantics of a command c is a map $[\![c]\!] : \mathrm{St} \to \mathrm{St}_\bot$, and is defined inductively as shown below:

$$
\begin{aligned}
[\![\mathtt{skip}]\!]\sigma &\triangleq \sigma,\\
[\![x := e]\!]\sigma &\triangleq \sigma[x \mapsto [\![e]\!]\sigma],\\
[\![c; c']\!]\sigma &\triangleq [\![c']\!]^\dagger([\![c]\!]\sigma),\\
[\![\mathtt{if}\ b\ \{c\}\ \mathtt{else}\ \{c'\}]\!]\sigma &\triangleq \text{if } ([\![b]\!]\sigma = \mathit{true}) \text{ then } [\![c]\!]\sigma \text{ else } [\![c']\!]\sigma,\\
[\![\mathtt{while}\ b\ \{c\}]\!]\sigma &\triangleq (\mathit{fix}\, F)(\sigma) \quad \text{where } F(f)(\sigma) \triangleq \text{if } ([\![b]\!]\sigma = \mathit{true}) \text{ then } f^\dagger([\![c]\!]\sigma) \text{ else } \sigma,\\
[\![x := \mathtt{sam}(n, d, \lambda y.e')]\!]\sigma &\triangleq \sigma[x \mapsto r,\ \mathit{val}_\mu \mapsto r, \mathit{pr}_\mu \mapsto [\![d]\!]\sigma(\sigma(\mu)),\ \mathit{cnt}_\mu \mapsto \sigma(\mathit{cnt}_\mu) + 1]\\
&\qquad \text{where } \mu \triangleq [\![n]\!]\sigma \text{ and } r \triangleq [\![e'[\mu/y]]\!]\sigma,\\
[\![\mathtt{obs}(d, r)]\!]\sigma &\triangleq \sigma[\mathit{like} \mapsto \sigma(\mathit{like}) \cdot [\![d]\!]\sigma(r)].
\end{aligned}
$$

The interpretation uses the least fixed-point operator *fix* for continuous maps F on the function space $[\mathrm{St} \to \mathrm{St}_\bot]$, where the function space is ordered pointwise and continuity means the one with respect to this order. It also uses the notation $e'[e''/y]$ to denote the substitution of y in e' by e''. According to this interpretation, $x := \mathtt{sam}(n, d, \lambda y.e')$ increments the cnt_μ variable for the name $n = \mu$ so that the variable, which has 0 initially, records the number of times that the random variable with the same name n is sampled during execution.

Having some cnt_μ variable increased by 2 or larger at some point of execution is not an intended behaviour of a command c. That is, if c is a well-designed command, every random variable with a fixed name should be sampled at most once during the execution of c. This intended behaviour of commands plays an important role in our results, and we refer to it using the following terminology.

Definition 3.1. An always-terminating command c *does not have a double-sampling error* if for any $\sigma \in \mathrm{St}$, we have $[\![c]\!]\sigma(\mathit{cnt}_\mu) - \sigma(\mathit{cnt}_\mu) \leq 1$ for all $\mu \in \mathrm{Name}$. □

Example 3.2 (Density semantics). Consider the following state $\sigma \in \mathrm{St}$: $\sigma \triangleq [x \mapsto 0, y \mapsto 0, (\mathtt{"a"}, 0) \mapsto 2, (\mathtt{"b"}, 0) \mapsto 4, \mathit{cnt}_{(\mathtt{"a"},0)} \mapsto 0, \mathit{cnt}_{(\mathtt{"b"},0)} \mapsto 0, \cdots]$, where $x, y \in \mathrm{PVar}$ denote program variables and $(\mathtt{"a"}, 0), (\mathtt{"b"}, 0) \in \mathrm{Name}$ denote random variables. Note that σ consists of three parts: the PVar part says that the values of x and y are both 0; the Name part says that the values of $(\mathtt{"a"}, 0)$ and $(\mathtt{"b"}, 0)$ are 2 and 4; and the AVar part says that $(\mathtt{"a"}, 0)$ and $(\mathtt{"b"}, 0)$ have not been sampled yet.

[8] There are multiple valid ways to convert a string-real pair to a name (i.e., to define *create_name*); we choose just one of them.

Next, consider a command $c \equiv \big(x := \mathtt{sam}(\mathtt{name}(\mathtt{"a"}, 0), \mathtt{dist}_{\mathsf{N}}(-3, 1), \lambda z.z);\ y := \mathtt{sam}(\mathtt{name}(\mathtt{"b"}, 0), \mathtt{dist}_{\mathsf{N}}(5, 1), \lambda z.z)\big)$. Given the command c and the input state σ, our density semantics computes the following output state $\sigma' \in \mathrm{St}$: $\sigma' \triangleq [\![c]\!]\sigma = [x \mapsto 2, y \mapsto 4, (\mathtt{"a"}, 0) \mapsto 2, (\mathtt{"b"}, 0) \mapsto 4, cnt_{(\mathtt{"a"},0)} \mapsto 1, cnt_{(\mathtt{"b"},0)} \mapsto 1, pr_{(\mathtt{"a"},0)} \mapsto \mathcal{N}(2; -3, 1), pr_{(\mathtt{"b"},0)} \mapsto \mathcal{N}(4; 5, 1), \cdots]$. The input/output states σ and σ' illustrate two aspects of our semantics. First, the semantics uses the Name part of an input state to determine the sampled values of sample commands: x and y in σ' take the values of $\sigma((\mathtt{"a"}, 0)) = 2$ and $\sigma((\mathtt{"b"}, 0)) = 4$. Second, the semantics records, in the AVar part of an output state, the densities of sample commands: $pr_{(\mathtt{"a"},0)}$ and $pr_{(\mathtt{"b"},0)}$ in σ' store the densities of the two sample commands in c evaluated at 2 and 4. □

Variational inference. We consider the most common form of *variational inference* for Pyro-like PPLs where we are asked to learn a good approximation of the posterior of a given model, i.e., the conditional distribution of the model given a dataset. Typically, a parameterised approximate posterior is given in variational inference, and learning corresponds to finding good values of those parameters. A popular approach is to measure the quality of parameter values by the so called evidence lower bound (ELBO), and to optimise ELBO.

To translate what we have described so far to our context, we need to explain a general recipe for generating a density p_c for a command c, which is in general unnormalised (i.e., does not integrate to 1). The recipe specifies p_c as follows: for each $\sigma_\theta \in \mathrm{St}[\theta]$ with $\theta \subseteq \mathrm{PVar}$, $p_{c,\sigma_\theta} : \mathrm{St}[\mathsf{Name}] \to [0, \infty)$ is defined by

$$p_{c,\sigma_\theta}(\sigma_n) \triangleq \begin{cases} [\![c]\!]\sigma(\mathit{like}) \cdot \prod_{\mu \in \mathsf{Name}} [\![c]\!]\sigma(pr_\mu) & \text{if } [\![c]\!]\sigma \in \mathrm{St} \text{ and } [\![c]\!]\sigma(cnt_\mu) \le 1 \text{ for all } \mu \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where $\sigma = \sigma_0 \oplus \sigma_\theta \oplus \sigma_n \in \mathrm{St}$, and the $\oplus$ operator combines two real-valued maps with disjoint domains in the standard way. Also, $\sigma_0 \in \mathrm{St}[(\mathrm{PVar} \setminus \theta) \uplus \mathrm{AVar}]$ maps *like* to 1, and pr_μ to $\mathcal{N}(\sigma_n(\mu); 0, 1)$ and val_μ to $\sigma_n(\mu)$ for every $\mu \in \mathsf{Name}$, and all other variables to 0. Here $\mathrm{St}[\mathsf{Name}]$ is understood as a measurable space constructed by taking the product of the $|\mathsf{Name}|$ copies of the measurable space $\mathbb{R}$ and the integral is taken over the uniform measure on $\mathrm{St}[\mathsf{Name}]$ (i.e., the product of the $|\mathsf{Name}|$ copies of the Lebesgue measure on $\mathbb{R}$).

In variational inference in our PPL context, we are given two commands c_m and c_g, called *model* and *guide*. We assume that (i) these commands always terminate and do not have a double-sampling error, (ii) some variables $\theta = \{\theta_1, \ldots, \theta_k\} \subseteq \mathrm{PVar}$ that only appear in c_g, not in c_m, are identified as parameters to be optimised, and (iii) the density p_{c_g,σ_θ} of the guide c_g integrates to 1 and defines a probability distribution.[9] Given the model-guide pair (c_m, c_g), a popular approach for variational inference is to solve the following optimisation problem approximately,

$$\operatorname*{argmax}_{\sigma_\theta} \mathbb{E}_{p_{c_g,\sigma_\theta}(\sigma_n)} \left[\log(p_{c_m}(\sigma_n)/p_{c_g,\sigma_\theta}(\sigma_n))\right], \tag{4}$$

when the expectation is well-defined for all σ_θ. The objective of this optimisation is the ELBO that we mentioned earlier. Here p_{c_m} means p_{c_m,σ'_θ} for some/any σ'_θ; the choice of σ'_θ does not matter since c_m does not access the parameters θ and so $p_{c_m,\sigma'_\theta} = p_{c_m,\sigma''_\theta}$ for all $\sigma'_\theta, \sigma''_\theta \in \mathrm{St}[\theta]$.

Often variational inference is applied when the model c_m is parameterised as well. In those cases, it asks for finding good parameters of the model c_m as well as those of the guide c_g. So, an algorithm for variational inference this time simultaneously learns a good model for given observations and a good approximate posterior for the learnt model. This more general form of variational inference

[9] In practice, one more assumption is required: the set of random variables sampled from the model should be the same as that from the guide. This assumption can be checked automatically, e.g., by [Lee et al. 2020; Lew et al. 2020]. In this work, however, this assumption is always satisfied: all random variables in Name are sampled by a sample command or at the beginning (via initialisation).

can be easily accommodated in our setup. We just need to drop the condition that the parameters may not appear in c_m, and to use p_{c_m,σ_θ} instead of p_{c_m} in the optimisation objective in Eq. (4):

$$\operatorname*{argmax}_{\sigma_\theta} \mathcal{L}(\sigma_\theta) \text{ for } \mathcal{L}(\sigma_\theta) \triangleq \mathbb{E}_{p_{c_g,\sigma_\theta}(\sigma_n)}\left[\log(p_{c_m,\sigma_\theta}(\sigma_n)/p_{c_g,\sigma_\theta}(\sigma_n))\right]. \tag{5}$$

The rest of the paper focuses on this general form of variational inference (often called model learning).

4 SELECTIVE PATHWISE GRADIENT ESTIMATOR

We consider a gradient-based algorithm for the optimisation problem in Eq. (5). The algorithm finds a good σ_θ by repeatedly estimating the gradient of the optimisation objective at the current σ_θ,

$$\text{grad\_est}(\sigma_\theta) \approx \nabla_\theta \mathbb{E}_{p_{c_g,\sigma_\theta}(\sigma_n)}\left[\log(p_{c_m,\sigma_\theta}(\sigma_n)/p_{c_g,\sigma_\theta}(\sigma_n))\right],$$

and updating σ_θ with the estimate under a learning rate $\eta > 0$, that is, $\sigma_\theta \leftarrow \sigma_\theta + \eta \cdot \text{grad\_est}(\sigma_\theta)$. Note that the core of the algorithm lies in the computation of $\text{grad\_est}(\sigma_\theta)$.

In this section, we describe a particular algorithm for the gradient computation, called *selective pathwise gradient estimator (SPGE)*, which is often regarded as the algorithm of choice and corresponds to the inference algorithm developed for stochastic computation graphs [Schulman et al. 2015] and implemented for Pyro. Our description of the SPGE takes the often-ignored aspect of customising the SPGE algorithm for PPLs seriously, and it is accompanied with a novel formal analysis of the customisation (§4.1 and §4.2). Our analysis clearly identifies information about probabilistic programs that is useful for this customised SPGE algorithm, and prepares the stage for our program analysis for smoothness properties in §5 (§4.2 and §4.3).

4.1 Program Transformation

We start by describing a program transformation that changes some sample commands in a given probabilistic program. This transformation is used crucially by the SPGE.

The key component of the transformation is a partial function π called *reparameterisation plan*, which has the type $\mathsf{NameEx} \times \mathsf{DistEx} \times \mathsf{LamEx} \rightharpoonup \mathsf{DistEx} \times \mathsf{LamEx}$. Here NameEx, DistEx, and LamEx denote the sets of name expressions, distribution expressions, and lambda expressions of the form $\lambda y.e$, respectively. The plan π specifies how we transform sample commands. Concretely, assume that we are given $x := \mathtt{sam}(n, d, \lambda y.e)$. We check whether $\pi(n, d, \lambda y.e)$ is defined or not. If not, we keep the original sample command. Otherwise, if $\pi(n, d, \lambda y.e)$ is $(d', \lambda y'.e')$, we replace the command with $x := \mathtt{sam}(n, d', \lambda y'.e')$.

A natural extension of this intended transformation of π leads to the following program transformation for a general command c, denoted by $\overline{c}^{\pi}$:

$$\begin{aligned}
\overline{\mathtt{skip}}^{\pi} &\triangleq \mathtt{skip},\\
\overline{x := e}^{\pi} &\triangleq x := e,\\
\overline{c; c'}^{\pi} &\triangleq \overline{c}^{\pi}; \overline{c'}^{\pi},\\
\overline{\mathtt{if}\ b\ \{c\}\ \mathtt{else}\ \{c'\}}^{\pi} &\triangleq \mathtt{if}\ b\ \{\overline{c}^{\pi}\}\ \mathtt{else}\ \{\overline{c'}^{\pi}\},\\
\overline{\mathtt{while}\ b\ \{c\}}^{\pi} &\triangleq \mathtt{while}\ b\ \{\overline{c}^{\pi}\},\\
\overline{x := \mathtt{sam}(n, d, l)}^{\pi} &\triangleq \begin{cases} x := \mathtt{sam}(n, d', l') & \text{if } \exists (d', l').\ \pi(n, d, l) = (d', l')\\ x := \mathtt{sam}(n, d, l) & \text{otherwise,}\end{cases}\\
\overline{\mathtt{obs}(d, r)}^{\pi} &\triangleq \mathtt{obs}(d, r).
\end{aligned}$$

The transformation recursively traverses c, and applies π to all the sample commands in c. Note that for any π, there exists a total function π' such that $\overline{c}^{\pi} = \overline{c}^{\pi'}$ for all c; the π' coincides with π in the domain of π, and outside of this domain, it is the identity function. But such π' loses information about the domain of π, which plays a crucial role in our formalisation of the SPGE.

We are primarily interested in semantics-preserving instances of $\overline{\cdot}^{\pi}$. The next definition helps us to identify such instances.

Definition 4.1. A reparameterisation plan π is *valid* if for all $n \in \mathsf{NameEx}$, $d, d' \in \mathsf{DistEx}$, and $(\lambda y.e), (\lambda y'.e') \in \mathsf{LamEx}$ such that $\pi(n, d, \lambda y.e) = (d', \lambda y'.e')$, the following condition holds: for all states $\sigma \in \mathsf{St}$ and measurable subsets $A \subseteq \mathbb{R}$,

$$\int \mathbf{1}_{[\![e[r/y]]\!]\sigma \in A]} \cdot [\![d]\!]\sigma(r)\, dr = \int \mathbf{1}_{[\![e'[r/y']]\!]\sigma \in A]} \cdot [\![d']\!]\sigma(r)\, dr. \qquad \square\,(6)$$

The condition says that the distribution obtained by sampling from d and applying $\lambda y.e$ is the same as that obtained by sampling from d' and applying $\lambda y'.e'$. An example of a widely-used valid reparameterisation plan maps its input as follows, whenever defined: $\pi_0(n,\ \mathtt{dist}_{\mathsf{N}}(e_1, e_2),\ \lambda y.e_3) = (\mathtt{dist}_{\mathsf{N}}(0,1),\ \lambda y.e_3[(y \times \sqrt{e_2} + e_1)/y])$, where we assume y does not appear in e_1 and e_2, the substitution in π_0 expresses the composition of two functions $\lambda y.e_3$ and $\lambda y.(y \times \sqrt{e_2} + e_1)$, and $\sqrt{-}$ denotes a square-root operator that handles non-positive arguments in the same way as $\mathtt{dist}_{\mathsf{N}}(e, -)$ does: if $[\![e_2]\!]\sigma \leq 0$ and $[\![\mathtt{dist}_{\mathsf{N}}(e_1, e_2)]\!]\sigma = \lambda r.\, \mathcal{N}(r; [\![e_1]\!]\sigma, r_2)$ for some $r_2 > 0$, then $[\![\sqrt{e_2}]\!]\sigma = \sqrt{r_2}$. The above plan satisfies the condition in Eq. (6), because $y \times \sqrt{r_2} + r_1$ with a sample y from $\mathcal{N}(0,1)$ is distributed by $\mathcal{N}(r_1, r_2)$.

We now show that $\overline{\cdot}^{\pi}$ with a valid π preserves semantics. For a command c and $\sigma_\theta \in \mathsf{St}[\theta]$, define the *value function* $v_{c,\sigma_\theta} : \mathsf{St}[\mathsf{Name}] \to \mathsf{St}[\mathsf{Name}]$ as follows:

$$v_{c,\sigma_\theta}(\sigma_n)(\mu) \triangleq \text{let } \sigma \triangleq \sigma_0 \oplus \sigma_\theta \oplus \sigma_n \text{ in } \begin{cases} [\![c]\!]\sigma(val_\mu) & \text{if } [\![c]\!]\sigma \in \mathsf{St} \text{ and } [\![c]\!]\sigma(cnt_{\mu'}) \leq 1 \text{ for all } \mu' \\ 0 & \text{otherwise} \end{cases}$$

where $\sigma_0 \in \mathsf{St}[(\mathsf{PVar} \setminus \theta) \uplus \mathsf{AVar}]$ maps *like* to 1, and pr_μ to $\mathcal{N}(\sigma_n(\mu); 0, 1)$ and val_μ to $\sigma_n(\mu)$ for every $\mu \in \mathsf{Name}$, and it also maps all the other variables to 0. The value function basically applies the lambda functions in c's sample commands to the corresponding random variables. The next theorem proves that if π is valid, the program transformation $\overline{\cdot}^{\pi}$ preserves the semantics in the sense that the integral of a function h remains the same under c and $\overline{c}^{\pi}$ for any c. Note that the two integrals in the theorem are connected via the value functions of c and $\overline{c}^{\pi}$.

Theorem 4.2. *Let π be a valid reparameterisation plan, and c be a command. Then, for all $\sigma_\theta \in \mathsf{St}[\theta]$ and all measurable $h : \mathsf{St}[\mathsf{Name}] \to \mathbb{R}$, we have*

$$\int d\sigma_n \Big(p_{c,\sigma_\theta}(\sigma_n) \cdot h(v_{c,\sigma_\theta}(\sigma_n)) \Big) = \int d\sigma_n \Big(p_{\overline{c}^{\pi},\sigma_\theta}(\sigma_n) \cdot h(v_{\overline{c}^{\pi},\sigma_\theta}(\sigma_n)) \Big)$$

where the left integral is defined if and only if the right integral is defined.

Remark 4.3. One immediate yet important consequence of the theorem is that if p_{c,σ_θ} is a probability density, so is $p_{\overline{c}^{\pi},\sigma_\theta}$. This consequence will be used in §4.2 and the proof of Theorem 4.5 later. $\square$

4.2 Gradient Estimator via Program Transformation

Let c be a command that always terminates and does not have a double-sampling error, and let $\sigma_\theta \in \mathsf{St}[\theta]$. We define the *partial density function* $p^{\langle S \rangle}_{c,\sigma_\theta}$ *of c over a subset $S \subseteq \mathsf{Name}$* as

$$p^{\langle S \rangle}_{c,\sigma_\theta} : \mathsf{St}[\mathsf{Name}] \to (0, \infty), \qquad p^{\langle S \rangle}_{c,\sigma_\theta}(\sigma_n) \triangleq \prod_{\mu \in S} [\![c]\!](\sigma_0 \oplus \sigma_\theta \oplus \sigma_n)(pr_\mu),$$

where σ_0 is set as in the definition of p_{c,σ_θ} in Eq. (3). The partial density $p^{\langle S \rangle}_{c,\sigma_\theta}$ is essentially the full density p_{c,σ_θ} in Eq. (3) with the omission of the factors not mentioned in S. Intuitively, it computes the density of the random variables in S conditioned on the random variables outside of S.

The SPGE computes an approximate gradient of the objective $\mathcal{L}$ in Eq. (5) using the program transformation in the previous subsection. Its inputs are a model c_m, a guide c_g, parameters θ to optimise, and a reparameterisation plan π, where

- c_m, c_g, and $\overline{c_g}^{\pi}$ always terminate and do not have a double-sampling error, and
- c_g defines the normalised probability density p_{c_g,σ_θ} for all $\sigma_\theta \in \mathsf{St}[\theta]$. (7)

Given these inputs, the SPGE computes an approximate gradient in three steps. First, it defines the set $rv(\pi) \subseteq \mathsf{Name}$ of random variables to be reparameterised:

$$rv(\pi) \triangleq \{(\alpha, i) \in \mathsf{Name} \mid (\mathsf{name}(\alpha, \_), \_, \_) \in \mathrm{dom}(\pi)\}$$

where _ means some existentially quantified (meta) variable. Second, the SPGE transforms the guide c_g to $\overline{c_g}^{\pi}$, and draws a sample $\hat{\sigma}_n$ from $p_{\overline{c_g}^{\pi},\sigma_\theta}$.[10] Drawing a sample $\hat{\sigma}_n$ makes sense here since $p_{\overline{c_g}^{\pi},\sigma_\theta}$ is a probability density (i.e., it normalises to 1) by Remark 4.3. Another important point is that drawing $\hat{\sigma}_n$ can be done simply by executing $\overline{c_g}^{\pi}$ in the standard sampling semantics (not in our density semantics), where each sample command is interpreted as a random draw, not as a density calculator. Third, the SPGE computes the following approximation of $\nabla_\theta \mathcal{L}(\sigma_\theta)$ and returns it as a result:

$$\begin{aligned}\mathrm{grad\_est}(\sigma_\theta; \hat{\sigma}_n) \triangleq\ & \left(\nabla_\theta \log p^{\langle \mathsf{Name} \setminus rv(\pi)\rangle}_{c_g,\sigma_\theta}(\sigma'_n)\right) \cdot \log(p_{c_m,\sigma_\theta}(\sigma'_n)/p_{c_g,\sigma_\theta}(\sigma'_n)) \\ & - \nabla_\theta \log p^{\langle rv(\pi)\rangle}_{c_g,\sigma_\theta}(\sigma'_n) + \nabla_\theta \log p_{c_m,\sigma_\theta}(\sigma'_n), \qquad \text{for } \sigma'_n \triangleq v_{\overline{c_g}^{\pi},\sigma_\theta}(\hat{\sigma}_n).\end{aligned} \tag{8}$$

Recall that if a command c always terminates, both the partial density $p^{\langle S\rangle}_{c,\sigma_\theta}(\sigma_n)$ and the full density $p_{c,\sigma_\theta}(\sigma_n)$ can be computed simply by executing c in our semantics and calculating the defining formulas of both densities from the final state of the execution. Thus, all the terms in grad_est can be computed by executing c_g and c_m according to our density semantics or differentiating the results of these executions via, for instance, automatic differentiation as done in Pyro. Note that grad_est applies two non-trivial optimisations, when compared with the (naive) SPGE explained in Eq. (2): its first term involves a partial density of c_g instead of the full density of $\overline{c_g}^{\pi}$, and its second term involves (again) a partial density of c_g instead of the full density of c_g.

Is the SPGE correct in any sense? The answer depends on its inputs. If the inputs satisfy the requirements that we will explain soon, the result of the SPGE is precisely $\nabla_\theta \mathcal{L}(\sigma_\theta)$ on average, that is, $\nabla_\theta \mathcal{L}(\sigma_\theta) = \mathbb{E}[\mathrm{grad\_est}(\sigma_\theta; \hat{\sigma}_n)]$, where the expectation is taken over the sample $\hat{\sigma}_n$ used by the SPGE. This property is called *unbiasedness*, and it plays the crucial role for ensuring that parameters updated iteratively with estimated gradients converge to a local optimum.

Let us now spell out the requirements on the inputs of the SPGE. To do so, we need to introduce one further concept for the reparameterisation plans π.

Definition 4.4. A reparameterisation plan π is *simple* if for all $(n, d, \lambda y.e)$ and $(n', d', \lambda y'.e')$ in $\mathsf{NameEx} \times \mathsf{DistEx} \times \mathsf{LamEx}$ such that n and n' have the same string part, we have $(n, d, \lambda y.e) \in \mathrm{dom}(\pi) \iff (n', d', \lambda y'.e') \in \mathrm{dom}(\pi)$. □

The simplicity is one of the requirements that the SPGE imposes on π. It ensures the following property of the set $rv(\pi)$, which the SPGE relies on when computing grad_est by Eq. (8): $rv(\pi)$ (and $\mathsf{Name} \setminus rv(\pi)$) over-approximates the set of random variables that, if sampled, are (and are not) reparameterised by $\overline{\cdot}^{\pi}$. Specifically, it forbids π from using any syntax-specific information of the arguments of a sample command when it decides whether to transform the command or not. All the requirements of the SPGE, including the simplicity just explained, are summarised in the next theorem.

Theorem 4.5. *Let c_m, c_g, and π be the inputs to the SPGE (i.e., they satisfy the assumptions in Eq. (7)). Suppose that $\mathcal{L}(\sigma_\theta)$ and $\nabla_\theta \mathcal{L}(\sigma_\theta)$ are well-defined for every $\sigma_\theta \in \mathsf{St}[\theta]$. Further, assume that every*

[10] In practice, the SPGE often draws a fixed number of independent samples $\hat{\sigma}^{(1)}_n, \ldots, \hat{\sigma}^{(M)}_n$ from $p_{\overline{c_g}^{\pi},\sigma_\theta}$ and computes $\frac{1}{M}\sum_{i=1}^{M} \mathrm{grad\_est}(\sigma_\theta; \hat{\sigma}^{(i)}_n)$ as an estimate of $\nabla_\theta \mathcal{L}(\sigma_\theta)$. The presented results hold for this more general case as well.

sample command in c_g has $\lambda y.y$ as its third argument, and c_g does not have observe commands. Then, for all $\sigma_\theta \in \mathrm{St}[\theta]$,

$$\nabla_\theta \mathcal{L}(\sigma_\theta) = \mathbb{E}_{p_{\overline{c_g}^\pi,\sigma_\theta}(\hat{\sigma}_n)}\left[\mathrm{grad\_est}(\sigma_\theta;\hat{\sigma}_n)\right] \tag{9}$$

if π satisfies the following requirements:

(R1) π is valid and simple.

(R2) The below functions from $\mathrm{St}[\theta] \times \mathrm{St}[\mathsf{Name}]$ to $(0,\infty)$ are differentiable in $\theta \cup rv(\pi)$ jointly:

$$(\sigma_\theta,\sigma_n) \longmapsto p_{c_m,\sigma_\theta}(\sigma_n), \qquad (\sigma_\theta,\sigma_n) \longmapsto p^{\langle rv(\pi)\rangle}_{c_g,\sigma_\theta}(\sigma_n), \qquad (\sigma_\theta,\sigma_n) \longmapsto p^{\langle \mathsf{Name}\setminus rv(\pi)\rangle}_{c_g,\sigma_\theta}(\sigma_n).$$

(R3) For all $\sigma_n \in \mathrm{St}[\mathsf{Name}]$, the below functions on $\mathrm{St}[\theta]$ are differentiable in θ jointly:

$$\sigma_\theta \longmapsto v_{\overline{c_g}^\pi,\sigma_\theta}(\sigma_n), \qquad \sigma_\theta \longmapsto p^{\langle rv(\pi)\rangle}_{\overline{c_g}^\pi,\sigma_\theta}(\sigma_n), \qquad \sigma_\theta \longmapsto p^{\langle \mathsf{Name}\setminus rv(\pi)\rangle}_{\overline{c_g}^\pi,\sigma_\theta}(\sigma_n).$$

(R4) For all $\sigma_\theta \in \mathrm{St}[\theta]$ and $\sigma_n \in \mathrm{St}[\mathsf{Name}]$, we have $\nabla_\theta p^{\langle rv(\pi)\rangle}_{\overline{c_g}^\pi,\sigma_\theta}(\sigma_n) = 0$.

(R5) The below equations hold for all $\sigma_\theta \in \mathrm{St}[\theta]$:

$$\nabla_\theta \int d\sigma_n \left(p_{\overline{c_g}^\pi,\sigma_\theta}(\sigma_n)\right) = \int d\sigma_n \nabla_\theta \left(p_{\overline{c_g}^\pi,\sigma_\theta}(\sigma_n)\right),$$

$$\nabla_\theta \int d\sigma_n \left(p_{\overline{c_g}^\pi,\sigma_\theta}(\sigma_n) \cdot \log \frac{p_{c_m,\sigma_\theta}(\sigma_n')}{p_{c_g,\sigma_\theta}(\sigma_n')}\right) = \int d\sigma_n \, \nabla_\theta \left(p_{\overline{c_g}^\pi,\sigma_\theta}(\sigma_n) \cdot \log \frac{p_{c_m,\sigma_\theta}(\sigma_n')}{p_{c_g,\sigma_\theta}(\sigma_n')}\right).$$

In the second equation, we write σ_n' for $v_{\overline{c_g}^\pi,\sigma_\theta}(\sigma_n)$.

To be clear, $f : \mathrm{St}[K] \to \mathbb{R}^n$ for $K \subseteq \mathrm{Var}$ is said to be *differentiable in $K' \subseteq K$ jointly* if for any $\tau \in \mathrm{St}[K \setminus K']$, $f|_{[\tau]} : \mathrm{St}[K'] \to \mathbb{R}^n$ is (jointly) differentiable, where $f|_{[\tau]}(\sigma) \triangleq f(\sigma \oplus \tau)$.

In this work, we focus on the requirements (R2) and (R3) about smoothness. They require that five density functions of c_m, c_g, and $\overline{c_g}^\pi$, and the value function of $\overline{c_g}^\pi$ be differentiable in certain variables. We will develop a program-analysis framework to check these differentiability requirements soundly and automatically (§5), and will describe an algorithm to the SPGE variable-selection problem, using the developed analysis framework (§6). The remaining requirements (R1), (R4), and (R5) are of less interest in this work. (R1) and (R4) can be guaranteed by simple syntactic checks and our way of constructing π (Lemma D.1). (R5) follows from (R2) and (R3) under a regularity condition on densities (Theorem D.2) which is usually satisfied in practice; in some cases, however, the condition might not hold, and we leave it as future work to automatically discharge the condition (which is about the integrability of local Lipschitz constants of densities) or more generally (R5).[11]

We point out that Pyro uses the SPGE in their inference engine, but without checking the above requirements. In particular, its default option simply uses the π that transforms all the continuous random variables in a guide, and this can easily violate the requirements and make the SPGE biased.

4.3 Local Lipschitzness for Relaxed Requirements

In Theorem 4.5, we considered the requirements (R2) and (R3) about the differentiability of density and value functions, as a sufficient condition for the unbiasedness of the SPGE. They are, however, sometimes too strong to hold in practice due to the use of popular non-differentiable functions (e.g., ReLU). As we will see in §7, the requirements are indeed violated by some representative Pyro programs even though the conclusion of Theorem 4.5 holds for those programs (i.e., the estimated gradients by the SPGE for those programs are unbiased).

To validate the unbiasedness of the SPGE for more examples in practice, we consider the following relaxation of the requirements (R2) and (R3), which changes differentiability to local Lipschitzness:

(R2') The functions in (R2) are locally Lipschitz in $\theta \cup rv(\pi)$ jointly.

(R3') For every $\sigma_n \in \mathrm{St}[\mathsf{Name}]$, the functions in (R3) are locally Lipschitz in θ jointly.

[11] Some sufficient conditions for a regularity condition similar to the one considered here have been studied for certain classes of model-guide pairs, e.g., in [Lee et al. 2020].

Here $f : V \to \mathbb{R}^m$ for $V \subseteq \mathbb{R}^n$ is *Lipschitz* if there is $C > 0$ such that $\|f(x) - f(x')\|_2 \leq C\|x - x'\|_2$ for all $x, x' \in V$; and f is *locally Lipschitz* if for all $x \in \mathbb{R}^n$, there is an open neighborhood $U \subseteq \mathbb{R}^n$ of x such that $f|_U : U \to \mathbb{R}^m$ is Lipschitz. Further, $f : \mathrm{St}[K] \to \mathbb{R}^m$ for $K \subseteq \mathrm{Var}$ is *locally Lipschitz in* $K' \subseteq K$ *jointly* if for any $\tau \in \mathrm{St}[K \setminus K']$, $f|_{[\tau]} : \mathrm{St}[K'] \to \mathbb{R}^m$ is locally Lipschitz, where $f|_{[\tau]}(\sigma) \triangleq f(\sigma \oplus \tau)$. Although differentiability does not imply local Lipschitzness, *continuous* differentiability does. Since most differentiable functions used in practice are continuously differentiable, asking for (R2') and (R3') amounts to relaxing the requirements of (R2) and (R3) in practice.

We choose local Lipschitzness as an alternative to differentiability in (R2) and (R3) for two main reasons. First, local Lipschitzness is satisfied by most functions used in practice, which can even be non-differentiable (e.g., ReLU). This would allow us to validate the unbiasedness of the SPGE for more programs, even when they use non-differentiable functions. Second, using local Lipschitzness in (R2) and (R3) does not break the results given in §4.2, even though local Lipschitzness is more practically permissive than differentiability (as explained above). In particular, we have a counterpart of Theorem 4.5 that uses (R2') and (R3') instead of (R2) and (R3):

Theorem 4.6. *Consider the setup of Theorem 4.5. Then, for all* $\sigma_\theta \in \mathrm{St}[\theta]$, *Eq.* (9) *holds if* π *satisfies the requirements (R1), (R2'), (R3'), (R4), and (R5).*

We can obtain Theorem 4.6 thanks to three properties of local Lipschitzness: locally Lipschitz functions are closed under function composition, have well-defined gradients almost everywhere, and satisfy the chain rule almost everywhere in restricted settings (Lemma E.1). Although the latter two properties are weaker than the corresponding properties of differentiability (i.e., differentiable functions always have well-definedness gradients and satisfy the chain rule), they are still strong enough to prove the theorem.

As in §4.2, we focus on the requirements (R2') and (R3') and less on (R1), (R4), and (R5). Note that (R1) and (R4) can be checked syntactically in the same way as discussed in §4.2. On the other hand, (R5) follows from (R2') and (R3') now under two (instead of one) regularity conditions on densities (Theorem E.2), where the first condition is the one mentioned in §4.2 and the second (new) condition is about some form of almost-everywhere differentiability of densities. Although we believe that the two regularity conditions would usually hold in practice, they can be violated in some cases that involve locally Lipschitz, non-differentiable functions; hence, it would be worthwhile to devise an automatic way of checking the two conditions or more generally (R5), which we leave as future work.

Because local Lipschitzness property has wider coverage than differentiability in practice while ensuring that the results in §4.2 remain valid, our implementation and experiments consider the option of using (R2') and (R3') as well as that of using (R2) and (R3); see §5.3–§7 for details.

5 PROGRAM ANALYSIS FOR SMOOTHNESS

Recall that our goal is to develop an algorithm for the SPGE variable-selection problem in Definition 2.1, which asks for finding a large set S of random variables with a certain property when given a model c_m, a guide c_g, and a reparameterisation plan π_0. When rephrased using the terminologies that we covered so far, finding such an S amounts to finding a restriction π of the given π_0 such that (c_m, c_g, π) satisfies the requirements in Theorem 4.5 (or 4.6). Thus, the key for developing a desired algorithm for the problem lies in constructing an automatic method for proving that the requirements in Theorem 4.5 (or 4.6), in particular, the smoothness requirements (R2) and (R3) (or (R2') and (R3')) are met. In this section, we propose a program analysis for smoothness properties, which can help find π that meets the smoothness requirements, and which, together with the optimiser in the next section, leads to an algorithm for solving the SPGE variable-selection problem.

We first define a parametric abstraction for smoothness properties (§5.1). We then describe a program analysis based on this abstraction and prove the soundness of the analysis (§5.2). We finally

instantiate the analysis to differentiability and local Lipschitzness (§5.3). The results in this section are not limited to PPLs, but are applicable to general imperative programming languages.

5.1 Parametric Abstraction for Smoothness Properties

At a high level, our parametric abstraction for smoothness properties is built out of two components. The first is a predicate over commands that expresses a target smoothness property but in a conditional form. The predicate is parameterised by two sets of variables, K for the input variables and L for the output variables. Intuitively, the predicate holds for a command if conditioning the input variables outside of K to any fixed values and varying only the ones in K makes the command a smooth function on the output variables in L. Our program analysis tracks a conditional smoothness property formalised by this predicate, and in so doing, it identifies a smooth part of a given command, even when the command fails to be so with respect to some variables.

The second component is also a predicate over commands, but it deals with dependency, instead of smoothness. It is again parameterised by K and L, and expresses that to compute the output variables in L, a command accesses only the input variables in K. Our program analysis tracks dependency formalised by this predicate, so as to achieve high precision, especially when handling sequential composition. To see this, imagine that we want to check the differentiability of a sequence $c; c'$. A natural approach is to use the chain rule. If the dependency-tracking part of our analysis is missing, in order to establish that the output on a variable v by the sequence is differentiable in an input variable u, the analysis should show the output on v by the second command c' is differentiable in *all* variables, and the output on any variable by the first command c is differentiable in u. This requirement on c and c' is too strong. Often, c' uses only a small number of variables to compute v, and it is sufficient to require that just on those used variables, c should be differentiable in u. Similarly, c commonly updates only a small number of variables using u, and it is enough to require that just in those u-dependent variables, the second command c' is differentiable when computing the output v. The dependency-tracking part lets our analysis carry out such reasoning and achieve better precision. Formally, this means our analysis uses a version of reduced product [Cousot and Cousot 1979] between dependency analysis and the analysis that tracks the target smoothness property.

We now formally describe each of these components as well as their combination.

5.1.1 Family of Smoothness Predicates. Our program analysis assumes that a target smoothness property is specified in terms of a family of predicates,

$$\phi = (\phi_{K,L} : K, L \subseteq \mathsf{Var}),$$

where $\phi_{K,L}$ is a set of partial functions from $\mathsf{St}[K]$ to $\mathsf{St}[L]$ (i.e., $\phi_{K,L} \subseteq [\mathsf{St}[K] \rightharpoonup \mathsf{St}[L]]$).

Example 5.1 (Differentiability). In the instantiation of our program analysis for differentiability, we use the family $\phi^{(d)}$ where for all $K, L \subseteq \mathsf{Var}$, a partial function $f : \mathsf{St}[K] \rightharpoonup \mathsf{St}[L]$ belongs to $\phi^{(d)}_{K,L}$ if and only if (i) $\mathrm{dom}(f)$ is open and (ii) f is (jointly) differentiable in its domain. □

At first, one may wonder why we use a family of $\phi_{K,L}$ predicates instead of a single predicate ϕ_0 over $[\mathsf{St} \to \mathsf{St}_\perp]$. The reason is that, as mentioned above, the analysis aims at a conditional variant of the traditional notion of smoothness. For instance, instead of checking that a function $f : \mathsf{St} \to \mathsf{St}_\perp$ is differentiable on $\mathsf{St} \backslash f^{-1}(\{\perp\})$, the analysis proves differentiability conditioned on certain variables being fixed: if we fix the input variables in $\mathsf{Var} \backslash K$ and vary just those in K in the initial state, and look at the output variables in L only, then the function f becomes differentiable, although it might not be so when all input/output variables are considered. To express this, we need the whole family of $\phi_{K,L}$ predicates.

This notion of conditional differentiability is similar to, but not the same as, so called partial differentiability. Partial differentiability in K says that, for every $v \in K$, if we fix all the input variables except v, including those in $K \setminus \{v\}$, and consider the output variables in L only, f becomes

differentiable.[12] As we will show in Remark 5.12, the set of partially-differentiable functions is not closed under a certain operator, but we need the closure to ensure that our program analysis is sound. Our conditional differentiability does not suffer from this issue.

5.1.2 Smoothness Abstraction. Based on the family ϕ, we build a predicate Φ that captures the smoothness of commands. The Φ constrains functions from St to $\mathrm{St}_\perp$, unlike $\phi_{K,L}$. For $K, L \subseteq \mathrm{Var}$ with $K \supseteq L$, define $\pi_{K,L}$ to be the projection from $\mathrm{St}[K]$ to $\mathrm{St}[L]$.

Definition 5.2. The *smoothness abstraction* Φ is the predicate over a function $f \in [\mathrm{St} \to \mathrm{St}_\perp]$ and variable sets $K, L \subseteq \mathrm{Var}$. It is satisfied by (f, K, L) if for all $\tau \in \mathrm{St}[\mathrm{Var} \setminus K]$, the predicate $\phi_{K,L}$ holds for the following partial function $g : \mathrm{St}[K] \rightharpoonup \mathrm{St}[L]$: $\mathrm{dom}(g) \triangleq \{\sigma \in \mathrm{St}[K] \mid f(\sigma \oplus \tau) \in \mathrm{St}\}$ and $g(\sigma) \triangleq (\pi_{\mathrm{Var},L} \circ f)(\sigma \oplus \tau)$ for $\sigma \in \mathrm{dom}(g)$. We denote the satisfaction of Φ by

$$\models \Phi(f, K, L). \qquad \square$$

Note that the function g is constructed from f by fixing the $\mathrm{Var} \setminus K$ part of the input state to τ, and looking at only the L part of the output. This construction is precisely the one used in the informal definition of conditional differentiability described above, and its use reflects the fact that our program analysis attempts to prove a conditional smoothness property.

5.1.3 Dependency Abstraction. Our abstract domain has a component for tracking dependency between input-output variables. Dependency here means that a given input variable is used for computing a given output variable. We define a predicate Δ that has a similar format as Φ. Intuitively, $\Delta(f, K, L)$ holds if and only if the L part of the output of f depends at most on the K part of the input to f. To define Δ formally, for $K \subseteq \mathrm{Var}$, let $\sim_K$ be the following equivalence relation over states: $\sigma \sim_K \sigma'$ if and only if $\sigma(v) = \sigma'(v)$ for all $v \in K$.

Definition 5.3. The *dependency abstraction* Δ is the predicate on $f \in [\mathrm{St} \to \mathrm{St}_\perp]$ and $K, L \subseteq \mathrm{Var}$ that holds if for all $\sigma, \sigma' \in \mathrm{St}$ with $\sigma \sim_K \sigma'$, we have $(f(\sigma) \in \mathrm{St} \iff f(\sigma') \in \mathrm{St})$ and $(f(\sigma) \in \mathrm{St} \implies f(\sigma) \sim_L f(\sigma'))$. We denote the satisfaction of Δ by

$$\models \Delta(f, K, L). \qquad \square$$

5.1.4 Combined Abstraction. We bring together the two abstractions that we just defined, and construct the final abstract domain $\mathcal{D}^\sharp$ used by our program analysis.

Intuitively, each element of $\mathcal{D}^\sharp$ is a predicate on a function $f \in [\mathrm{St} \to \mathrm{St}_\perp]$ expressed by the conjunction of the following form: $\bigwedge_{i=1}^m \Phi(f, K_i, L_i) \wedge \bigwedge_{j=1}^n \Delta(f, K'_j, L'_j)$. A direct but naive way of implementing this intuition is to let $\mathcal{D}^\sharp$ be the collection of all the constraints of this form, but it permits too many constraints and leads to a costly program analysis. We take a more economical alternative that further restricts the allowed form of the constraints. The alternative requires that the conjunction from above should be constructed out of two mappings p and d from output variables to input variable sets, and a set V of input variables. The p component describes smoothness, and the d and V components dependency. They together encode the constraint

$$\bigwedge_{v \in \mathrm{Var}} \Phi(f, p(v), \{v\}) \wedge \bigwedge_{u \in \mathrm{Var}} \Delta(f, d(u), \{u\}) \wedge \Delta(f, V, \emptyset).$$

Thus, a function $f \in [\mathrm{St} \to \mathrm{St}_\perp]$ satisfies the constraint encoded by p, d, and V if (i) for every output variable v, when we fix the values of all the input variables outside of $p(v)$, the (partial) function $\sigma \longmapsto f(\sigma)(v)$ is smooth (e.g., differentiable); (ii) for every output variable u, the (partial) function $\sigma \longmapsto f(\sigma)(u)$ does not access any variable outside of $d(u)$ to compute the value of u; and (iii) the values of input variables in V determine whether f returns $\perp$ or not.

[12] Conditional differentiability extends partial differentiability in the sense that the latter can be expressed as a conjunction of the former (but not vice versa): f is partially differentiable in K if and only if f is conditionally differentiable in $\{v\}$ for all $v \in K$.

Definition 5.4. The *abstract domain* $\mathcal{D}^\sharp$ consists of triples $(p, d, V) \in [\mathsf{Var} \to \mathcal{P}(\mathsf{Var})]^2 \times \mathcal{P}(\mathsf{Var})$, called *abstract state*, such that $p(v) \supseteq d(v)^c$ and $d(v) \supseteq V$ for all $v \in \mathsf{Var}$, where $-^c$ is the standard operation for set complement. That is,

$$\mathcal{D}^\sharp \triangleq \{(p, d, V) \in [\mathsf{Var} \to \mathcal{P}(\mathsf{Var})]^2 \times \mathcal{P}(\mathsf{Var}) \mid p(v) \supseteq d(v)^c \text{ and } d(v) \supseteq V \text{ for all } v \in \mathsf{Var}\}.$$

We order abstract states as follows: $(p, d, V) \sqsubseteq (p', d', V')$ if and only if $V \subseteq V'$ and for all $v \in \mathsf{Var}$, $p(v) \supseteq p'(v)$ and $d(v) \subseteq d'(v)$. These abstract states are concretised by $\gamma : \mathcal{D}^\sharp \to \mathcal{P}([\mathsf{St} \to \mathsf{St}_\perp])$:

$$f \in \gamma(p, d, V) \iff \models \Delta(f, V, \emptyset),\ \models \Phi(f, p(v), \{v\}),\ \text{and } \models \Delta(f, d(v), \{v\}) \text{ for all } v \in \mathsf{Var}. \quad \square (10)$$

Note that the definition of $\mathcal{D}^\sharp$ contains two conditions. The first condition $p(v) \supseteq d(v)^c$ comes from our assumption that if a function does not depend on a variable u, it is smooth in u. This and other assumptions of the analysis will be explained shortly in §5.2.2. The other condition $d(v) \supseteq V$ originates from the relationship that if $\Delta(f, K, \{v\})$ holds, so does $\Delta(f, K, \emptyset)$.

Example 5.5 (Differentiability). Consider the setup of Example 5.1 and the program $c \equiv (y := x * x;\ \mathtt{if}\ (x \geq 0)\ \{s := 1\}\ \mathtt{else}\ \{s := -1\})$. Let (p, d, V) be the smallest abstract state that describes the program. In this program, s is not differentiable in x, but y is. So, $p(s) = \mathsf{Var} \setminus \{x\} \supseteq \{y, s\}$ and $p(y) = \mathsf{Var} \supseteq \{x, y, s\}$. Note that $p(s)$ contains the input variables s and y because by not depending on those input variables, the output s is differentiable in those variables. For the dependency part, we have $d(s) = d(y) = \{x\}$ and $V = \emptyset$. $\square$

5.2 Parametric Static Program Analysis

Our program analysis is based on abstract interpretation [Cousot and Cousot 1977], and computes an approximation of the concrete semantics $[\![c]\!]$ of a given command c using the abstract domain $\mathcal{D}^\sharp$. We formalise this computation by the *abstract semantics* of c, which defines $[\![c]\!]^\sharp \in \mathcal{D}^\sharp$ by induction on the structure of commands, and over-approximates $[\![c]\!]$ in the sense of the concretization γ in Eq. (10).

5.2.1 Analysis Definition. Fig. 3 shows the abstract semantics of $[\![c]\!]^\sharp$. The overall structure of the semantics follows the standard compositional semantics of an imperative language. For instance, the abstract semantics of sequential composition is defined in terms of those of constituent commands, and the semantics of a loop is the least fixed point of a monotone operator over $\mathcal{D}^\sharp$. However, the specifics of the semantics include non-standard details, and we spell them out by going through the defining clauses of $[\![c]\!]^\sharp$.

The definition of $[\![\mathtt{skip}]\!]^\sharp$ formalises the effect of `skip` on smoothness and dependency. The definition says that `skip` computes each output variable v in a smooth manner in all input variables, and in so doing, it creates the dependency between the variable v to itself at the input state. The V part of $[\![\mathtt{skip}]\!]^\sharp$ is the empty set since `skip` always terminates.

The next case is $x := e$. Its abstract semantics records the smoothness and dependency information of the updated variable x by analysing the expression e. For the smoothness part, the semantics invokes the subroutine $(\!|e|\!)^\sharp$ that computes an under-approximation of the set of variables in which the expression e is smooth (and thus over-approximates the smoothness property of e).[13] For the dependency part, $[\![x := e]\!]^\sharp$ computes the set of all the free variables of e so as to get an over-approximation of all variables that may affect the value of e. For variables other than x, $[\![x := e]\!]^\sharp$ behaves like $[\![\mathtt{skip}]\!]^\sharp$.

The abstract semantics of a sequence $c; c'$ composes those of the sub-commands c and c'. It uses the liftings $f_\cup, f_\cap : \mathcal{P}(\mathsf{Var}) \to \mathcal{P}(\mathsf{Var})$ of functions f of type $\mathsf{Var} \to \mathcal{P}(\mathsf{Var})$, which are defined as follows: $f_\cup(V) \triangleq \bigcup_{v \in V} f(v)$ and $f_\cap(V) \triangleq \bigcap_{v \in V} f(v)$. The abstract semantics $[\![c; c']\!]^\sharp$ constructs

[13] The subroutine $(\!|e|\!)^\sharp \subseteq \mathsf{Var}$ is defined inductively on e, and differs for different target smoothness properties. For instance, if the target property is differentiability, we have $(\!|r|\!)^\sharp \triangleq (\!|x|\!)^\sharp \triangleq \mathsf{Var}$, $(\!|e_1 + e_2|\!)^\sharp \triangleq (\!|e_1|\!)^\sharp \cap (\!|e_2|\!)^\sharp$, $(\!|\mathrm{ReLU}(e)|\!)^\sharp \triangleq \mathsf{Var} \cap fv(e)^c$, etc. On the other hand, if the target property is local Lipschitzness, the subroutine changes for some cases: e.g., $(\!|\mathrm{ReLU}(e)|\!)^\sharp \triangleq (\!|e|\!)^\sharp$.

$$
\begin{aligned}
[\![\texttt{skip}]\!]^\sharp &\triangleq (\lambda v.\mathsf{Var},\ \lambda v.\{v\},\ \emptyset),\\
[\![x := e]\!]^\sharp &\triangleq ((\lambda v.\, v \equiv x\ ?\ (\!|e|\!)^\sharp : \mathsf{Var}),\ (\lambda v.\, v \equiv x\ ?\ fv(e) : \{v\}),\ \emptyset),\\
[\![c; c']\!]^\sharp &\triangleq \text{let } (p, d, V) \triangleq [\![c]\!]^\sharp \text{ and } (p', d', V') \triangleq [\![c']\!]^\sharp \text{ in } (p'', d'', V'')\\
&\quad \text{where } p''(v) \triangleq (V \cup p_\cap(d'(v))^c \cup d_\cup(p'(v)^c))^c,\\
&\quad\quad d''(v) \triangleq V \cup d_\cup(d'(v)), \text{ and } V'' \triangleq V \cup d_\cup(V'),\\
[\![\texttt{if}\ b\ \{c\}\ \texttt{else}\ \{c'\}]\!]^\sharp &\triangleq \text{let } (p, d, V) \triangleq [\![c]\!]^\sharp \text{ and } (p', d', V') \triangleq [\![c']\!]^\sharp \text{ in } (p'', d'', V'')\\
&\quad \text{where } p''(v) \triangleq fv(b)^c \cap p(v) \cap p'(v),\\
&\quad\quad d''(v) \triangleq fv(b) \cup d(v) \cup d'(v), \text{ and } V'' \triangleq fv(b) \cup V \cup V',\\
[\![\texttt{while}\ b\ \{c\}]\!]^\sharp &\triangleq \text{let } (p, d, V) \triangleq [\![c]\!]^\sharp \text{ in } \mathit{fix}\ F^\sharp\\
&\quad \text{where } F^\sharp(p_0, d_0, V_0) \triangleq (p', d', V'),\\
&\quad\quad p'(v) \triangleq fv(b)^c \cap (V \cup p_\cap(d_0(v))^c \cup d_\cup(p_0(v)^c))^c,\\
&\quad\quad d'(v) \triangleq fv(b) \cup (V \cup d_\cup(d_0(v))) \cup \{v\}, \text{ and } V' \triangleq fv(b) \cup (V \cup d_\cup(V_0)),\\
[\![\texttt{obs}(\texttt{dist}_\mathsf{N}(e_1, e_2), r)]\!]^\sharp &\triangleq (p, d, \emptyset)\\
&\quad \text{where } p(v) \triangleq (v \equiv \mathit{like})\ ?\ (\!|\mathit{like} \times \texttt{pdf}_\mathsf{N}(r; e_1, e_2)|\!)^\sharp : \mathsf{Var},\\
&\quad\quad \text{and } d(v) \triangleq (v \equiv \mathit{like})\ ?\ \{\mathit{like}\} \cup fv(e_1) \cup fv(e_2) : \{v\},\\
[\![x := \texttt{sam}(n, \texttt{dist}_\mathsf{N}(e_1, e_2), \lambda y.e')]\!]^\sharp &\triangleq (p, d, \emptyset) \quad \text{for } n = \texttt{name}(\alpha, r) \text{ with } r \in \mathbb{R}\\
&\quad \text{where } \mu \triangleq \mathit{create\_name}(\alpha, r),\\
&\quad\quad p(v) \triangleq \begin{cases} (\!|e'[\mu/y]|\!)^\sharp & \text{if } v \in \{x, \mathit{val}_\mu\}\\ (\!|\texttt{pdf}_\mathsf{N}(\mu; e_1, e_2)|\!)^\sharp & \text{if } v \equiv \mathit{pr}_\mu\\ (\!|\mathit{cnt}_\mu + 1|\!)^\sharp & \text{if } v \equiv \mathit{cnt}_\mu\\ \mathsf{Var} & \text{otherwise,}\end{cases}\\
&\quad\quad \text{and } d(v) \triangleq \begin{cases} fv(e'[\mu/y]) & \text{if } v \in \{x, \mathit{val}_\mu\}\\ \{\mu\} \cup fv(e_1) \cup fv(e_2) & \text{if } v \equiv \mathit{pr}_\mu\\ \{v\} & \text{otherwise,}\end{cases}\\
[\![x := \texttt{sam}(n, \texttt{dist}_\mathsf{N}(e_1, e_2), \lambda y.e')]\!]^\sharp &\triangleq (p, d, \emptyset) \quad \text{for } n = \texttt{name}(\alpha, e) \text{ with } e \notin \mathbb{R}\\
&\quad \text{where } p(v) \triangleq \begin{cases} fv(e)^c \cap \bigcap_{\mu=(\alpha,\_)\in\mathsf{Name}} (\!|e'[\mu/y]|\!)^\sharp & \text{if } v \equiv x\\ fv(e)^c \cap (\!|e'[\mu/y]|\!)^\sharp & \text{if } v \equiv \mathit{val}_\mu \text{ for } \mu = (\alpha, \_)\\ fv(e)^c \cap (\!|\texttt{pdf}_\mathsf{N}(\mu; e_1, e_2)|\!)^\sharp & \text{if } v \equiv \mathit{pr}_\mu \text{ for } \mu = (\alpha, \_)\\ fv(e)^c \cap (\!|\mathit{cnt}_\mu + 1|\!)^\sharp & \text{if } v \equiv \mathit{cnt}_\mu \text{ for } \mu = (\alpha, \_)\\ \mathsf{Var} & \text{otherwise,}\end{cases}\\
&\quad\quad \text{and } d(v) \triangleq \begin{cases} fv(e) \cup \bigcup_{\mu=(\alpha,\_)\in\mathsf{Name}} fv(e'[\mu/y]) & \text{if } v \equiv x\\ fv(e) \cup \{\mathit{val}_\mu\} \cup fv(e'[\mu/y]) & \text{if } v \equiv \mathit{val}_\mu \text{ for } \mu = (\alpha, \_)\\ fv(e) \cup \{\mathit{pr}_\mu, \mu\} \cup fv(e_1) \cup fv(e_2) & \text{if } v \equiv \mathit{pr}_\mu \text{ for } \mu = (\alpha, \_)\\ fv(e) \cup \{\mathit{cnt}_\mu\} & \text{if } v \equiv \mathit{cnt}_\mu \text{ for } \mu = (\alpha, \_)\\ \{v\} & \text{otherwise.}\end{cases}
\end{aligned}
$$

Fig. 3. Abstract semantics of commands defining $[\![c]\!]^\sharp \in \mathcal{D}^\sharp$.

the dependency part d'' by composing d from $[\![c]\!]^\sharp$ and d' from $[\![c']\!]^\sharp$ after lifting the former. Note the inclusion of the set V in the definition of d''. This is to account for the case that $d'(v)$ in the definition is the empty set; in that case, $d_\cup(d'(v))$ is empty as well and does not have any information about termination. The smoothness part p'' of $[\![c; c']\!]^\sharp$ is more involved, and implements the intuition described briefly in §5.1. In order to conclude that input variables in V_0 together smoothly affect an output variable v in the computation of $c; c'$, the p'' considers the intermediate state after the first command c, and forms two groups of variables at that intermediate state: $d'(v)$ and $p'(v)^c$. Note that

the desired smoothness property for the input variables in V_0 and the output variable v may fail if the first command c uses some variable $u_0 \in V_0$ non-smoothly to update a variable u_0' in $d'(v)$, or it uses some variable $u_1 \in V_0$ to compute the value of a variable $u_1' \in p'(v)^c$. In the former case, the non-smoothness of c causes an issue, and in the latter case, the non-smoothness of c' causes an issue. The p'' collects the input variables that avoid these two failure modes and also do not influence the termination of the sequence. As we show in our soundness theorem, doing so is sufficient because it amounts to using a version of chain rule for the target smoothness property.

The abstract semantics of an if command conservatively assumes that any variable in its condition b may affect the value of any output variable (by influencing whether the true or false branch of the command gets executed) and this influence is potentially non-smooth. For every output variable v, the smooth set $p''(v)$ for the if command implements this assumption by excluding free variables in b, and the computed dependency set does the same but this time by including free variables in b.

The abstract semantics of a loop computes the least fixed point of a monotone operator $F^\sharp : \mathcal{D}^\sharp \to \mathcal{D}^\sharp$ using the standard Kleene iteration. The operator $F^\sharp$ describes the effect of one iteration of the loop, and it is derived from the standard loop unrolling and our abstract semantics of sequencing and the if command.

The abstract semantics of an observe command $\texttt{obs}(\texttt{dist}_\texttt{N}(e_1, e_2), r)$ uses the fact that the command has the same concrete semantics as the assignment $\mathit{like} := \mathit{like} \times \texttt{pdf}_\texttt{N}(r; e_1, e_2)$, where $\texttt{pdf}_\texttt{N}$ is the density function of the normal distribution. The semantics computes (p, d, V) according to that of the assignment, which we explained earlier.

The final case is the abstract semantics of a sample command. The semantics performs a case analysis on the first argument of sam. If it is a constant expression not involving any variables, then the abstract semantics constructs the name μ of the sampled random variable, and updates p, d, and V according to the concrete semantics of the command. Otherwise, the abstract semantics acknowledges that the precise name μ of the random variable cannot be known statically, and performs so called *weak update* by joining two pieces of information before and after the update of the command in the concrete semantics. Note that the abstract semantics does not require the third argument of sam should be the identity function. The ability of dealing with a general function in the third argument is needed since our analysis is intended to be applied to programs after the transformation of the SPGE, which may introduce such an argument.

The abstract semantics is well-defined under the following relatively weak assumption:

Assumption 1 (Expression analysis and free variables). $(\!|e|\!)^\sharp \supseteq fv(e)^c$ *for all expressions* e.

This assumption is satisfied by the instantiations of the semantics with differentiability and local Lipschitzness, which are used in our implementation. It will be assumed in the rest of the paper.

Theorem 5.6. *If Assumption 1 holds, then for all commands* c*, we have* $[\![c]\!]^\sharp \in \mathcal{D}^\sharp$*, that is, when we let* $(p, d, V) \triangleq [\![c]\!]^\sharp$*, we have* $p(v) \supseteq d(v)^c$ *and* $d(v) \supseteq V$ *for all variables* $v \in \mathsf{Var}$.

Example 5.7 (Differentiability). Consider the differentiability property and the example program of Example 5.5. Let (p_1, d_1, V_1) and (p_2, d_2, V_2) be the results of analysing the first assignment command $y := x * x$ and the following if command of the program. Then,

$$(p_1, d_1, V_1) = (\lambda v.\mathsf{Var},\ \lambda v.\ (v{\equiv}y)\ ?\ \{x\} : \{v\},\ \emptyset),\quad (p_2, d_2, V_2) = (\lambda v.\{x\}^c,\ \lambda v.\ (v{\equiv}s)\ ?\ \{x\} : \{x, v\},\ \{x\}).$$

Let (p, d, V) be the analysis result for the entire program. Then,

$$p(v) = \left(V_1 \cup (p_1)_\cap(d_2(v))^c \cup (d_1)_\cup(p_2(v)^c)\right)^c = d_1(x)^c = \{x\}^c, \qquad V = V_1 \cup (d_1)_\cup(V_2) = \{x\}.$$

Also, $d(v) = V_1 \cup (d_1)_\cup(d_2(v)) = \{x\}$. As shown in Fig. 3, the variable x that may affect the condition expression of the if command is removed from the smoothness sets, and $p(s) = p(y) = \{x\}^c$. Note that this result is conservative with respect to y. □

5.2.2 Analysis Soundness and Assumptions. The soundness of our analysis states that for every command c, its abstract semantics $[\![c]\!]^\sharp \in \mathcal{D}^\sharp$ over-approximates the concrete semantics $[\![c]\!]$ via γ: $[\![c]\!] \in \gamma([\![c]\!]^\sharp)$. The soundness is conditioned on Assumption 1 and six new assumptions. One of the new assumptions is about the soundness of $(\!|e|\!)^\sharp$. The remaining five assumptions are concerned with the predicate family for the target smoothness property $\phi = (\phi_{K,L} : K, L \subseteq \mathrm{Var})$, and say that certain canonical operators are smooth according to ϕ so that using them in the abstract semantics should not cause an issue. In this subsection, we present the six assumptions one by one, and sketch how those assumptions are related to the soundness.

We start with the assumption that the analysis of each expression $(\!|e|\!)^\sharp$ under-approximates the set of variables in which the evaluation of e is smooth (and thus over-approximates the smoothness property of e).

Assumption 2 (Expression analysis soundness). *For all expressions e, variables x, subsets $K, L \subseteq \mathrm{Var}$, and states $\tau \in \mathrm{St}[\mathrm{Var} \setminus K]$ such that $K = (\!|e|\!)^\sharp$ and $L = \{x\}$, if we let $g : \mathrm{St}[K] \to \mathrm{St}[L]$ be the function defined by $g(\sigma) \triangleq [x \mapsto [\![e]\!](\sigma \oplus \tau)]$, the function g satisfies $\phi_{K,L}$ (i.e., $g \in \phi_{K,L}$).*

This assumption is used in our soundness argument whenever the abstract semantics uses $(\!|e|\!)^\sharp$ for computing smoothness information about an expression e.

The next two assumptions assert the smoothness of the standard operators on the product spaces.

Assumption 3 (Projection). *For all $K, L \subseteq \mathrm{Var}$ with $K \supseteq L$, the projection $\pi_{K,L}$ satisfies $\phi_{K,L}$.*

Assumption 4 (Pairing). *For all $K, L, M \subseteq \mathrm{Var}$ with $L \cap M = \emptyset$, if $f \in \phi_{K,L}$ and $g \in \phi_{K,M}$, we have $\langle f, g\rangle \in \phi_{K,L\cup M}$, where $\langle f, g\rangle$ is the pairing of two partial functions: $\langle f, g\rangle(\sigma) \triangleq$ if $(\sigma \in \mathrm{dom}(f) \cap \mathrm{dom}(g))$ then $f(\sigma) \oplus g(\sigma)$ else undefined.*

Note that $\mathrm{St}[L \cup M]$ is isomorphic to $\mathrm{St}[L] \times \mathrm{St}[M]$, the product space that we referred to above. The assumptions say that the projection is smooth, and the pairing of smooth functions is smooth. Our analysis uses Assumption 3 to deal with variables not modified by a command. For instance, when analysing an assignment $x := e$, the analysis uses Assumption 3 and concludes that on every output variable v other than x, the assignment is smooth in all the input variables. Assumption 4 is used to justify the handling of a sequence $c; c'$ by our analysis, in particular, the part that the analysis combines smoothness information over multiple output variables after the first command c.

The projection and pairing assumptions are about how shrinking and expanding output variables affect the target smoothness property. The next restriction assumption is about shrinking the input variables. It validates the weakening of the K part of $\models \Phi(f, K, L)$, and is used in the abstract semantics of $c; c'$ (and other composite commands).

Assumption 5 (Restriction). *For all $K, K', L \subseteq \mathrm{Var}$ with $K \supseteq K'$, and $\tau \in \mathrm{St}[K \setminus K']$, if $f \in \phi_{K,L}$, then we have $g \in \phi_{K',L}$, where $g(\sigma) \triangleq$ if $(\sigma \oplus \tau \in \mathrm{dom}(f))$ then $f(\sigma \oplus \tau)$ else undefined.*

The following assumption says that the function composition preserves smoothness. It is related to the chain rule for differentiation, and used to justify the abstract semantics of a sequence $c; c'$.

Assumption 6 (Composition). *For all $K, L, M \subseteq \mathrm{Var}$, if $f \in \phi_{K,L}$ and $g \in \phi_{L,M}$, we have $g \circ f \in \phi_{K,M}$, where $g \circ f$ is the standard composition of two partial functions: $(g \circ f)(\sigma) \triangleq$ if $(\sigma \in \mathrm{dom}(f) \wedge f(\sigma) \in \mathrm{dom}(g))$ then $g(f(\sigma))$ else undefined.*

The final assumption lets the analysis infer smoothness information about the completely-undefined function. It is used to justify the handling of loops by our analysis.

Assumption 7 (Strictness). *For all $K, L \subseteq \mathrm{Var}$, we have $(\lambda\sigma \in \mathrm{St}[K].\mathit{undefined}) \in \phi_{K,L}$.*

Theorem 5.8 (Soundness). *If Assumptions 1–7 hold, the analysis computes the sound abstraction of the concrete semantics of commands in the following sense: for all commands c,*

$$[\![c]\!] \in \gamma([\![c]\!]^\sharp).$$

Remark 5.9. A standard method for proving a property of a loop or more generally a recursively defined function is so called Scott induction. In this method, we view a property as a set $\mathcal{T}$ of state transformers and a loop as the least fixed point of a continuous function F on state transformers. Then, we prove the three conditions: (i) $\mathcal{T}$ contains the least state transformer, (ii) it is closed under the least upper bound of any increasing sequence of state transformers, and (iii) $\mathcal{T}$ is preserved by F. The first and second conditions are called strictness and admissibility, respectively, and these three conditions imply that the least fixed point of F belongs to $\mathcal{T}$.

Our soundness proof for the loop case deviates slightly from this standard method. If it followed the method instead, we would need, in addition to the strictness assumption, the following assumption, which corresponds to the second admissibility condition:

Assumption 8 (Admissibility). *Let $K, L \subseteq \mathrm{Var}$, and order partial functions in $[\mathrm{St}[K] \rightharpoonup \mathrm{St}[L]]$ by the inclusion of the graphs of partial functions. Then, for every increasing sequence $\{f_n : \mathrm{St}[K] \rightharpoonup \mathrm{St}[L]\}_{n\in\mathbb{N}}$ (i.e., the graph of f_{n+1} includes that of f_n for all $n \in \mathbb{N}$), if every f_n satisfies $\phi_{K,L}$, so does the least upper bound f_∞ of the sequence (defined by its graph being the union of the graphs of all f_n's).*

The inclusion of this admissibility assumption would, then, limit the applicability of our program analysis, since some well-known smoothness properties, such as (global) Lipschitz continuity and local boundedness, do not satisfy the assumption, although they satisfy our five assumptions (Assumptions 3–7) (see Table 2). On the plus side, the inclusion of the admissibility assumption could enable our analysis to handle loops more accurately, possibly by tracking the impact of the boolean condition of each loop on smoothness more precisely. Our soundness proof avoids the admissibility assumption by exploiting the fact that our analysis handles loop conditions conservatively: our analysis drops all the variables that loop conditions may depend on from the set of smooth variables, so that it avoids finding too precise inductive predicates that can break soundness.[14] □

5.3 Instantiations

Our program analysis requires that the family of smoothness predicates should satisfy Assumptions 3–7. Although these assumptions are violated by some smoothness properties, such as partial differentiability and partial continuity, they are met by our leading example $\phi^{(d)}$ for differentiability (Example 5.1), and also by the predicate family $\phi^{(l)}$ for local Lipschitzness, which is used in our implementation. Recall the definitions of the predicate families $\phi^{(d)}$ and $\phi^{(l)}$: for all $K, L \subseteq \mathrm{Var}$,

$$\begin{aligned}
\phi^{(d)}_{K,L} &\triangleq \{f : \mathrm{St}[K] \rightharpoonup \mathrm{St}[L] \mid \mathrm{dom}(f) \text{ is open and } f \text{ is (jointly) differentiable in its domain}\},\\
\phi^{(l)}_{K,L} &\triangleq \{f : \mathrm{St}[K] \rightharpoonup \mathrm{St}[L] \mid \mathrm{dom}(f) \text{ is open, and for all } \sigma \in \mathrm{dom}(f), \text{ there are } C > 0 \text{ and}\\
&\qquad \text{an open } O \subseteq \mathrm{dom}(f) \text{ s.t. } \sigma \in O \text{ and } \|f(\sigma_0) - f(\sigma_1)\|_2 \le C\|\sigma_0 - \sigma_1\|_2 \text{ for all } \sigma_0, \sigma_1 \in O\}.
\end{aligned}$$

Theorem 5.10. *Both $\phi^{(d)}$ and $\phi^{(l)}$ satisfy Assumptions 3–7.*

Remark 5.11. The requirement of open domain in $\phi^{(d)}$ is sometimes too constraining and hurts the accuracy of the analysis. It can, however, be relaxed, and we can generalise $\phi^{(d)}$ to the following

[14] To be precise, our analysis does not require the admissibility assumption, not because our abstract domain is finite, but because given a loop, our analysis finds an inductive predicate that is "sufficiently admissible" in the sense that it is closed under the least upper bound of any chain *that matters for soundness*: the chain should be definable by some loop. More details are in §F.4.

Table 1. Failure cases of the composition assumption. For each given ϕ, we have $f, g \in \phi$ but $g \circ f \notin \phi$, where f and g are interpreted as (total or partial) functions from $\mathrm{St}[K]$ to $\mathrm{St}[L]$. Let $c_1 \equiv (y = x^2; z = g(y))$ and $c_2 \equiv (y = x; z = g(x, y))$. Then, for each i-th ϕ, $[\![c_i]\!]^\sharp$ incorrectly concludes that z is smooth with respect to x.

ϕ	f	g
$\phi_{K,L}^{(d'')}$	$f(x) = x^2$ defined on $\mathbb{R}$	$g(x) = \mathbf{1}_{[x>0]}$ defined on $[0, 1]$
$\phi_{K,L}^{(pd)}, \phi_{K,L}^{(pc)}$	$f(x) = (x, x)$ defined on $\mathbb{R}$	$g(x, y) = \begin{cases} xy/(x^2+y^2) & \text{if } (x,y) \neq (0,0) \\ 0 & \text{otherwise} \end{cases}$ defined on $\mathbb{R}^2$

predicate family $\phi^{(d')}$, which corresponds to the standard definition of differentiability on a manifold *with boundary* in differential geometry [Lee 2012, Chapter 2]:

$$\phi_{K,L}^{(d')} \triangleq \{f : \mathrm{St}[K] \rightharpoonup \mathrm{St}[L] \mid \text{for all } \sigma \in \mathrm{dom}(f), \text{there exist an open } U \subseteq \mathrm{St}[K] \text{ and } g : U \to \mathrm{St}[L] \text{ such that } \sigma \in U, f = g \text{ on } U \cap \mathrm{dom}(f), \text{and } g \text{ is (jointly) differentiable}\}.$$

Note the weakening of open-domain requirement in $\phi^{(d')}$: the open domain U in the above definition does not have to be included in $\mathrm{dom}(f)$. The family $\phi^{(d')}$ satisfies Assumptions 3–7, and can lead to a more permissive instantiation of our program analysis than the family $\phi^{(d)}$, especially in handling atomic commands, such as assignment, sample, and observe. We point out that $\phi^{(d')}$ does not satisfy Assumption 8 (i.e., the admissibility assumption), while $\phi^{(d)}$ does satisfy it. As a result, if the handling of loops in our analysis is changed such that loop conditions are analysed more accurately, the analysis may remain sound only for $\phi^{(d)}$, not for $\phi^{(d')}$, as explained in Remark 5.9. □

Remark 5.12. At this point, the reader might feel that Assumptions 3–7 are satisfied by nearly all smoothness properties. This impression is not accurate. For instance, the composition assumption does not hold for the notions of differentiability of partial functions formalised by the following $\phi^{(d'')}$ and $\phi^{(pd)}$, nor for the partial continuity formalised by $\phi^{(pc)}$:

$$\phi_{K,L}^{(d'')} \triangleq \{f : \mathrm{St}[K] \rightharpoonup \mathrm{St}[L] \mid f \text{ is (jointly) differentiable in the interior of its domain}\},$$
$$\phi_{K,L}^{(pd)} \triangleq \{f : \mathrm{St}[K] \rightharpoonup \mathrm{St}[L] \mid \mathrm{dom}(f) \text{ is open, and for all } v \in K, f \text{ is partially differentiable in } v\},$$
$$\phi_{K,L}^{(pc)} \triangleq \{f : \mathrm{St}[K] \rightharpoonup \mathrm{St}[L] \mid \mathrm{dom}(f) \text{ is open, and for all } v \in K, f \text{ is partially continuous in } v\}.$$

Table 1 contains counterexamples that show the failure of the composition assumption for these predicate families. In fact, when instantiated with these families, our program analysis is not sound. The same table shows example programs and incorrect conclusions derived by our analysis.

Table 2 shows more target smoothness properties from mathematics, and whether each property satisfies Assumptions 3–7 (and Assumption 8). Recall that our program analysis does not require Assumption 8 for soundness; the table shows the assumption just for reference. The target properties from "cont." to "real analytic" in the table (and three more) satisfy Assumptions 3–7, so that our analysis can be immediately applied to those target properties while remaining sound. □

6 ALGORITHM FOR THE SPGE VARIABLE-SELECTION PROBLEM

We now put together the results from §4 and §5 to formally define and soundly (yet approximately) solve the SPGE variable-selection problem. We start with the formal definition of the problem:

Definition 6.1 (SPGE Variable-Selection Problem; Formal). Assume we are given a model c_m, a guide c_g, and a (initial) simple reparameterisation plan π_0 such that c_m, c_g, and $\overline{c_g}^{\pi}$ always terminate and have no double-sampling errors for all $\pi \sqsubseteq \pi_0$. Here we write $\pi \sqsubseteq \pi'$ if the graph of π is included in that of π'. Given these c_m, c_g, and π_0, find a reparameterisation plan $\pi \sqsubseteq \pi_0$ such that (i) π is simple and satisfies (R2) and (R3) in §4.2, and (ii) $|rv(\pi)|$ is maximised. We say that π is a *sound solution* if it satisfies (i), and an *optimal solution* if it satisfies (i) and (ii). □

Table 2. Various well-known target smoothness properties from mathematics, and whether each of them satisfies Assumptions 3–7 (and Assumption 8). Here "cont." and "diff." denote "continuous" and "differentiable". The properties above the double horizontal line are defined such that they include the open-domain requirement as in $\phi^{(d)}$ and $\phi^{(l)}$, and the properties below the line are defined without the open-domain requirement.

Target smoothness property	A3 (proj.)	A4 (pair.)	A5 (rest.)	A6 (comp.)	A7 (stri.)	A8 (admi.)
cont. (C^0)	○	○	○	○	○	○
locally Lipschitz (= $\phi^{(l)}$)	○	○	○	○	○	○
uniformly cont.	○	○	○	○	○	×
Lipschitz cont.	○	○	○	○	○	×
diff. (= $\phi^{(d)}$)	○	○	○	○	○	○
continuously diff. (C^1)	○	○	○	○	○	○
smooth (C^∞)	○	○	○	○	○	○
real analytic (C^ω)	○	○	○	○	○	○
partially cont. (= $\phi^{(pc)}$)	○	○	○	×	○	○
partially diff. (= $\phi^{(pd)}$)	○	○	○	×	○	○
almost-everywhere cont.	○	○	×	×	○	○
almost-everywhere diff.	○	○	×	×	○	○
coordinatewise non-decreasing	○	○	○	○	○	○
locally bounded	○	○	○	○	○	×
bounded	×	○	○	○	○	×
Borel measurable	○	○	○	○	○	○
locally integrable	○	○	×	×	○	×
integrable	×	○	×	×	○	×

The input π_0 in the problem is a newcomer. It fixes a semantics-preserving transformation for all the sample commands. Typically, π_0 is defined on the entire $\mathsf{NameEx} \times \mathsf{DistEx} \times \mathsf{LamEx}$, and remains fixed across all input model-guide pairs (c_m, c_g). More importantly, it is valid so that the change of any sample command by π_0 preserves the semantics of the command when we take into account both the second distribution argument and the third lambda argument of the sample command. The validity of π_0 is inherited by any sound solution π of the SPGE variable-selection problem since validity as a property on reparameterisation plans is down-closed with respect to the $\sqsubseteq$ order. In our setup, π_0 is fixed to be the following reparameterisation plan from §4.1:

$$\pi_0(n, \mathtt{dist}_{\mathsf{N}}(e_1, e_2), \lambda y.e_3) \triangleq (\mathtt{dist}_{\mathsf{N}}(0, 1), \lambda y.e_3[(y \times \sqrt{e_2} + e_1)/y]) \tag{11}$$

for all $n \in \mathsf{NameEx}$ and expressions e_1, e_2, and e_3.

As an example of the SPGE variable-selection problem, consider the problem for the π_0 in Eq. (11) and the model-guide pair (c_m, c_g) given in Fig. 1, where $\texttt{"z}_\texttt{i}\texttt{"}$ in the figure is interpreted as $\mathtt{name}(\texttt{"z}_\texttt{i}\texttt{"}, 0)$. Then, as discussed in §2, the problem has the following optimal solution: $\pi \triangleq \pi_0|_{S \times \mathsf{DistEx} \times \mathsf{LamEx}}$ for $S \triangleq \{\mathtt{name}(\alpha, e) \in \mathsf{NameEx} \mid \alpha \not\equiv \texttt{"z}_2\texttt{"}\}$.

We present an algorithm for computing a sound (yet possibly suboptimal) solution to the problem.

(1) By running our program analysis instantiated with differentiability (described in §5.2 and §5.3), compute $(\mathbb{p}_m, \mathbb{d}_m, \mathbb{V}_m) \triangleq [\![c_m]\!]^\sharp$ and $(\mathbb{p}_g, \mathbb{d}_g, \mathbb{V}_g) \triangleq [\![c_g]\!]^\sharp$, where we use $\mathbb{p}$, $\mathbb{d}$, and $\mathbb{V}$ for the output of the analysis to distinguish them from densities p and distributions d.

(2) Using $\mathbb{p}_m$ and $\mathbb{p}_g$, check

$$\theta \subseteq K, \quad \text{where } K \triangleq \mathbb{p}_m(\mathit{like}) \cap \bigcap_{\mu \in \mathsf{Name}} \mathbb{p}_m(\mathit{pr}_\mu) \cap \bigcap_{\mu \in \mathsf{Name}} \mathbb{p}_g(\mathit{pr}_\mu). \tag{12}$$

If the check fails, return an error message that our analysis cannot discharge (R2) for *any* π, since the analysis concludes that the density function of c_m or c_g can be non-differentiable in θ (even when $rv(\pi) = \emptyset$). If the check passes, initialise the set of reparameterised random variables by

$$S \triangleq \{(\alpha, i) \in \mathsf{Name} \mid \text{for all } i' \in \mathbb{N},\ (\alpha, i') \in \mathsf{Name} \implies (\alpha, i') \in K\}.$$

(3) Using S and π_0, construct a reparameterisation plan $\pi \sqsubseteq \pi_0$ by $\pi \triangleq \pi_0[S]$, where $\pi_0[S](n,d,l)$ is $\pi_0(n,d,l)$ if $(n,d,l) \in \mathrm{dom}(\pi_0)$, $n = \mathtt{name}(\alpha, \_)$, and $(\alpha, \_) \in S$; otherwise, it is undefined.
(4) By running the differentiability analysis on $\overline{c_g}^{\pi}$, compute $(\overline{\mathbb{p}_g}, \overline{\mathbb{d}_g}, \overline{\mathbb{V}_g}) \triangleq [\![\overline{c_g}^{\pi}]\!]^{\sharp}$ and check

$$\theta \subseteq \bigcap_{\mu \in \mathsf{Name}} \overline{\mathbb{p}_g}(pr_\mu) \cap \bigcap_{\mu \in \mathsf{Name}} \overline{\mathbb{p}_g}(val_\mu). \tag{13}$$

If the check passes, return π as the output of the algorithm. If not, update S by $S \setminus \{(\alpha, i) \in \mathsf{Name}\}$ after choosing some $(\alpha, \_) \in S$, and then repeat the above procedure (from the step (3)), the point where we construct π using S) until S becomes empty.

Our algorithm computes a sound solution (in the sense stated in Definition 6.1), partly because of the soundness of our program analysis:

Theorem 6.2. *Let c_m, c_g, and π_0 be the inputs to the SPGE variable-selection problem. If the above algorithm returns π for (c_m, c_g, π_0), then π is a sound solution for the problem.*

We point out that the theorem holds for the local Lipschitzness case as well (in addition to the differentiability case). That is, if the above algorithm runs our program analysis instantiated with local Lipschitzness (instead of differentiability), and if it returns an output π, then π is a sound solution to the SPGE variable-selection problem that uses (R2') and (R3') (instead of (R2) and (R3)).

Our algorithm solves the problem only approximately: there is no formal guarantee that it always computes an optimal solution. The suboptimality may arise due to two approximations: the over-approximation of our program analysis when it computes differentiability (or local Lipschitzness) information, and the heuristic choices made by our algorithm when the algorithm computes the random-variable set S. We demonstrate, however, that our algorithm finds optimal solutions for all the benchmarks in §7.

Our algorithm calls our program analysis at most $|\{\alpha \in \mathsf{Str} \mid (\alpha, \_) \in S_0\}| + 2$ times, where S_0 is the initial value of S (i.e., the set of random variables whose sample commands are to be transformed) in the algorithm. However, for all the benchmarks in §7, our algorithm terminated with the initial set S_0 and thus called our analysis only 3 times (on the model, the guide, and the reparameterised guide according to S_0). Based on these results and our intuition on the algorithm, we conjecture that our algorithm always terminates with the initial set S_0 under a mild condition on π_0 and our analysis (e.g., $(\!|e|\!)^{\sharp}$ is computed inductively on e). Since the conjecture is still open, our algorithm might not succeed in the first iteration, and if so, it continues to search for a sound solution greedily. Note that there are many other ways to continue the search and our algorithm uses just one of them (as it is linear-time).

7 EXPERIMENTAL EVALUATION

In our experiments, we consider two research questions. First, can the analysis proposed in §5 be instantiated and implemented so that it can produce meaningful smoothness results on real-world probabilistic programs? Second, can the algorithm proposed in §6 find near-optimal solutions to the SPGE variable-selection problem on real-world probabilistic programs? To assess the two questions, we have implemented a static smoothness analyser for Pyro programs based on §5, and a variable selector based on §6 which (approximately) solves the variable-selection problem.[15] Our analyser and variable selector are implemented in OCaml, and support a subset of the Pyro PPL and two smoothness properties: differentiability and local Lipschitzness.

Implementation. Although the analysis described in §5 may look simple when considering a basic PPL, real-world PPLs such as Pyro are of a much higher degree of complexity. First, they provide a large panel of continuous/discrete probability distributions for sample and observe commands, and

[15] Our implementation is available at https://github.com/wonyeol/smoothness-analysis and [Lee et al. 2022a].

library functions for tensors and neural networks. Second, programs in real-world PPLs may fail to be smooth for reasons other than if-else and while commands. In particular, values sampled from discrete distributions, and arguments to operators and distribution constructors that are well-defined only on a strict subset of values, may induce non-smoothness. A straightforward treatment of these will result in an overly conservative analysis, treating far too many variables as potentially non-smooth. Third, Pyro programs typically rely on tensors (of large, statically unknown size) to deal with large datasets, and it is generally infeasible to reason about each (real-valued) element of tensors individually. In the following, we discuss how our static analyser addresses these issues and provides sound, useful information about smoothness of Pyro programs.

Distributions and library functions. Our analyser supports 17 distributions (continuous or discrete). Each distribution is characterized by a pair (b, a) for a boolean b and an array of booleans a, where b (or a_i) denotes whether its probability density is differentiable or locally Lipschitz with respect to the sampled value (or the i-th argument) of the distribution. For example, a normal distribution is described by `(true,[true,true])` (assuming that the second argument is positive) and a Poisson distribution by `(false,[true])`. Similarly, the analyser supports a large number of PyTorch/Pyro library functions for tensors and neural networks, and assumes the correct smoothness information about them. For instance, the ReLU function is considered locally Lipschitz but not differentiable.

Refining smoothness information based on safety pre-analysis. Although the expression `x/y` is generally non-smooth with respect to `y` (even if it is well-defined for `y=0`), if more information is available, for instance that `y` always lies in range $[1, 10]$, we can safely consider it smooth with respect to both `x` and `y`. Likewise, the density of a normal distribution is generally non-smooth with respect to the standard deviation argument σ (even if it is well-defined for $\sigma \leq 0$), so more precise smoothness information can be produced when σ is known to be always positive. Thus, establishing precise smoothness information requires to first establish safety properties related to program operations. To achieve this, our tool actually performs two analyses in sequence: (i) a safety pre-analysis infers ranges over all numerical variables and marks each argument to an operator or a distribution constructor as either "safe" or "potentially unsafe"; (ii) the program analysis formalised in §5 utilises information computed in the first phase to produce precise smoothness information. The first phase boils down to a forward abstract interpretation based on basic abstract domains like intervals and signs [Cousot and Cousot 1977]. It logs safety information for each program statement just like static analyses for runtime errors and undefined behaviors [Blanchet et al. 2003]. As formalised in §5.2, the second analysis is compositional. Due to their different nature, the two analyses need to be done in sequence.

Tensors. Pyro programs commonly use nested loops and indexed tensors. As the number of scalar values in each tensor is often statically unknown (or known but huge), treating each scalar as a separate variable is not feasible; so we rely on a conservative summarisation of tensors. Intuitively, we treat all scalars in a tensor as a "block": e.g., when density might not be smooth with respect to some scalar(s) of a tensor, the analysis conservatively concludes that it might not be smooth with respect to all scalars in the tensor. In our experiments, this abstraction does not result in any precision loss.

Evaluation. We evaluated our analyser and variable selector on 13 representative Pyro examples from the Pyro webpage [Uber AI Labs 2022] that use standard SVI engines and contain explicitly written model-guide pairs (without AutoGuide). They include advanced models with deep neural networks such as attend-infer-repeat [Eslami et al. 2016] and single-cell annotation using variational inference [Xu et al. 2021]. Additionally, we included the example in Fig. 1, for which Pyro offers an unsound reparameterisation plan. Table 3 lists half of these 14 Pyro examples with their code size and conceptual complexity (see §I for the rest). Experiments were performed on a Macbook Pro with 2.3GHz Core i9 and 32GB RAM.

Table 3. Subset of Pyro examples used in experiments and their key features (see §I for the rest). The last five columns show the total number of code lines (excluding comments), loops, sample commands, observe commands, and learnable parameters (declared explicitly by `pyro.param` or implicitly by a neural network module). Each number is the sum of the counts in the model and guide.

Name	Probabilistic model	LoC	`while`	`sam`	`obs`	`param`
`spnor`	Splitting normal example in Fig. 1	16	0	2	1	2
`sgdef`	Deep exponential family	105	0	12	1	12
`dmm`	Deep Markov model	112	3	2	1	13
`mhmm`	Hidden Markov models	137	1	5	5	12
`scanvi`	Single-cell annotation using variational inference	147	0	7	2	21
`air`	Attend-infer-repeat	174	2	6	1	16
`cvae`	Conditional variational autoencoder	205	0	2	1	15

Table 4. Results of smoothness analyses. "Manual" and "Ours" denote the number of continuous random variables and learnable parameters in which the density of the program is smooth, computed by hand and by our analyser. "Time" denotes the runtime of our analyser in seconds. "#CRP" denotes the total number of continuous random variables and learnable parameters in the program. `-m` and `-g` denote model and guide. We consider $\{(\alpha, i) \in \text{Name}\}$ as one random variable for each $\alpha \in \text{Name}$. See §I for the rest of results.

	Differentiable			Locally Lipschitz			
Name	Manual	Ours	Time	Manual	Ours	Time	#CRP
`spnor-m`	1	1	0.006	1	1	0.009	2
`spnor-g`	4	4	0.007	4	4	0.008	4
`sgdef-m`	6	6	0.003	6	6	0.006	6
`sgdef-g`	18	18	0.016	18	18	0.015	18
`dmm-m`	4	4	0.014	10	10	0.016	10
`dmm-g`	4	4	0.026	5	5	0.020	5
`mhmm-m`	10	10	0.063	10	10	0.075	10
`mhmm-g`	6	6	0.007	6	6	0.008	6
`scanvi-m`	6	6	0.032	12	12	0.032	12
`scanvi-g`	8	8	0.052	15	15	0.058	15
`air-m`	1	1	0.108	4	4	0.105	4
`air-g`	3	3	0.075	15	15	0.072	16
`cvae-m`	3	3	0.025	8	8	0.027	8
`cvae-g`	5	5	0.031	9	9	0.023	9

Smoothness analyser. We assess our smoothness analyser on the 14 Pyro examples for differentiability and local Lipschitzness (§5.3), and show a subset of results in Table 4 (see §I for the rest). The results demonstrate that our analysis can cope successfully with real-world Pyro programs. First, our analysis is accurate. For all examples, the analysis identifies the exact ground-truth set of random variables and parameters in which the density of the program is differentiable (or locally Lipschitz). In many of them, information computed by the pre-analysis is required to achieve these exact results; e.g., some examples (e.g., `dpmm` and `air`) require precise information about which distribution arguments can be proved to be always in the proper range of values. Second, the runtime of our analysis is low. Typical probabilistic programming applications are not of a very large size, and conceptual complexity is generally the main issue, thus the analysis performance presents no scalability concern.

We draw two more observations from the results. First, for `spnor-m` and `air-g`, the density of each program is not locally Lipschitz in one continuous random variable. These non-local-Lipschitznesses arise as follows: for the former, the random variable (`"z2"` in Fig. 1) is used in the branch condition of an if-else command that contains observe commands, thereby creating discontinuity; and for the latter, the random variable (`"z_where"`) is passed into the denominator of a division operator, thereby causing a division-by-zero error for some value.

Second, for all the other examples, the density is locally Lipschitz in all continuous random variables and parameters, but is often non-differentiable in many parameters (and continuous random

Table 5. Results of variable selections. "Ours-Time" denote the runtime of our variable selector in seconds. "Ours-Sound" and "Pyro \ Ours" denote the number of random variables in the example that are in π_{ours}, and that are in π_0 but not in π_{ours}, respectively, where π_{ours} and π_0 denote the reparameterisation plans given by our variable selector and by Pyro. "Pyro \ Ours" is partitioned into "Sound" and "Unsound": the latter denotes the number of random variables that make (R2') or (R3') violated when added to π_{ours}, and the former denotes the number of the rest. "#CR" and "#DR" denote the total number of continuous and discrete random variables in the example. We consider $\{(\alpha, i) \in \text{Name}\}$ as one random variable for each $\alpha \in \text{Name}$. See §I for the rest of results.

	Ours		Pyro \ Ours			
Name	Time	Sound	Sound	Unsound	#CR	#DR
`spnor`	0.021	1	0	1	2	0
`sgdef`	0.034	6	0	0	6	0
`dmm`	0.054	1	0	0	1	0
`mhmm`	0.083	2	0	0	2	1
`scanvi`	0.143	3	0	0	3	1
`air`	0.247	1	0	1	2	1
`cvae`	0.063	1	0	0	1	0

variables too); see, for instance, `scanvi` and `cvae`. Due to this, the requirement (R2) is not satisfied for these examples even with the empty reparameterisation plan (corresponding to the score estimator); that is, if we use the differentiability requirements (R2) and (R3) to validate the unbiasedness of gradient estimators, even the score estimator cannot be validated for these examples. From manual inspection, we checked that the non-differentiabilities from these examples all arise by the use of locally Lipschitz but non-differentiable operators (e.g., `relu` and `grid_sample`). Since many practical models (and guides) use locally Lipschitz but non-differentiable operators, this observation strongly suggests that a right smoothness requirement for validating gradient estimators is not differentiability (which has been used as a standard requirement), but rather local Lipschitzness (e.g., (R2') and (R3')).

Variable selector. To evaluate our variable selector, we consider the SPGE variable-selection problem with local Lipschitzness requirements, i.e., the problem that uses (R2') and (R3') in §4.3 instead of (R2) and (R3) in §4.2. We do not consider the original problem (with differentiability requirements), since for many examples the differentiability requirements are not satisfied even by the empty reparameterisation plan (i.e., score estimator) as observed above. For an initial reparameterisation plan π_0 for the problem, we use the plan given by Pyro's default variable selector: it is defined for all continuous random variables and applies standard reparameterisations (e.g., Eq. (11) for a normal distribution). In this settings, we apply our variable selector to the problem on the 14 Pyro examples. Table 5 displays the results (only for 7 examples; see §I for the rest) and compares them with π_0.

The results demonstrate that for all examples, our variable selector finds the optimal reparameterisation plan with a small runtime. We also observe that for all cases, it terminates in the first iteration and calls our smoothness analyser only three times, as mentioned in §6. Note that the reparameterisation plan given by Pyro is also optimal for all but two examples. We emphasise, however, that our variable selector not only finds a reparameterisation plan but also verifies the local Lipschitzness requirements (R2') and (R3'), whereas Pyro's default variable selector does not do so. Indeed, for two examples, Pyro's reparameterisation plan is unsound as it violates the local Lipschitzness requirements. Hence, these results should be interpreted as: for all but two examples, our variable selector (and smoothness analyser) successfully verifies that the default gradient estimator used by Pyro satisfies important smoothness-related preconditions for unbiasedness, namely the local Lipschitzness requirements.

The two examples for which Pyro becomes unsound are `spnor` and `air`. Recall that they have two continuous random variables (one for each) in which their densities are not locally Lipschitz. The unsoundness of Pyro on these examples stems precisely from the fact that it reparameterises the two non-locally-Lipschitz random variables without checking any local Lipschitzness requirements.

8 RELATED WORK

The high-level idea of using program transformation for improved posterior inference and model learning in PPLs has been explored previously [Claret et al. 2013; Gorinova et al. 2020; Nori et al. 2014; Ritchie et al. 2016b; Schulman et al. 2015]. In particular, Schulman et al. [2015] proposed a method for implementing the SPGE for stochastic computation graphs via graph transformation, and this method was adopted in the implementation of the same estimator in Pyro and also in our work. However, the method lacks a formal analysis on the implemented estimator especially in the context of probabilistic programs; it does not have a version of Theorem 4.5, which formally identifies requirements for the unbiasedness of the estimator. Also, the method does not check the required smoothness properties of given probabilistic programs. Our work fills in these gaps. Gorinova et al. [2020] proposed an automatic technique to transform models in a PPL using the same or closely-related transformation of sample commands in the SPGE. The work is, however, concerned with transforming models and taming their posterior distributions, while ours focuses on transforming guides. Also, the work does not check smoothness properties of transformed models that are required for running efficient inference algorithms, such as Hamiltonian Monte Carlo, on those models, while our work checks those properties using our program analysis.

Program analyses or type systems for PPLs have been developed to detect common errors [Lee et al. 2020; Lew et al. 2020; Wang et al. 2021], infer important probabilistic properties such as conditional independence [Gorinova et al. 2022], or automatically plan inference algorithms [WebPPL 2019] as in our work. In particular, WebPPL runs a simple program analysis (checking if there are interleaving sample and observe commands) to decide if it is worth applying sequential Monte Carlo.

The smoothness properties computed by our program analysis, such as differentiability and local Lipschitzness, fall in the scope of *hyperliveness* in the hierarchy of hyperproperties [Clarkson and Schneider 2008]. Intuitively, hyperliveness properties are those that cannot be refuted based on any finite counterexample (i.e., made of finitely-many finite execution traces), and counterexamples for differentiability and local Lipschitzness should indeed require infinitely-many execution traces due to the use of limit or all neighbouring inputs in their definitions. Not so many analyses have considered such hyperliveness properties. Among those, the most relevant to our work are the continuity analyses of Chaudhuri et al. [2010, 2012]. It uses a program abstraction that is rather similar to ours, but their analyses suffer from soundness issues, partly due to the incorrect joining of continuity sets [Chaudhuri et al. 2010] and also to an unsound rule for sequential composition [Chaudhuri et al. 2012] (see §A for details). We do not claim that these issues are difficult to fix. Our point is just that developing program analyses for smoothness properties requires special care. Chaudhuri et al.'s work focuses on proving smoothness properties of control software, or revealing the unexpected continuity of discrete algorithms. On the other hand, our program analysis is designed to assist variational inference and model learning for probabilistic programs. Barthe et al. [2020] proposed a refinement type system, which considers a higher-order functional language and ensures that every typable first-order program is continuous in all variables. On the other hand, our program analysis considers a first-order imperative language and can prove that a program is continuous in some (not necessarily all) variables. Other existing program analyses for smoothness properties include [Laurel et al. 2022] which over-approximates the Clarke generalised Jacobian, and [Mangal et al. 2020] which proves probabilistic Lipschitzness.

ACKNOWLEDGMENTS

We thank Hangyeol Yu for helping us prove Theorems 4.2 and 4.5, and anonymous reviewers for giving constructive comments. Lee was supported by Samsung Scholarship. Yang was 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 the Institute for Basic Science (IBS-R029-C1). Rival was supported by the French ANR VeriAMOS project.

REFERENCES

Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein Generative Adversarial Networks. In *International Conference on Machine Learning (ICML)*. 214–223.

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

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.

B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. 2003. A Static Analyzer for Large Safety Critical Software. In *Programming Languages, Design and Implementation (PLDI)*. 196–207. https://doi.org/10.1145/781131.781153

Vladimir I. Bogachev. 2007. *Measure Theory* (first ed.). Springer. https://doi.org/10.1007/978-3-540-34514-5

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* 76, 1 (2017), 1–32. https://doi.org/10.18637/jss.v076.i01

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.

Swarat Chaudhuri, Sumit Gulwani, and Roberto Lublinerman. 2010. Continuity analysis of programs. In *Principles of Programming Languages (POPL)*. 57–70. https://doi.org/10.1145/1706299.1706308

Swarat Chaudhuri, Sumit Gulwani, and Roberto Lublinerman. 2012. Continuity and robustness of programs. *Commun. ACM* 55, 8 (2012), 107–115. https://doi.org/10.1145/2240236.2240262

Guillaume Claret, Sriram K. Rajamani, Aditya V. Nori, Andrew D. Gordon, and Johannes Borgström. 2013. Bayesian inference using data flow analysis. In *Foundations of Software Engineering (FSE)*. 92–102. https://doi.org/10.1145/2491411.2491423

M. R. Clarkson and F. B. Schneider. 2008. Hyperproperties. In *Computer Security Foundations (CSF)*. 51–65. https://doi.org/10.1109/CSF.2008.7

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. https://doi.org/10.1145/512950.512973

Patrick Cousot and Radhia Cousot. 1979. Systematic design of program analysis frameworks. In *Principles of Programming Languages (POPL)*. 269–282. https://doi.org/10.1145/567752.567778

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.

Hong Ge, Kai Xu, and Zoubin Ghahramani. 2018. Turing: A Language for Flexible Probabilistic Inference. In *Artificial Intelligence and Statistics (AISTATS)*. 1682–1690.

Timon Gehr, Sasa Misailovic, and Martin T. Vechev. 2016. PSI: Exact Symbolic Inference for Probabilistic Programs. In *Computer Aided Verification (CAV)*. 62–83. https://doi.org/10.1007/978-3-319-41528-4_4

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. https://doi.org/10.1145/2578855.2535850

Maria I. Gorinova, Andrew D. Gordon, Charles Sutton, and Matthijs Vákár. 2022. Conditional Independence by Typing. *ACM Trans. Program. Lang. Syst.* 44, 1 (2022), 4:1–4:54. https://doi.org/10.1145/3490421

Maria I. Gorinova, Dave Moore, and Matthew D. Hoffman. 2020. Automatic Reparameterisation of Probabilistic Programs. In *International Conference on Machine Learning (ICML)*. 3648–3657.

Steven Holtzen, Guy Van den Broeck, and Todd D. Millstein. 2020. Scaling exact inference for discrete probabilistic programs. *Proc. ACM Program. Lang.* 4, OOPSLA (2020), 140:1–140:31. https://doi.org/10.1145/3428208

Hyunjik Kim, George Papamakarios, and Andriy Mnih. 2021. The Lipschitz Constant of Self-Attention. In *International Conference on Machine Learning (ICML)*. 5562–5571.

Diederik P. Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes. In *International Conference on Learning Representations (ICLR)*.

Alp Kucukelbir, Rajesh Ranganath, Andrew Gelman, and David M. Blei. 2015. Automatic Variational Inference in Stan. In *Neural Information Processing Systems (NIPS)*. 568–576.

Jacob Laurel, Rem Yang, Gagandeep Singh, and Sasa Misailovic. 2022. A dual number abstraction for static analysis of Clarke Jacobians. *Proc. ACM Program. Lang.* 6, POPL (2022), 1–30. https://doi.org/10.1145/3498718

John M. Lee. 2012. *Introduction to Smooth Manifolds* (second ed.). Springer. https://doi.org/10.1007/978-1-4419-9982-5

Wonyeol Lee, Xavier Rival, and Hongseok Yang. 2022a. Artifact for the Paper "Smoothness Analysis for Probabilistic Programs with Application to Optimised Variational Inference". https://doi.org/10.5281/zenodo.7246597

Wonyeol Lee, Xavier Rival, and Hongseok Yang. 2022b. Smoothness Analysis for Probabilistic Programs with Application to Optimised Variational Inference. *arXiv:2208.10530* (2022). https://doi.org/10.48550/ARXIV.2208.10530

Wonyeol Lee, Hangyeol Yu, Xavier Rival, and Hongseok Yang. 2020. Towards verified stochastic variational inference for probabilistic programs. *Proc. ACM Program. Lang.* 4, POPL (2020), 16:1–16:33. https://doi.org/10.1145/3371084

Alexander K. Lew, Marco F. Cusumano-Towner, Benjamin Sherman, Michael Carbin, and Vikash K. Mansinghka. 2020. Trace types and denotational semantics for sound programmable inference in probabilistic languages. *Proc. ACM Program. Lang.* 4, POPL (2020), 19:1–19:32. https://doi.org/10.1145/3371087

Ravi Mangal, Kartik Sarangmath, Aditya V. Nori, and Alessandro Orso. 2020. Probabilistic Lipschitz Analysis of Neural Networks. In *Static Analysis Symposium (SAS)*. 274–309. https://doi.org/10.1007/978-3-030-65474-0_13

Vikash K. Mansinghka, Daniel Selsam, and Yura N. Perov. 2014. Venture: a higher-order probabilistic programming platform with programmable inference. *arXiv:1404.0099* (2014). https://doi.org/10.48550/ARXIV.1404.0099

T. Minka, J.M. Winn, J.P. Guiver, S. Webster, Y. Zaykov, B. Yangel, A. Spengler, and J. Bronskill. 2014. Infer.NET 2.6. https://dotnet.github.io/infer/.

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. https://doi.org/10.1007/978-3-319-29604-3_5

Radford M. Neal. 2011. MCMC Using Hamiltonian Dynamics. In *Handbook of Markov Chain Monte Carlo*. 113–162. https://doi.org/10.1201/b10905

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. https://doi.org/10.1609/aaai.v28i1.9060

André Platzer. 2018. *Logical Foundations of Cyber-Physical Systems*. Springer. https://doi.org/10.1007/978-3-319-63588-0

Rajesh Ranganath, Sean Gerrish, and David M. Blei. 2014. Black Box Variational Inference. In *Artificial Intelligence and Statistics (AISTATS)*. 814–822.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. 2014. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. In *International Conference on Machine Learning (ICML)*. 7344–7353.

Daniel Ritchie, Paul Horsfall, and Noah D. Goodman. 2016a. Deep Amortized Inference for Probabilistic Programs. *arXiv:1610.05735* (2016). https://doi.org/10.48550/ARXIV.1610.05735

Daniel Ritchie, Andreas Stuhlmüller, and Noah D. Goodman. 2016b. C3: Lightweight Incrementalized MCMC for Probabilistic Programs using Continuations and Callsite Caching. In *Artificial Intelligence and Statistics (AISTATS)*. 28–37.

John Salvatier, Thomas V. Wiecki, and Christopher Fonnesbeck. 2016. Probabilistic programming in Python using PyMC3. *PeerJ Comput. Sci.* 2 (2016), e55. https://doi.org/10.7717/peerj-cs.55

John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. 2015. Gradient Estimation Using Stochastic Computation Graphs. In *Neural Information Processing Systems (NIPS)*. 3528–3536.

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.

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. https://doi.org/10.1145/2933575.2935313

David Tolpin, Jan-Willem van de Meent, Hongseok Yang, and Frank D. Wood. 2016. Design and Implementation of Probabilistic Programming Language Anglican. In *Implementation and Application of Functional Programming Languages (IFL)*. 6:1–6:12. https://doi.org/10.1145/3064899.3064910

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). https://doi.org/10.48550/ARXIV.1610.09787

Uber AI Labs. 2022. Pyro examples. http://pyro.ai/examples/. Version used: June 18, 2022.

Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. 2018. An Introduction to Probabilistic Programming. *arXiv:1809.10756* (2018). https://doi.org/10.48550/ARXIV.1809.10756

Di Wang, Jan Hoffmann, and Thomas W. Reps. 2021. Sound probabilistic inference via guide types. In *Programming Language Design and Implementation (PLDI)*. 788–803. https://doi.org/10.1145/3453483.3454077

WebPPL. 2019. https://github.com/probmods/webppl/blob/v0.9.15/src/header.wppl#L510.

Ronald J. Williams. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning* 8, 3-4 (1992), 229–256. https://doi.org/10.1007/BF00992696

David Wingate, Noah D. Goodman, Andreas Stuhlmüller, and Jeffrey Mark Siskind. 2011a. Nonstandard Interpretations of Probabilistic Programs for Efficient Inference. In *Neural Information Processing Systems (NIPS)*. 1152–1160.

David Wingate, Andreas Stuhlmüller, and Noah D. Goodman. 2011b. Lightweight Implementations of Probabilistic Programming Languages Via Transformational Compilation. In *Artificial Intelligence and Statistics (AISTATS)*. 770–778.

David Wingate and Theophane Weber. 2013. Automated Variational Inference in Probabilistic Programming. *arXiv:1301.1299* (2013). https://doi.org/10.48550/ARXIV.1301.1299

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.

Chenling Xu, Romain Lopez, Edouard Mehlman, Jeffrey Regier, Michael I Jordan, and Nir Yosef. 2021. Probabilistic harmonization and annotation of single-cell transcriptomics data with deep generative models. *Molecular systems biology* 17, 1 (2021), e9620. https://doi.org/10.15252/msb.20209620

Yuan Zhou, Hongseok Yang, Yee Whye Teh, and Tom Rainforth. 2020. Divide, Conquer, and Combine: a New Inference Strategy for Probabilistic Programs with Stochastic Support. In *International Conference on Machine Learning (ICML)*. 11534–11545.

Received 2022-07-07; accepted 2022-11-07

A DEFERRED RESULTS IN §1

A.1 Unsoundness of Continuity Analyses in [Chaudhuri et al. 2010, 2012]

The continuity analysis in [Chaudhuri et al. 2010] considers joint continuity, whereas the continuity analysis in [Chaudhuri et al. 2012] considers partial continuity. That is, given a command c, an output variable v of c, and some input variables $u_1, \ldots, u_m$ to c, the former analyses whether v is continuous in $\{u_1, \ldots, u_m\}$ jointly, whereas the latter analyses whether v is continuous in u_i separately for every $1 \leq i \leq m$.

Join and Sequence rules. The former analysis contains a rule called Join [Chaudhuri et al. 2010, Figure 3], and the latter analysis contains a rule called Sequence [Chaudhuri et al. 2012, Figure 1]. The two rules can be rewritten (with some simplifications) as follows, in terms of functions between $\mathbb{R}^n$: for any $f, g : \mathbb{R}^n \to \mathbb{R}^n$ and $S, S', T, U \subseteq \{1, \ldots, n\}$,

$$\frac{\begin{array}{c}\text{For each } j \in T,\ f_j \text{ is continuous in } \{x_i \mid i \in S\} \\ \text{For each } j \in T,\ f_j \text{ is continuous in } \{x_i \mid i \in S'\}\end{array}}{\text{For each } j \in T,\ f_j \text{ is continuous in } \{x_i \mid i \in S \cup S'\}}\ \text{(Join)}$$

$$\frac{\begin{array}{c}\text{For each } j \in T,\ f_j \text{ is continuous in } x_i \text{ for each } i \in S \\ \text{For each } k \in U,\ g_k \text{ is continuous in } y_j \text{ for each } j \in T\end{array}}{\text{For each } k \in U,\ (g \circ f)_k \text{ is continuous in } x_i \text{ for each } i \in S}\ \text{(Sequence)}$$

where f and g are functions of variables $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$, respectively, and $h_i \triangleq \mathit{proj}_i \circ h$ for $h : \mathbb{R}^n \to \mathbb{R}^n$ and $i \in \{1, \ldots, n\}$ denotes the i-th component of h. As mentioned above, the Join rule analyses joint continuity, while the Sequence rule analyses partial continuity. Further, the Join rule says that joint continuity is preserved under the union of input variables, while the Sequence rule says that partial continuity is preserved under the composition of functions.

The two rules, however, are unsound with the following counterexamples. Let $h : \mathbb{R}^2 \to \mathbb{R}^2$ be the function

$$h(x_1, x_2) \triangleq \begin{cases} (x_1 x_2 / (x_1^2 + x_2^2), x_2) & \text{if } (x_1, x_2) \neq (0, 0) \\ (0, x_2) & \text{otherwise.} \end{cases}$$

Note that h_1 is continuous in x_1 and in x_2 separately, but not in $\{x_1, x_2\}$ jointly. First, for the Join rule, consider the following $f : \mathbb{R}^2 \to \mathbb{R}^2$ and $S, S', T \subseteq \{1, 2\}$:

$$f(x_1, x_2) \triangleq h(x_1, x_2), \quad S \triangleq \{1\}, \quad S' \triangleq \{2\}, \quad T \triangleq \{1, 2\}.$$

Then, the premise of the Join rule holds, and so the conclusion of the rule must hold. But this is *not* the case since $f_1 = h_1$ is not continuous in $\{x_1, x_2\}$. Hence, the Join rule is unsound. Next, for the Sequence rule, consider the following $f, g : \mathbb{R}^2 \to \mathbb{R}^2$ and $S, T, U \subseteq \{1, 2\}$:

$$f(x_1, x_2) \triangleq (x_1, x_1), \quad g(x_1, x_2) \triangleq h(x_1, x_2), \quad S \triangleq T \triangleq U \triangleq \{1, 2\}.$$

Then, the premise of the Sequence rule holds, and so the conclusion of the rule must hold. But this is *not* the case since $(g \circ f)_1$ is not continuous in x_1 (due to $(g \circ f)_1(x_1, x_2) = \mathbf{1}_{[x_1 \neq 0]} \cdot \frac{1}{2}$). Hence, the Sequence rule is unsound. These counterexamples show that joint continuity is *not* preserved under the union of input variables, and partial continuity is *not* preserved under the composition of functions.

The two aforementioned counterexamples can be easily translated into programs: the first becomes $c_1 \equiv (z := h_1(x, y))$ and the second becomes $c_2 \equiv (y := x;\ z := h_1(x, y))$, where x, y, and z are program variables and h_1 is the binary operator defined above. The analysis in [Chaudhuri et al. 2010] deduces that in c_1, z is continuous in x and y (jointly), and the analysis in [Chaudhuri et al. 2012] deduces that in c_2, z is continuous in x (separately). Both deductions, however, are incorrect as seen above, and the two analyses are thus unsound.

Sequence rule (again). Both of the two analyses in [Chaudhuri et al. 2010, 2012] contain the Sequence rule (discussed above) which has the following rule as an instance: for all variables v_0, v_1, v_2 that are not necessarily distinct, and for all commands c_1, c_2,

$$\frac{\text{In } c_1, v_1 \text{ is continuous in } v_0 \quad \text{In } c_2, v_2 \text{ is continuous in } v_1}{\text{In } (c_1; c_2), v_2 \text{ is continuous in } v_0}$$

The above instance of the Sequence rule is, however, unsound because it incorrectly handles the dependencies between variables. For instance, consider commands $c_1 \equiv (\texttt{if}\ (x < 0)\ \{y := 0\}\ \texttt{else}\ \{y := 1\})$ and $c_2 \equiv (z := x + y)$, and variables $v_0 \equiv x$, $v_1 \equiv x$, and $v_2 \equiv z$. Then, in $(c_1; c_2)$, z is *not* continuous in x (because z after $(c_1; c_2)$ is x if $x < 0$, and is $x + 1$ if $x \geq 0$). However, the premise of the above rule holds for this case, and so the rule incorrectly concludes that z is continuous in x under $(c_1; c_2)$, by ignoring the dependency of z on y (which is discontinuous in x).

Loop rule. The analysis in [Chaudhuri et al. 2010] contains a rule called Simple-loop (Figure 5 in the paper) to analyse loops, and the analysis in [Chaudhuri et al. 2012] contains a rule called Loop (Figure 1 in the paper) which is essentially the same as the Simple-loop rule.

The two rules, however, incorrectly assume that (joint or partial) continuity without any restriction on the domain of functions satisfies the admissibility assumption discussed in Remark 5.9; and this in turn makes the rules unsound. To illustrate the unsoundness, consider a command $c \equiv (\texttt{while}\ (0 < x < 1)\ \{c'\})$ with $c' \equiv (y := y + f(x);\ x := g(x))$, where f and g are the continuous operators defined by:

$$f(x) \triangleq \begin{cases} x & \text{if } 0 < x \leq 1/2 \\ 1 - x & \text{if } 1/2 < x \leq 1 \\ 0 & \text{otherwise,} \end{cases} \qquad g(x) \triangleq \begin{cases} 0 & \text{if } x \leq 0 \\ 2x & \text{if } 0 < x \leq 1/2 \\ 1 & \text{otherwise.} \end{cases}$$

Then, the premises of the two rules are satisfied, mainly because x and y after c' are continuous jointly in x and y before c', and x and y do not change in c' if $x = 0$ or $x = 1$ (i.e., at the "boundary" of the loop condition of c). Hence, the two rules conclude that x and y after c are continuous in x and y before c (jointly or separately). However, this is an unsound conclusion because in c, y is *not* continuous in x at $x = 0$: we have $y' = y$ if $x = 0$, but $y' \to y + 1$ as $x \to 0^+$ (more precisely, $y' = y$ if $x \leq 0$ or $x \geq 1$, and $y' = y + (1 - x)$ if $0 < x < 1$), where x' and y' denote the values of x and y after c.

B DEFERRED RESULTS IN §2

B.1 Table Summarising §2

Table 6. Gradient estimators for variational inference, and requirements for each estimator. "Req." denotes "Requirement" and "diff." denotes "differentiable". Recall that $f_\theta(z) \triangleq \log(p_{c_m}(z)/p_{c_g,\theta}(z))$.

		SCE	PGE	SPGE
Setup	$q_\theta(z)$	$p_{c_g,\theta}(z)$	$p_{c'_g}(z)$	$p_{c''_g,\theta}(z)$
	$v_\theta(z)$	z	$v_{c'_g,\theta}(z)$	$v_{c''_g,\theta}(z)$
	$g_\theta(z)$	$f_\theta(v_\theta(z)) \cdot \nabla_\theta \log q_\theta(z)$	$\nabla_\theta(f_\theta(v_\theta(z)))$	$\nabla_\theta(f_\theta(v_\theta(z))) + f_\theta(v_\theta(z)) \cdot \nabla_\theta \log q_\theta(z)$
Req.	$q_\theta(z)$	diff. in θ	–	diff. in θ
	$v_\theta(z)$	–	diff. in θ	diff. in θ
	$p_{c_m}(z)$	–	diff. in θ and z	diff. in θ and "selected" z_i's
	$p_{c_g,\theta}(z)$	–	diff. in θ and z	diff. in θ and "selected" z_i's

Table 6 compares key aspects of the three gradient estimators (SCE, PGE, and SPGE) explained in §2.

C DEFERRED RESULTS IN §4.1

C.1 Proof of Theorem 4.2

We introduce several definitions, state lemmas, and prove Theorem 4.2 using the lemmas. We prove the lemmas in §C.2 and §C.3.

Recall the partition $\mathsf{Var} = \mathsf{PVar} \uplus \mathsf{Name} \uplus \mathsf{AVar}$ of Var. We use the following letters to denote the values of each part: $\sigma_p \in \mathsf{St}[\mathsf{PVar}]$, $\sigma_n \in \mathsf{St}[\mathsf{Name}]$, and $\sigma_a \in \mathsf{St}[\mathsf{AVar}]$. Based on the partition, we define the next functions:

$$prs(c) : \mathsf{St}[\mathsf{PVar}] \times \mathsf{St}[\mathsf{Name}] \times \mathsf{St}[\mathsf{AVar}] \to [0, \infty),$$

$$prs(c)(\sigma_p, \sigma_n, \sigma_a) \triangleq \begin{cases} [\![c]\!](\sigma_p \oplus \sigma_n \oplus \sigma_a)(like) \\ \quad \cdot \prod_{\mu \in \mathsf{Name}} [\![c]\!](\sigma_p \oplus \sigma_n \oplus \sigma_a)(pr_\mu) & \text{if } noerr(c, \sigma_p \oplus \sigma_n \oplus \sigma_a) \\ 0 & \text{otherwise,} \end{cases}$$

$$vals(c) : \mathsf{St}[\mathsf{PVar}] \times \mathsf{St}[\mathsf{Name}] \times \mathsf{St}[\mathsf{AVar}] \to \mathsf{St}[\mathsf{Name}],$$

$$vals(c)(\sigma_p, \sigma_n, \sigma_a) \triangleq \begin{cases} \lambda\mu \in \mathsf{Name}.\, [\![c]\!](\sigma_p \oplus \sigma_n \oplus \sigma_a)(val_\mu) & \text{if } noerr(c, \sigma_p \oplus \sigma_n \oplus \sigma_a) \\ \lambda\mu \in \mathsf{Name}.\, 0 & \text{otherwise,} \end{cases}$$

where $noerr(c, \sigma)$ is a predicate for a command c and $\sigma \in \mathsf{St}$, defined by

$$noerr(c, \sigma) \iff [\![c]\!]\sigma \in \mathsf{St} \wedge \big(\forall \mu \in \mathsf{Name}.\, [\![c]\!]\sigma(cnt_\mu) - \sigma(cnt_\mu) \leq 1\big).$$

The predicate $noerr(c, \sigma)$ says that c terminates for σ without a double-sampling error. The functions prs and $vals$ generalise the density function p and the value function v, respectively; in particular, they do not assume a particular initial state σ_0 used in Eq. (3). We consider the generalisation of p and v so as to enable inductive proofs.

Although generalising p and v, the functions prs and $vals$ are not sufficient to enable inductive proofs since their inputs and outputs contain some unnecessary parts, which stops induction from working well (especially in the sequential composition case): namely, the part of $\mathsf{St}[\mathsf{Name}]$ that is not read during execution, and the part of $\mathsf{St}[\mathsf{AVar}]$ that is not updated during execution. To exclude those unnecessary parts, we first define the set of substates of $\mathsf{St}[\mathsf{Name}]$ as follows:

$$\xi_n \in \mathsf{St}_\square[\mathsf{Name}] \triangleq \bigcup_{K \subseteq \mathsf{Name}} \mathsf{St}[K].$$

Based on these substates, we define the next functions:

$$prs_\square(c) : \mathsf{St}[\mathsf{PVar}] \times \mathsf{St}_\square[\mathsf{Name}] \to [0, \infty),$$

$$prs_\square(c)(\sigma_p, \xi_n) \triangleq \begin{cases} [\![c]\!](\sigma_p \oplus \xi_n \oplus \sigma_r)(like) \\ \quad \cdot \prod_{\mu \in \mathrm{dom}(\xi_n)} [\![c]\!](\sigma_p \oplus \xi_n \oplus \sigma_r)(pr_\mu) & \text{if } \exists \sigma_r.\, used(c, \sigma_p \oplus \xi_n \oplus \sigma_r, \xi_n) \\ 0 & \text{otherwise,} \end{cases}$$

$$vals_\square(c) : \mathsf{St}[\mathsf{PVar}] \times \mathsf{St}_\square[\mathsf{Name}] \to \mathsf{St}_\square[\mathsf{Name}],$$

$$vals_\square(c)(\sigma_p, \xi_n) \triangleq \begin{cases} \lambda\mu \in \mathrm{dom}(\xi_n).\, [\![c]\!](\sigma_p \oplus \xi_n \oplus \sigma_r)(val_\mu) & \text{if } \exists \sigma_r.\, used(c, \sigma_p \oplus \xi_n \oplus \sigma_r, \xi_n) \\ \lambda\mu \in \mathrm{dom}(\xi_n).\, 0 & \text{otherwise,} \end{cases}$$

$$pvars_\square(c) : \mathsf{St}[\mathsf{PVar}] \times \mathsf{St}_\square[\mathsf{Name}] \to \mathsf{St}[\mathsf{PVar}],$$

$$pvars_\square(c)(\sigma_p, \xi_n) \triangleq \begin{cases} \lambda x \in \mathsf{PVar}.\, [\![c]\!](\sigma_p \oplus \xi_n \oplus \sigma_r)(x) & \text{if } \exists \sigma_r.\, used(c, \sigma_p \oplus \xi_n \oplus \sigma_r, \xi_n) \\ \lambda x \in \mathsf{PVar}.\, 0 & \text{otherwise,} \end{cases}$$

where $used(c, \sigma, \xi_n)$ is a predicate for a command c, $\sigma \in \mathsf{St}$, and $\xi_n \in \mathsf{St}_\square[\mathsf{Name}]$, defined by

$$used(c, \sigma, \xi_n) \iff noerr(c, \sigma) \wedge \big(\sigma(like) = 1\big) \wedge \big(\xi_n = \sigma|_{\mathrm{dom}(\xi_n)}\big) \\ \wedge \big(\mathrm{dom}(\xi_n) = \{\mu \in \mathsf{Name} \mid [\![c]\!]\sigma(cnt_\mu) - \sigma(cnt_\mu) = 1\}\big).$$

The predicate $used(c, \sigma, \xi_n)$ says that c terminates for σ without a double-sampling error, *like* is initialised to 1 in σ, and ξ_n is the Name part of σ that is sampled during the execution of c from σ. By using $used(-,-,-)$, the three functions do not take the unnecessary part of a state as an input, and do not return the unnecessary part of a state in the output. The three functions are well-defined.

Lemma C.1. *$prs_\square$, $vals_\square$, and $pvars_\square$ are well-defined, i.e., they do not depend on the choice of σ_r.*

We now state two main lemmas for Theorem 4.2. The first lemma describes how *prs* and *vals* are connected with $prs_\square$ and $vals_\square$. The second lemma says that a particular integral involving $prs_\square$, $vals_\square$, and $pvars_\square$ is the same for c and $\overline{c}^\pi$ if a reparameterisation plan π is valid.

Lemma C.2. *Let c be a command, and $f_i : \mathbb{R} \to \mathbb{R}$ for $i \in \{1, 2, 3\}$ be measurable functions such that $f_1(r) \geq 0$ for all $r \in \mathbb{R}$. Define $f_* : \mathsf{St}[\mathsf{Name}] \to \mathsf{St}[\mathsf{AVar}]$ by*

$$f_*(\sigma_n)(a) \triangleq \begin{cases} 1 & \text{if } a \equiv like \\ f_1(\sigma_n(\mu)) & \text{if } a \equiv pr_\mu \text{ for } \mu \in \mathsf{Name} \\ f_2(\sigma_n(\mu)) & \text{if } a \equiv val_\mu \text{ for } \mu \in \mathsf{Name} \\ f_3(\sigma_n(\mu)) & \text{if } a \equiv cnt_\mu \text{ for } \mu \in \mathsf{Name}. \end{cases}$$

Then, for all $\sigma_p \in \mathsf{St}[\mathsf{PVar}]$ and all measurable $h : \mathsf{St}[\mathsf{Name}] \to \mathbb{R}$,

$$\int d\sigma_n \Big(prs(c)(\sigma_p, \sigma_n, f_*(\sigma_n)) \cdot h\Big(vals(c)(\sigma_p, \sigma_n, f_*(\sigma_n))\Big)\Big) \\ = \int d\xi_n \Big(prs_\square(c)(\sigma_p, \xi_n) \cdot g\Big(vals_\square(c)(\sigma_p, \xi_n)\Big)\Big)$$

where the integral on the LHS is defined if and only if the one on the RHS is defined, and the function $g : \mathsf{St}_\square[\mathsf{Name}] \to \mathbb{R}$ is defined by

$$g(\xi_n'') = \int d\xi_n' \Big(\mathbf{1}_{[\mathrm{dom}(\xi_n'') \uplus \mathrm{dom}(\xi_n') = \mathsf{Name}]} \cdot \Big(\prod_{\mu \in \mathrm{dom}(\xi_n')} f_1(\xi_n'(\mu))\Big) \cdot h\Big(\xi_n'' \oplus \lambda\mu \in \mathrm{dom}(\xi_n').\, f_2(\xi_n'(\mu))\Big)\Big). \tag{14}$$

Lemma C.3. *Let c be a command and $g : \mathsf{St}[\mathsf{PVar}] \times \mathsf{St}_\square[\mathsf{Name}] \to \mathbb{R}$ be a measurable function. Then, for all $\sigma_p \in \mathsf{St}[\mathsf{PVar}]$,*

$$\int d\xi_n \Big(prs_\square(c)(\sigma_p, \xi_n) \cdot g\Big(pvars_\square(c)(\sigma_p, \xi_n), vals_\square(c)(\sigma_p, \xi_n)\Big)\Big) \\ = \int d\xi_n \Big(prs_\square(\overline{c}^\pi)(\sigma_p, \xi_n) \cdot g\Big(pvars_\square(\overline{c}^\pi)(\sigma_p, \xi_n), vals_\square(\overline{c}^\pi)(\sigma_p, \xi_n)\Big)\Big)$$

where the integral on the LHS is defined if and only if the one on the RHS is defined.

We now prove Theorem 4.2 using these two lemmas.

Proof of Theorem 4.2. Let π be a valid reparameterisation plan, c be a command, $\sigma_\theta \in \mathsf{St}[\theta]$, and $h : \mathsf{St}[\mathsf{Name}] \to \mathbb{R}$ be a measurable function. Suppose that the integral on the LHS of Theorem 4.2 is defined. Recall that for a given $\sigma_n \in \mathsf{St}[\mathsf{Name}]$, the definitions of p and v (in §3 and §4.1) use the

initial state $\sigma \triangleq \sigma_\theta \oplus \sigma_n \oplus \sigma_0 \in \mathrm{St}$, where $\sigma_0 \in \mathrm{St}[(\mathrm{PVar} \setminus \theta) \cup \mathrm{AVar}]$ depends on σ_n and has the following definition:

$$\sigma_0(v) \triangleq \begin{cases} 0 & \text{if } v \in \mathrm{PVar} \setminus \theta \\ 1 & \text{if } v \equiv \mathit{like} \\ \mathcal{N}(\sigma_n(\mu); 0, 1) & \text{if } v \equiv \mathit{pr}_\mu \text{ for } \mu \in \mathrm{Name} \\ \sigma_n(\mu) & \text{if } v \equiv \mathit{val}_\mu \text{ for } \mu \in \mathrm{Name} \\ 0 & \text{if } v \equiv \mathit{cnt}_\mu \text{ for } \mu \in \mathrm{Name}. \end{cases}$$

The initial state σ can be re-expressed as

$$\sigma = \sigma_p \oplus \sigma_n \oplus f_*(\sigma_n) \tag{15}$$

using the following $\sigma_p \in \mathrm{St}[\mathrm{PVar}]$ and $f_* : \mathrm{St}[\mathrm{Name}] \to \mathrm{St}[\mathrm{AVar}]$:

$$\sigma_p(x) \triangleq \begin{cases} \sigma_\theta(x) & \text{if } x \in \theta \\ 0 & \text{if } x \in \mathrm{PVar} \setminus \theta, \end{cases} \qquad f_*(\sigma_n)(a) \triangleq \begin{cases} 1 & \text{if } a \equiv \mathit{like} \\ f_1(\sigma_n(\mu)) & \text{if } a \equiv \mathit{pr}_\mu \text{ for } \mu \in \mathrm{Name} \\ f_2(\sigma_n(\mu)) & \text{if } a \equiv \mathit{val}_\mu \text{ for } \mu \in \mathrm{Name} \\ f_3(\sigma_n(\mu)) & \text{if } a \equiv \mathit{cnt}_\mu \text{ for } \mu \in \mathrm{Name}, \end{cases}$$

where $f_1(r) \triangleq \mathcal{N}(r; 0, 1)$, $f_2(r) \triangleq r$, and $f_3(r) \triangleq 0$. Using this, we get the desired equation:

$$\begin{aligned} &\int d\sigma_n \Big(p_{c,\sigma_\theta}(\sigma_n) \cdot h\Big(v_{c,\sigma_\theta}(\sigma_n) \Big) \Big) \\ &= \int d\sigma_n \Big(\mathit{prs}(c)(\sigma_p, \sigma_n, f_*(\sigma_n)) \cdot h\Big(\mathit{vals}(c)(\sigma_p, \sigma_n, f_*(\sigma_n)) \Big) \Big) \\ &= \int d\xi_n \Big(\mathit{prs}_\square(c)(\sigma_p, \xi_n) \cdot g\Big(\mathit{vals}_\square(c)(\sigma_p, \xi_n) \Big) \Big) \\ &= \int d\xi_n \Big(\mathit{prs}_\square(\overline{c}^\pi)(\sigma_p, \xi_n) \cdot g\Big(\mathit{vals}_\square(\overline{c}^\pi)(\sigma_p, \xi_n) \Big) \Big) \\ &= \int d\sigma_n \Big(\mathit{prs}(\overline{c}^\pi)(\sigma_p, \sigma_n, f_*(\sigma_n)) \cdot h\Big(\mathit{vals}(\overline{c}^\pi)(\sigma_p, \sigma_n, f_*(\sigma_n)) \Big) \Big) \\ &= \int d\sigma_n \Big(p_{\overline{c}^\pi,\sigma_\theta}(\sigma_n) \cdot h\Big(v_{\overline{c}^\pi,\sigma_\theta}(\sigma_n) \Big) \Big) \end{aligned}$$

where $g : \mathrm{St}_\square[\mathrm{Name}] \to \mathbb{R}$ is defined as Eq. (14). The first equality holds by Eq. (15) and the definition of p_{c,σ_θ}, v_{c,σ_θ}, *prs*, and *vals*. The second equality holds by Lemma C.2 (applied to c). The third equality follows from Lemma C.3. The fourth equality holds by Lemma C.2 (applied to $\overline{c}^\pi$). The fifth equality holds by Eq. (15) and the definitions of $p_{\overline{c}^\pi,\sigma_\theta}$, $v_{\overline{c}^\pi,\sigma_\theta}$, *prs*, and *vals*. Note that the same equational reasoning with the reverse direction can be used to prove the claimed equation of the theorem when the integral on the RHS of the equation is defined. □

C.2 Proofs of Lemmas C.1 and C.2

Proof of Lemma C.1. Let c be a command, $\sigma_p \in \mathrm{St}[\mathrm{PVar}]$, and $\xi_n \in \mathrm{St}_\square[\mathrm{Name}]$. Consider $\sigma_r \in \mathrm{St}[\mathrm{Var} \setminus (\mathrm{dom}(\sigma_p) \cup \mathrm{dom}(\xi_n))]$ such that $\mathit{used}(c, \sigma_p \oplus \xi_n \oplus \sigma_r, \xi_n)$. We want to show that $\mathit{prs}_\square(c)(\sigma_p, \xi_n)$, $\mathit{vals}_\square(c)(\sigma_p, \xi_n)$, and $\mathit{pvars}_\square(c)(\sigma_p, \xi_n)$ do not depend on the choice of σ_r. To do so, consider $\sigma_r' \in \mathrm{St}[\mathrm{Var} \setminus (\mathrm{dom}(\sigma_p) \cup \mathrm{dom}(\xi_n))]$ such that $\mathit{used}(c, \sigma_p \oplus \xi_n \oplus \sigma_r', \xi_n)$. Let $\sigma \triangleq \sigma_p \oplus \xi_n \oplus \sigma_r$

and $\sigma' \triangleq \sigma_p \oplus \xi_n \oplus \sigma'_r$. Then, it suffices to show that

$$\begin{aligned} [\![c]\!]\sigma(like) \cdot \textstyle\prod_{\mu\in\mathrm{dom}(\xi_n)} [\![c]\!]\sigma(pr_\mu) &= [\![c]\!]\sigma'(like) \cdot \textstyle\prod_{\mu\in\mathrm{dom}(\xi_n)} [\![c]\!]\sigma'(pr_\mu), \\ [\![c]\!]\sigma(val_\mu) &= [\![c]\!]\sigma'(val_\mu) \quad \text{for all } \mu \in \mathrm{dom}(\xi_n), \\ [\![c]\!]\sigma(x) &= [\![c]\!]\sigma'(x) \quad \text{for all } x \in \mathrm{PVar}. \end{aligned} \tag{16}$$

Since $used(c, \sigma, \xi_n)$ and $used(c, \sigma', \xi_n)$, we have $\sigma(like) = 1 = \sigma'(like)$ and so $\sigma|_V = \sigma'|_V$ for $V = \mathrm{PVar} \cup \mathrm{dom}(\xi_n) \cup \{like\}$. Using this and $used(c, \sigma, \xi_n)$, we can apply Lemma C.6-(3) and -(4) to get $[\![c]\!]\sigma(v) = [\![c]\!]\sigma'(v)$ for all $v \in \mathrm{PVar} \cup \{like\} \cup \{pr_\mu, val_\mu \mid \mu \in \mathrm{dom}(\xi_n)\}$. Hence, we obtain the desired equations in Eq. (16). □

Proof of Lemma C.2. Let c be a command, $h : \mathrm{St}[\mathsf{Name}] \to \mathbb{R}$ be a measurable function, $f_* : \mathrm{St}[\mathsf{Name}] \to \mathrm{St}[\mathsf{AVar}]$ be the function defined in the statement of this lemma, and $\sigma_p \in \mathrm{St}[\mathrm{PVar}]$.

We first prove that the following equations hold for any measurable $h' : \mathrm{St}[\mathsf{Name}] \to \mathbb{R}$:

$$\begin{aligned}
&\int d\sigma_n \Big(\mathbf{1}_{[noerr(c,\sigma_p\oplus\sigma_n\oplus f_*(\sigma_n))]} \cdot h'(\sigma_n) \Big) \\
&= \int d\sigma_n \Big(\mathbf{1}_{[noerr(c,\sigma_p\oplus\sigma_n\oplus f_*(\sigma_n))]} \cdot h'(\sigma_n) \cdot \sum_{K\subseteq\mathsf{Name}} \mathbf{1}_{[used(c,\sigma_p\oplus\sigma_n\oplus f_*(\sigma_n),\sigma_n|_K)]} \Big) \\
&= \sum_{K\subseteq\mathsf{Name}} \int d\sigma_n \Big(\mathbf{1}_{[used(c,\sigma_p\oplus\sigma_n\oplus f_*(\sigma_n),\sigma_n|_K)]} \cdot \mathbf{1}_{[noerr(c,\sigma_p\oplus\sigma_n\oplus f_*(\sigma_n))]} \cdot h'(\sigma_n) \Big) \\
&= \sum_{K\subseteq\mathsf{Name}} \int_{[K\to\mathbb{R}]} d\xi_n \int_{[\mathsf{Name}\setminus K\to\mathbb{R}]} d\xi'_n \Big(\mathbf{1}_{[used(c,\sigma_p\oplus(\xi_n\oplus\xi'_n)\oplus f_*(\xi_n\oplus\xi'_n),\xi_n)]} \\
&\qquad \cdot \mathbf{1}_{[noerr(c,\sigma_p\oplus(\xi_n\oplus\xi'_n)\oplus f_*(\xi_n\oplus\xi'_n))]} \cdot h'(\xi_n \oplus \xi'_n) \Big) \\
&= \sum_{K\subseteq\mathsf{Name}} \int_{[K\to\mathbb{R}]} d\xi_n \Big(\sum_{L\subseteq\mathsf{Name}} \int_{[L\to\mathbb{R}]} d\xi'_n \Big(\mathbf{1}_{[\mathrm{dom}(\xi_n)\uplus\mathrm{dom}(\xi'_n)=\mathsf{Name}]} \cdot \mathbf{1}_{[used(c,\sigma_p\oplus(\xi_n\oplus\xi'_n)\oplus f_*(\xi_n\oplus\xi'_n),\xi_n)]} \\
&\qquad \cdot \mathbf{1}_{[noerr(c,\sigma_p\oplus(\xi_n\oplus\xi'_n)\oplus f_*(\xi_n\oplus\xi'_n))]} \cdot h'(\xi_n \oplus \xi'_n) \Big) \Big) \\
&= \int d\xi_n \int d\xi'_n \Big(\mathbf{1}_{[\mathrm{dom}(\xi_n)\uplus\mathrm{dom}(\xi'_n)=\mathsf{Name}]} \cdot \mathbf{1}_{[used(c,\sigma_p\oplus(\xi_n\oplus\xi'_n)\oplus f_*(\xi_n\oplus\xi'_n),\xi_n)]} \\
&\qquad \cdot \mathbf{1}_{[noerr(c,\sigma_p\oplus(\xi_n\oplus\xi'_n)\oplus f_*(\xi_n\oplus\xi'_n))]} \cdot h'(\xi_n \oplus \xi'_n) \Big).
\end{aligned}$$

All of these equations mean that one side of the equation is defined if and only if the other side is defined, and when both sides are defined, they are the same. The first equality holds because $noerr(c, \sigma_p \oplus \sigma_n \oplus f_*(\sigma_n))$ implies that there exists a unique $K \subseteq \mathsf{Name}$ with $used(c, \sigma_p \oplus \sigma_n \oplus f_*(\sigma_n), \sigma_n|_K)$; here we use $f_*(\sigma_n)(like) = 1$. The second equality holds since Name is finite. The third equality holds because $\mathrm{St}[\mathsf{Name}]$ is isomorphic to $[K \to \mathbb{R}] \times [\mathsf{Name} \setminus K \to \mathbb{R}]$. The fourth equality holds since $\xi'_n \in [L \to \mathbb{R}]$ with $L \neq \mathsf{Name} \setminus K$ implies $\mathbf{1}_{[\mathrm{dom}(\xi_n)\uplus\mathrm{dom}(\xi'_n)=\mathsf{Name}]} = 0$. The fifth equality holds by the definition of $\mathrm{St}_\square[\mathsf{Name}]$ and its underlying measure.

Using this result, we obtain the desired equation:

$$\begin{aligned}
&\int d\sigma_n \Big(prs(c)(\sigma_p, \sigma_n, f_*(\sigma_n)) \cdot h\Big(vals(c)(\sigma_p, \sigma_n, f_*(\sigma_n)) \Big) \Big) \\
&= \int d\sigma_n \Big(\mathbf{1}_{[noerr(c,\sigma_p\oplus\sigma_n\oplus f_*(\sigma_n))]} \cdot prs(c)(\sigma_p, \sigma_n, f_*(\sigma_n)) \cdot h\Big(vals(c)(\sigma_p, \sigma_n, f_*(\sigma_n)) \Big) \Big) \\
&= \int d\xi_n \int d\xi'_n \Big(\mathbf{1}_{[\mathrm{dom}(\xi_n)\uplus\mathrm{dom}(\xi'_n)=\mathsf{Name}]} \cdot \mathbf{1}_{[used(c,\sigma_p\oplus(\xi_n\oplus\xi'_n)\oplus f_*(\xi_n\oplus\xi'_n),\xi_n)]}
\end{aligned}$$

$$
\begin{aligned}
&\cdot \mathbf{1}_{[noerr(c,\sigma_p\oplus(\xi_n\oplus\xi'_n)\oplus f_*(\xi_n\oplus\xi'_n))]} \\
&\cdot prs(c)(\sigma_p, \xi_n\oplus\xi'_n, f_*(\xi_n\oplus\xi'_n))\cdot h\Big(vals(c)(\sigma_p,\xi_n\oplus\xi'_n, f_*(\xi_n\oplus\xi'_n))\Big)\Big) \\
=\int d\xi_n \int d\xi'_n\Big(&\mathbf{1}_{[\mathrm{dom}(\xi_n)\uplus\mathrm{dom}(\xi'_n)=\mathsf{Name}]}\cdot \mathbf{1}_{[used(c,\sigma_p\oplus(\xi_n\oplus\xi'_n)\oplus f_*(\xi_n\oplus\xi'_n),\xi_n)]} \\
&\cdot prs(c)(\sigma_p, \xi_n\oplus\xi'_n, f_*(\xi_n\oplus\xi'_n))\cdot h\Big(vals(c)(\sigma_p,\xi_n\oplus\xi'_n, f_*(\xi_n\oplus\xi'_n))\Big)\Big) \\
=\int d\xi_n \int d\xi'_n\Big(&\mathbf{1}_{[\mathrm{dom}(\xi_n)\uplus\mathrm{dom}(\xi'_n)=\mathsf{Name}]}\cdot \mathbf{1}_{[used(c,\sigma_p\oplus(\xi_n\oplus\xi'_n)\oplus f_*(\xi_n\oplus\xi'_n),\xi_n)]} \\
&\cdot prs_\square(c)(\sigma_p,\xi_n)\cdot\Big(\prod_{\mu\in\mathrm{dom}(\xi'_n)} f_1(\xi'_n(\mu))\Big)\cdot h\Big(vals_\square(c)(\sigma_p,\xi_n)\oplus\Big(\lambda\mu\in\mathrm{dom}(\xi'_n).\, f_2(\xi'_n(\mu))\Big)\Big)\Big) \\
=\int d\xi_n \int d\xi'_n\Big(&\mathbf{1}_{[\mathrm{dom}(\xi_n)\uplus\mathrm{dom}(\xi'_n)=\mathsf{Name}]} \\
&\cdot prs_\square(c)(\sigma_p,\xi_n)\cdot\Big(\prod_{\mu\in\mathrm{dom}(\xi'_n)} f_1(\xi'_n(\mu))\Big)\cdot h\Big(vals_\square(c)(\sigma_p,\xi_n)\oplus\Big(\lambda\mu\in\mathrm{dom}(\xi'_n).\, f_2(\xi'_n(\mu))\Big)\Big)\Big) \\
=\int d\xi_n\Big(prs_\square(c)(\sigma_p,\xi_n)\cdot\int d\xi'_n\Big(&\mathbf{1}_{[\mathrm{dom}(vals_\square(c)(\sigma_p,\xi_n))\uplus\mathrm{dom}(\xi'_n)=\mathsf{Name}]}\cdot\Big(\prod_{\mu\in\mathrm{dom}(\xi'_n)} f_1(\xi'_n(\mu))\Big) \\
&\cdot h\Big(vals_\square(c)(\sigma_p,\xi_n)\oplus\Big(\lambda\mu\in\mathrm{dom}(\xi'_n).\, f_2(\xi'_n(\mu))\Big)\Big)\Big)\Big) \\
=\int d\xi_n\Big(prs_\square(c)(\sigma_p,\xi_n)\cdot g\Big(&vals_\square(c)(\sigma_p,\xi_n)\Big)\Big)
\end{aligned}
$$

where $g : \mathrm{St}_\square[\mathsf{Name}] \to \mathbb{R}$ is defined as in the statement of this lemma, and each equation again means that one side of it is defined if and only if the other side is defined, and when both sides are defined, they are the same. The first and third equalities hold because $prs(c)(\sigma_p,\sigma_n,f_*(\sigma_n)) \neq 0$ implies $\mathbf{1}_{[noerr(c,\sigma_p\oplus\sigma_n\oplus f_*(\sigma_n))]} = 1$. The second equality uses the equation that we have shown in the previous paragraph. The fourth equality holds because of the following reason: if

$$\mathbf{1}_{[\mathrm{dom}(\xi_n)\uplus\mathrm{dom}(\xi'_n)=\mathsf{Name}]}\cdot\mathbf{1}_{[used(c,\sigma_p\oplus\xi_n\oplus\sigma_r,\xi_n)]} = 1 \quad \text{for } \sigma_r \triangleq \xi'_n\oplus f_*(\xi_n\oplus\xi'_n),$$

then

$$
\begin{aligned}
prs(c)(\sigma_p,\xi_n\oplus\xi'_n,f_*(\xi_n\oplus\xi'_n)) &= [\![c]\!](\sigma_p\oplus\xi_n\oplus\sigma_r)(like)\cdot\textstyle\prod_{\mu\in\mathsf{Name}}[\![c]\!](\sigma_p\oplus\xi_n\oplus\sigma_r)(pr_\mu) \\
&= [\![c]\!](\sigma_p\oplus\xi_n\oplus\sigma_r)(like)\cdot\textstyle\prod_{\mu\in\mathrm{dom}(\xi_n)}[\![c]\!](\sigma_p\oplus\xi_n\oplus\sigma_r)(pr_\mu) \\
&\quad\cdot\textstyle\prod_{\mu\in\mathsf{Name}\setminus\mathrm{dom}(\xi_n)}(\sigma_p\oplus\xi_n\oplus\sigma_r)(pr_\mu) \\
&= prs_\square(c)(\sigma_p,\xi_n)\cdot\textstyle\prod_{\mu\in\mathsf{Name}\setminus\mathrm{dom}(\xi_n)}(\sigma_p\oplus\xi_n\oplus\sigma_r)(pr_\mu) \\
&= prs_\square(c)(\sigma_p,\xi_n)\cdot\textstyle\prod_{\mu\in\mathrm{dom}(\xi'_n)} f_1(\xi'_n(\mu))
\end{aligned}
$$

and

$$
\begin{aligned}
vals(c)(\sigma_p,\xi_n\oplus\xi'_n,f_*(\xi_n\oplus\xi'_n)) &= \lambda\mu\in\mathsf{Name}.\,[\![c]\!](\sigma_p\oplus\xi_n\oplus\sigma_r)(val_\mu) \\
&= \big(\lambda\mu\in\mathrm{dom}(\xi_n).\,[\![c]\!](\sigma_p\oplus\xi_n\oplus\sigma_r)(val_\mu)\big) \\
&\quad\oplus\big(\lambda\mu\in\mathsf{Name}\setminus\mathrm{dom}(\xi_n).\,(\sigma_p\oplus\xi_n\oplus\sigma_r)(val_\mu)\big) \\
&= vals_\square(c)(\sigma_p,\xi_n)\oplus\big(\lambda\mu\in\mathsf{Name}\setminus\mathrm{dom}(\xi'_n).\,(\sigma_p\oplus\xi_n\oplus\sigma_r)(val_\mu)\big) \\
&= vals_\square(c)(\sigma_p,\xi_n)\oplus\big(\lambda\mu\in\mathrm{dom}(\xi'_n).\,f_2(\xi'_n(\mu))\big).
\end{aligned}
$$

These equalities for $prs(c)$ and $vals(c)$ themselves hold for the below reasons:

- The first equalities hold by $used(c, \sigma_p \oplus \xi_n \oplus \sigma_r, \xi_n)$ and the definitions of $prs(c)$ and $vals(c)$.
- The second equalities hold by Lemma C.6-(2), which is applicable since $used(c, \sigma_p \oplus \xi_n \oplus \sigma_r, \xi_n)$.
- The third equalities hold by $used(c, \sigma_p \oplus \xi_n \oplus \sigma_r, \xi_n)$ and the definitions of $prs_\Box(c)$ and $vals_\Box(c)$.
- The fourth equalities hold by $\mathrm{dom}(\xi_n) \uplus \mathrm{dom}(\xi'_n) = \mathsf{Name}$ and the definition of f_*.

Returning back to the main equations, we point out that the fifth equality comes from the next fact:

$$\left(\mathbf{1}_{[\mathrm{dom}(\xi_n) \uplus \mathrm{dom}(\xi'_n) = \mathsf{Name}]} \cdot prs_\Box(c)(\sigma_p, \xi_n)\right) \neq 0 \implies \mathbf{1}_{[used(c, \sigma_p \oplus (\xi_n \oplus \xi'_n) \oplus f_*(\xi_n \oplus \xi'_n), \xi_n)]} = 1.$$

The justification for this implication is given below:

- If the premise holds, then there exists σ_r such that $used(c, \sigma_p \oplus \xi_n \oplus \sigma_r, \xi_n)$. Since $\sigma_r(like) = 1 = f_*(-)(like)$, $\sigma_p \oplus \xi_n \oplus \sigma_r$ and $\sigma_p \oplus (\xi_n \oplus \xi'_n) \oplus f_*(\xi_n \oplus \xi'_n)$ coincide on $\mathsf{PVar} \cup \mathrm{dom}(\xi_n) \cup \{like\}$. Thus, Lemma C.7 gives the conclusion.

Again back to the main equations, we note that the sixth equality holds since

$$\mathrm{dom}(\xi_n) = \mathrm{dom}(vals_\Box(c)(\sigma_p, \xi_n)),$$

and the seventh equality follows from the definition of g. This completes the proof. □

Lemma C.4. *For all commands c and states $\sigma \in \mathsf{St}$ such that $[\![c]\!]\sigma \in \mathsf{St}$, we have*

$$[\![c]\!]\sigma(cnt_\mu) \geq \sigma(cnt_\mu) \quad \textit{and} \quad [\![c]\!]\sigma(\mu) = \sigma(\mu)$$

for all $\mu \in \mathsf{Name}$.

Proof. We prove the lemma by induction on the structure of c. Let $\sigma \in \mathsf{St}$ such that $[\![c]\!]\sigma \in \mathsf{St}$.

Cases $c \equiv \mathtt{skip}$, or $c \equiv (x := e)$, or $c \equiv \mathtt{obs}(d, r)$. In these cases, $[\![c]\!]\sigma(cnt_\mu) = \sigma(cnt_\mu)$ and $[\![c]\!]\sigma(\mu) = \mu$ for all μ. The claim of the lemma, thus, follows.

Case $c \equiv (x := \mathtt{sam}(n, d, \lambda y.e'))$. Let $\mu \triangleq [\![n]\!]\sigma$, $p \triangleq [\![d]\!]\sigma$, and $r \triangleq [\![e'[\mu/y]]\!]\sigma$. Then,

$$[\![c]\!]\sigma = \sigma[x \mapsto r, val_\mu \mapsto r, pr_\mu \mapsto p(r), cnt_\mu \mapsto \sigma(cnt_\mu) + 1].$$

Thus, the claim of the lemma follows.

Case $c \equiv (c'; c'')$. Pick $\mu \in \mathsf{Name}$. Then,

$$[\![c'; c'']\!]\sigma(\mu) = [\![c'']\!]([\![c']\!]\sigma)(\mu) = [\![c']\!]\sigma(\mu) = \sigma(\mu).$$

Here the second and third equalities use induction hypothesis on c' and c'', respectively. Also,

$$\begin{aligned}[\![c'; c'']\!]\sigma(cnt_\mu) - \sigma(cnt_\mu) &= \left([\![c'']\!]([\![c']\!]\sigma)(cnt_\mu) - [\![c']\!]\sigma(cnt_\mu)\right) + \left([\![c']\!]\sigma(cnt_\mu) - \sigma(cnt_\mu)\right) \\ &\geq 0.\end{aligned}$$

The inequality here uses induction hypothesis on c' and c''.

Case $c \equiv (\mathtt{if}\ b\ \{c'\}\ \mathtt{else}\ \{c''\})$. Assume that $[\![b]\!]\sigma = \mathit{true}$. We will prove the claims of the lemma under this assumption. The other case of $[\![b]\!] = \mathit{false}$ can be proved similarly. Pick $\mu \in \mathsf{Name}$. Then, by induction hypothesis on c',

$$[\![c]\!]\sigma(\mu) = [\![c']\!]\sigma(\mu) = \sigma(\mu) \quad \text{and} \quad [\![c]\!]\sigma(cnt_\mu) = [\![']\!]\sigma(cnt_\mu) \geq \sigma(cnt_\mu).$$

Case $c \equiv (\texttt{while}\ b\ \{c'\})$**.** Let $\mathcal{T}$ be the following subset of $[\mathsf{St} \to \mathsf{St}_\perp]$:

$$f \in \mathcal{T} \iff \forall \sigma \in \mathsf{St}.\ \Big(f(\sigma) \neq \perp \implies \forall \mu \in \mathsf{Name}.\ f(\sigma)(\mu) = \sigma(\mu) \wedge f(\sigma)(cnt_\mu) \geq \sigma(cnt_\mu)\Big).$$

Let F be the operator on $[\mathsf{St} \to \mathsf{St}_\perp]$ whose least fixed point becomes the semantics of the loop c. The desired conclusion follows if we show that $\mathcal{T}$ contains $\lambda\sigma.\perp$ and is closed under taking the limit of a chain in $\mathcal{T}$, and F preserves $\mathcal{T}$. The least element $\lambda\sigma.\perp$ belongs to $\mathcal{T}$ since there are no states σ with $(\lambda\sigma.\perp)(\sigma) \neq \perp$. Consider an increasing sequence $f_0, f_1, \ldots$ in $\mathcal{T}$, and let $f_\infty \triangleq \bigsqcup_{n\in\mathbb{N}} f_n$. Pick σ such that $f_\infty(\sigma) \neq \perp$. Then, $f_\infty(\sigma) = f_m(\sigma)$ for some $m \in \mathbb{N}$. Since $f_m \in \mathcal{T}$, we have

$$f_m(\sigma)(\mu) = \sigma(\mu) \quad \text{and} \quad f_m(\sigma)(cnt_\mu) \geq \sigma(cnt_\mu)$$

for all $\mu \in \mathsf{Name}$. Since $f_m(\sigma) = f_\infty(\sigma)$, we also have, for every $\mu \in \mathsf{Name}$, $f_\infty(\sigma)(\mu) = \sigma(\mu)$ and $f_\infty(\sigma)(cnt_\mu) \geq \sigma(cnt_\mu)$, as desired. It remains to show that $F(f) \in \mathcal{T}$ for all $f \in \mathcal{T}$. Pick $f \in \mathcal{T}$ and $\sigma \in \mathsf{St}$ such that $F(f)(\sigma) \in \mathsf{St}$. If $[\![b]\!]\sigma = \mathit{false}$, we have $F(f)(\sigma) = \sigma$, and the claims of the lemma follow. Otherwise, $F(f)(\sigma) = f([\![c']\!]\sigma)$. Pick $\mu \in \mathsf{Name}$. Then, by induction hypothesis on c' and the membership $f \in \mathcal{T}$,

$$F(f)(\sigma)(\mu) = f([\![c']\!]\sigma)(\mu) = [\![c']\!]\sigma(\mu) = \sigma(\mu),$$

and

$$F(f)(\sigma)(cnt_\mu) = f([\![c']\!]\sigma)(cnt_\mu) \geq [\![c']\!]\sigma(cnt_\mu) \geq \sigma(cnt_\mu).$$

We have just shown that $F(f) \in \mathcal{T}$, as desired. □

Definition C.5. Define $used_-$ as the predicate *used* but without the condition that *like* should be 1. That is, for all commands c, states $\sigma \in \mathsf{St}$, and $\xi_n \in \mathsf{St}_\square[\mathsf{Name}]$,

$$\begin{aligned} used_-(c, \sigma, \xi_n) \iff [\![c]\!]\sigma \in \mathsf{St} \wedge \big([\![c]\!]\sigma(cnt_\mu) - \sigma(cnt_\mu) \leq 1 \text{ for all } \mu \in \mathsf{Name}\big) \\ \wedge\ \xi_n = \sigma|_{\mathrm{dom}(\xi_n)} \\ \wedge\ \mathrm{dom}(\xi_n) = \{\mu \in \mathsf{Name} \mid [\![c]\!]\sigma(cnt_\mu) - \sigma(cnt_\mu) = 1\}. \end{aligned}$$

Lemma C.6. *Let c be a command, $\sigma_0, \sigma_1 \in \mathsf{St}$, and $\xi_n \in \mathsf{St}_\square[\mathsf{Name}]$. Suppose that $used_-(c, \sigma_0, \xi_n)$ and $\sigma_1|_V = \sigma_0|_V$ for $V \triangleq \mathsf{PVar} \cup \mathrm{dom}(\xi_n)$. Then, the following properties hold:*

(1) $[\![c]\!]\sigma_1 \in \mathsf{St}$.
(2) $[\![c]\!]\sigma_1(a) = \sigma_1(a)$ *for all* $a \in \{pr_\mu, val_\mu, cnt_\mu \mid \mu \in \mathsf{Name} \setminus \mathrm{dom}(\xi_n)\}$.
(3) $[\![c]\!]\sigma_1(v) = [\![c]\!]\sigma_0(v)$ *for all* $v \in \mathsf{PVar} \cup \{pr_\mu, val_\mu \mid \mu \in \mathrm{dom}(\xi_n)\}$.
(4) $[\![c]\!]\sigma_1(\mathit{like}) = [\![c]\!]\sigma_0(\mathit{like})$, *if* $\sigma_0(\mathit{like}) = \sigma_1(\mathit{like})$.
(5) $[\![c]\!]\sigma_1(a) - \sigma_1(a) = [\![c]\!]\sigma_0(a) - \sigma_0(a)$ *for all* $a \in \{cnt_\mu \mid \mu \in \mathsf{Name}\}$.

Proof. For $\sigma_0', \sigma_1' \in \mathsf{St}$ and $\xi_n' \in \mathsf{St}_\square[\mathsf{Name}]$, write

$$\sigma_0' \sim_{\xi_n'} \sigma_1'$$

to mean that $\sigma_0'|_V = \sigma_1'|_V$ for $V \triangleq \mathsf{PVar} \cup \mathrm{dom}(\xi_n')$. Note that using this notation, we can write the conditions of the lemma as follows:

$$used_-(c, \sigma_0, \xi_n) \wedge \sigma_0 \sim_{\xi_n} \sigma_1.$$

We will prove, by induction on the structure of c, that these conditions imply the five properties claimed by the lemma. Our proof will sometimes use a simple observation that the five properties claimed by the lemma and the relationship $\sigma_0 \sim_{\xi_n} \sigma_1$ imply $used_-(c, \sigma_1, \xi_n)$. One consequence of the observation is that if our lemma holds, its five properties also hold with σ_0 and σ_1 swapped. We will often use this consequence.

Case $c \equiv \texttt{skip}$**.** In this case, $[\![c]\!]\sigma_0 = \sigma_0$ and $[\![c]\!]\sigma_1 = \sigma_1$. From these equalities, the claimed properties (1), (2), (4) and (5) follow. For the remaining property (3), we note that $\mathrm{dom}(\xi_n) = \emptyset$ and the property, thus, follows from $\sigma_0 \sim_{\xi_n} \sigma_1$.

Case $c \equiv (x := e)$**.** In this case, $[\![c]\!]\sigma_0 = \sigma_0[x \mapsto [\![e]\!]\sigma_0]$ and $[\![c]\!]\sigma_1 = \sigma_1[x \mapsto [\![e]\!]\sigma_1]$. The results are not $\bot$, and they are identical to the pre-states σ_0 and σ_1 as far as auxiliary variables in AVar are concerned. Also, expressions in commands do not depend on variables other than program variables, so that $\sigma_0 \sim_{\xi_n} \sigma_1$ gives $[\![e]\!]\sigma_0 = [\![e]\!]\sigma_1$ and $[\![c]\!]\sigma_0(x) = [\![c]\!]\sigma_1(x)$ for all $x \in \mathrm{PVar}$. From all of these observations, the claimed properties (1)–(5) follow.

Case $c \equiv (x := \texttt{sam}(n, d, \lambda y.e'))$**.** Since $\sigma_0(x) = \sigma_1(x)$ for all $x \in \mathrm{PVar}$, we have $[\![n]\!]\sigma_0 = [\![n]\!]\sigma_1$ and $[\![d]\!]\sigma_0 = [\![d]\!]\sigma_1$. Let $\mu \triangleq [\![n]\!]\sigma_0$, $p \triangleq [\![d]\!]\sigma_0$, and $r \triangleq [\![e'[\mu/y]]\!]\sigma_0$. By the semantics of the sample commands, we have

$$[\![c]\!]\sigma_0 = \sigma_0[x \mapsto r, \mathit{val}_\mu \mapsto r, \mathit{pr}_\mu \mapsto p(\sigma_0(\mu)), \mathit{cnt}_\mu \mapsto \sigma_0(\mathit{cnt}_\mu) + 1].$$

Since $\mathit{used}_-(c, \sigma_0, \xi_n)$ holds, we have $\xi_n = \sigma_0|_{\{\mu\}}$, which in turn implies $\sigma_0(\mu) = \sigma_1(\mu)$ because $\sigma_0 \sim_{\xi_n} \sigma_1$. Thus, $[\![e'[\mu/y]]\!]\sigma_1 = [\![e'[\mu/y]]\!]\sigma_0 = r$, and

$$\begin{aligned}[\![c]\!]\sigma_1 &= \sigma_1[x \mapsto [\![e'[\mu/y]]\!]\sigma_1, \mathit{val}_\mu \mapsto [\![e'[\mu/y]]\!]\sigma_1, \mathit{pr}_\mu \mapsto p(\sigma_1(\mu)), \mathit{cnt}_\mu \mapsto \sigma_1(\mathit{cnt}_\mu) + 1] \\ &= \sigma_1[x \mapsto r, \mathit{val}_\mu \mapsto r, \mathit{pr}_\mu \mapsto p(\sigma_0(\mu)), \mathit{cnt}_\mu \mapsto \sigma_1(\mathit{cnt}_\mu) + 1].\end{aligned}$$

The RHS of the last equality implies that the five properties claimed by the lemma hold.

Case $c \equiv \texttt{obs}(d, r)$**.** We have $[\![d]\!]\sigma_0 = [\![d]\!]\sigma_1$ since $\sigma_0(x) = \sigma_1(x)$ for all $x \in \mathrm{PVar}$. Let $p \triangleq [\![d]\!]\sigma_0$. Then,

$$[\![c]\!]\sigma_0 = \sigma_0[\mathit{like} \mapsto \sigma_0(\mathit{like}) \cdot p(r)] \quad \text{and} \quad [\![c]\!]\sigma_1 = \sigma_1[\mathit{like} \mapsto \sigma_1(\mathit{like}) \cdot p(r)].$$

Also, $\mathrm{dom}(\xi_n) = \emptyset$ since $\mathit{used}_-(c, \sigma_0, \xi_n)$ holds. From what we have proved and also the agreement of σ_0 and σ_1 on program variables, the five properties claimed by the lemma follow.

Case $c \equiv (c'; c'')$**.** Since $[\![c]\!]\sigma_0 = [\![c'']\!]^\dagger([\![c']\!]\sigma_0) \in \mathrm{St}$, we have $[\![c']\!]\sigma_0 \in \mathrm{St}$. Let

$$\begin{aligned}\sigma_0' &\triangleq [\![c']\!]\sigma_0, \\ N_0 &\triangleq \{\mu \in \mathrm{Name} \mid [\![c'']\!]\sigma_0'(\mathit{cnt}_\mu) - \sigma_0(\mathit{cnt}_\mu) = 1\}, \\ N_0' &\triangleq \{\mu \in \mathrm{Name} \mid \sigma_0'(\mathit{cnt}_\mu) - \sigma_0(\mathit{cnt}_\mu) = 1\}.\end{aligned}$$

Then, $N_0 = \mathrm{dom}(\xi_n)$ because $\mathit{used}_-(c'; c'', \sigma_0, \xi_n)$ holds. We will prove the following facts:

(1) $N_0' \subseteq N_0$.
(2) Let $\xi_n' \triangleq \xi_n|_{N_0'}$, and $\xi_n'' \triangleq \xi_n|_{(N_0 \setminus N_0')}$. Then, $\mathit{used}_-(c', \sigma_0, \xi_n')$ and $\mathit{used}_-(c'', \sigma_0', \xi_n'')$ hold.
(3) $[\![c']\!]\sigma_1 \in \mathrm{St}$.
(4) Let $\sigma_1' \triangleq [\![c']\!]\sigma_1$. Then, $\sigma_0' \sim_{\xi_n''} \sigma_1'$.

These four facts imply the five properties claimed by the lemma. Here is the reason. Note that $\sigma_0 \sim_{\xi_n'} \sigma_1$ since $\mathrm{dom}(\xi_n') = N_0' \subseteq N_0 = \mathrm{dom}(\xi_n)$. This relationship between σ_0 and σ_1 and the second fact let us use induction hypothesis on $(c', \xi_n', \sigma_0, \sigma_1)$. Also, the second and fourth facts allow us to use induction hypothesis on $(c'', \xi_n'', \sigma_0', \sigma_1')$. We can derive the five properties from what we get from these two applications of induction hypothesis:

(1) By induction hypothesis on $(c'', \xi_n'', \sigma_0', \sigma_1')$, we have $[\![c'; c'']\!]\sigma_1 = [\![c'']\!]\sigma_1' \in \mathrm{St}$.
(2) For all $a \in \{\mathit{pr}_\mu, \mathit{val}_\mu, \mathit{cnt}_\mu \mid \mu \in \mathrm{Name} \setminus \mathrm{dom}(\xi_n)\}$,

$$[\![c'; c'']\!]\sigma_1(a) = [\![c'']\!]\sigma_1'(a) = \sigma_1'(a) = [\![c']\!]\sigma_1(a) = \sigma_1(a).$$

The second equality comes from induction hypothesis on $(c'', \xi_n'', \sigma_0', \sigma_1')$ and $\mathrm{dom}(\xi_n'') \subseteq \mathrm{dom}(\xi_n)$, and the fourth equality from induction hypothesis on $(c', \xi_n', \sigma_0, \sigma_1)$ and $\mathrm{dom}(\xi_n') \subseteq \mathrm{dom}(\xi_n)$.

(3) For all $v \in \mathsf{PVar} \cup \{pr_\mu, val_\mu \mid \mu \in \mathrm{dom}(\xi_n'')\}$, by induction hypothesis on $(c'', \xi_n'', \sigma_0', \sigma_1')$,

$$[\![c'; c'']\!]\sigma_1(v) = [\![c'']\!]\sigma_1'(v) = [\![c'']\!]\sigma_0'(v) = [\![c'; c'']\!]\sigma_0(v).$$

Also, for all $a \in \{pr_\mu, val_\mu \mid \mu \in \mathrm{dom}(\xi_n')\}$, we have $a \in \{pr_\mu, val_\mu \mid \mu \in \mathsf{Name} \setminus \mathrm{dom}(\xi_n'')\}$, and we can calculate:

$$\begin{aligned}[\![c'; c'']\!]\sigma_1(a) &= [\![c'']\!]\sigma_1'(a) = \sigma_1'(a)\\ &= [\![c']\!]\sigma_1(a) = [\![c']\!]\sigma_0(a)\\ &= \sigma_0'(a) = [\![c'']\!]\sigma_0'(a) = [\![c'; c'']\!]\sigma_0(a).\end{aligned}$$

The second equality uses induction hypothesis on $(c'', \xi_n'', \sigma_0', \sigma_1')$, and the fourth comes from the induction hypothesis on $(c', \xi_n', \sigma_0, \sigma_1)$. The sixth equality follows from induction hypothesis applied to $(c'', \xi_n'', \sigma_0', \sigma_1')$ and again to the same tuple but with σ_0' and σ_1' swapped.

(4) If $\sigma_0(like) = \sigma_1(like)$, by induction hypothesis on $(c', \xi_n', \sigma_0, \sigma_1)$,

$$\sigma_0'(like) = [\![c']\!]\sigma_0(like) = [\![c']\!]\sigma_1(like) = \sigma_1'(like),$$

which in turn implies, by induction hypothesis on $(c'', \xi_n'', \sigma_0', \sigma_1')$,

$$[\![c'; c'']\!]\sigma_0(like) = [\![c'']\!]\sigma_0'(like) = [\![c'']\!]\sigma_1'(like) = [\![c'; c'']\!]\sigma_1(like).$$

(5) For all $a \in \{cnt_\mu \mid \mu \in \mathsf{Name}\}$,

$$\begin{aligned}[\![c'; c'']\!]\sigma_1(a) - \sigma_1(a) &= [\![c'; c'']\!]\sigma_1(a) - [\![c']\!]\sigma_1(a) + [\![c']\!]\sigma_1(a) - \sigma_1(a)\\ &= [\![c'']\!]\sigma_1'(a) - \sigma_1'(a) + [\![c']\!]\sigma_1(a) - \sigma_1(a)\\ &= [\![c'']\!]\sigma_0'(a) - \sigma_0'(a) + [\![c']\!]\sigma_0(a) - \sigma_0(a)\\ &= [\![c'; c'']\!]\sigma_0(a) - \sigma_0(a).\end{aligned}$$

The only non-trivial inequality is the third one, and it follows from induction hypothesis on $(c', \xi_n', \sigma_0, \sigma_1)$ and $(c'', \xi_n'', \sigma_0', \sigma_1')$.

We prove the four facts as follows:

(1) Let $\mu \in N_0'$. Since $used_-(c'; c'', \sigma_0, \xi_n)$, we have

$$[\![c'']\!]\sigma_0'(\mu) - \sigma_0(\mu) = [\![c'; c'']\!]\sigma_0 - \sigma_0(\mu) \leq 1.$$

Also, by Lemma C.4 and the definition of N_0',

$$[\![c'']\!]\sigma_0'(\mu) - \sigma_0(\mu) \geq \sigma_0'(\mu) - \sigma_0(\mu) = 1.$$

Thus, $[\![c'']\!]\sigma_0'(\mu) - \sigma_0(\mu) = 1$, which implies that $\mu \in N_0$, as desired.

(2) We should show that $used_-(c', \sigma_0, \xi_n')$ and $used_-(c'', \sigma_0', \xi_n'')$ hold. The conjuncts in the definition of $used_-(c', \sigma_0, \xi_n')$ except the second follow immediately from $used_-(c'; c'', \sigma_0, \xi_n)$ and the definition of ξ_n'. For the remaining second conjunct, we use Lemma C.4 and $used_-(c'; c'', \sigma_0, \xi_n)$, and prove the conjunct as shown below: for all $\mu \in \mathsf{Name}$,

$$[\![c']\!]\sigma_0(cnt_\mu) - \sigma_0(cnt_\mu) \leq [\![c'']\!]([\![c']\!]\sigma_0)(cnt_\mu) - \sigma_0(cnt_\mu) \leq 1.$$

For $used_-(c'', \sigma_0', \xi_n'')$, we first note that the first and third conjuncts in its definition are direct consequences of $used_-(c'; c'', \sigma_0, \xi_n)$ and the definition of ξ_n''. We prove the second conjunct in the definition as follows: for all $\mu \in \mathsf{Name}$,

$$[\![c'']\!]\sigma_0'(cnt_\mu) - \sigma_0'(cnt_\mu) = [\![c'; c'']\!]\sigma_0(cnt_\mu) - [\![c']\!]cnt_0(cnt_\mu)$$

$$\leq [\![c'; c'']\!]\sigma_0(cnt_\mu) - cnt_0(cnt_\mu)$$
$$\leq 1.$$

The first inequality uses Lemma C.4, and the second comes from $used_-(c'; c'', \sigma_0, \xi_n)$. It remains to show the fourth conjunct in the definition of $used_-(c'', \sigma_0', \xi_n'')$, which we do below: for all $\mu \in \mathsf{Name}$,

$$\begin{aligned}
&[\![c'']\!]\sigma_0'(cnt_\mu) - \sigma_0'(cnt_\mu) = 1 \\
&\quad\iff [\![c'']\!]\sigma_0'(cnt_\mu) - \sigma_0'(cnt_\mu) = 1 \wedge \sigma_0'(cnt_\mu) - \sigma_0(cnt_\mu) = 0 \\
&\quad\iff \mu \in N_0 \wedge \mu \notin N_0' \\
&\quad\iff \mu \in \mathrm{dom}(\xi_n'').
\end{aligned}$$

The first equivalence comes from Lemma C.4 and $[\![c'']\!]\sigma_0'(cnt_\mu) - \sigma_0(cnt_\mu) \leq 1$, which holds because of $used_-(c'; c'', \sigma_0, \xi_n)$. The second equivalence follows from the definitions of N_0 and N_0'.

(3) Since $\sigma_0 \sim_{\xi_n} \sigma_1$ implies $\sigma_0 \sim_{\xi_n'} \sigma_1$ and we have $used_-(c', \sigma_0, \xi_n')$, we can apply induction hypothesis to $(c', \xi_n', \sigma_0, \sigma_1)$, and get $[\![c']\!]\sigma_1 \in \mathsf{St}$.

(4) We continue our reasoning in the previous item, and derive from induction hypothesis on $(c', \xi_n', \sigma_0, \sigma_1)$ the fact that for all $x \in \mathsf{PVar}$,

$$\sigma_0'(x) = [\![c']\!]\sigma_0(x) = [\![c']\!]\sigma_1(x) = \sigma_1'(x).$$

Also, for all $\mu \in \mathrm{dom}(\xi_n'')$,

$$\sigma_0'(\mu) = \sigma_0(\mu) = \sigma_1(\mu) = \sigma_1'(\mu),$$

where the first and third equalities come from Lemma C.4, and the second equality follows from the assumption that $\sigma_0 \sim_{\xi_n} \sigma_1$.

Case $c \equiv (\texttt{if}\ b\ \{c'\}\ \texttt{else}\ c'')$**.** Assume that $[\![b]\!]\sigma_0 = \mathit{true}$. Then, $[\![c]\!]\sigma_0 = [\![c']\!]\sigma_0$. We prove the five properties claimed by the lemma under this assumption. The proof for the other possibility, namely, $[\![b]\!]\sigma_0 = \mathit{false}$ is similar. Since $\sigma_0 \sim_{\xi_n} \sigma_1$, the states σ_0 and σ_1 coincide for the values of program variables. Thus, $[\![b]\!]\sigma_1 = \mathit{true}$, and $[\![c]\!]\sigma_1 = [\![c']\!]\sigma_1$. Since $[\![c]\!]\sigma_0 = [\![c']\!]\sigma_0$ as well, it suffices to show the five properties claimed by the lemma for $(c', \sigma_0, \sigma_1, \xi_n)$. This sufficient condition follows from induction hypothesis on $(c', \sigma_0, \sigma_1, \xi_n)$, since $used_-(c, \sigma_0, \xi_n)$ and $[\![b]\!]\sigma_0 = \mathit{true}$ imply $used_-(c', \sigma_0, \xi_n)$.

Case $c \equiv (\texttt{while}\ b\ \{c'\})$**.** Consider the version of $used_-$ where the first parameter can be a state transformer $f : \mathsf{St} \to \mathsf{St}_\perp$, instead of a command. Similarly, consider the version of the five properties claimed by the lemma where we use a state transformer $f : \mathsf{St} \to \mathsf{St}_\perp$, again instead of a command. We denote both versions by $used_-(f, \sigma_0'', \xi_n'')$ and $\varphi(f, \xi_n'', \sigma_0'', \sigma_1'')$. Let $\mathcal{T}$ be the subset of $\mathsf{St} \to \mathsf{St}_\perp$ defined by

$$f \in \mathcal{T} \iff \forall \sigma_0'', \sigma_1'' \in \mathsf{St}.\, \forall \xi_n'' \in \mathsf{St}_\square.\, \Big(\Big(used_-(f, \sigma_0'', \xi_n'') \wedge \sigma_0'' \sim_{\xi_n''} \sigma_1''\Big) \implies \varphi(f, \xi_n'', \sigma_0'', \sigma_1'')\Big),$$

and $F : [\mathsf{St} \to \mathsf{St}_\perp] \to [\mathsf{St} \to \mathsf{St}_\perp]$ be the following operator used in the semantics of the loop $[\![c]\!]$:

$$F(f)(\sigma) \triangleq \text{if } ([\![b]\!]\sigma = \mathit{true}) \text{ then } f^\dagger([\![c']\!]\sigma) \text{ else } \sigma.$$

We will show that $\mathcal{T}$ contains $\lambda\sigma.\, \perp$ and is closed under taking the least upper bound of an increasing chain in $[\mathsf{St} \to \mathsf{St}_\perp]$, and the operator F preserves $\mathcal{T}$. These three conditions imply that $[\![c]\!]$ is in $\mathcal{T}$, which in turn gives the five properties claimed by the lemma.

The first condition holds simply because $used_-((\lambda\sigma.\perp), \sigma_0'', \xi_n'')$ is false for all σ_0'' and ξ_n''. To prove the closure under the least upper bound of a chain, consider an increasing sequence $f_0, f_1, \ldots$ in $\mathcal{T}$. Let $f_\infty \triangleq \bigsqcup_{n \in \mathbb{N}} f_n$. Consider $\sigma_0'', \sigma_1'' \in \mathsf{St}$ and $\xi_n'' \in \mathsf{St}_\square$ such that $\sigma_0'' \sim_{\xi_n''} \sigma_1''$ and $used_-(f_\infty, \sigma_0'', \xi_n'')$. We should show that $\varphi(f_\infty, \xi_n'', \sigma_0'', \sigma_1'')$ holds. By the definition of f_∞, there exists $m \in \mathbb{N}$ such that

$f_\infty(\sigma_0) = f_m(\sigma_0)$. Then, the assumption $used_\_(f_\infty, \sigma_0'', \xi_n'')$ implies $used_\_(f_m, \sigma_0'', \xi_n'')$. This in turn gives $\varphi(f_m, \xi_n'', \sigma_0'', \sigma_1'')$ because $f_m \in \mathcal{T}$. By what we have proved and the definition of f_∞, we have

$$f_m(\sigma_0'') = f_\infty(\sigma_0'') \in \mathsf{St} \quad \text{and} \quad f_m(\sigma_1'') = f_\infty(\sigma_1'') \in \mathsf{St}.$$

Thus, $\varphi(f_m, \xi_n'', \sigma_0'', \sigma_1'')$ entails $\varphi(f_\infty, \xi_n'', \sigma_0'', \sigma_1'')$, as desired. It remains to show that $F(f) \in \mathcal{T}$ for all $f \in \mathcal{T}$. Pick $f \in \mathcal{T}$. We first replay our proof for the sequential-composition case after viewing $f^\dagger \circ [\![c']\!]$ as the sequential composition of c' and f. This replay, then, gives the membership $f^\dagger \circ [\![c']\!] \in \mathcal{T}$. Next, we replay our proof for the if case on $F(f)$ after viewing $f^\dagger \circ [\![c']\!]$ as the true branch and $\lambda\sigma.\, \sigma = [\![\mathtt{skip}]\!]$ as the false branch. This replay implies the required $F(f) \in \mathcal{T}$. □

Lemma C.7. *Let c be a command, $\sigma, \sigma' \in \mathsf{St}$, and $\xi_n \in \mathsf{St}_\square[\mathsf{Name}]$.*

- *If $\sigma|_V = \sigma'|_V$ for $V \triangleq \mathsf{PVar} \cup \mathrm{dom}(\xi_n) \cup \{like\}$, then $used(c, \sigma, \xi_n)$ implies $used(c, \sigma', \xi_n)$.*
- *If $\sigma|_U = \sigma'|_U$ for $U \triangleq \mathsf{PVar} \cup \mathrm{dom}(\xi_n)$, then $used_\_(c, \sigma, \xi_n)$ implies $used_\_(c, \sigma', \xi_n)$.*

Proof. Assume the settings in the statement of this lemma. For the first claim, assume $used(c, \sigma, \xi_n)$. Then, by the definition of *used* and *noerr*,

$$[\![c]\!]\sigma \in \mathsf{St} \wedge \big(\forall \mu \in \mathsf{Name}.\, [\![c]\!]\sigma(cnt_\mu) - \sigma(cnt_\mu) \le 1\big) \wedge \big(\sigma(like) = 1\big)$$
$$\wedge \big(\xi_n = \sigma|_{\mathrm{dom}(\xi_n)}\big) \wedge \big(\mathrm{dom}(\xi_n) = \{\mu \in \mathsf{Name} \mid [\![c]\!]\sigma(cnt_\mu) - \sigma(cnt_\mu) = 1\}\big).$$

From this and Lemma C.6 (which is applicable since $used(c, \sigma, \xi_n)$ and $\sigma|_V = \sigma'|_V$), we obtain

$$[\![c]\!]\sigma' \in \mathsf{St} \wedge \big(\forall \mu \in \mathsf{Name}.\, [\![c]\!]\sigma'(cnt_\mu) - \sigma'(cnt_\mu) \le 1\big) \wedge \big(\sigma'(like) = 1\big)$$
$$\wedge \big(\xi_n = \sigma'|_{\mathrm{dom}(\xi_n)}\big) \wedge \big(\mathrm{dom}(\xi_n) = \{\mu \in \mathsf{Name} \mid [\![c]\!]\sigma'(cnt_\mu) - \sigma'(cnt_\mu) = 1\}\big).$$

Note that we have the first clause by Lemma C.6-(1), the second and fifth clauses by Lemma C.6–(5), and the third and fourth clauses by $\sigma|_V = \sigma'|_V$. Hence, $used(c, \sigma', \xi_n)$ holds. The proof of the second claim is exactly the same except that we apply Lemma C.6 to $\sigma|_U = \sigma'|_U$ to prove only the four clauses of *used* that exclude $\sigma'(like) = 1$. □

C.3 Proof of Lemma C.3

Proof of Lemma C.3. We prove this lemma by induction on the structure of c. Let $g : \mathsf{St}[\mathsf{PVar}] \times \mathsf{St}_\square[\mathsf{Name}] \to \mathbb{R}$ be a measurable function and $\sigma_p \in \mathsf{St}[\mathsf{PVar}]$. In this proof, each equation involving integrals means (otherwise noted) that one side of the equation is defined if and only if the other side is defined, and when both sides are defined, they are the same.

Case $c \equiv \mathtt{skip}$, $c \equiv (x := e)$, or $c \equiv \mathtt{obs}(d, r)$. In this case, $\overline{c}^\pi \equiv c$ so the desired equation holds.

Case $c \equiv (x := \mathtt{sam}(n, d, \lambda y.e))$. If $(n, d, \lambda y.e) \notin \mathrm{dom}(\pi)$, then $\overline{c}^\pi \equiv c$ and thus the desired equation holds. So assume that $\pi(n, d, \lambda y.e) = (d', \lambda y'.e')$ for some d' and $\lambda y'.e'$. Then, $\overline{c}^\pi \equiv (x := \mathtt{sam}(n, d', \lambda y'.e'))$.

First, by the validity of π, for all states $\sigma \in \mathsf{St}$ and measurable subsets $A \subseteq \mathbb{R}$,

$$\int \mathbf{1}_{[\![e[r/y]]\!]\sigma \in A]} \cdot [\![d]\!]\sigma(r)\, dr = \int \mathbf{1}_{[\![e'[r/y']]\!]\sigma \in A]} \cdot [\![d']\!]\sigma(r)\, dr,$$

where both sides are always defined. Using this and the monotone convergence theorem, we can show that for all measurable $f : \mathbb{R} \to \mathbb{R}$,

$$\int f([\![e[r/y]]\!]\sigma) \cdot [\![d]\!]\sigma(r)\, dr = \int f([\![e'[r/y']]\!]\sigma) \cdot [\![d']\!]\sigma(r)\, dr. \tag{17}$$

Next, choose any $\sigma_{r_0} \in \mathsf{St}[\mathsf{Var} \setminus \mathsf{PVar}]$. Since $fv(n) \subseteq \mathsf{PVar}$, there exists $\mu \in \mathsf{Name}$ such that

$$[\![n]\!](\sigma_p \oplus \sigma_r) = \mu \quad \text{for all } \sigma_r \in \mathsf{St}[\mathsf{Var} \setminus \mathsf{PVar}].$$

Using this and $fv(e), fv(d) \subseteq \mathsf{PVar}$, we obtain the following: for any $\sigma_r \in \mathsf{St}[\mathsf{Var} \setminus \mathsf{PVar}]$,

$$\begin{aligned}
[\![c]\!](\sigma_p \oplus \sigma_r)(like) &= 1, \\
[\![c]\!](\sigma_p \oplus \sigma_r)(pr_\mu) &= [\![d]\!](\sigma_p \oplus \sigma_r)(\sigma_r(\mu)) = [\![d]\!](\sigma_p \oplus \sigma_{r_0})(\sigma_r(\mu)), \\
[\![c]\!](\sigma_p \oplus \sigma_r)(val_\mu) &= [\![e[\sigma_r(\mu)/y]]\!](\sigma_p \oplus \sigma_r) = [\![e[\sigma_r(\mu)/y]]\!](\sigma_p \oplus \sigma_{r_0}), \\
[\![c]\!](\sigma_p \oplus \sigma_r)(cnt_\mu) &= 1, \qquad [\![c]\!](\sigma_p \oplus \sigma_r)(cnt_{\mu'}) = 0 \quad \text{for } \mu' \not\equiv \mu, \\
[\![c]\!](\sigma_p \oplus \sigma_r)|_{\mathsf{PVar}} &= \sigma_p[x \mapsto [\![e[\sigma_r(\mu)/y]]\!](\sigma_p \oplus \sigma_{r_0})].
\end{aligned}$$

This implies that for any $\xi_n \in \mathsf{St}_\square[\mathsf{Name}]$, if $prs_\square(c)(\sigma_p, \xi_n) \neq 0$, then

$$\begin{aligned}
\mathrm{dom}(\xi_n) &= \{\mu\}, \\
prs_\square(c)(\sigma_p, \xi_n) &= 1 \cdot [\![d]\!](\sigma_p \oplus \sigma_{r_0})(\xi_n(\mu)), \\
pvars_\square(c)(\sigma_p, \xi_n) &= \sigma_p[x \mapsto [\![e[\xi_n(\mu)/y]]\!](\sigma_p \oplus \sigma_{r_0})], \\
vals_\square(c)(\sigma_p, \xi_n) &= [\mu \mapsto [\![e[\xi_n(\mu)/y]]\!](\sigma_p \oplus \sigma_{r_0})].
\end{aligned}$$

Note that the same equations hold for $\overline{c}^\pi$, except that we replace d, e, and y in the RHS of the above equations by d', e', and y'. Using these, we obtain:

$$\begin{aligned}
&\int d\xi_n \Big(prs_\square(c)(\sigma_p, \xi_n) \cdot g\Big(pvars_\square(c)(\sigma_p, \xi_n), vals_\square(c)(\sigma_p, \xi_n) \Big) \Big) \\
&= \int_{[\{\mu\} \to \mathbb{R}]} d\xi_n \Big([\![d]\!](\sigma_p \oplus \sigma_{r_0})(\xi_n(\mu)) \cdot \widehat{g}\Big([\![e[\xi_n(\mu)/y]]\!](\sigma_p \oplus \sigma_{r_0}) \Big) \Big) \\
&= \int_{\mathbb{R}} dr \Big([\![d]\!](\sigma_p \oplus \sigma_{r_0})(r) \cdot \widehat{g}\Big([\![e[r/y]]\!](\sigma_p \oplus \sigma_{r_0}) \Big) \Big) \\
&= \int_{\mathbb{R}} dr \Big([\![d']\!](\sigma_p \oplus \sigma_{r_0})(r) \cdot \widehat{g}\Big([\![e'[r/y']]\!](\sigma_p \oplus \sigma_{r_0}) \Big) \Big) \\
&= \int_{[\{\mu\} \to \mathbb{R}]} d\xi_n \Big([\![d']\!](\sigma_p \oplus \sigma_{r_0})(\xi_n(\mu)) \cdot \widehat{g}\Big([\![e'[\xi_n(\mu)/y']]\!](\sigma_p \oplus \sigma_{r_0}) \Big) \Big) \\
&= \int d\xi_n \Big(prs_\square(\overline{c}^\pi)(\sigma_p, \xi_n) \cdot g\Big(pvars_\square(\overline{c}^\pi)(\sigma_p, \xi_n), vals_\square(\overline{c}^\pi)(\sigma_p, \xi_n) \Big) \Big)
\end{aligned}$$

where $\widehat{g} : \mathbb{R} \to \mathbb{R}$ is defined as $\widehat{g}(r) = g(\sigma_p[x \mapsto r], [\mu \mapsto r])$. Here the first and fifth equalities use the equations proven above, the second and fourth equalities use that $[\{\mu\} \to \mathbb{R}]$ is isomorphic to $\mathbb{R}$, and the third equality uses Eq. (17). This proves the desired equation.

Case $c \equiv (\texttt{if}\ b\ \{c'\}\ \texttt{else}\ c'')$**.** In this case, since $fv(b) \subseteq \mathsf{PVar}$ and $\overline{c}^\pi \equiv \texttt{if}\ b\ \{\overline{c'}^\pi\}\ \texttt{else}\ \overline{c''}^\pi$, we have only two subcases:

- For all $\sigma_r \in \mathsf{St}[\mathsf{Var} \setminus \mathsf{PVar}]$, $[\![c]\!](\sigma_p \oplus \sigma_r) = [\![c']\!](\sigma_p \oplus \sigma_r)$ and $[\![\overline{c}^\pi]\!](\sigma_p \oplus \sigma_r) = [\![\overline{c'}^\pi]\!](\sigma_p \oplus \sigma_r)$.
- For all $\sigma_r \in \mathsf{St}[\mathsf{Var} \setminus \mathsf{PVar}]$, $[\![c]\!](\sigma_p \oplus \sigma_r) = [\![c'']\!](\sigma_p \oplus \sigma_r)$ and $[\![\overline{c}^\pi]\!](\sigma_p \oplus \sigma_r) = [\![\overline{c''}^\pi]\!](\sigma_p \oplus \sigma_r)$.

If the first subcase holds, we have

$$\begin{aligned}
&\int d\xi_n \Big(prs_\square(c)(\sigma_p, \xi_n) \cdot g\Big(pvars_\square(c)(\sigma_p, \xi_n), vals_\square(c)(\sigma_p, \xi_n) \Big) \Big) \\
&= \int d\xi_n \Big(prs_\square(c')(\sigma_p, \xi_n) \cdot g\Big(pvars_\square(c')(\sigma_p, \xi_n), vals_\square(c')(\sigma_p, \xi_n) \Big) \Big) \\
&= \int d\xi_n \Big(prs_\square(\overline{c'}^\pi)(\sigma_p, \xi_n) \cdot g\Big(pvars_\square(\overline{c'}^\pi)(\sigma_p, \xi_n), vals_\square(\overline{c'}^\pi)(\sigma_p, \xi_n) \Big) \Big)
\end{aligned}$$

$$= \int d\xi_n \Big(prs_\square(\overline{c}^\pi)(\sigma_p, \xi_n) \cdot g\Big(pvars_\square(\overline{c}^\pi)(\sigma_p, \xi_n), vals_\square(\overline{c}^\pi)(\sigma_p, \xi_n) \Big) \Big)$$

where the second equality is by IH on c'. If the second subcase holds, we obtain a similar equation by IH on c''. Hence, the desired equation holds in all subcases.

Case $c \equiv (c'; c'')$. In this case, we obtain the following equation:

$$\begin{aligned}
&\int d\xi_n \Big(prs_\square(c';c'')(\sigma_p, \xi_n) \cdot g\Big(pvars_\square(c';c'')(\sigma_p, \xi_n), vals_\square(c';c'')(\sigma_p, \xi_n) \Big) \Big) \\
&= \int d\xi'_n \Big(prs_\square(c')(\sigma_p, \xi'_n) \cdot \int d\xi''_n \Big(prs_\square(c'')\big(pvars_\square(c')(\sigma_p, \xi'_n), \xi''_n\big) \cdot \mathbf{1}_{[\mathrm{dom}(\xi'_n) \cap \mathrm{dom}(\xi''_n) = \emptyset]} \\
&\qquad \cdot g\Big(pvars_\square(c'')\big(pvars_\square(c')(\sigma_p, \xi'_n), \xi''_n\big), vals_\square(c')(\sigma_p, \xi'_n) \oplus vals_\square(c'')\big(pvars_\square(c')(\sigma_p, \xi'_n), \xi''_n\big) \Big) \Big) \Big) \\
&= \int d\xi'_n \Big(prs_\square(c')(\sigma_p, \xi'_n) \cdot g'\Big(pvars_\square(c')(\sigma_p, \xi'_n), vals_\square(c')(\sigma_p, \xi'_n) \Big) \Big) \\
&\qquad \text{where } g'(\widehat{\sigma'_p}, \widehat{\xi'_n}) \triangleq \int d\xi''_n \Big(prs_\square(c'')(\widehat{\sigma'_p}, \xi''_n) \cdot \mathbf{1}_{[\mathrm{dom}(\widehat{\xi'_n}) \cap \mathrm{dom}(\xi''_n) = \emptyset]} \\
&\qquad\qquad \cdot g\Big(pvars_\square(c'')(\widehat{\sigma'_p}, \xi''_n), \widehat{\xi'_n} \oplus vals_\square(c'')(\widehat{\sigma'_p}, \xi''_n) \Big) \Big) \\
&= \int d\xi'_n \Big(prs_\square(\overline{c'}^\pi)(\sigma_p, \xi'_n) \cdot g'\Big(pvars_\square(\overline{c'}^\pi)(\sigma_p, \xi'_n), vals_\square(\overline{c'}^\pi)(\sigma_p, \xi'_n) \Big) \Big) \quad \cdots \quad (*)
\end{aligned}$$

where the first equality is from Lemma C.9, the second equality uses $\mathrm{dom}(\xi'_n) = \mathrm{dom}(vals_\square(c')(\sigma_p, \xi'_n))$, and the third equality is by IH on c'. We now analyse $g'(\widehat{\sigma'_p}, \widehat{\xi'_n})$ as follows:

$$\begin{aligned}
&g'(\widehat{\sigma'_p}, \widehat{\xi'_n}) \\
&= \int d\xi''_n \Big(prs_\square(c'')(\widehat{\sigma'_p}, \xi''_n) \cdot \mathbf{1}_{[\mathrm{dom}(\widehat{\xi'_n}) \cap \mathrm{dom}(\xi''_n) = \emptyset]} \cdot g\Big(pvars_\square(c'')(\widehat{\sigma'_p}, \xi''_n), \widehat{\xi'_n} \oplus vals_\square(c'')(\widehat{\sigma'_p}, \xi''_n) \Big) \Big) \\
&= \int d\xi''_n \Big(prs_\square(c'')(\widehat{\sigma'_p}, \xi''_n) \cdot g''\Big(pvars_\square(c'')(\widehat{\sigma'_p}, \xi''_n), vals_\square(c'')(\widehat{\sigma'_p}, \xi''_n) \Big) \Big) \\
&\qquad \text{where } g''(\widehat{\sigma''_p}, \widehat{\xi''_n}) \triangleq \mathbf{1}_{[\mathrm{dom}(\widehat{\xi'_n}) \cap \mathrm{dom}(\widehat{\xi''_n}) = \emptyset]} \cdot g\Big(\widehat{\sigma''_p}, \widehat{\xi'_n} \oplus \widehat{\xi''_n} \Big) \\
&= \int d\xi''_n \Big(prs_\square(\overline{c''}^\pi)(\widehat{\sigma'_p}, \xi''_n) \cdot g''\Big(pvars_\square(\overline{c''}^\pi)(\widehat{\sigma'_p}, \xi''_n), vals_\square(\overline{c''}^\pi)(\widehat{\sigma'_p}, \xi''_n) \Big) \Big) \\
&= \int d\xi''_n \Big(prs_\square(\overline{c''}^\pi)(\widehat{\sigma'_p}, \xi''_n) \cdot \mathbf{1}_{[\mathrm{dom}(\widehat{\xi'_n}) \cap \mathrm{dom}(\xi''_n) = \emptyset]} \cdot g\Big(pvars_\square(\overline{c''}^\pi)(\widehat{\sigma'_p}, \xi''_n), \widehat{\xi'_n} \oplus vals_\square(\overline{c''}^\pi)(\widehat{\sigma'_p}, \xi''_n) \Big) \Big)
\end{aligned}$$

where the second and fourth equalities use $\mathrm{dom}(\xi''_n) = \mathrm{dom}(vals_\square(c'')(\widehat{\sigma'_p}, \xi''_n))$, and the third equality is by IH on c''. Using this, we obtain the following equation for the main quantity $(*)$:

$$\begin{aligned}
(*) &= \int d\xi'_n \Big(prs_\square(\overline{c'}^\pi)(\sigma_p, \xi'_n) \cdot \int d\xi''_n \Big(prs_\square(\overline{c''}^\pi)\big(pvars_\square(\overline{c'}^\pi)(\sigma_p, \xi'_n), \xi''_n\big) \cdot \mathbf{1}_{[\mathrm{dom}(\xi'_n) \cap \mathrm{dom}(\xi''_n) = \emptyset]} \\
&\qquad \cdot g\Big(pvars_\square(\overline{c''}^\pi)\big(pvars_\square(\overline{c'}^\pi)(\sigma_p, \xi'_n), \xi''_n\big), \\
&\qquad\quad vals_\square(\overline{c'}^\pi)(\sigma_p, \xi'_n) \oplus vals_\square(\overline{c''}^\pi)\big(pvars_\square(\overline{c'}^\pi)(\sigma_p, \xi'_n), \xi''_n\big) \Big) \Big) \Big) \\
&= \int d\xi_n \Big(prs_\square(\overline{c'}^\pi; \overline{c''}^\pi)(\sigma_p, \xi_n) \cdot g\Big(pvars_\square(\overline{c'}^\pi; \overline{c''}^\pi)(\sigma_p, \xi_n), vals_\square(\overline{c'}^\pi; \overline{c''}^\pi)(\sigma_p, \xi_n) \Big) \Big)
\end{aligned}$$

where the first equality uses $\mathrm{dom}(\xi'_n) = \mathrm{dom}(vals_\square(c')(\sigma_p, \xi'_n))$, and the second equality is by Lemma C.9, as we did above. By $\overline{c'; c''}^\pi \equiv \overline{c'}^\pi; \overline{c''}^\pi$, we get the desired equation.

Case $c \equiv (\texttt{while}\ b\ \{c'\})$**.** In this case, $\overline{c}^{\pi} \equiv (\texttt{while}\ b\ \{\overline{c'}^{\pi}\})$. Without loss of generality, assume that g is a nonnegative function; we can prove the general case of g directly from the nonnegative case of g, by considering the nonnegative part and the negative part of g separately.

Consider the version of $prs_{\square}(-)$, $pvars_{\square}(-)$, and $vals_{\square}(-)$, where the parameter can be a state transformer $f : \mathrm{St} \to \mathrm{St}_{\bot}$, instead of a command. We denote the versions by $prs_{\square}(f)$, $pvars_{\square}(f)$, and $vals_{\square}(f)$. Define $\mathcal{T} \subseteq [\mathrm{St} \to \mathrm{St}_{\bot}]^2$ and $T : [\mathrm{St} \to \mathrm{St}_{\bot}]^2 \to [\mathrm{St} \to \mathrm{St}_{\bot}]^2$ by

$$(f, \overline{f}) \in \mathcal{T} \iff \int d\xi_n\, G_{g',\sigma'_p}(f)(\xi_n) = \int d\xi_n\, G_{g',\sigma'_p}(\overline{f})(\xi_n)$$
$$\text{for all measurable } g' : \mathrm{St}[\mathrm{PVar}] \times \mathrm{St}_{\square}[\mathrm{Name}] \to \mathbb{R}_{\geq 0} \text{ and } \sigma'_p \in \mathrm{St}[\mathrm{PVar}],$$
$$T(f, \overline{f}) \triangleq (F(f), \overline{F}(\overline{f})),$$

where $G_{g',\sigma'_p}(f) \in \mathrm{St}_{\square}[\mathrm{Name}] \to \mathbb{R}_{\geq 0}$ and $F, \overline{F} : [\mathrm{St} \to \mathrm{St}_{\bot}] \to [\mathrm{St} \to \mathrm{St}_{\bot}]$ are defined by

$$G_{g',\sigma'_p}(f)(\xi_n) \triangleq prs_{\square}(f)(\sigma'_p, \xi_n) \cdot g'\Big(pvars_{\square}(f)(\sigma'_p, \xi_n), vals_{\square}(f)(\sigma'_p, \xi_n)\Big),$$
$$F(f)(\sigma) \triangleq \text{if } (\llbracket b \rrbracket \sigma = \mathit{true}) \text{ then } (f^{\dagger} \circ \llbracket c' \rrbracket)(\sigma) \text{ else } \sigma,$$
$$\overline{F}(\overline{f})(\sigma) \triangleq \text{if } (\llbracket b \rrbracket \sigma = \mathit{true}) \text{ then } (\overline{f}^{\dagger} \circ \llbracket \overline{c'}^{\pi} \rrbracket)(\sigma) \text{ else } \sigma.$$

Note that F and $\overline{F}$ are the operators used in the semantics of the loops $\llbracket c \rrbracket$ and $\llbracket \overline{c}^{\pi} \rrbracket$, respectively. We will show that $\mathcal{T}$ contains $(\lambda\sigma.\, \bot, \lambda\sigma.\, \bot)$, the operator T preserves $\mathcal{T}$, and $\mathcal{T}$ is closed under taking the least upper bound of an increasing chain in $[\mathrm{St} \to \mathrm{St}_{\bot}]^2$, where the order on $[\mathrm{St} \to \mathrm{St}_{\bot}]^2$ is defined as: $(f_0, \overline{f_0}) \sqsubseteq (f_1, \overline{f_1}) \iff f_0 \sqsubseteq f_1 \wedge \overline{f_0} \sqsubseteq \overline{f_1}$. These three conditions imply $(\llbracket c \rrbracket, \llbracket \overline{c}^{\pi} \rrbracket) \in \mathcal{T}$, which in turn proves the desired equation:

$$\begin{aligned}
&\int d\xi_n \Big(prs_{\square}(c)(\sigma_p, \xi_n) \cdot g\Big(pvars_{\square}(c)(\sigma_p, \xi_n), vals_{\square}(c)(\sigma_p, \xi_n)\Big)\Big) \\
&= \int d\xi_n\, G_{g,\sigma_p}(\llbracket c \rrbracket)(\xi_n) \\
&= \int d\xi_n\, G_{g,\sigma_p}(\llbracket \overline{c}^{\pi} \rrbracket)(\xi_n) \\
&= \int d\xi_n \Big(prs_{\square}(\overline{c}^{\pi})(\sigma_p, \xi_n) \cdot g\Big(pvars_{\square}(\overline{c}^{\pi})(\sigma_p, \xi_n), vals_{\square}(\overline{c}^{\pi})(\sigma_p, \xi_n)\Big)\Big),
\end{aligned}$$

where the second equality follows from $(\llbracket c \rrbracket, \llbracket \overline{c}^{\pi} \rrbracket) \in \mathcal{T}$.

The first condition holds simply because $G_{g',\sigma'_p}(\lambda\sigma.\, \bot)(\xi_n) = 0$ for all g', σ'_p, and ξ_n. To show the second condition, pick $(f, \overline{f}) \in \mathcal{T}$. Our goal is to show $T(f, \overline{f}) \in \mathcal{T}$. We first replay our proof for the sequential-composition case on $(f^{\dagger} \circ \llbracket c' \rrbracket, \overline{f}^{\dagger} \circ \llbracket \overline{c'}^{\pi} \rrbracket)$, after viewing $f^{\dagger} \circ \llbracket c' \rrbracket$ and $\overline{f}^{\dagger} \circ \llbracket \overline{c'}^{\pi} \rrbracket$ as the sequential composition of c' and f, and of $\overline{c'}^{\pi}$ and $\overline{f}$, respectively. This replay, then, gives the membership $(f^{\dagger} \circ \llbracket c' \rrbracket, \overline{f}^{\dagger} \circ \llbracket \overline{c'}^{\pi} \rrbracket) \in \mathcal{T}$. Next, we replay our proof for the if case on $(F(f), \overline{F}(\overline{f}))$, after viewing $f^{\dagger} \circ \llbracket c' \rrbracket$ and $\overline{f}^{\dagger} \circ \llbracket \overline{c'}^{\pi} \rrbracket$ as the true branches, and $\lambda\sigma.\, \sigma = \llbracket \texttt{skip} \rrbracket$ as the false branch. This replay implies the required $T(f, \overline{f}) = (F(f), \overline{F}(\overline{f})) \in \mathcal{T}$.

To show the third condition, consider an increasing sequence $\{(f_k, \overline{f_k})\}_{k \in \mathbb{N}}$ in $\mathcal{T}$. Let $f_{\infty} \triangleq \bigsqcup_{k \in \mathbb{N}} f_k$ and $\overline{f}_{\infty} \triangleq \bigsqcup_{k \in \mathbb{N}} \overline{f_k}$. Consider a measurable $g' : \mathrm{St}[\mathrm{PVar}] \times \mathrm{St}_{\square}[\mathrm{Name}] \to \mathbb{R}_{\geq 0}$ and $\sigma'_p \in \mathrm{St}[\mathrm{PVar}]$. We should show that $\int d\xi_n\, G_{g',\sigma'_p}(f_{\infty})(\xi_n) = \int d\xi_n\, G_{g',\sigma'_p}(\overline{f_{\infty}})(\xi_n)$. Since $\{f_k\}_{k \in \mathbb{N}}$ is increasing, for any $\sigma \in \mathrm{St}$, $f_k(\sigma) \in \mathrm{St}$ implies that $f_{k'}(\sigma) = f_k(\sigma) \in \mathrm{St}$ for all $k' \geq k$. Hence, $\{G_{g',\sigma'_p}(f_k)\}_{k \in \mathbb{N}}$ is a pointwise increasing sequence: for all $\xi_n \in \mathrm{St}_{\square}[\mathrm{Name}]$,

$$0 \leq G_{g',\sigma'_p}(f_k)(\xi_n) \leq G_{g',\sigma'_p}(f_{k+1})(\xi_n) \quad \text{for all } k \in \mathbb{N}.$$

Also, by the definition of f_∞, for any $\sigma \in \mathrm{St}$, there exists $K \in \mathbb{N}$ such that $f_\infty(\sigma) = f_K(\sigma)$; thus, $G_{g',\sigma'_p}(f_\infty)$ is the pointwise limit of $\{G_{g',\sigma'_p}(f_k)\}_{k\in\mathbb{N}}$: for all $\xi_n \in \mathrm{St}_\square[\mathsf{Name}]$,

$$G_{g',\sigma'_p}(f_\infty)(\xi_n) = \lim_{k\to\infty} G_{g',\sigma'_p}(f_k)(\xi_n).$$

Note that the corresponding results hold for $\overline{f_\infty}$ and $\overline{f_k}$. Using these results, we finally obtain the following as desired:

$$\begin{aligned}\int d\xi_n\, G_{g',\sigma'_p}(f_\infty)(\xi_n) &= \lim_{k\to\infty} \int d\xi_n\, G_{g',\sigma'_p}(f_k)(\xi_n)\\ &= \lim_{k\to\infty} \int d\xi_n\, G_{g',\sigma'_p}(\overline{f_k})(\xi_n) = \int d\xi_n\, G_{g',\sigma'_p}(\overline{f_\infty})(\xi_n).\end{aligned}$$

The first and third equalities follow from the monotone convergence theorem, applied to the above results. The second equality holds since $(f_k, \overline{f_k}) \in \mathcal{T}$. This completes the proof of the while case. □

Lemma C.8. *Let c be a command, $\sigma_0, \sigma_1 \in \mathrm{St}$, and $r_0 \in \mathbb{R}$. Suppose that $\sigma_1(like) = \sigma_0(like) \cdot r_0$ and $\sigma_1|_V = \sigma_0|_V$ for $V \triangleq \mathrm{Var} \setminus \{like\}$. If $[\![c]\!]\sigma_0 \in \mathrm{St}$, then*

$$[\![c]\!]\sigma_1 \in \mathrm{St}, \qquad [\![c]\!]\sigma_1(like) = [\![c]\!]\sigma_0(like) \cdot r_0, \qquad ([\![c]\!]\sigma_1)|_V = ([\![c]\!]\sigma_0)|_V.$$

Proof. Let $V \triangleq \mathrm{Var} \setminus \{like\}$. Pick an arbitrary command c. We prove the lemma by induction on the structure of c. Let $\sigma_0, \sigma_1 \in \mathrm{St}$ and $r_0 \in \mathbb{R}$ such that $[\![c]\!]\sigma_0 \in \mathrm{St}$, $\sigma_1(like) = \sigma_0(like) \cdot r_0$, and $\sigma_1|_V = \sigma_0|_V$. We should show that $[\![c]\!]\sigma_1 \in \mathrm{St}$, $[\![c]\!]\sigma_1(like) = [\![c]\!]\sigma_0(like) \cdot r_0$, and $[\![c]\!]\sigma_1|_V = [\![c]\!]\sigma_0|_V$.

Case $c \equiv \mathtt{skip}$**.** In this case, what we need to prove is identical to the assumption on $(\sigma_0, \sigma_1, r_0)$.

Case $c \equiv (x := e)$**.** By the semantics of the assignments, we have $[\![c]\!]\sigma_1 \in \mathrm{St}$, and

$$[\![c]\!]\sigma_1(like) = \sigma_1(like) = \sigma_0(like) \cdot r_0 = [\![c]\!]\sigma_0(like) \cdot r_0.$$

The last requirement also holds since $[\![e]\!]\sigma_0 = [\![e]\!]\sigma_1$ and $\sigma_0|_V = \sigma_1|V$.

Case $c \equiv (x := \mathtt{sam}(n, d, \lambda y.e'))$**.** By the semantics of the sample commands, we have $[\![c]\!]\sigma_1 \in \mathrm{St}$. Also, the assignments do not change the value of *like*, so that $[\![c]\!]\sigma_1(like) = \sigma_1(like) = \sigma_0(like) \cdot r_0 = [\![c]\!]\sigma_0(like) \cdot r_0$. It remains to show that $[\![c]\!]\sigma_0|_V = [\![c]\!]\sigma_1|_V$. Since $\sigma_0|_V = \sigma_1|_V$, we have $[\![n]\!]\sigma_0 = [\![n]\!]\sigma_1$ and $[\![d]\!]\sigma_0 = [\![d]\!]\sigma_1$. Let $\mu \triangleq [\![n]\!]\sigma_0$. Then, by the same reason, $[\![e'[\mu/y]]\!]\sigma_0 = [\![e'[\mu/y]]\!]\sigma_1$. Let $f \triangleq [\![d]\!]\sigma_0$ and $r \triangleq [\![e'[\mu/y]]\!]\sigma_0$. We prove the required equality as follows:

$$\begin{aligned}[\![c]\!]\sigma_0|_V &= \sigma_0[x \mapsto r, val_\mu \mapsto r, pr_\mu \mapsto f(\sigma_0(\mu)), cnt_\mu \mapsto \sigma_0(cnt_\mu) + 1]|_V\\ &= \sigma_1[x \mapsto r, val_\mu \mapsto r, pr_\mu \mapsto f(\sigma_0(\mu)), cnt_\mu \mapsto \sigma_0(cnt_\mu) + 1]|_V\\ &= \sigma_1[x \mapsto r, val_\mu \mapsto r, pr_\mu \mapsto f(\sigma_1(\mu)), cnt_\mu \mapsto \sigma_1(cnt_\mu) + 1]|_V\\ &= [\![c]\!]\sigma_1|_V.\end{aligned}$$

Case $c \equiv (\mathtt{obs}(d, r))$**.** By the semantics of the observe commands, we have $[\![c]\!]\sigma_1 \in \mathrm{St}$. Also, the observe commands do not change any variable except *like*. So, $[\![c]\!]\sigma_0|_V = \sigma_0|V = \sigma_1|_V = [\![c]\!]\sigma_1|_V$. The remaining requirement for *like* can be proved as follows:

$$[\![c]\!]\sigma_1(like) = \sigma_1(like) \cdot [\![d]\!]\sigma_1(r) = \sigma_0(like) \cdot r_0 \cdot [\![d]\!]\sigma_1(r) = \sigma_0(like) \cdot r_0 \cdot [\![d]\!]\sigma_0(r) = [\![c]\!]\sigma_0(like) \cdot r_0.$$

Case $c \equiv (c'; c'')$**.** We have $[\![c']\!]\sigma_0 \in \mathrm{St}$ and $[\![c'']\!]([\![c']\!]\sigma_0) \in \mathrm{St}$. We apply induction hypothesis first to $(c', \sigma_0, \sigma_1, r_0)$, and then to $(c'', [\![c']\!]\sigma_0, [\![c']\!]\sigma_1, r_0)$. Then, we get the requirements of the lemma.

Case $c \equiv$ (`if` b $\{c'\}$ `else` $\{c''\}$)**.** We deal with the case that $[\![b]\!]\sigma_0 = \mathit{true}$. The other case of $[\![b]\!]\sigma_0 = \mathit{false}$ can be proved similarly. Since $[\![b]\!]\sigma_0 = \mathit{true}$, we have $[\![c']\!]\sigma_0 = [\![c]\!]\sigma_0 \in \mathsf{St}$. Thus, we can apply induction hypothesis to c'. If we do so, we get

$$[\![c']\!]\sigma_1 \in \mathsf{St}, \quad [\![c']\!]\sigma_1(\mathit{like}) = [\![c']\!]\sigma_0(\mathit{like}) \cdot r_0, \quad \text{and} \quad [\![c']\!]\sigma_1|_V = [\![c']\!]\sigma_0|V.$$

This gives the desired conclusion because $[\![b]\!]\sigma_1 = [\![b]\!]\sigma_0 = \mathit{true}$ and so $[\![c]\!]\sigma_1 = [\![c']\!]\sigma_1$, and $[\![c]\!]\sigma_0 = [\![c']\!]\sigma_0$.

Case $c \equiv$ (`while` b $\{c'\}$)**.** Let F be the operator on $[\mathsf{St} \to \mathsf{St}_\perp]$ such that $[\![c]\!]$ is the least fixed point of F. Define a subset $\mathcal{T}$ of $[\mathsf{St} \to \mathsf{St}_\perp]$ as follows: a function $f \in [\mathsf{St} \to \mathsf{St}_\perp]$ is in $\mathcal{T}$ if and only if for all $\sigma_0', \sigma_1' \in \mathsf{St}$ such that $\sigma_1'|_V = \sigma_0'|_V$ and $\sigma_1'(\mathit{like}) = \sigma_0'(\mathit{like}) \cdot r_0$, we have

$$f(\sigma_0') \neq \perp \implies \Big(f(\sigma_1') \neq \perp \wedge f(\sigma_1')|_V = f(\sigma_0')|_V \wedge f(\sigma_1')(\mathit{like}) = f(\sigma_0')(\mathit{like}) \cdot r_0\Big).$$

The set $\mathcal{T}$ contains the least function $\lambda\sigma.\perp$, and is closed under the least upper bound of any chain in $[\mathsf{St} \to \mathsf{St}_\perp]$. It is also closed under F. This F-closure follows essentially from our arguments for sequential composition, if command, and skip, and induction hypothesis on c'. What we have shown for $\mathcal{T}$ implies that $\mathcal{T}$ contains the least fixed point of F, which gives the desired property for c. □

Lemma C.9. *Let c', c'' be commands and $g : \mathsf{St}[\mathsf{PVar}] \times \mathsf{St}_\square[\mathsf{Name}] \to \mathbb{R}$ be a measurable function. Then, for any $\sigma_p \in \mathsf{St}[\mathsf{PVar}]$,*

$$\begin{aligned}
&\int d\xi_n \Big(\mathit{prs}_\square(c';c'')(\sigma_p,\xi_n) \cdot g\Big(\mathit{pvars}_\square(c';c'')(\sigma_p,\xi_n), \mathit{vals}_\square(c';c'')(\sigma_p,\xi_n)\Big)\Big) \\
&= \int d\xi_n' \Big(\mathit{prs}_\square(c')(\sigma_p,\xi_n') \cdot \int d\xi_n'' \Big(\mathit{prs}_\square(c'')(\mathit{pvars}_\square(c')(\sigma_p,\xi_n'),\xi_n'') \cdot \mathbf{1}_{[\mathrm{dom}(\xi_n')\cap\mathrm{dom}(\xi_n'')=\emptyset]} \\
&\qquad \cdot g\Big(\mathit{pvars}_\square(c'')(\mathit{pvars}_\square(c')(\sigma_p,\xi_n'),\xi_n''), \mathit{vals}_\square(c')(\sigma_p,\xi_n') \oplus \mathit{vals}_\square(c'')(\mathit{pvars}_\square(c')(\sigma_p,\xi_n'),\xi_n'')\Big)\Big)\Big).
\end{aligned}$$

Proof. Let c', c'' be commands, $g : \mathsf{St}[\mathsf{PVar}] \times \mathsf{St}_\square[\mathsf{Name}] \to \mathbb{R}$ a measurable function, and $\sigma_p \in \mathsf{St}[\mathsf{PVar}]$. In this proof, each equation involving integrals means (otherwise noted) that one side of the equation is defined if and only if the other side is defined, and when both sides are defined, they are the same.

First, to convert a single-integral on ξ_n to a double-integral on ξ_n' and ξ_n'' as in the desired equation, we show the following claim: for any measurable $f : \mathsf{St}_\square[\mathsf{Name}] \to \mathsf{St}$ and $h : \mathsf{St}_\square[\mathsf{Name}] \to \mathbb{R}$ such that $f(\xi_n)|_{\mathrm{dom}(\xi_n)} = \xi_n$ for all $\xi_n \in \mathsf{St}_\square[\mathsf{Name}]$, we have

$$\begin{aligned}
&\int d\xi_n \Big(\mathbf{1}_{[\mathit{used}(c';c'',f(\xi_n),\xi_n)]} \cdot h(\xi_n)\Big) \\
&= \sum_{K\subseteq\mathsf{Name}} \int_{[K\to\mathbb{R}]} d\xi_n \Big(\mathbf{1}_{[\mathit{used}(c';c'',f(\xi_n),\xi_n)]} \cdot h(\xi_n)\Big) \\
&= \sum_{K\subseteq\mathsf{Name}} \int_{[K\to\mathbb{R}]} d\xi_n \Big(\mathbf{1}_{[\mathit{used}(c';c'',f(\xi_n),\xi_n)]} \cdot h(\xi_n) \cdot \sum_{L\subseteq K} \mathbf{1}_{[\mathit{used}(c',f(\xi_n),\xi_n|_L)]}\Big) \\
&= \sum_{K\subseteq\mathsf{Name}} \sum_{L\subseteq K} \int_{[K\to\mathbb{R}]} d\xi_n \Big(\mathbf{1}_{[\mathit{used}(c',f(\xi_n),\xi_n|_L)]} \cdot \mathbf{1}_{[\mathit{used}(c';c'',f(\xi_n),\xi_n)]} \cdot h(\xi_n)\Big) \\
&= \sum_{K\subseteq\mathsf{Name}} \sum_{L\subseteq K} \int_{[L\to\mathbb{R}]} d\xi_n' \int_{[K\setminus L\to\mathbb{R}]} d\xi_n'' \Big(\mathbf{1}_{[\mathit{used}(c',f(\xi_n'\oplus\xi_n''),\xi_n')]} \cdot \mathbf{1}_{[\mathit{used}(c';c'',f(\xi_n'\oplus\xi_n''),\xi_n'\oplus\xi_n'')]} \cdot h(\xi_n'\oplus\xi_n'')\Big) \\
&= \sum_{L'\subseteq\mathsf{Name}} \sum_{\substack{M'\subseteq\mathsf{Name}\\ L'\cap M'=\emptyset}} \int_{[L'\to\mathbb{R}]} d\xi_n' \int_{[M'\to\mathbb{R}]} d\xi_n'' \Big(\mathbf{1}_{[\mathit{used}(c',f(\xi_n'\oplus\xi_n''),\xi_n')]} \cdot \mathbf{1}_{[\mathit{used}(c';c'',f(\xi_n'\oplus\xi_n''),\xi_n'\oplus\xi_n'')]} \cdot h(\xi_n'\oplus\xi_n'')\Big)
\end{aligned}$$

$$= \sum_{L' \subseteq \mathsf{Name}} \int_{[L' \to \mathbb{R}]} d\xi'_n \Big(\sum_{M' \subseteq \mathsf{Name}} \int_{[M' \to \mathbb{R}]} d\xi''_n \Big(\mathbf{1}_{[\mathrm{dom}(\xi'_n) \cap \mathrm{dom}(\xi''_n) = \emptyset]} \cdot \mathbf{1}_{[used(c', f(\xi'_n \oplus \xi''_n), \xi'_n)]} \cdot \mathbf{1}_{[used(c'; c'', f(\xi'_n \oplus \xi''_n), \xi'_n \oplus \xi''_n)]} \cdot h(\xi'_n \oplus \xi''_n) \Big) \Big)$$

$$= \int d\xi'_n \int d\xi''_n \Big(\mathbf{1}_{[\mathrm{dom}(\xi'_n) \cap \mathrm{dom}(\xi''_n) = \emptyset]} \cdot \mathbf{1}_{[used(c', f(\xi'_n \oplus \xi''_n), \xi'_n)]} \cdot \mathbf{1}_{[used(c'; c'', f(\xi'_n \oplus \xi''_n), \xi'_n \oplus \xi''_n)]} \cdot h(\xi'_n \oplus \xi''_n) \Big).$$

The first equality uses the definition of $\mathrm{St}_\square[\mathsf{Name}]$ and its measure. The second equality uses that $used(c'; c'', \sigma, \xi_n)$ implies a unique existence of $L \subseteq \mathrm{dom}(\xi_n)$ such that $used(c', \sigma, \xi_n|L)$; we already showed the existence of such L in the proof of Lemma C.6 (for the sequential composition case), and the uniqueness follows from the definition of *used*. The third equality uses that K is finite, and the fourth equality uses that $[K \to \mathbb{R}]$ is isomorphic to $[L \to \mathbb{R}] \times [K \setminus L \to \mathbb{R}]$ for any $L \subseteq K$. The fifth equality holds uses that $\{(L, K \setminus L) \mid K \subseteq \mathsf{Name}, L \subseteq K\} = \{(L', M') \mid L', M' \subseteq \mathsf{Name}, L' \cap M' = \emptyset\}$. The sixth equality uses that Name is finite, $\mathrm{dom}(\xi'_n) = L'$, and $\mathrm{dom}(\xi''_n) = M'$. The seventh equality uses the definition of $\mathrm{St}_\square[\mathsf{Name}]$ and its measure.

Second, to decompose $prs_\square(c'; c'')$, $pvars_\square(c'; c'')$, and $vals_\square(c'; c'')$ as in the desired equation, we show the following claim. Suppose that $\sigma \in \mathrm{St}$ and $\xi'_n, \xi''_n \in \mathrm{St}_\square[\mathsf{Name}]$ with $\mathrm{dom}(\xi'_n) \cap \mathrm{dom}(\xi''_n) = \emptyset$ satisfy $used(c'; c'', \sigma, \xi'_n \oplus \xi''_n)$ and $used(c', \sigma, \xi'_n)$. Then, we first get

$$\begin{aligned}
prs_\square(c'; c'')(\sigma|_{\mathsf{PVar}}, \xi'_n \oplus \xi''_n) &= [\![c'; c'']\!]\sigma(like) \cdot \textstyle\prod_{\mu \in \mathrm{dom}(\xi'_n \oplus \xi''_n)} [\![c'; c'']\!]\sigma(pr_\mu) \\
&= [\![c'']\!]([\![c']\!]\sigma)(like) \\
&\quad \cdot \textstyle\prod_{\mu \in \mathrm{dom}(\xi'_n)} [\![c'']\!]([\![c']\!]\sigma)(pr_\mu) \cdot \prod_{\mu \in \mathrm{dom}(\xi''_n)} [\![c'']\!]([\![c']\!]\sigma)(pr_\mu) \\
&= [\![c']\!]\sigma(like) \cdot [\![c'']\!]\big(([\![c']\!]\sigma)[like \mapsto 1]\big)(like) \\
&\quad \cdot \textstyle\prod_{\mu \in \mathrm{dom}(\xi'_n)} [\![c']\!]\sigma(pr_\mu) \cdot \prod_{\mu \in \mathrm{dom}(\xi''_n)} [\![c'']\!]\big(([\![c']\!]\sigma)[like \mapsto 1]\big)(pr_\mu) \\
&= prs_\square(c')(\sigma|_{\mathsf{PVar}}, \xi'_n) \cdot prs_\square(c'')\big(([\![c']\!]\sigma)[like \mapsto 1]|_{\mathsf{PVar}}, \xi''_n\big) \\
&= prs_\square(c')(\sigma|_{\mathsf{PVar}}, \xi'_n) \cdot prs_\square(c'')\big(pvars_\square(c')(\sigma|_{\mathsf{PVar}}, \xi'_n), \xi''_n\big).
\end{aligned}$$

Here is the proof of each equality.

- The first equality uses $used(c'; c'', \sigma, \xi'_n \oplus \xi''_n)$.
- The second equality uses $noerr(c'; c'', \sigma)$, which comes from $used(c'; c'', \sigma, -)$.
- The third equality comes from Lemma C.8, Lemma C.6-(2), and Lemma C.6-(3). The two applications of Lemma C.6 are valid since $used_-(c'', [\![c']\!]\sigma, \xi''_n)$ and $\mathrm{dom}(\xi'_n) \cap \mathrm{dom}(\xi''_n) = \emptyset$, where the first predicate follows from $used(c'; c'', \sigma, \xi'_n \oplus \xi''_n)$ and $used(c', \sigma, \xi'_n)$ by the claim in the proof of Lemma C.6 (for the sequential composition case).
- The fourth equality uses that $used(c', \sigma, \xi'_n)$ and $used(c'', ([\![c']\!]\sigma)[like \mapsto 1], \xi''_n)$, where the second predicate follows from $used_-(c'', [\![c']\!]\sigma, \xi''_n)$ and Lemma C.7.
- The fifth equality uses $([\![c']\!]\sigma)[like \mapsto 1]|_{\mathsf{PVar}} = ([\![c']\!]\sigma)|_{\mathsf{PVar}} = pvars_\square(c')(\sigma|_{\mathsf{PVar}}, \xi'_n)$, where the second part of the equation comes from $used(c', \sigma, \xi'_n)$.

By the same argument so far (except that pr_μ and $\times$ are replaced by val_μ and $\oplus$), we next get

$$vals_\square(c'; c'')(\sigma|_{\mathsf{PVar}}, \xi'_n \oplus \xi''_n) = vals_\square(c')(\sigma|_{\mathsf{PVar}}, \xi'_n) \oplus vals_\square(c'')\big(pvars_\square(c')(\sigma|_{\mathsf{PVar}}, \xi'_n), \xi''_n\big).$$

By a similar argument, we lastly get

$$\begin{aligned}
pvars_\square(c'; c'')(\sigma|_{\mathsf{PVar}}, \xi'_n \oplus \xi''_n) &= [\![c'; c'']\!]\sigma|_{\mathsf{PVar}} \\
&= [\![c'']\!]([\![c']\!]\sigma)|_{\mathsf{PVar}} \\
&= [\![c'']\!]\big(([\![c']\!]\sigma)[like \mapsto 1]\big)|_{\mathsf{PVar}}
\end{aligned}$$

$$= pvars_{\Box}(c'')\big((\llbracket c' \rrbracket \sigma)[like \mapsto 1]|_{\mathsf{PVar}}, \xi_n''\big)$$
$$= pvars_{\Box}(c'')\big(pvars_{\Box}(c')(\sigma|_{\mathsf{PVar}}, \xi_n'), \xi_n''\big).$$

Here is the proof of each equality.

- The first equality uses $used(c'; c'', \sigma, \xi_n' \oplus \xi_n'')$.
- The second equality uses $noerr(c'; c'', \sigma)$ (shown above).
- The third equality uses $used_{\_}(c'', \llbracket c' \rrbracket \sigma, \xi_n'')$ (shown above) and Lemma C.6-(3).
- The fourth equality uses $used(c'', (\llbracket c' \rrbracket \sigma)[like \mapsto 1], \xi_n'')$ (shown above).
- The fifth equality uses $(\llbracket c' \rrbracket \sigma)[like \mapsto 1]|_{\mathsf{PVar}} = pvars_{\Box}(c')(\sigma|_{\mathsf{PVar}}, \xi_n')$ (shown above).

Third, to remove some indicator terms that will appear in our derivation, we show the next claim: for any $\sigma \in \mathsf{St}$ and $\xi_n', \xi_n'' \in \mathsf{St}_{\Box}[\mathsf{Name}]$ with $\mathrm{dom}(\xi_n') \cap \mathrm{dom}(\xi_n'') = \emptyset$, $used_{\_}(c', \sigma, \xi_n')$ and $used_{\_}(c'', \llbracket c' \rrbracket \sigma, \xi_n'')$ imply $used_{\_}(c'; c'', \sigma, \xi_n' \oplus \xi_n'')$. Assume the premise. Then, we have

$$\begin{aligned}
\llbracket c' \rrbracket \sigma \in \mathsf{St} \wedge \xi_n' &= \sigma|_{\mathrm{dom}(\xi_n')} \\
&\wedge \big(\forall \mu \in \mathsf{Name}.\ \llbracket c' \rrbracket \sigma(cnt_\mu) - \sigma(cnt_\mu) \le 1\big) \\
&\wedge \mathrm{dom}(\xi_n') = \{\mu \in \mathsf{Name} \mid \llbracket c' \rrbracket \sigma'(cnt_\mu) - \sigma(cnt_\mu) = 1\}, \\
\llbracket c'' \rrbracket(\llbracket c' \rrbracket \sigma) \in \mathsf{St} \wedge \xi_n'' &= (\llbracket c' \rrbracket \sigma)|_{\mathrm{dom}(\xi_n'')} \\
&\wedge \big(\forall \mu \in \mathsf{Name}.\ \llbracket c'' \rrbracket(\llbracket c' \rrbracket \sigma)(cnt_\mu) - \llbracket c' \rrbracket \sigma(cnt_\mu) \le 1\big) \\
&\wedge \mathrm{dom}(\xi_n'') = \{\mu \in \mathsf{Name} \mid \llbracket c'' \rrbracket(\llbracket c' \rrbracket \sigma)(cnt_\mu) - \llbracket c' \rrbracket \sigma(cnt_\mu) = 1\}.
\end{aligned}$$

We should show

$$\begin{aligned}
\llbracket c'; c'' \rrbracket \sigma \in \mathsf{St} \wedge \xi_n' \oplus \xi_n'' &= \sigma|_{\mathrm{dom}(\xi_n' \oplus \xi_n'')} \\
&\wedge \big(\forall \mu \in \mathsf{Name}.\ \llbracket c'; c'' \rrbracket \sigma(cnt_\mu) - \sigma(cnt_\mu) \le 1\big) \\
&\wedge \mathrm{dom}(\xi_n' \oplus \xi_n'') = \{\mu \in \mathsf{Name} \mid \llbracket c'; c'' \rrbracket \sigma'(cnt_\mu) - \sigma(cnt_\mu) = 1\}.
\end{aligned}$$

We obtain the four clauses as follows. The first clause follows from $\llbracket c'' \rrbracket(\llbracket c' \rrbracket \sigma) \in \mathsf{St}$. The second clause comes from $\xi_n' = \sigma|_{\mathrm{dom}(\xi_n')}$ and $\xi_n'' = (\llbracket c' \rrbracket \sigma)|_{\mathrm{dom}(\xi_n'')} = \sigma|_{\mathrm{dom}(\xi_n'')}$, where the last equality comes from Lemma C.4. The third and fourth clauses hold by the following:

$$\begin{aligned}
&\llbracket c'; c'' \rrbracket \sigma(cnt_\mu) - \sigma(cnt_\mu) \\
&= \llbracket c'' \rrbracket(\llbracket c' \rrbracket \sigma)(cnt_\mu) - \sigma(cnt_\mu) \\
&= \begin{cases}
\llbracket c' \rrbracket \sigma(cnt_\mu) - \sigma(cnt_\mu) = 1 & \text{if } \mu \in \mathrm{dom}(\xi_n') \\
\llbracket c'' \rrbracket(\llbracket c' \rrbracket \sigma)(cnt_\mu) - \llbracket c' \rrbracket \sigma(cnt_\mu) = 1 & \text{if } \mu \in \mathrm{dom}(\xi_n'') \\
\sigma(cnt_\mu) - \sigma(cnt_\mu) = 0 & \text{if } \mu \in \mathsf{Name} \setminus (\mathrm{dom}(\xi_n') \cup \mathrm{dom}(\xi_n'')).
\end{cases}
\end{aligned}$$

- The first case uses Lemma C.6-(2) (applied to $used_{\_}(c'', \llbracket c' \rrbracket \sigma, \xi_n'')$ and $\mathrm{dom}(\xi_n') \cap \mathrm{dom}(\xi_n'') = \emptyset$) and the fourth clause of $used_{\_}(c', \sigma, \xi_n')$.
- The second case uses Lemma C.6-(2) (applied to $used_{\_}(c', \sigma, \xi_n')$ and $\mathrm{dom}(\xi_n') \cap \mathrm{dom}(\xi_n'') = \emptyset$) and the fourth clause of $used_{\_}(c'', \llbracket c' \rrbracket \sigma, \xi_n'')$.
- The third case uses Lemma C.6-(2) (applied to $used_{\_}(c'', \llbracket c' \rrbracket \sigma, \xi_n'')$ and $used_{\_}(c', \sigma, \xi_n')$).

Finally, we put the three results together. Define $f : \mathsf{St}_{\Box}[\mathsf{Name}] \to \mathsf{St}$ as $f(\xi_n) \triangleq \sigma_p \oplus \xi_n \oplus (\lambda v \in \mathsf{Var} \setminus (\mathsf{PVar} \cup \mathrm{dom}(\xi_n)).\ 1)$. Then, we obtain the desired equation as follows:

$$\int d\xi_n \Big(prs_{\Box}(c'; c'')(\sigma_p, \xi_n) \cdot g\Big(pvars_{\Box}(c'; c'')(\sigma_p, \xi_n), vals_{\Box}(c'; c'')(\sigma_p, \xi_n)\Big)\Big)$$
$$= \int d\xi_n \Big(\mathbf{1}_{[used(c'; c'', f(\xi_n), \xi_n)]} \cdot prs_{\Box}(c'; c'')(\sigma_p, \xi_n) \cdot g\Big(pvars_{\Box}(c'; c'')(\sigma_p, \xi_n), vals_{\Box}(c'; c'')(\sigma_p, \xi_n)\Big)\Big)$$

$$= \int d\xi'_n \int d\xi''_n \Big(\mathbf{1}_{[\mathrm{dom}(\xi'_n)\cap\mathrm{dom}(\xi''_n)=\emptyset]} \cdot \mathbf{1}_{[used(c', f(\xi'_n\oplus\xi''_n),\xi'_n)]} \cdot \mathbf{1}_{[used(c';c'', f(\xi'_n\oplus\xi''_n),\xi'_n\oplus\xi''_n)]} \cdot prs_\square(c';c'')(\sigma_p, \xi'_n \oplus \xi''_n) \cdot g\Big(pvars_\square(c';c'')(\sigma_p, \xi'_n \oplus \xi''_n), vals_\square(c';c'')(\sigma_p, \xi'_n \oplus \xi''_n)\Big)\Big)$$

$$= \int d\xi'_n \int d\xi''_n \Big(\mathbf{1}_{[\mathrm{dom}(\xi'_n)\cap\mathrm{dom}(\xi''_n)=\emptyset]} \cdot \mathbf{1}_{[used(c', f(\xi'_n\oplus\xi''_n),\xi'_n)]} \cdot \mathbf{1}_{[used(c';c'', f(\xi'_n\oplus\xi''_n),\xi'_n\oplus\xi''_n)]} \cdot prs_\square(c')(\sigma_p, \xi'_n) \cdot prs_\square(c'')\big(pvars_\square(c')(\sigma_p, \xi'_n), \xi''_n\big) \cdot g\Big(pvars_\square(c'')\big(pvars_\square(c')(\sigma_p, \xi'_n), \xi''_n\big), vals_\square(c')(\sigma_p, \xi'_n) \oplus vals_\square(c'')\big(pvars_\square(c')(\sigma_p, \xi'_n), \xi''_n\big)\Big)\Big)$$

$$= \int d\xi'_n \int d\xi''_n \Big(\mathbf{1}_{[\mathrm{dom}(\xi'_n)\cap\mathrm{dom}(\xi''_n)=\emptyset]} \cdot prs_\square(c')(\sigma_p, \xi'_n) \cdot prs_\square(c'')\big(pvars_\square(c')(\sigma_p, \xi'_n), \xi''_n\big) \cdot g\Big(pvars_\square(c'')\big(pvars_\square(c')(\sigma_p, \xi'_n), \xi''_n\big), vals_\square(c')(\sigma_p, \xi'_n) \oplus vals_\square(c'')\big(pvars_\square(c')(\sigma_p, \xi'_n), \xi''_n\big)\Big)\Big).$$

The first equality uses that $prs_\square(c';c'')(\sigma_p, \xi_n) \neq 0$ implies $\mathbf{1}_{[used(c';c'', f(\xi_n),\xi_n)]} = 1$:

- Since $prs_\square(c';c'')(\sigma_p, \xi_n) \neq 0$, there is $\sigma_r \in \mathrm{St}[\mathrm{Var}\backslash(\mathrm{PVar}\cup\mathrm{dom}(\xi_n))]$ such that $used(c';c'', \sigma_p\oplus \xi_n \oplus \sigma_r, \xi_n)$. Note $(\sigma_p \oplus \xi_n \oplus \sigma_r)|_V = f(\xi_n)|_V$ for $V = \mathrm{PVar} \cup \mathrm{dom}(\xi_n) \cup \{like\}$. From these and Lemma C.7, we get $used(c';c'', f(\xi_n), \xi_n)$.

The second and third equalities use the first and second results we proved above, respectively. The fourth equality uses the next claim: $prs_\square(c')(\sigma_p, \xi'_n) \cdot prs_\square(c'')\big(pvars_\square(c')(\sigma_p, \xi'_n), \xi''_n\big) \neq 0$ and $\mathrm{dom}(\xi'_n) \cap \mathrm{dom}(\xi''_n) = \emptyset$ imply $\mathbf{1}_{[used(c', f(\xi'_n\oplus\xi''_n),\xi'_n)]} \cdot \mathbf{1}_{[used(c';c'', f(\xi'_n\oplus\xi''_n),\xi'_n\oplus\xi''_n)]} = 1$. We prove the claim using the third result we proved above:

- Assume the premise. From $prs_\square(c')(\sigma_p, \xi'_n) \neq 0$, there is $\sigma'_r \in \mathrm{St}[\mathrm{Var} \setminus (\mathrm{PVar} \cup \mathrm{dom}(\xi'_n))]$ such that $used(c', \sigma_p \oplus \xi'_n \oplus \sigma'_r, \xi'_n)$. Note $(\sigma_p \oplus \xi'_n \oplus \sigma'_r)|_{V'} = f(\xi'_n \oplus \xi''_n)|_{V'}$ for $V' = \mathrm{PVar} \cup \mathrm{dom}(\xi'_n) \cup \{like\}$. From these and Lemma C.7, we get $used(c', f(\xi'_n \oplus \xi''_n), \xi'_n)$ as desired.
- From $prs_\square(c'')\big(pvars_\square(c')(\sigma_p, \xi'_n), \xi''_n\big) \neq 0$, there is $\sigma''_r \in \mathrm{St}[\mathrm{Var}\backslash(\mathrm{PVar}\cup\mathrm{dom}(\xi''_n))]$ such that
$$used(c'', pvars_\square(c')(\sigma_p, \xi'_n) \oplus \xi''_n \oplus \sigma''_r, \xi''_n).$$
Since $used(c', f(\xi'_n \oplus \xi''_n), \xi'_n)$, we have $[\![c']\!](f(\xi'_n \oplus \xi''_n)) \in \mathrm{St}$ and $pvars_\square(c')(\sigma_p, \xi'_n) = pvars_\square(c')(f(\xi'_n \oplus \xi''_n)|_{\mathrm{PVar}}, \xi'_n) = [\![c']\!](f(\xi'_n \oplus \xi''_n))|_{\mathrm{PVar}}$; also, by Lemma C.4, $\xi''_n = [\![c']\!](f(\xi'_n \oplus \xi''_n))|_{\mathrm{dom}(\xi''_n)}$. Thus,
$$(pvars_\square(c')(\sigma_p, \xi'_n) \oplus \xi''_n \oplus \sigma''_r)|_{V''} = [\![c]\!](f(\xi'_n \oplus \xi''_n))|_{V''}$$
for $V'' = \mathrm{PVar} \cup \mathrm{dom}(\xi''_n)$. From these and Lemma C.7, we get $used_\_(c'', [\![c']\!](f(\xi'_n \oplus \xi''_n)), \xi''_n)$.
- From $\mathrm{dom}(\xi'_n) \cap \mathrm{dom}(\xi''_n) = \emptyset$, $used_\_(c', f(\xi'_n \oplus \xi''_n), \xi'_n)$, and $used_\_(c'', [\![c']\!](f(\xi'_n \oplus \xi''_n)), \xi''_n)$, we can apply the third result proved above, and get $used_\_(c';c'', f(\xi'_n \oplus \xi''_n), \xi'_n \oplus \xi''_n)$. Since $f(\xi'_n \oplus \xi''_n)(like) = 1$, we get $used(c';c'', f(\xi'_n \oplus \xi''_n), \xi'_n \oplus \xi''_n)$ as desired.

This completes the proof. □

D DEFERRED RESULTS IN §4.2

D.1 Deferred Statements and Their Proofs

Lemma D.1. *Let c be a command and π be a simple reparameterisation plan. Suppose that for all $n \in \mathrm{NameEx}$, $d, d' \in \mathrm{DistEx}$, and $(\lambda y.e) \in \mathrm{LamEx}$ such that $\pi(n, d, \lambda y.e) = (d', \_)$, we have*

$$d' = \mathtt{dist}_{\mathtt{N}}(r'_1, r'_2) \quad \textit{for some } r'_1, r'_2 \in \mathbb{R}. \tag{18}$$

Further, assume that for all $\sigma_n \in \mathrm{St}[\mathrm{Name}]$, the function

$$\sigma_\theta \in \mathrm{St}[\theta] \longmapsto p^{\langle rv(\pi)\rangle}_{\overline{c}^\pi, \sigma_\theta}(\sigma_n) \tag{19}$$

is continuous. Then, for all $\sigma_\theta \in \mathrm{St}[\theta]$ *and* $\sigma_n \in \mathrm{St}[\mathrm{Name}]$,

$$\nabla_\theta p_{\overline{c}^\pi, \sigma_\theta}^{\langle rv(\pi) \rangle}(\sigma_n) = 0. \tag{20}$$

Proof. Consider c and π that satisfies the given conditions. Fix $\sigma_n \in \mathrm{St}[\mathrm{Name}]$. Let $f : \mathrm{St}[\theta] \to \mathbb{R}$ be the function in Eq. (19). Suppose that Eq. (20) does not hold. Then, f is not a constant function.

On the one hand, since f is continuous (by assumption) and not constant, the image of f over its domain (i.e., $f(\mathrm{St}[\theta]) \subseteq \mathbb{R}$) is an uncountable set. This can be shown as follows: since the image of a connected set over a continuous function is connected, $f(\mathrm{St}[\theta])$ is a connected set in $\mathbb{R}$; since f is not constant, $f(\mathrm{St}[\theta])$ contains at least two points; since $f(\mathrm{St}[\theta])$ is connected, it should contain a non-empty interval, so it should be an uncountable set.

On the other hand, since π is simple and satisfies Eq. (18), and since c has only finitely many sample commands, $f(\mathrm{St}[\theta])$ is a finite set. So this contradicts to that $f(\mathrm{St}[\theta])$ is an uncountable set. Hence, f should satisfy Eq. (20). □

Theorem D.2. *Let* $f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}$ *be a measurable function that satisfies the next three conditions:*

- *For all* $x \in \mathbb{R}^n$, $f(-, x) : \mathbb{R} \to \mathbb{R}$ *is differentiable.*
- *For all* $\theta \in \mathbb{R}$, $\int_{\mathbb{R}^n} f(\theta, x)\, dx$ *is finite.*
- *For all* $\theta \in \mathbb{R}$, *there is an open* $U \subseteq \mathbb{R}$ *such that* $\theta \in U$ *and* $\int_{\mathbb{R}^n} \mathrm{Lip}\big(f(-, x)|_U\big)\, dx$ *is finite.*

Here $\mathrm{Lip}(g)$ *for a function* $g : V \to \mathbb{R}$ *with* $V \subseteq \mathbb{R}$ *denotes the smallest Lipschitz constant of* g*:*

$$\mathrm{Lip}(g) \triangleq \sup_{r, r' \in V,\, r \neq r'} \frac{|g(r') - g(r)|}{|r' - r|} \in \mathbb{R} \cup \{\infty\}.$$

Then, for all $\theta \in \mathbb{R}$, *both sides of the following are well-defined and equal:*

$$\nabla_\theta \int_{\mathbb{R}^n} f(\theta, x)\, dx = \int_{\mathbb{R}^n} \nabla_\theta f(\theta, x)\, dx$$

where ∇_θ *denotes the partial differentiation operator with respect to* θ*.*

Proof. This theorem follows from Theorem E.2 due to the following: the first condition of this theorem implies the first and second conditions of Theorem E.2 (as differentiability implies continuity); the second and third conditions of this theorem are identical to the third and fourth conditions of Theorem E.2; and the conclusion of this theorem is identical to that of Theorem E.2. □

D.2 Proof of Theorem 4.5

The proof of Theorem 4.5 relies on the following two lemmas, which are proven in §D.3. The first lemma states that if a command contains no observe commands, then its (full) density function can be decomposed into its partial density functions over S and $\mathrm{Name} \setminus S$ for any $S \subseteq \mathrm{Name}$. The second lemma states that if π is simple and c uses only $\lambda y.y$ as the third argument of its sample commands, then the partial density function of $\overline{c}^\pi$ over non-transformed random variables (i.e., variables in $\mathrm{Name} \setminus rv(\pi)$) is connected to that of c via the value function of $\overline{c}^\pi$.

Lemma D.3. *Let* c *be a command. If* c *does not contain observe commands, then, for all* $S \subseteq \mathrm{Name}$, $\sigma_\theta \in \mathrm{St}[\theta]$, *and* $\sigma_n \in \mathrm{St}[\mathrm{Name}]$,

$$p_{c, \sigma_\theta}(\sigma_n) = p_{c, \sigma_\theta}^{\langle S \rangle}(\sigma_n) \cdot p_{c, \sigma_\theta}^{\langle \mathrm{Name} \setminus S \rangle}(\sigma_n).$$

Lemma D.4. *Let* c *be a command and* π *be a reparameterisation plan. Suppose that every sample command in* c *has* $\lambda y.y$ *as its third argument. Then, for all* $\sigma_\theta \in \mathrm{St}[\theta]$ *and* $\sigma_n \in \mathrm{St}[\mathrm{Name}]$, *if* $p_{\overline{c}^\pi, \sigma_\theta}(\sigma_n) > 0$, *then*

$$p_{\overline{c}^\pi, \sigma_\theta}^{\langle \mathrm{Name} \setminus rv(\pi) \rangle}(\sigma_n) = p_{c, \sigma_\theta}^{\langle \mathrm{Name} \setminus rv(\pi) \rangle}(v_{\overline{c}^\pi, \sigma_\theta}(\sigma_n)).$$

We now prove Theorem 4.5 using the two lemmas.

Proof of Theorem 4.5. Let $S = rv(\pi)$. Before starting the main derivation of the selective gradient estimator, we show the differentiability of several functions which are to be used in the derivation. From (R2) and (R3), the next functions over $\mathrm{St}[\theta]$ are differentiable for all σ_n by the preservation of differentiability under function composition:

$$\sigma_\theta \longmapsto p_{c_m,\sigma_\theta}(v_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n)), \quad \sigma_\theta \longmapsto p^{\langle S\rangle}_{c_g,\sigma_\theta}(v_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n)), \quad \sigma_\theta \longmapsto p^{\langle \mathsf{Name}\setminus S\rangle}_{c_g,\sigma_\theta}(v_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n)),$$
$$\sigma_\theta \longmapsto p^{\langle S\rangle}_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n), \quad \sigma_\theta \longmapsto p^{\langle \mathsf{Name}\setminus S\rangle}_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n).$$

From this, the next functions over $\mathrm{St}[\theta]$ are also differentiable for all σ_n by Lemma D.3 with c_g and $\overline{c_g}^{\pi}$ and by the fact that the multiplication of differentiable functions is differentiable:

$$\sigma_\theta \longmapsto p_{c_g,\sigma_\theta}(v_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n)), \qquad \sigma_\theta \longmapsto p_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n).$$

These differentiability results are required in the below proof to apply several gradients rules (e.g., $\nabla_\theta(f(\theta)+g(\theta)) = \nabla_\theta f(\theta) + \nabla_\theta g(\theta)$) which may fail for non-differentiable functions.

Fix $\sigma_\theta \in \mathrm{St}[\theta]$. Using the above differentiability results, we derive the selective gradient estimator as follows, where we write σ'_n for $v_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n)$:

$$\begin{aligned}
&\nabla_\theta \mathcal{L}_\theta \\
&= \nabla_\theta \int d\sigma_n \left(p_{c_g,\sigma_\theta}(\sigma_n) \cdot \log \frac{p_{c_m,\sigma_\theta}(\sigma_n)}{p_{c_g,\sigma_\theta}(\sigma_n)} \right) \\
&= \nabla_\theta \int d\sigma_n \left(p_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n) \cdot \log \frac{p_{c_m,\sigma_\theta}(\sigma'_n)}{p_{c_g,\sigma_\theta}(\sigma'_n)} \right) \\
&= \int d\sigma_n \, \nabla_\theta \left(p_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n) \cdot \log \frac{p_{c_m,\sigma_\theta}(\sigma'_n)}{p_{c_g,\sigma_\theta}(\sigma'_n)} \right) \\
&= \int d\sigma_n \left(\nabla_\theta p_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n) \cdot \log \frac{p_{c_m,\sigma_\theta}(\sigma'_n)}{p_{c_g,\sigma_\theta}(\sigma'_n)} + p_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n) \cdot \nabla_\theta \log \frac{p_{c_m,\sigma_\theta}(\sigma'_n)}{p_{c_g,\sigma_\theta}(\sigma'_n)} \right) \\
&= \int d\sigma_n \, p_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n) \left(\nabla_\theta \log p_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n) \cdot \log \frac{p_{c_m,\sigma_\theta}(\sigma'_n)}{p_{c_g,\sigma_\theta}(\sigma'_n)} - \nabla_\theta \log p_{c_g,\sigma_\theta}(\sigma'_n) + \nabla_\theta \log p_{c_m,\sigma_\theta}(\sigma'_n) \right) \\
&= \int d\sigma_n \, p_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n) \Bigg[\left(\nabla_\theta \log p^{\langle S\rangle}_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n) + \nabla_\theta \log p^{\langle \mathsf{Name}\setminus S\rangle}_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n) \right) \cdot \log \frac{p_{c_m,\sigma_\theta}(\sigma'_n)}{p_{c_g,\sigma_\theta}(\sigma'_n)} \\
&\qquad\qquad - \left(\nabla_\theta \log p^{\langle S\rangle}_{c_g,\sigma_\theta}(\sigma'_n) + \nabla_\theta \log p^{\langle \mathsf{Name}\setminus S\rangle}_{c_g,\sigma_\theta}(\sigma'_n) \right) + \nabla_\theta \log p_{c_m,\sigma_\theta}(\sigma'_n) \Bigg] \\
&= \int d\sigma_n \, p_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n) \Bigg[\left(0 + \nabla_\theta \log p^{\langle \mathsf{Name}\setminus S\rangle}_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n) \right) \cdot \log \frac{p_{c_m,\sigma_\theta}(\sigma'_n)}{p_{c_g,\sigma_\theta}(\sigma'_n)} \\
&\qquad\qquad - \left(\nabla_\theta \log p^{\langle S\rangle}_{c_g,\sigma_\theta}(\sigma'_n) + 0 \right) + \nabla_\theta \log p_{c_m,\sigma_\theta}(\sigma'_n) \Bigg] \\
&= \int d\sigma_n \, p_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n) \Bigg(\nabla_\theta \log p^{\langle \mathsf{Name}\setminus S\rangle}_{c_g,\sigma_\theta}(\sigma'_n) \cdot \log \frac{p_{c_m,\sigma_\theta}(\sigma'_n)}{p_{c_g,\sigma_\theta}(\sigma'_n)} \\
&\qquad\qquad - \nabla_\theta \log p^{\langle S\rangle}_{c_g,\sigma_\theta}(\sigma'_n) + \nabla_\theta \log p_{c_m,\sigma_\theta}(\sigma'_n) \Bigg).
\end{aligned}$$

We justify key steps of the above derivation below.

- The second equality comes from Theorem 4.2 and the fact that v_{c_g,σ_θ} is the identity function (since the third argument of every sample command in c_g is the identity function $\lambda y.y$).

- The third equality holds because differentiation there commutes with integration by (R5).
- The fourth comes from the product rule for differentiation: $\nabla_\theta(f(\theta)\cdot g(\theta)) = \nabla_\theta f(\theta)\cdot g(\theta) + f(\theta)\cdot\nabla_\theta g(\theta)$ for all differentiable f and g. Here f and g in the original equation are differentiable because differentiability is preserved under division and log for positive-valued functions.
- The fifth equality holds because $\nabla_\theta f(\theta) = f(\theta)\cdot\nabla_\theta \log f(\theta)$ for all differentiable and positive-valued f.
- The sixth equality follows from Lemma D.3 applied to c_g and $\overline{c_g}^{\pi}$ (both of which do not contain observe commands), and from the linearity of differentiation: $\nabla_\theta(f(\theta)+g(\theta)) = \nabla_\theta f(\theta) + \nabla_\theta g(\theta)$ for all differentiable f and g. Here f and g in the original equation are differentiable because differentiability is preserved under log for positive-valued functions.
- The seventh equality follows from (R4) and

$$\mathbb{E}_{p_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n)}\left[\nabla_\theta \log p^{\langle \mathrm{Name}\setminus S\rangle}_{c_g,\sigma_\theta}(\sigma_n')\right] = 0. \tag{21}$$

  The proof of Eq. (21) will be given after we complete this justification of the derivation.
- The last equality comes from Lemma D.4 applied to c_g.

The only remaining part is to prove Eq. (21). We derive the equation as follows:

$$\begin{aligned}
&\int d\sigma_n \left(p_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n)\cdot\nabla_\theta \log p^{\langle \mathrm{Name}\setminus S\rangle}_{c_g,\sigma_\theta}(\sigma_n')\right)\\
&= \int d\sigma_n \left(p_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n)\cdot\nabla_\theta \log p^{\langle \mathrm{Name}\setminus S\rangle}_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n)\right)\\
&= \int d\sigma_n \left(p_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n)\cdot\left(\nabla_\theta \log p^{\langle \mathrm{Name}\setminus S\rangle}_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n) + \nabla_\theta \log p^{\langle S\rangle}_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n)\right)\right)\\
&= \int d\sigma_n \left(p_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n)\cdot\nabla_\theta \log p_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n)\right)\\
&= \int d\sigma_n\, \nabla_\theta p_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n)\\
&= \nabla_\theta \int d\sigma_n\, p_{\overline{c_g}^{\pi},\sigma_\theta}(\sigma_n)\\
&= \nabla_\theta 1 = 0.
\end{aligned}$$

Here is the justification of the above derivation:

- The first equality comes from Lemma D.4 applied to c_g.
- The second equality follows from (R4).
- The third equality holds because of Lemma D.3 applied to $\overline{c_g}^{\pi}$ (which does not contain observe commands), and the linearity of differentiation: $\nabla_\theta(f(\theta)+g(\theta)) = \nabla_\theta f(\theta) + \nabla_\theta g(\theta)$ for all differentiable f and g. Here f and g in the original equation are differentiable because differentiability is preserved under log for positive-valued functions.
- The fourth equality holds because $\nabla_\theta f(\theta) = f(\theta)\cdot\nabla_\theta \log f(\theta)$ for all differentiable and positive-valued f.
- The fifth equality uses (R5), which states the commutativity between differentiation and integration in the equality.
- The six equality comes from that $p_{\overline{c_g}^{\pi},\sigma_\theta}$ is a probability density by Remark 4.3.

This completes the proof. □

D.3 Proofs of Lemmas D.3 and D.4

We define the partial density version of $prs_{\square}^{\langle S \rangle}(c)$ for $S \subseteq \mathsf{Name}$:

$$prs_{\square}^{\langle S \rangle}(c) : \mathsf{St}[\mathsf{PVar}] \times \mathsf{St}_{\square}[\mathsf{Name}] \to [0, \infty),$$

$$prs_{\square}^{\langle S \rangle}(c)(\sigma_p, \xi_n) \triangleq \begin{cases} \prod_{\mu \in \mathrm{dom}(\xi_n) \cap S} [\![c]\!](\sigma_p \oplus \xi_n \oplus \sigma_r)(pr_\mu) & \text{if } \exists \sigma_r.\ used(c, \sigma_p \oplus \xi_n \oplus \sigma_r, \xi_n) \\ 0 & \text{otherwise.} \end{cases}$$

$prs_{\square}^{\langle S \rangle}(c)$ enjoys many of the properties that $prs_{\square}(c)$ has. For instance, $prs_{\square}^{\langle S \rangle}(c)$ is a well-defined function (i.e., its value does not depend on the choice of σ_r), as $prs_{\square}(c)$ does. Since the proof of those properties of $prs_{\square}^{\langle S \rangle}(c)$ would be almost identical to that of $prs_{\square}(c)$, we will use them in the following proofs without explicitly (re)proving them.

Proof of Lemma D.3. Let c be a command that has no observe commands. Let $S \subseteq \mathsf{Name}$, $\sigma_\theta \in \mathsf{St}[\theta]$, and $\sigma_n \in \mathsf{St}[\mathsf{Name}]$. We set σ_0 as in the definition of p_{c,σ_θ} in Eq. (3) (as a function of σ_n). Let $\sigma = \sigma_\theta \oplus \sigma_n \oplus \sigma_0$. If $noerr(c, \sigma)$ does not hold, then the LHS and RHS of the desired equation become zero, so the equation holds. If $noerr(c, \sigma)$ holds, we get the desired equation as follows:

$$\begin{aligned} p_{c,\sigma_\theta}^{\langle S \rangle}(\sigma_n) \cdot p_{c,\sigma_\theta}^{\langle \mathsf{Name} \setminus S \rangle}(\sigma_n) &= \left([\![c]\!]\sigma(like) \cdot \textstyle\prod_{\mu \in S} [\![c]\!]\sigma(pr_\mu)\right) \cdot \left([\![c]\!]\sigma(like) \cdot \textstyle\prod_{\mu \in \mathsf{Name} \setminus S} [\![c]\!]\sigma(pr_\mu)\right) \\ &= \left([\![c]\!]\sigma(like)\right)^2 \cdot \textstyle\prod_{\mu \in \mathsf{Name}} [\![c]\!]\sigma(pr_\mu) \\ &= [\![c]\!]\sigma(like) \cdot p_{c,\sigma_\theta}(\sigma_n) \\ &= \sigma(like) \cdot p_{c,\sigma_\theta}(\sigma_n) \\ &= p_{c,\sigma_\theta}(\sigma_n). \end{aligned}$$

The second last equality uses Lemma D.5, and the last equality uses $\sigma(like) = 1$ (which holds by the definition of σ_0). This completes the proof. □

Proof of Lemma D.4. Consider a command c, a reparameterisation plan π, $\sigma_\theta \in \mathsf{St}[\theta]$, and $\sigma_n \in \mathsf{St}[\mathsf{Name}]$. Assume that all the sample commands of c have $\lambda y.y$ as their third arguments, and $p_{\overline{c}^\pi,\sigma_\theta}(\sigma_n) > 0$.

We first define several objects and make observations on them. Let $S \triangleq \mathsf{Name} \setminus rv(\pi)$. Define $f_* : \mathsf{St}[\mathsf{Name}] \to \mathsf{St}[\mathsf{AVar}]$ to be the function for constructing an initial state:

$$f_*(\sigma_n)(v) \triangleq \begin{cases} f_{pr}(\sigma_n(\mu)) & \text{if } v \equiv pr_\mu \text{ for some } \mu \\ f_{val}(\sigma_n(\mu)) & \text{if } v \equiv val_\mu \text{ for some } \mu \\ f_{cnt}(\sigma_n(\mu)) & \text{if } v \equiv cnt_\mu \text{ for some } \mu \\ 1 & \text{if } v \equiv like, \end{cases}$$

where $f_{val}(r) \triangleq r$, $f_{pr}(r) \triangleq \mathcal{N}(r; 0, 1)$, and $f_{cnt}(r) \triangleq 0$. Define initial states $\overline{\sigma}, \sigma \in \mathsf{St}$ for $p_{\overline{c}^\pi,\sigma_\theta}(\sigma_n)$ and $p_{c,\sigma_\theta}(v_{\overline{c}^\pi,\sigma_\theta}(\sigma_n))$, respectively, as

$$\overline{\sigma} \triangleq \sigma_p \oplus \sigma_n \oplus f_*(\sigma_n), \qquad \sigma \triangleq \sigma_p \oplus v_{\overline{c}^\pi,\sigma_\theta}(\sigma_n) \oplus f_*(v_{\overline{c}^\pi,\sigma_\theta}(\sigma_n)),$$

where $\sigma_p \triangleq \sigma_\theta \oplus (\lambda v \in \mathsf{PVar} \setminus \theta.\, 0) \in \mathsf{St}[\mathsf{PVar}]$. Then, the assumption $p_{\overline{c}^\pi,\sigma_\theta}(\sigma_n) > 0$ implies $noerr(\overline{c}^\pi, \overline{\sigma})$ by the definition of p. From this, $\overline{\sigma}(like) = 1$, and the definition of $used$, there exists $\overline{\xi_n} \in \mathsf{St}_{\square}[\mathsf{Name}]$ such that $used(\overline{c}^\pi, \overline{\sigma}, \overline{\xi_n})$. From this, we have

$$\overline{\sigma} = \sigma_p \oplus \overline{\xi_n} \oplus \overline{\sigma_r}, \qquad used(\overline{c}^\pi, \sigma_p \oplus \overline{\xi_n} \oplus \overline{\sigma_r}, \overline{\xi_n}),$$

for some $\overline{\sigma_r}$. Next, let

$$\xi_n \triangleq vals_{\square}(\overline{c}^\pi)(\sigma_p, \overline{\xi_n}).$$

We can apply Lemma D.6 to $used(\overline{c}^{\pi}, \overline{\sigma}, \overline{\xi_n})$, since all the sample commands of c have $\lambda y.y$ in their third arguments (by assumption). The application of the lemma gives:

$$\forall \sigma'_r \in \mathsf{St}[\mathsf{Var} \setminus (\mathsf{PVar} \cup \mathrm{dom}(\xi_n))].\ \ \sigma'_r(like) = 1 \implies used(c, \sigma_p \oplus \xi_n \oplus \sigma'_r, \xi_n),$$
$$prs^{\langle S \rangle}_{\square}(\overline{c}^{\pi})(\sigma_p, \overline{\xi_n}) = prs^{\langle S \rangle}_{\square}(c)(\sigma_p, \xi_n),$$

where the for-all part comes from Lemma C.7.

We now show two claims. The first claims is: there exists σ_r such that

$$\sigma = \sigma_p \oplus \xi_n \oplus \sigma_r, \qquad\qquad used(c, \sigma_p \oplus \xi_n \oplus \sigma_r, \xi_n).$$

By the definition of σ, it suffices to show that $\xi_n = \left(v_{\overline{c}^{\pi}, \sigma_\theta}(\sigma_n)\right)|_{\mathrm{dom}(\xi_n)}$. This indeed holds as follows: for any $\mu \in \mathrm{dom}(\xi_n)$, $\xi_n(\mu) = vals_{\square}(\overline{c}^{\pi})(\sigma_p, \overline{\xi_n})(\mu) = [\![\overline{c}^{\pi}]\!]\overline{\sigma}(val_\mu) = v_{\overline{c}^{\pi}, \sigma_\theta}(\sigma_n)(\mu)$, where the second equality uses $used(\overline{c}^{\pi}, \sigma_p \oplus \overline{\xi_n} \oplus \overline{\sigma_r}, \overline{\xi_n})$. The second claim is: for all $\mu \in S \setminus \mathrm{dom}(\xi_n)$,

$$\sigma(pr_\mu) = \overline{\sigma}(pr_\mu).$$

Here is the proof of the claim: $\sigma(pr_\mu) = f_{pr}(\sigma(\mu)) = f_{pr}(v_{\overline{c}^{\pi}, \sigma_\theta}(\sigma_n)(\mu)) = f_{pr}([\![\overline{c}^{\pi}]\!]\overline{\sigma}(val_\mu)) = f_{pr}(\overline{\sigma}(val_\mu)) = f_{pr}(\overline{\sigma}(\mu))$; and $\overline{\sigma}(pr_\mu) = f_{pr}(\overline{\sigma}(\mu))$; here the second last equality in the first equation uses Lemma C.6-(2) with $\mu \notin \mathrm{dom}(\overline{\xi_n})$ and $used(\overline{c}^{\pi}, \sigma_p \oplus \overline{\xi_n} \oplus \overline{\sigma_r}, \overline{\xi_n})$, and the last equality in the first equation uses $f_{val}(r) = r$.

Based on the observations made so far, we show the desired equation as follows:

$$\begin{aligned}
p^{\langle S \rangle}_{\overline{c}^{\pi}, \sigma_\theta}(\sigma_n) &= \prod_{\mu \in S \cap \mathrm{dom}(\overline{\xi_n})} [\![\overline{c}^{\pi}]\!]\overline{\sigma}(pr_\mu) \cdot \prod_{\mu \in S \setminus \mathrm{dom}(\overline{\xi_n})} [\![\overline{c}^{\pi}]\!]\overline{\sigma}(pr_\mu) \\
&= prs^{\langle S \rangle}_{\square}(\overline{c}^{\pi})(\sigma_p, \overline{\xi_n}) \cdot \prod_{\mu \in S \setminus \mathrm{dom}(\overline{\xi_n})} \overline{\sigma}(pr_\mu) \\
&= prs^{\langle S \rangle}_{\square}(c)(\sigma_p, \xi_n) \cdot \prod_{\mu \in S \setminus \mathrm{dom}(\xi_n)} \sigma(pr_\mu) \\
&= \prod_{\mu \in S \cap \mathrm{dom}(\xi_n)} [\![c]\!]\sigma(pr_\mu) \cdot \prod_{\mu \in S \setminus \mathrm{dom}(\xi_n)} [\![c]\!]\sigma(pr_\mu) \\
&= p^{\langle S \rangle}_{c, \sigma_\theta}(v_{\overline{c}^{\pi}, \sigma_\theta}(\sigma_n)).
\end{aligned}$$

The first and last equalities are by the definition of p. The second equality uses $used(\overline{c}^{\pi}, \sigma_p \oplus \overline{\xi_n} \oplus \overline{\sigma_r}, \overline{\xi_n})$ and Lemma C.6-(2) with $\mu \notin \mathrm{dom}(\overline{\xi_n})$. The third equality uses $\mathrm{dom}(\overline{\xi_n}) = \mathrm{dom}(\xi_n)$, the observation made in the first paragraph, and the second claim in the above. The fourth equality uses the first claim in the above, and Lemma C.6-(2) with $\mu \notin \mathrm{dom}(\xi_n)$. □

Lemma D.5. *Let c be a command and $\sigma \in \mathsf{St}$. If c has no observe commands and $[\![c]\!]\sigma \in \mathsf{St}$, then*

$$[\![c]\!]\sigma(like) = \sigma(like).$$

Proof. Let c be a command that does not contain an observe command. We show the claim of the lemma by induction on the structure of c. Pick $\sigma \in \mathsf{St}$ such that $[\![c]\!]\sigma \in \mathsf{St}$. We will show that $[\![c]\!]\sigma(like) = \sigma(like)$.

Case $c \equiv \mathtt{skip}$**.** In this case, $[\![c]\!]\sigma(like) = \sigma(like)$ by the definition of the semantics.

Case $c \equiv (x := e)$**.** Again, $[\![c]\!]\sigma(like) = \sigma(like)$ by the definition of the semantics.

Case $c \equiv (x := \mathtt{sam}(n, d, \lambda y.e'))$**.** Once more, $[\![c]\!]\sigma(like) = \sigma(like)$ by the definition of the semantics.

Case $c \equiv (c'; c'')$**.** We have $[\![c']\!]\sigma \in \mathrm{St}$ and $[\![c'']\!]([\![c']\!]\sigma) \in \mathrm{St}$. We apply induction hypothesis first to (c', σ), and again to $(c', [\![c']\!]\sigma)$. The first application gives $[\![c']\!]\sigma(\mathit{like}) = \sigma(\mathit{like})$, and the second $[\![c'; c'']\!]\sigma(\mathit{like}) = [\![c']\!]\sigma(\mathit{like})$. The desired conclusion follows from these two equalities.

Case $c \equiv (\texttt{if}\ b\ \{c'\}\ \texttt{else}\ \{c''\})$**.** We deal with the case that $[\![b]\!]\sigma = \mathit{true}$. The other case of $[\![b]\!]\sigma = \mathit{false}$ can be proved similarly. Since $[\![b]\!]\sigma = \mathit{true}$, we have $[\![c']\!]\sigma = [\![c]\!]\sigma \in \mathrm{St}$. Thus, we can apply induction hypothesis to c'. If we do so, we get $[\![c']\!]\sigma(\mathit{like}) = \sigma(\mathit{like})$. This gives the desired conclusion because $[\![c]\!]\sigma = [\![c']\!]\sigma$.

Case $c \equiv (\texttt{while}\ b\ \{c'\})$**.** Let F be the operator on $[\mathrm{St} \to \mathrm{St}_\perp]$ such that $[\![c]\!]$ is the least fixed point of F. Define a subset $\mathcal{T}$ of $[\mathrm{St} \to \mathrm{St}_\perp]$ as follows:

$$f \in \mathcal{T} \iff \Big(\forall \sigma' \in \mathrm{St}.\, f(\sigma') \in \mathrm{St} \implies f(\sigma')(\mathit{like}) = \sigma'(\mathit{like}).\Big)$$

The set $\mathcal{T}$ contains the least function $\lambda\sigma.\perp$, and is closed under the least upper bound of any chain in $[\mathrm{St} \to \mathrm{St}_\perp]$. It is also closed under F. This F-closure follows essentially from our arguments for sequential composition, if command, and skip, and induction hypothesis on c'. What we have shown for $\mathcal{T}$ implies that $\mathcal{T}$ contains the least fixed point of F, which gives the desired property for c. □

Lemma D.6. *Let c be a command and π be a reparameterisation plan. Suppose that every sample command in c has $\lambda y.y$ as its third argument. Then, for all $\sigma_p \in \mathrm{St}[\mathrm{PVar}]$ and $\overline{\xi_n} \in \mathrm{St}_\square[\mathrm{Name}]$, if $\mathit{used}(\overline{c}^\pi, \sigma_p \oplus \overline{\xi_n} \oplus \overline{\sigma_r}, \overline{\xi_n})$ for some $\overline{\sigma_r}$, then*

$$\exists \sigma_r.\ \mathit{used}(c, \sigma_p \oplus \xi_n \oplus \sigma_r, \xi_n),$$
$$\mathit{pvars}_\square(\overline{c}^\pi)(\sigma_p, \overline{\xi_n}) = \mathit{pvars}_\square(c)(\sigma_p, \xi_n),$$
$$\mathit{prs}_\square^{\langle \mathrm{Name} \setminus rv(\pi)\rangle}(\overline{c}^\pi)(\sigma_p, \overline{\xi_n}) = \mathit{prs}_\square^{\langle \mathrm{Name} \setminus rv(\pi)\rangle}(c)(\sigma_p, \xi_n),$$

where

$$\xi_n \triangleq \mathit{vals}_\square(\overline{c}^\pi)(\sigma_p, \overline{\xi_n}).$$

Proof. Fix a reparameterisation plan π. The proof proceeds by induction on the structure of c. Let $\sigma_p \in \mathrm{St}[\mathrm{PVar}]$, and $\overline{\xi_n} \in \mathrm{St}_\square[\mathrm{Name}]$. Assume that c uses only $\lambda y.y$ in the third argument of its sample commands, and $\mathit{used}(\overline{c}^\pi, \sigma_p \oplus \overline{\xi_n} \oplus \overline{\sigma_r}, \overline{\xi_n})$ for some $\overline{\sigma_r}$. Let $\overline{\sigma} \triangleq \sigma_p \oplus \overline{\xi_n} \oplus \overline{\sigma_r}$ and $S \triangleq \mathrm{Name} \setminus rv(\pi)$. Then, we simply have $\mathit{used}(\overline{c}^\pi, \overline{\sigma}, \overline{\xi_n})$.

Cases $c \equiv \texttt{skip}$**,** $c \equiv (x := e)$**, or** $c \equiv \texttt{obs}(d, r)$**.** In this case, $[\![c]\!]\sigma(\mathit{cnt}_\mu) = \sigma(\mathit{cnt}_\mu)$ for all $\sigma \in \mathrm{St}$ and $\mu \in \mathrm{Name}$. So $\mathrm{dom}(\overline{\xi_n}) = \mathrm{dom}(\xi_n) = \emptyset$ and thus $\overline{\xi_n} = \xi_n$. We also know $\overline{c}^\pi \equiv c$. From these, all of the three conclusions follow immediately.

Case $c \equiv (x := \texttt{sam}(n, d, \lambda y.e))$**.** Since $fv(n) \subseteq \mathrm{PVar}$, there exists $\mu \in \mathrm{Name}$ such that $[\![n]\!](\sigma_p \oplus \sigma_r) = \mu$ for all $\sigma_r \in \mathrm{St}[\mathrm{Var} \setminus \mathrm{PVar}]$. So, for all $\sigma_r \in \mathrm{St}[\mathrm{Var} \setminus \mathrm{PVar}]$ and $\mu' \in \mathrm{Name} \setminus \{\mu\}$,

$$[\![c]\!](\sigma_p \oplus \sigma_r)(\mathit{cnt}_{\mu'}) = \begin{cases} (\sigma_p \oplus \sigma_r)(\mathit{cnt}_{\mu'}) + 1 & \text{if } \mu' = \mu \\ (\sigma_p \oplus \sigma_r)(\mathit{cnt}_{\mu'}) & \text{otherwise.} \end{cases} \tag{22}$$

From this, we get $\mathrm{dom}(\overline{\xi_n}) = \mathrm{dom}(\xi_n) = \{\mu\}$. Further, by assumption, we get $e \equiv y$. We now prove the three conclusions based on these observations and case analysis on $(n, d, \lambda y.e)$.

First, assume $(n, d, \lambda y.e) \notin \mathrm{dom}(\pi)$. Then, $\overline{c}^\pi \equiv c$ and

$$\xi_n = \mathit{vals}_\square(\overline{c}^\pi)(\sigma_p, \overline{\xi_n}) = [\mu \mapsto [\![\overline{c}^\pi]\!]\overline{\sigma}(\mathit{val}_\mu)] = [\mu \mapsto [\![e[\overline{\sigma}(\mu)/y]]\!]\overline{\sigma}] = [\mu \mapsto \overline{\xi_n}(\mu)] = \overline{\xi_n},$$

where the second last equality uses $e \equiv y$. Hence, the three conclusions clearly hold.

Next, assume $(n, d, \lambda y.e) \in \mathrm{dom}(\pi)$. Suppose that $\pi(n, d, \lambda y.e) = (\overline{d}, \lambda\overline{y}.\overline{e})$. Then, $\overline{c}^{\pi} \equiv (x := \mathsf{sam}(n, \overline{d}, \lambda\overline{y}.\overline{e}))$ and

$$\xi_n = vals_{\square}(\overline{c}^{\pi})(\sigma_p, \overline{\xi_n}) = [\mu \mapsto [\![\overline{c}^{\pi}]\!]\overline{\sigma}(val_{\mu})] = [\mu \mapsto [\![\overline{e}[\overline{\sigma}(\mu)/\overline{y}]]\!]\overline{\sigma}] = [\mu \mapsto [\![\overline{e}[\overline{\xi_n}(\mu)/\overline{y}]]\!]\overline{\sigma}].$$

Since Eq. (22) holds also for $[\![\overline{c}^{\pi}]\!]$, and since $\mathrm{dom}(\xi_n) = \{\mu\}$, we get the first conclusion:

$$used(c, \sigma_p \oplus \xi_n \oplus \overline{\sigma_r}, \xi_n).$$

To prove the second conclusion, let $\sigma = \sigma_p \oplus \xi_n \oplus \overline{\sigma_r}$. Then, for all $v \in \mathrm{PVar}$,

$$pvars_{\square}(\overline{c}^{\pi})(\sigma_p, \overline{\xi_n})(v) = [\![\overline{c}^{\pi}]\!]\overline{\sigma}(v) = \begin{cases} [\![\overline{e}[\overline{\sigma}(\mu)/\overline{y}]]\!]\overline{\sigma} = [\![\overline{e}[\overline{\xi_n}(\mu)/\overline{y}]]\!]\overline{\sigma} & \text{if } v \equiv x \\ \overline{\sigma}(v) = \sigma_p(v) & \text{otherwise,} \end{cases}$$

$$pvars_{\square}(c)(\sigma_p, \xi_n)(v) = [\![c]\!]\sigma(v) = \begin{cases} [\![e[\sigma(\mu)/y]]\!]\sigma = \xi_n(\mu) = [\![\overline{e}[\overline{\xi_n}(\mu)/\overline{y}]]\!]\overline{\sigma} & \text{if } v \equiv x \\ \sigma(v) = \sigma_p(v), & \text{otherwise,} \end{cases}$$

where the second equation uses $e \equiv y$. Hence, the second conclusion holds. For the third conclusion, let $n = \mathsf{name}(\alpha, \_)$. Then, $\mu = (\alpha, \_) \in \{(\alpha, i) \in \mathsf{Name} \mid i \in \mathbb{N}\} \subseteq rv(\pi)$. Thus, $\mathrm{dom}(\overline{\xi_n}) \cap S = \mathrm{dom}(\xi_n) \cap S = \{\mu\} \cap S = \{\mu\} \cap (\mathsf{Name} \setminus rv(\pi)) = \emptyset$. From this, we get the third conclusion:

$$prs_{\square}^{\langle S \rangle}(\overline{c}^{\pi})(\sigma_p, \overline{\xi_n}) = 1 = prs_{\square}^{\langle S \rangle}(c)(\sigma_p, \xi_n).$$

Case $c \equiv (c'; c'')$. First, we make several observations necessary to prove the conclusion. By $used(\overline{c}^{\pi}, \overline{\sigma}, \overline{\xi_n})$, we have $[\![\overline{c'}^{\pi}]\!]\overline{\sigma} \in \mathrm{St}$ and $[\![\overline{c''}^{\pi}]\!]([\![\overline{c'}^{\pi}]\!]\overline{\sigma}) \in \mathrm{St}$. Let

$$\overline{\sigma'} \triangleq [\![\overline{c'}^{\pi}]\!]\overline{\sigma}, \qquad \overline{\sigma''} \triangleq [\![\overline{c''}^{\pi}]\!]\overline{\sigma'}, \qquad \sigma'_p \triangleq \overline{\sigma'}|_{\mathrm{PVar}}, \qquad \sigma''_p \triangleq \overline{\sigma''}|_{\mathrm{PVar}}.$$

Then, by $used(\overline{c}^{\pi}, \overline{\sigma}, \overline{\xi_n})$ and the claim in the proof of Lemma C.6 (for the sequential composition case), there exist $\overline{\xi'_n}$ and $\overline{\xi''_n}$ such that

$$\overline{\xi_n} = \overline{\xi'_n} \oplus \overline{\xi''_n}, \qquad used(\overline{c'}^{\pi}, \overline{\sigma}, \overline{\xi'_n}), \qquad used_{\_}(\overline{c''}^{\pi}, \overline{\sigma'}, \overline{\xi''_n}).$$

By the latter two, we can apply induction to $(c', \sigma_p, \overline{\xi'_n})$ and $(c'', \sigma'_p, \overline{\xi''_n})$, and IH gives the following:

$$\begin{aligned} pvars_{\square}(c')(\sigma_p, \xi'_n) &= pvars_{\square}(\overline{c'}^{\pi})(\sigma_p, \overline{\xi'_n}) && \text{[By IH on } c'\text{]} \\ &= ([\![\overline{c'}^{\pi}]\!](\overline{\sigma}[like \mapsto 1]))|_{\mathrm{PVar}} && \text{[By } used(\overline{c'}^{\pi}, \overline{\sigma}, \overline{\xi'_n})\text{]} \\ &= ([\![\overline{c'}^{\pi}]\!]\overline{\sigma})|_{\mathrm{PVar}} = \overline{\sigma'}|_{\mathrm{PVar}} = \sigma'_p, && \text{[By Lemma C.6-(3)]} \\ pvars_{\square}(c'')(\sigma'_p, \xi''_n) &= pvars_{\square}(\overline{c''}^{\pi})(\sigma'_p, \overline{\xi''_n}) && \text{[By IH on } c''\text{]} \\ &= ([\![\overline{c''}^{\pi}]\!](\overline{\sigma'}[like \mapsto 1]))|_{\mathrm{PVar}} && \text{[By } used_{\_}(\overline{c''}^{\pi}, \overline{\sigma'}, \overline{\xi''_n})\text{]} \\ &= ([\![\overline{c''}^{\pi}]\!]\overline{\sigma'})|_{\mathrm{PVar}} = \overline{\sigma''}|_{\mathrm{PVar}} = \sigma''_p, && \text{[By Lemma C.6-(3)]} \end{aligned}$$

where

$$\xi'_n \triangleq vals_{\square}(\overline{c'}^{\pi})(\sigma_p, \overline{\xi'_n}), \qquad \xi''_n \triangleq vals_{\square}(\overline{c''}^{\pi})(\sigma'_p, \overline{\xi''_n}).$$

By the former equation, we get

$$\begin{aligned} \xi_n &= vals_{\square}(\overline{c}^{\pi})(\sigma_p, \overline{\xi_n}) \\ &= vals_{\square}(\overline{c'}^{\pi}; \overline{c''}^{\pi})(\sigma_p, \overline{\xi'_n} \oplus \overline{\xi''_n}) && \text{[By } \overline{\xi_n} = \overline{\xi'_n} \oplus \overline{\xi''_n}\text{]} \\ &= vals_{\square}(\overline{c'}^{\pi})(\sigma_p, \overline{\xi'_n}) \oplus vals_{\square}(\overline{c''}^{\pi})(pvars_{\square}(\overline{c'}^{\pi})(\sigma_p, \overline{\xi'_n}), \overline{\xi''_n}) \\ &= vals_{\square}(\overline{c'}^{\pi})(\sigma_p, \overline{\xi'_n}) \oplus vals_{\square}(\overline{c''}^{\pi})(\sigma'_p, \overline{\xi''_n}) && \text{[By the former equation]} \end{aligned}$$

$$= \xi'_n \oplus \xi''_n$$

where the third equality uses $used(\overline{c'}^{\pi}, \overline{\sigma}, \overline{\xi'_n})$, $used(\overline{c'}^{\pi}; \overline{c''}^{\pi}, \overline{\sigma}, \overline{\xi'_n} \oplus \overline{\xi''_n})$, and the second claim in the proof of Lemma C.9.

We now show the first conclusion. By IH on $(c', \sigma_p, \overline{\xi'_n})$ and $(c'', \sigma'_p, \overline{\xi''_n})$, we get

$$\exists \sigma'_r.\ used(c', \sigma_p \oplus \xi'_n \oplus \sigma'_r, \xi'_n), \qquad \exists \sigma''_r.\ used(c'', \sigma'_p \oplus \xi''_n \oplus \sigma''_r, \xi''_n).$$

Let

$$\sigma \triangleq \sigma_p \oplus (\xi'_n \oplus \xi''_n) \oplus \sigma'_r|_{\mathrm{dom}(\sigma'_r) \setminus \mathrm{dom}(\xi''_n)}.$$

Then, $[\![c']\!]\sigma \in \mathrm{St}$ by $used(c', \sigma_p \oplus \xi'_n \oplus \sigma'_r, \xi'_n)$ and Lemma C.6-(1), and we have

$$\begin{aligned}
\sigma|_{\mathsf{PVar}} = \sigma_p, \qquad ([\![c']\!]\sigma)|_{\mathsf{PVar}} &= ([\![c']\!](\sigma_p \oplus \xi'_n \oplus \sigma'_r))|_{\mathsf{PVar}} && \text{[By Lemma C.6-(3)]}\\
&= pvars(c')(\sigma_p, \xi'_n) = \sigma'_p, && \text{[By the above]}\\
\sigma|_{\mathrm{dom}(\xi'_n)} = \xi'_n, \qquad ([\![c']\!]\sigma)|_{\mathrm{dom}(\xi''_n)} &= \sigma|_{\mathrm{dom}(\xi''_n)} = \xi''_n. && \text{[By Lemma C.4]}
\end{aligned}$$

By these, $used(c', \sigma_p \oplus \xi'_n \oplus \sigma'_r, \xi'_n)$, $used(c'', \sigma'_p \oplus \xi''_n \oplus \sigma''_r, \xi''_n)$, and Lemma C.7, we get

$$used(c', \sigma, \xi'_n), \qquad used_{\_}(c'', [\![c']\!]\sigma, \xi''_n).$$

By these and the third claim in the proof of Lemma C.9, we get

$$used(c'; c'', \sigma, \xi'_n \oplus \xi''_n).$$

By this and Lemma C.7, we get the following as desired, since $\xi_n = \xi'_n \oplus \xi''_n$ (shown in the above) and $\sigma = \sigma_p \oplus (\xi'_n \oplus \xi''_n) \oplus \sigma_r$ for some σ_r:

$$used(c, \sigma_p \oplus \xi_n \oplus \sigma_r, \xi_n).$$

Next, we show the second conclusion as follows:

$$\begin{aligned}
pvars_{\square}(\overline{c}^{\pi})(\sigma_p, \xi_n) &= ([\![\overline{c}^{\pi}]\!]\overline{\sigma})|_{\mathsf{PVar}} && \text{[By } used(\overline{c}^{\pi}, \overline{\sigma}, \overline{\xi_n})]\\
&= ([\![\overline{c''}^{\pi}]\!]([\![\overline{c'}^{\pi}]\!]\overline{\sigma}))|_{\mathsf{PVar}}\\
&= \overline{\sigma''}|_{\mathsf{PVar}} = \sigma''_p,\\
pvars_{\square}(c)(\sigma_p, \xi_n) &= pvars_{\square}(c'; c'')(\sigma_p, \xi'_n \oplus \xi''_n) && \text{[By } \xi_n = \xi'_n \oplus \xi''_n]\\
&= pvars_{\square}(c'')(pvars_{\square}(c')(\sigma_p, \xi'_n), \xi''_n)\\
&= pvars_{\square}(c'')(\sigma'_p, \xi''_n) = \sigma''_p, && \text{[By the above]}
\end{aligned}$$

where the second last equality uses $used(c', \sigma, \xi'_n)$, $used(c'; c'', \sigma, \xi'_n \oplus \xi''_n)$, and the second claim in the proof of Lemma C.9.

Lastly, we show the third conclusion as follows:

$$\begin{aligned}
prs_{\square}^{\langle S \rangle}(\overline{c}^{\pi})(\sigma_p, \overline{\xi_n}) &= prs_{\square}^{\langle S \rangle}(\overline{c'; c''}^{\pi})(\sigma_p, \overline{\xi'_n} \oplus \overline{\xi''_n}) && \text{[By } \overline{\xi_n} = \overline{\xi'_n} \oplus \overline{\xi''_n}]\\
&= prs_{\square}^{\langle S \rangle}(\overline{c'}^{\pi})(\sigma_p, \overline{\xi'_n}) \cdot prs_{\square}^{\langle S \rangle}(\overline{c''}^{\pi})(pvars(\overline{c'}^{\pi})(\sigma_p, \overline{\xi'_n}), \overline{\xi''_n})\\
&= prs_{\square}^{\langle S \rangle}(\overline{c'}^{\pi})(\sigma_p, \overline{\xi'_n}) \cdot prs_{\square}^{\langle S \rangle}(\overline{c''}^{\pi})(pvars(c')(\sigma_p, \xi'_n), \overline{\xi''_n}) && \text{[By IH on } c']\\
&= prs_{\square}^{\langle S \rangle}(c')(\sigma_p, \xi'_n) \cdot prs_{\square}^{\langle S \rangle}(c'')(pvars(c')(\sigma_p, \xi'_n), \xi''_n) && \text{[By IH on } c' \text{ and } c'']\\
&= prs_{\square}^{\langle S \rangle}(c)(\sigma_p, \xi'_n \oplus \xi''_n)\\
&= prs_{\square}^{\langle S \rangle}(c)(\sigma_p, \xi_n). && \text{[By } \xi_n = \xi'_n \oplus \xi''_n]
\end{aligned}$$

Here the second and fifth equalities use the second claim in the proof of Lemma C.9 with the following: $used(\overline{c'}^{\pi}, \overline{\sigma}, \overline{\xi'_n})$, $used(\overline{c'}^{\pi}; \overline{c''}^{\pi}, \overline{\sigma}, \overline{\xi'_n} \oplus \overline{\xi''_n})$, $used(c', \sigma, \xi'_n)$, and $used(c'; c'', \sigma, \xi'_n \oplus \xi''_n)$.

Case $c \equiv (\texttt{if}\ b\ \{c'\}\ \texttt{else}\ \{c''\})$**.** In this case, $\overline{c}^{\pi} \equiv (\texttt{if}\ b\ \{\overline{c'}^{\pi}\}\ \texttt{else}\ \{\overline{c''}^{\pi}\})$. Since $fv(b) \subseteq \mathrm{PVar}$, $[\![b]\!](\sigma_p \oplus \sigma_r)$ is constant for all $\sigma_r \in \mathrm{St}[\mathrm{Var} \setminus \mathrm{PVar}]$. Without loss of generality, assume $[\![b]\!](\sigma_p \oplus \sigma_r) = \mathit{true}$. Then, $[\![c]\!](\sigma_p \oplus \sigma_r) = [\![c']\!](\sigma_p \oplus \sigma_r)$ and $[\![\overline{c}^{\pi}]\!](\sigma_p \oplus \sigma_r) = [\![\overline{c'}^{\pi}]\!](\sigma_p \oplus \sigma_r)$ for all $\sigma_r \in \mathrm{St}[\mathrm{Var} \setminus \mathrm{PVar}]$. Hence, by IH on $(c', \sigma_p, \overline{\xi_n})$, we get the three conclusions directly.

Case $c \equiv (\texttt{while}\ b\ \{c'\})$**.** In this case, $\overline{c}^{\pi} \equiv (\texttt{while}\ b\ \{\overline{c'}^{\pi}\})$. Consider the version of $prs_{\square}(-)$ where the parameter can be a state transformer $\overline{f} : \mathrm{St} \to \mathrm{St}_{\perp}$, instead of a command. Similarly, consider the version of the three conclusions where we use two state transformers $\overline{f}, f : \mathrm{St} \to \mathrm{St}_{\perp}$, again instead of a command. We denote the versions by $prs_{\square}(\overline{f})$ and $\varphi(\overline{f}, f, \sigma_p, \overline{\xi_n})$. We write $\overline{f} \sim f$ if $prs_{\square}(\overline{f})(\sigma_p, \overline{\xi_n}) > 0$ implies $\varphi(\overline{f}, f, \sigma_p, \overline{\xi_n})$ for all $\sigma_p \in \mathrm{St}[\mathrm{PVar}]$ and $\overline{\xi_n} \in \mathrm{St}_{\square}[\mathrm{Name}]$. Further, we define $F^{\pi'} : [\mathrm{St} \to \mathrm{St}_{\perp}] \to [\mathrm{St} \to \mathrm{St}_{\perp}]$ as

$$F^{\pi'}(f)(\sigma) \triangleq \text{if } ([\![b]\!]\sigma = \mathit{true}) \text{ then } (f^{\dagger} \circ [\![\overline{c'}^{\pi'}]\!])(\sigma) \text{ else } \sigma.$$

Note that F^{π} and F^{π_0} are the operators used in the semantics of the loops $[\![\overline{c}^{\pi}]\!]$ and $[\![c]\!]$, respectively, where π_0 denotes the empty reparameterisation plan. We will show three claims: $\lambda\sigma.\perp \sim \lambda\sigma.\perp$; if $\overline{f} \sim f$, then $F^{\pi}(\overline{f}) \sim F^{\pi_0}(f)$; and if increasing sequences $\{\overline{f_k}\}_{k\in\mathbb{N}}$ and $\{f_k\}_{k\in\mathbb{N}}$ satisfy $\overline{f_k} \sim f_k$ for all $k \in \mathbb{N}$, then $\overline{f_\infty} \sim f_\infty$ holds for $\overline{f_\infty} = \bigsqcup_{k\in\mathbb{N}} \overline{f_k}$ and $f_\infty = \bigsqcup_{k\in\mathbb{N}} f_k$. These three claims imply $[\![\overline{c}^{\pi}]\!] \sim [\![c]\!]$, which in turn proves the desired three conclusions.

The first claim holds simply because $prs_{\square}(\lambda\sigma.\perp)(-,-)$ is always 0. To show the second claim, consider $\overline{f}, f : \mathrm{St} \to \mathrm{St}_{\perp}$ such that $\overline{f} \sim f$. We first replay our proof for the sequential-composition case on $(\overline{f}, f)$ after viewing $\overline{f}^{\dagger} \circ [\![\overline{c'}^{\pi}]\!]$ and $f^{\dagger} \circ [\![c']\!]$ as the sequential composition of $\overline{c'}^{\pi}$ and $\overline{f}$, and of c' and f, respectively. This replay, then, gives the relationship $\overline{f}^{\dagger} \circ [\![\overline{c'}^{\pi}]\!] \sim f^{\dagger} \circ [\![c']\!]$. Next, we replay our proof for the if case on $(F^{\pi}(\overline{f}), F^{\pi_0}(f))$, after viewing $\overline{f}^{\dagger} \circ [\![\overline{c'}^{\pi}]\!]$ and $f^{\dagger} \circ [\![c']\!]$ as the true branches, and $\lambda\sigma.\,\sigma = [\![\texttt{skip}]\!]$ as the false branch. This replay implies the required relationship $F^{\pi}(\overline{f}) \sim F^{\pi_0}(f)$.

To show the third condition, consider increasing sequences $\{\overline{f_k}\}_{k\in\mathbb{N}}$ and $\{f_k\}_{k\in\mathbb{N}}$ such that $\overline{f_k} \sim f_k$ for all $k \in \mathbb{N}$. Let $\overline{f_\infty} = \bigsqcup_{k\in\mathbb{N}} \overline{f_k}$ and $f_\infty = \bigsqcup_{k\in\mathbb{N}} f_k$. Consider any σ_p and $\overline{\xi_n}$ such that $prs_{\square}(\overline{f_\infty})(\sigma_p, \overline{\xi_n}) > 0$. We should show $\varphi(\overline{f_\infty}, f_\infty, \sigma_p, \overline{\xi_n})$. Pick any $\sigma_r \in \mathrm{St}[\mathrm{Var} \setminus (\mathrm{PVar} \cup \mathrm{dom}(\overline{\xi_n}))]$ with $\sigma_r(\mathit{like}) = 1$. Let $\overline{\sigma} = \sigma_p \oplus \overline{\xi_n} \oplus \sigma_r$ and $\sigma = \sigma_p \oplus \xi_n \oplus \sigma_r$. Note that the value of each term in $\varphi(\cdots)$ (i.e., $\mathit{used}(\cdots)$, $pvars_{\square}(\cdots)$, and $prs_{\square}^{\langle S \rangle}(\cdots)$) is independent of the choice of σ_r by Lemma C.7 and the well-definedness of $pvars_{\square}$ and $prs_{\square}^{\langle S \rangle}$. Since the two given sequences are increasing, there exists $K \in \mathbb{N}$ such that $\overline{f_\infty}(\overline{\sigma}) = \overline{f_K}(\overline{\sigma})$ and $f_\infty(\sigma) = f_K(\sigma)$. From this and $prs_{\square}(\overline{f_\infty})(\sigma_p, \overline{\xi_n}) > 0$, we have $prs_{\square}(\overline{f_K})(\sigma_p, \overline{\xi_n}) > 0$. This in turn gives $\varphi(\overline{f_K}, f_K, \sigma_p, \overline{\xi_n})$ since $\overline{f_K} \sim f_K$. Lastly, again by $\overline{f_\infty}(\overline{\sigma}) = \overline{f_K}(\overline{\sigma})$ and $f_\infty(\sigma) = f_K(\sigma)$, we obtain $\varphi(\overline{f_\infty}, f_\infty, \sigma_p, \overline{\xi_n})$ as desired. □

E DEFERRED RESULTS IN §4.3

E.1 Deferred Statements and Their Proofs

Lemma E.1. *Let $f, g : \mathbb{R}^n \to \mathbb{R}$ be locally Lipschitz functions. Then, the following differentiation rules hold for almost every $x \in \mathbb{R}^n$:*

$$\begin{aligned} \nabla(f+g)(x) &= \nabla f(x) + \nabla g(x), \\ \nabla(f \cdot g)(x) &= \nabla f(x) \cdot g(x) + f(x) \cdot \nabla g(x), \\ \nabla \log f(x) &= 1/f(x) \cdot \nabla f(x), \end{aligned}$$

where for the third rule we assume $f(y) > 0$ for all $y \in \mathbb{R}^n$.

Proof. Note that the three functions $+ : \mathbb{R}^2 \to \mathbb{R}$, $\cdot : \mathbb{R}^2 \to \mathbb{R}$, and $\log : \mathbb{R}_{>0} \to \mathbb{R}$ are all differentiable. Hence, applying Lemma E.4 produces the claim. □

Theorem E.2. *Let $f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}$ be a measurable function that satisfies the next four conditions:*

- *For all $x \in \mathbb{R}^n$, $f(-, x) : \mathbb{R} \to \mathbb{R}$ is continuous.*
- *For all $\theta \in \mathbb{R}$, $\nabla_\theta f(\theta, x)$ is well-defined for almost all $x \in \mathbb{R}^n$.*
- *For all $\theta \in \mathbb{R}$, $\int_{\mathbb{R}^n} f(\theta, x)\, dx$ is finite.*
- *For all $\theta \in \mathbb{R}$, there is an open $U \subseteq \mathbb{R}$ such that $\theta \in U$ and $\int_{\mathbb{R}^n} \mathrm{Lip}\big(f(-, x)|_U\big)\, dx$ is finite.*

Here $\nabla_\theta(-)$ and $\mathrm{Lip}(-)$ are defined as in Theorem D.2, and "almost all" is with respect to the Lebesgue measure. Then, for all $\theta \in \mathbb{R}$, both sides of the following are well-defined and equal:

$$\nabla_\theta \int_{\mathbb{R}^n} f(\theta, x)\, dx = \int_{\mathbb{R}^n} \nabla_\theta f(\theta, x)\, dx.$$

Proof. Before starting the main proof, we show that for any open $U' \subseteq \mathbb{R}$, the following function $L : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ is measurable:

$$L(x) \triangleq \mathrm{Lip}\big(f(-, x)|_{U'}\big).$$

Define $V \subseteq \mathbb{R}^2$ and $\ell : V \times \mathbb{R}^n \to \mathbb{R}$ as

$$V \triangleq \{v \in U' \times U' \mid v_1 \neq v_2\} \subseteq \mathbb{R}^2, \qquad \ell(v, x) \triangleq \frac{|f(v_1, x) - f(v_2, x)|}{|v_1 - v_2|}.$$

Let V' be a countable, dense subset of V. Then, for all $x \in \mathbb{R}^n$,

$$L(x) = \sup_{v \in V} \ell(v, x) = \sup_{v' \in V'} \ell(v', x),$$

where the first equality is by the definition of $\mathrm{Lip}(-)$, and the second equality holds since $\ell(-, x) : V \to \mathbb{R}$ is continuous for all $x \in \mathbb{R}^n$ by the first condition of this theorem. Since L is the supremum of countably many measurable functions $\{\ell(v', -) : \mathbb{R}^n \to \mathbb{R} \mid v' \in V'\}$, L itself is a measurable function as desired. Note that the measurability of L ensures that the integral $\int_{\mathbb{R}^n} L(x)\, dx$ in the fourth condition of this theorem is well-defined (as a value in $\mathbb{R} \cup \{\infty\}$).

We now start the main proof. Pick any $\theta' \in \mathbb{R}$. Define $g : \mathbb{R} \setminus \{\theta'\} \to \mathbb{R}$ as

$$g(\theta) \triangleq \int_{\mathbb{R}^n} \frac{f(\theta, x) - f(\theta', x)}{\theta - \theta'}\, dx,$$

where $g(\theta)$ is finite by the third condition of this theorem. Then,

$$\Big(\nabla_\theta \int_{\mathbb{R}^n} f(\theta, x)\, dx\Big)\Big|_{\theta=\theta'} = \lim_{\theta \to \theta'} \frac{1}{\theta - \theta'} \Big(\int_{\mathbb{R}^n} f(\theta, x)\, dx - \int_{\mathbb{R}^n} f(\theta', x)\, dx \Big) = \lim_{\theta \to \theta'} g(\theta),$$

where each equality denotes that LHS is well-defined if and only if RHS is well-defined, and if so, LHS and RHS are equal. Here the first equality is from the definition of ∇_θ, and the second equality from the third condition of this theorem. Note that the following are equivalent for any $r \in \mathbb{R}$:

(i) $\lim_{\theta \to \theta'} g(\theta) = r$.
(ii) For any $\{\theta_i\}_{i \in \mathbb{N}} \subseteq \mathbb{R} \setminus \{\theta'\}$, $\lim_{i \to \infty} \theta_i = \theta'$ implies $\lim_{i \to \infty} g(\theta_i) = r$.

So it suffices to show that (ii) holds with an appropriate choice of r.

To do so, consider $\{\theta_i\}_{i \in \mathbb{N}} \subseteq \mathbb{R} \setminus \{\theta'\}$ such that $\lim_{i \to \infty} \theta_i = \theta'$. Define $h_i : \mathbb{R}^n \to \mathbb{R}$ as

$$h_i(x) \triangleq \frac{f(\theta_i, x) - f(\theta', x)}{\theta_i - \theta'}.$$

Then, $g(\theta_i) = \int_{\mathbb{R}^n} h_i(x)\,dx$ by the definition of h_i, and $\lim_{i\to\infty} h_i(x) = (\nabla_\theta f(\theta, x))|_{\theta=\theta'}$ for almost all $x \in \mathbb{R}^n$ by the second condition of this theorem. So, if the dominated convergence theorem is applicable to $\lim_{i\to\infty} \int_{\mathbb{R}^n} h_i(x)\,dx$, we would have

$$\lim_{i\to\infty} g(\theta_i) = \lim_{i\to\infty} \int_{\mathbb{R}^n} h_i(x)\,dx = \int_{\mathbb{R}^n} \Big(\nabla_\theta f(\theta, x)\Big)\Big|_{\theta=\theta'} dx,$$

where each equality denotes that both LHS and RHS are well-defined and are equal. Therefore, it suffices to show that the preconditions of the dominated convergence theorem for $\lim_{i\to\infty} \int_{\mathbb{R}^n} h_i(x)\,dx$ are satisfied.

We show that the preconditions indeed hold, which concludes the proof:

- "h_i is measurable for all $i \in \mathbb{N}$": This holds by the measurability of f.
- "$\lim_{i\to\infty} h_i(x) = (\nabla_\theta f(\theta, x))|_{\theta=\theta'}$ for almost all $x \in \mathbb{R}^n$": This was already shown above.
- "There exist $H : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ and $I \in \mathbb{N}$ such that (a) $\int_{\mathbb{R}^n} H(x)\,dx$ is finite, and (b) for all $i \geq I$, $|h_i(x)| \leq H(x)$ for almost all $x \in \mathbb{R}^n$": By the fourth condition of this theorem, there is an open $U \subseteq \mathbb{R}$ such that $\theta' \in U$ and $\int_{\mathbb{R}^n} \mathrm{Lip}(f(-, x)|_U)\,dx$ is finite. Let $H : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ be

  $$H(x) \triangleq \mathrm{Lip}\big(f(-, x)|_U\big).$$

  Then, (a) clearly holds since $\int_{\mathbb{R}^n} \mathrm{Lip}(f(-, x)|_U)\,dx$ is finite. Further, (b) holds as follows: by $\lim_{i\to\infty} \theta_i = \theta'$ and U being an open neighborhood of θ', there is $I \in \mathbb{N}$ such that $\theta_i \in U$ for all $i \geq I$; therefore, for all $i \geq I$,

  $$|h_i(x)| = \frac{|f(\theta_i, x) - f(\theta', x)|}{|\theta_i - \theta'|} \leq \mathrm{Lip}\big(f(-, x)|_U\big) = H(x) \quad \text{for all } x \in \mathbb{R}^n,$$

  where the inequality holds by the definition of $\mathrm{Lip}(-)$ with $\theta_i, \theta' \in U$ and $\theta_i \neq \theta'$. □

Remark E.3. The second condition $(*)$ of Theorem E.2 is weaker than the following, corresponding condition $(*')$ of a standard theorem for interchanging differentiation and integral (e.g., [Bogachev 2007, Corollary 2.8.7]): "for almost all $x \in \mathbb{R}^n$, $\nabla_\theta f(\theta, x)$ is well-defined for all $\theta \in \mathbb{R}$." The difference arises from whether a proof uses the mean value theorem or not: a proof of the standard theorem finds the function H for the dominated convergence theorem, by applying the mean value theorem which requires the stronger condition $(*')$; the proof of Theorem E.2 finds H not by applying the mean value theorem (but by using the fourth condition of the theorem), so the weaker condition $(*)$ is sufficient for the proof. In this sense, Theorem E.2 is close to [Bogachev 2007, Exercise 2.12.68]. □

Lemma E.4. *Let $f : X_1 \to X_2$ and $g : X_2 \to X_3$ for some open sets $X_i \subseteq \mathbb{R}^{n_i}$. Suppose that f is locally Lipschitz and g is differentiable. Then, $g \circ f : X_1 \to X_3$ is differentiable almost everywhere and the chain rule for $g \circ f$ holds almost everywhere, i.e.,*

$$D(g \circ f)(x) = D(g)(f(x)) \cdot D(f)(x)$$

for almost every $x \in X_1$. Here we use the Lebesgue measure as an underlying measure.

Proof. Since local Lipschitzness is preserved under a function composition, $g \circ f$ is locally Lipschitz and thus differentiable almost everywhere. Since f is also differentiable almost everywhere and g is differentiable everywhere, the set

$$\begin{aligned} U = X_1 \setminus \{x \in X_1 \mid\ & (g \circ f \text{ is differentiable at } x) \\ & \wedge\ (g \text{ is differentiable at } f(x)) \\ & \wedge\ (f \text{ is differentiable at } x)\} \end{aligned}$$

has Lebesgue measure zero. Note that the differentiability of g is importantly used here; if g is non-differentiable even at a point, U can have positive measure. The chain rule for $g \circ f$ holds for each $x \in U$ and this concludes the proof. □

E.2 Proof of Theorem 4.6

Proof of Theorem 4.6. The proof is essentially the same as the proof of Theorem 4.5, except that we invoke the following properties of local Lipschitzness (instead of differentiability): the composition of locally Lipschitz functions is again locally Lipschitz, and the differentiation rules for $+$, $\times$, and *log* hold almost everywhere for locally Lipschitz functions (Lemma E.1). □

F DEFERRED RESULTS IN §5.2

F.1 Proof of Theorem 5.6

Proof of Theorem 5.6. We prove the theorem by induction on the structure of c. Let $(p, d, V) \triangleq [\![c]\!]^\sharp$, and pick $v \in \mathrm{Var}$. We have to show that $p(v) \supseteq d(v)^c$ and $d(v) \supseteq V$. We call these two requirements as conditions (i) and (ii).

Case $c \equiv \mathtt{skip}$**.** In this case, $p(v) = \mathrm{Var}$ and $V = \emptyset$, from which the conditions (i) and (ii) follow.

Case $c \equiv x := e$**.** In this case, $V = \emptyset$. So, the condition (ii) holds. For the proof of the condition (i), we do case analysis on v. If v is the updated variable x, we have $p(v) = (\!|e|\!)^\sharp$ and $d(v) = fv(e)$. Since $(\!|e|\!)^\sharp \supseteq fv(e)^c$, the condition holds. If v is different from x, then $p(v)$ is Var, and so it includes $d(v)^c$.

Case $c \equiv \mathtt{obs}(\mathtt{dist}_{\mathsf{N}}(e_1, e_2), r)$**.** The proof of this case is similar to the one for the assignments. Since $V = \emptyset$, the condition (ii) holds. If v is the variable *like*, then

$$p(v) = (\!|like \times \mathtt{pdf}_{\mathsf{N}}(r; e_1, e_2)|\!)^\sharp \supseteq fv(like \times \mathtt{pdf}_{\mathsf{N}}(r; e_1, e_2))^c = \Big(\{like\} \cup fv(e_1) \cup fv(e_2)\Big)^c = d(v)^c.$$

So, the condition (i) holds in this case. If v is not the variable *like*, then $p(v) = \mathrm{Var}$, from which the condition (i) follows.

Case $c \equiv x := \mathtt{sam}(\mathtt{name}(\alpha, e), \mathtt{dist}_{\mathsf{N}}(e_1, e_2), \lambda y.e')$**.** In this case, $V = \emptyset$, from which the condition (ii) follows. We do case analysis on whether e is a real constant r or not. During the case analysis, we use the assumption that $fv(e)^c \subseteq (\!|e|\!)^\sharp$ for all e, without mentioning it explicitly.

First, we deal with the case that $e \equiv r$. Let $\mu \triangleq create\_name(\alpha, r)$. If v is none of x, val_μ, pr_μ, and cnt_μ, we have $p(v) = \mathrm{Var}$, which gives the condition (i). If $v \in \{x, val_\mu\}$, we prove the condition (i) as follows:

$$d(v)^c = fv(e'[\mu/y])^c \subseteq (\!|e'[\mu/y]|\!)^\sharp = p(v).$$

If $v \equiv pr_\mu$, we calculate the condition (i) as follows:

$$d(pr_\mu)^c = (\{\mu\} \cup fv(e_1) \cup fv(e_2))^c = fv(\mathtt{pdf}_{\mathsf{N}}(\mu; e_1, e_2))^c \subseteq (\!|\mathtt{pdf}_{\mathsf{N}}(\mu; e_1, e_2)|\!)^\sharp = p(pr_\mu).$$

If $v \equiv cnt_\mu$, we derive the condition (i) as follows:

$$d(cnt_\mu)^c = \{cnt_\mu\}^c = fv(cnt_\mu + 1)^c \subseteq (\!|cnt_\mu + 1|\!)^\sharp = p(cnt_\mu).$$

Next, we handle the case that e is not a real constant. If v is none of x, val_μ, pr_μ, and cnt_μ for some $\mu = (\alpha, \_)$, we have $p(v) = \mathrm{Var}$, which implies the condition (i). If $v \equiv x$, we show the condition (i) as follows:

$$p(v) = \Big(fv(e)^c \cap \bigcap_{\mu=(\alpha,\_)\in\mathrm{Name}} (\!|e'[\mu/y]|\!)^\sharp\Big) \supseteq \Big(fv(e) \cup \bigcup_{\mu=(\alpha,\_)\in\mathrm{Name}} fv(e'[\mu/y])\Big)^c = d(v)^c.$$

If $v \equiv val_\mu$ for some $\mu = (\alpha, \_)$, we calculate the condition (i) as follows:

$$p(v) = \left(fv(e)^c \cap (\!|e'[\mu/y]|\!)^\sharp\right) \supseteq \left(fv(e) \cup \{val_\mu\} \cup fv(e'[\mu/y])\right)^c = d(v)^c.$$

If $v \equiv pr_\mu$ for some $\mu = (\alpha, \_)$, we prove the condition (i) as follows:

$$\begin{aligned} p(v) = fv(e)^c \cap (\!|\mathtt{pdf}_{\mathsf{N}}(\mu; e_1, e_2)|\!)^\sharp &\supseteq fv(e)^c \cap fv(\mathtt{pdf}_{\mathsf{N}}(\mu; e_1, e_2))^c \\ &\supseteq \left(fv(e) \cup \{\mu, pr_\mu\} \cup fv(e_1) \cup fv(e_2)\right)^c = p(pr_\mu). \end{aligned}$$

Finally, if $v \equiv cnt_\mu$ for some $\mu = (\alpha, \_)$, we show the condition (i) as follows:

$$p(v) = fv(e)^c \cap (\!|cnt_\mu + 1|\!)^\sharp \supseteq fv(e)^c \cap fv(cnt_\mu + 1)^c = (fv(e) \cup \{cnt_\mu\})^c = d(v)^c.$$

Case $c \equiv c'; c''$**.** Let $(p', d', V') \triangleq [\![c']\!]^\sharp$ and $(p'', d'', V'') \triangleq [\![c'']\!]^\sharp$. The condition (ii) holds since

$$d(v) = V' \cup d'_\cup(d''(v)) \supseteq V' \cup d'_\cup(V'') = V.$$

For the condition (ii), we prove the required subset relationship as follows:

$$\begin{aligned} d(v)^c = (V' \cup d'_\cup(d''(v)))^c &= (V')^c \cap \bigcap_{w \in d''(v)} d'(w)^c \\ &\subseteq (V')^c \cap \bigcap_{w \in d''(v)} p'(w) \cap \bigcap_{w \in p''(v)^c} d'(w)^c \\ &= \left(V' \cup p'_\cap(d''(v))^c \cup d'_\cup(p''(v)^c)\right)^c = p(v). \end{aligned}$$

The subset relationship in the above derivation holds because $d'(w)^c \subseteq p'(w)$ and $d''(v) \supseteq p''(v)^c$ by induction hypothesis.

Case $c \equiv \mathtt{if}\ b\ \{c'\}\ \mathtt{else}\ \{c''\}$**.** Let $(p', d', V') \triangleq [\![c']\!]^\sharp$ and $(p'', d'', V'') \triangleq [\![c'']\!]^\sharp$. Then, by induction hypothesis,

$$V = \left(fv(b) \cup V' \cup V''\right) \subseteq \left(fv(b) \cup d'(v) \cup d''(v)\right) = d(v),$$

which implies the condition (ii). Also, by induction hypothesis again,

$$d(v)^c = \left(fv(b)^c \cap d'(v)^c \cap d''(v)^c\right) \subseteq \left(fv(b)^c \cap p'(v) \cap p''(v)\right) = p(v),$$

which shows the condition (ii).

Case $c \equiv \mathtt{while}\ b\ \{c'\}$**.** Let $(p', d', V') \triangleq [\![c']\!]^\sharp$, and $F^\sharp$ be the operator in the abstract semantics of c. Note that the abstract domain $\mathcal{D}^\sharp$ contains $(p_\bot, d_\bot, V_\bot) = ((\lambda v.\mathrm{Var}),\ (\lambda v.\emptyset),\ \emptyset)$. Thus, it is sufficient to show that $F^\sharp$ is a well-defined monotone function on $\mathcal{D}^\sharp$, because then the least fixed point of $F^\sharp$ is also in $\mathcal{D}^\sharp$ and satisfies the conditions (i) and (ii). The monotonicity of $F^\sharp$ holds because when $(p_1, d_1, V_1) \triangleq F^\sharp(p_0, d_0, V_0)$, the inputs p_0, d_0, and V_0 are used in the right polarity in the definitions of p_1, d_1, and V_1; for instance, p_0 is used only in the positive position (with respect to the subset order) when it is used to define p_1. To prove well-definedness of $F^\sharp$, assume that $p_0(v_0) \supseteq d_0(v_0)^c$ and $d_0(v_0) \supseteq V_0$ for all $v_0 \in \mathrm{Var}$, and pick a variable $v_1 \in \mathrm{Var}$. Then, since $V_0 \subseteq d_0(v_1)$,

$$V_1 = \left(fv(b) \cup d'_\cup(V_0) \cup V'\right) \subseteq \left(fv(b) \cup d'_\cup(d_0(v_1)) \cup V' \cup \{v_1\}\right) = d_1(v_1).$$

Also, by the induction hypothesis on the loop body c' and the relationship $d_0(v_1) \supseteq p_0(v_1)^c$,

$$d_1(v_1)^c = fv(b)^c \cap (V')^c \cap \bigcap_{w \in d_0(v_1)} d'(w)^c \cap \{v_1\}^c$$

$$\subseteq fv(b)^c \cap (V')^c \cap \bigcap_{w \in d_0(v_1)} p'(w) \cap \bigcap_{w \in p_0(v_1)^c} d'(w)^c$$
$$= fv(b)^c \cap \left(V' \cup p'_{\cap}(d_0(v_1))^c \cup d'_{\cup}(p_0(v_1)^c)\right)^c = p_1(v_1).$$

Thus, (p_1, d_1, V_1) is also in $\mathcal{D}^\sharp$. □

F.2 Proof of Theorem 5.8

Our program analysis consists of two parts, one for tracking the dependency information and the other for tracking the smoothness information. The first part does not depend on the second, although it is used crucially by the second part. We exploit this one-way relationship between the two parts of our analysis, and prove the soundness of the dependency-tracking part first and then that of the other smoothness-tracking part. Consider a command c, and let $(p, d, V) \triangleq [\![c]\!]^\sharp$. Then, we have:

Theorem F.1. *For all* $v \in \mathrm{Var}$, *we have* $\models \Delta([\![c]\!], d(v), \{v\})$. *Also,* $\models \Delta([\![c]\!], V, \emptyset)$.

Theorem F.2. *For all* $v \in \mathrm{Var}$, *we have* $\models \Phi([\![c]\!], p(v), \{v\})$.

We prove the two soundness results in §F.3 and §F.4. From these, we immediately obtain the main soundness theorem:

Proof of Theorem 5.8. Let c be a command and $(p, d, V) = [\![c]\!]^\sharp$. Then, by Theorems F.1 and F.2, we have $\models \Delta([\![c]\!], d(v), \{v\})$, $\models \Delta([\![c]\!], V, \emptyset)$, and $\models \Phi([\![c]\!], p(v), \{v\})$ for all $v \in \mathrm{Var}$. Hence, by the definition of γ (i.e., Eq. (10)), we have $[\![c]\!] \in \gamma([\![c]\!]^\sharp)$ as desired. □

F.3 Proof of Theorem F.1

Proof of Theorem F.1. We prove the theorem by induction on the structure of c. Let $(p, d, V) \triangleq [\![c]\!]^\sharp$. Pick a variable $v \in \mathrm{Var}$ and states $\sigma, \sigma', \sigma_0, \sigma'_0 \in \mathrm{St}$ such that

$$\sigma \sim_{d(v)} \sigma' \text{ and } \sigma_0 \sim_V \sigma'_0.$$

We will show that (i) if $[\![c]\!]\sigma_0 \in \mathrm{St}$, then $[\![c]\!]\sigma'_0 \in \mathrm{St}$, and (ii) if $[\![c]\!]\sigma \in \mathrm{St}$ and $[\![c]\!]\sigma' \in \mathrm{St}$, then $[\![c]\!]\sigma \sim_{\{v\}} [\![c]\!]\sigma'$, i.e., $[\![c]\!]\sigma(v) = [\![c]\!]\sigma'(v)$. Since $V \subseteq d(v)$, these two imply the claim of the theorem. We refer to these two properties as conditions (i) and (ii) in the rest of the proof.

Case $c \equiv \mathtt{skip}$**.** In this case, $d(v) = \{v\}$ and $V = \emptyset$. The condition (i) holds since `skip` always terminates. The condition (ii) also holds because $[\![c]\!]\sigma'' = \sigma''$ for all σ'', and the relation $\sim_{d(v)}$ coincides with $\sim_{\{v\}}$.

Case $c \equiv (x := e)$**.** In this case, $V = \emptyset$, and the condition (i) holds since the assignments always terminate. For the condition (ii), we do case analysis on the variable v.

- Case $v \equiv x$. In this case, $d(v) = fv(e)$. This implies $[\![e]\!]\sigma = [\![e]\!]\sigma'$. Thus, $[\![c]\!]\sigma(x) = [\![e]\!]\sigma = [\![e]\!]\sigma' = [\![c]\!]\sigma'(x)$. This implies the desired $[\![c]\!]\sigma \sim_{\{x\}} [\![c]\!]\sigma'$.
- Case $v \not\equiv x$. In this case, $d(v) = \{v\}$, and so $\sigma(v) = \sigma'(v)$. This implies that $[\![c]\!]\sigma(v) = \sigma(v) = \sigma'(v) = [\![c]\!]\sigma'(v)$, which gives the desired relationship.

Case $c \equiv \mathtt{obs}(\mathtt{dist}_{\mathrm{N}}(e_1, e_2), r)$**.** The observe commands always terminate. Thus, the condition (i) holds. We prove the condition (ii) by case analysis on the variable v. If v is not *like*, then $d(v) = \{v\}$, $[\![c]\!]\sigma(v) = \sigma(v)$, and $[\![c]\!]\sigma'(v) = \sigma'(v)$. Thus, in this case, the assumption $\sigma \sim_{d(v)} \sigma'$ implies $[\![c]\!]\sigma(v) = [\![c]\!]\sigma'(v)$, as desired. If v is *like*, then $d(v) = fv(e_1) \cup fv(e_2) \cup \{like\}$, and for some function $g : \mathbb{R}^4 \to \mathbb{R}$,

$$[\![c]\!]\sigma(v) = g(\sigma(like), r, [\![e_1]\!]\sigma, [\![e_2]\!]\sigma) \text{ and } [\![c]\!]\sigma'(v) = g(\sigma'(like), r, [\![e_1]\!]\sigma', [\![e_2]\!]\sigma').$$

Therefore, from the assumption $\sigma \sim_{d(v)} \sigma'$, it follows that $[\![c]\!]\sigma(v) = [\![c]\!]\sigma'(v)$, as desired.

Case $c \equiv (x := \mathtt{sam}(n, \mathtt{dist}_{\mathtt{N}}(e_1, e_2), \lambda y.e'))$**.** The sample commands always terminate. So, the condition (i) holds. We prove the condition (ii) by case analysis on n.

The first case is that n is a constant expression, i.e., it is an expression of the form $\mathtt{name}(\alpha, r)$ for some $\alpha \in \mathrm{Str}$ and real number r. Let $\mu \triangleq \mathit{create\_name}(\alpha, r)$. If v is not one of x, val_μ, and pr_μ, then $d(v) = \{v\}$, and $[\![c]\!]\sigma(v) = g(\sigma(v))$ and $[\![c]\!]\sigma'(v) = g(\sigma'(v))$ for some function $g : \mathbb{R} \to \mathbb{R}$, so that the assumption $\sigma \sim_{d(v)} \sigma'$ implies that $[\![c]\!]\sigma(v) = [\![c]\!]\sigma'(v)$, as desired. If v is x or val_μ, then $d(v) = \mathit{fv}(e'[\mu/y])$, $[\![c]\!]\sigma(v) = [\![e'[\mu/y]]\!]\sigma$, and $[\![c]\!]\sigma'(v) = [\![e'[\mu/y]]\!]\sigma'$, so that the required $[\![c]\!]\sigma(v) = [\![c]\!]\sigma'(v)$ holds. Finally, if $v = \mathit{pr}_\mu$, then $d(v) = \{\mu\} \cup \mathit{fv}(e_1) \cup \mathit{fv}(e_2)$, and so, the assumption $\sigma \sim_{d(v)} \sigma'$ implies that

$$[\![c]\!]\sigma(\mathit{pr}_\mu) = [\![\mathtt{pdf}_{\mathtt{N}}(\mu; e_1, e_2)]\!]\sigma = [\![\mathtt{pdf}_{\mathtt{N}}(\mu; e_1, e_2)]\!]\sigma' = [\![c]\!]\sigma'(\mathit{pr}_\mu),$$

which is precisely the equality that we want.

The next case is that n is not a constant expression. Let $\mathtt{name}(\alpha, e)$ be the form of n. If v is not one of x, val_μ, pr_μ, and cnt_μ for some μ of the form $(\alpha, \_)$, then $d(v) = \{v\}$, $[\![c]\!]\sigma(v) = \sigma(v)$, and $[\![c]\!]\sigma'(v) = \sigma'(v)$, so that the assumption $\sigma \sim_{d(v)} \sigma'$ implies the desired $[\![c]\!]\sigma(v) = [\![c]\!]\sigma'(v)$. Assume that v is one of x, val_μ, pr_μ, and cnt_μ for some μ with $\mu = (\alpha, \_)$. Let $\mu_0 \triangleq [\![n]\!]\sigma$ and $\mu_0' \triangleq [\![n]\!]\sigma'$. Since $d(v) \supseteq \mathit{fv}(n)$ in this case, the assumption $\sigma \sim_{d(v)} \sigma'$ ensures that $\mu_0 = \mu_0'$. If v is x, then $\mathit{fv}(e'[\mu_0/y]) \subseteq d(v)$, so that the assumption $\sigma \sim_{d(v)} \sigma'$ gives the desired

$$[\![c]\!]\sigma(v) = [\![e'[\mu_0/y]]\!]\sigma = [\![e'[\mu_0'/y]]\!]\sigma' = [\![c]\!]\sigma'(v).$$

If v is cnt_μ for some μ of the form $(\alpha, \_)$, then $\mathit{cnt}_\mu \in d(v)$, and $[\![c]\!]\sigma(\mathit{cnt}_\mu) = g(\sigma(\mathit{cnt}_\mu))$ and $[\![c]\!]\sigma'(\mathit{cnt}_\mu) = g(\sigma'(\mathit{cnt}_\mu))$ for some function $g : \mathbb{R} \to \mathbb{R}$, so that $[\![c]\!]\sigma(v) = [\![c]\!]\sigma'(v)$, as desired. If v is val_μ for $\mu = (\alpha, \_)$, then $d(v) \supseteq \{\mathit{val}_\mu\} \cup \mathit{fv}(e'[\mu/y])$, and $[\![c]\!]\sigma(\mathit{val}_\mu) = h([\![e'[\mu/y]]\!]\sigma, \sigma(\mathit{val}_\mu))$ and $[\![c]\!]\sigma'(\mathit{val}_\mu) = h([\![e'[\mu/y]]\!]\sigma', \sigma'(\mathit{val}_\mu))$ for some $h : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, so that $[\![c]\!]\sigma(\mathit{val}_\mu) = [\![c]\!]\sigma'(\mathit{val}_\mu)$ as desired. Finally, if v is pr_μ for some μ of the form $(\alpha, \_)$, then $d(v) \supseteq \{\mathit{pr}_\mu, \mu\} \cup \mathit{fv}(e_1) \cup \mathit{fv}(e_2)$, and for some $k : \mathbb{R}^4 \to \mathbb{R}$,

$$[\![c]\!]\sigma(v) = k(\sigma(\mathit{pr}_\mu), \sigma(\mu), [\![e_1]\!]\sigma, [\![e_2]\!]\sigma) \text{ and } [\![c]\!]\sigma'(v) = k(\sigma'(\mathit{pr}_\mu), \sigma'(\mu), [\![e_1]\!]\sigma', [\![e_2]\!]\sigma'),$$

so that the assumption $\sigma \sim_{d(v)} \sigma'$ guarantees that $[\![c]\!]\sigma(v) = [\![c]\!]\sigma'(v)$, as desired.

Case $c \equiv (c'; c'')$**.** Let $(p', d', V') \triangleq [\![c']\!]^\sharp$ and $(p'', d'', V'') \triangleq [\![c'']\!]^\sharp$. Recall that

$$d(v) = V' \cup (d')_\cup(d''(v)) = V' \cup \bigcup \{d'(v'') \mid v'' \in d''(v)\} \text{ and } V = V' \cup (d')_\cup(V'').$$

Let us handle the condition (i) first. Since $[\![c'; c'']\!]\sigma_0 \in \mathrm{St}$, we have $[\![c']\!]\sigma_0 \in \mathrm{St}$. But $\sigma_0 \sim_{V'} \sigma_0'$, because σ_0 and σ_0' are $\sim_V$-related and V includes V'. Thus, $[\![c']\!]\sigma_0' \in \mathrm{St}$ as well by induction hypothesis, and it is sufficient to show $[\![c']\!]\sigma_0 \sim_{V''} [\![c']\!]\sigma_0'$. Note that for every $v'' \in V''$, by the definition of V, we have $V \supseteq d'(v'')$, and so $\sigma_0 \sim_{d'(v'')} \sigma_0'$, which implies, by induction hypothesis, that $[\![c']\!]\sigma_0 \sim_{\{v''\}} [\![c']\!]\sigma_0'$. As a result, we have the desired $[\![c']\!]\sigma_0 \sim_{V''} [\![c']\!]\sigma_0'$.

Next, we deal with the condition (ii). Since $[\![c'; c'']\!]\sigma$ and $[\![c'; c'']\!]\sigma'$ are both in St, there exist states σ_1, σ_1' such that $[\![c']\!]\sigma = \sigma_1$ and $[\![c']\!]\sigma' = \sigma_1'$. We apply the induction hypothesis to c' and get $\sigma_1 \sim_{d''(v)} \sigma_1'$. Since $[\![c'']\!]\sigma_1$ and $[\![c'']\!]\sigma_1'$ are in St, we apply the induction hypothesis again but this time to c'', σ_1, and σ_1', and obtain $[\![c'']\!]\sigma_1 \sim_{\{v\}} [\![c'']\!]\sigma_1'$, which implies the desired

$$[\![c]\!]\sigma(v) = [\![c'']\!]\sigma_1(v) = [\![c'']\!]\sigma_1'(v) = [\![c]\!]\sigma'(v).$$

Case $c \equiv (\mathtt{if}\, b\, \{c'\}\, \mathtt{else}\, \{c''\})$**.** Let $(p', d', V') \triangleq [\![c']\!]^\sharp$ and $(p'', d'', V'') \triangleq [\![c'']\!]^\sharp$. Then, $d(v) = \mathit{fv}(b) \cup d'(v) \cup d''(v)$ and $V = \mathit{fv}(b) \cup V' \cup V''$.

We prove the condition (i) under the assumption that $[\![b]\!]\sigma_0 = \mathit{true}$. Essentially the same proof applies to the other case that $[\![b]\!]\sigma_0 = \mathit{false}$. Since V includes $\mathit{fv}(b)$, we also have $[\![b]\!]\sigma_0' = \mathit{true}$. Furthermore, since $V' \subseteq V$ and so $\sigma_0 \sim_{V'} \sigma_0'$ by the induction hypothesis, we get that $[\![c']\!]\sigma_0' \in \mathrm{St}$.

Next we show the condition (ii) under the assumption that $[\![b]\!]\sigma = \mathit{true}$. As before, the proof of the other case $[\![b]\!]\sigma = \mathit{false}$ is essentially the same. Since $d(v)$ includes $fv(b)$ and $d'(v)$, we have $[\![b]\!]\sigma' = \mathit{true}$ and $\sigma \sim_{d'(v)} \sigma'$. Also, because $[\![c]\!]\sigma = [\![c']\!]\sigma$ and $[\![c]\!]\sigma' = [\![c']\!]\sigma'$, both $[\![c']\!]\sigma$ and $[\![c']\!]\sigma'$ are in St. Thus, by induction hypothesis, $[\![c']\!]\sigma \sim_{\{v\}} [\![c']\!]\sigma'$, which implies that

$$[\![c]\!]\sigma(v) = [\![c']\!]\sigma(v) = [\![c']\!]\sigma'(v) = [\![c]\!]\sigma'(v),$$

as desired.

Case $c \equiv (\texttt{while}\, b\, \{c_0\})$**.** Let $(d_0, p_0, V_0) \triangleq [\![c_0]\!]^\sharp$, and $F^\sharp$ be the operator in the abstract semantics of c such that (p, d, V) is the least fixed point of $F^\sharp$. Also, let F be the operator in the concrete semantics of c such that $[\![c]\!]$ is the least fixed point of F. Now define

$$T \triangleq \{f \in [\mathrm{St} \to \mathrm{St}_\perp] \mid \text{for all } v \in \mathrm{Var},\ \models \Delta(f, d(v), \{v\}) \text{ and } \models \Delta(f, V, \emptyset)\}.$$

We will show that (i) T contains the empty function $\perp_{\mathrm{St}\to\mathrm{St}_\perp} \triangleq \lambda\sigma.\,\mathrm{undefined}$, (ii) it is closed under the least upper bounds of increasing chains, and (iii) the function F maps functions in T to some functions in the same set. These three imply that the least fixed point of F, namely, $[\![c]\!]$, is in T, which gives the desired conclusion.

The membership of $\perp_{\mathrm{St}\to\mathrm{St}_\perp}$ to T is immediate, since we have $\models \Delta(\perp_{\mathrm{St}\to\mathrm{St}_\perp}, U, U')$ for all $U, U' \subseteq \mathrm{Var}$.

To prove the next requirement, namely, the closure under the least upper bounds of increasing chains, consider a chain $(f_n)_{n\in\mathbb{N}}$ in T, i.e., a sequence such that $f_n(\sigma) = f_{n+1}(\sigma)$ for all $n \in \mathbb{N}$ and σ with $f_n(\sigma) \in \mathrm{St}$. Let f_∞ be the least upper bound of the f_n's (i.e., $f_\infty(\sigma) = f_n(\sigma)$ if $f_n(\sigma) \in \mathrm{St}$ and $f_\infty(\sigma) = \perp$ if $f_n(\sigma) = \perp$ for all $n \in \mathbb{N}$). As in all the other cases so far, we pick an arbitrary variable $v \in \mathrm{Var}$ and arbitrary states $\sigma_0, \sigma_0', \sigma$, and σ' such that

$$\sigma_0 \sim_V \sigma_0', \qquad f_\infty(\sigma_0) \in \mathrm{St}, \qquad \sigma \sim_{d(v)} \sigma', \qquad f_\infty(\sigma), f_\infty(\sigma') \in \mathrm{St}.$$

We will show that $f_\infty(\sigma_0') \in \mathrm{St}$ and $f_\infty(\sigma) \sim_{\{v\}} f_\infty(\sigma')$, which correspond to what we have called conditions (i) and (ii) in the previous cases. Since $f_\infty(\sigma_0) \in \mathrm{St}$, there exists $n \in \mathbb{N}$ such that $f_n(\sigma_0) \in \mathrm{St}$. Because $\models \Delta(f_n, V, \emptyset)$ and $\sigma_0 \sim_V \sigma_0'$, we have $f_n(\sigma_0') \in \mathrm{St}$, which implies that $f_\infty(\sigma_0') = f_n(\sigma_0') \in \mathrm{St}$, as desired. Our proof of the condition (ii) has a similar form. Since both $f_\infty(\sigma)$ and $f_\infty(\sigma')$ are in St, there exists $n \in \mathbb{N}$ such that $f_\infty(\sigma) = f_n(\sigma)$ and $f_\infty(\sigma') = f_n(\sigma')$. By assumption, $\sigma \sim_{d(v)} \sigma'$, and $f_n \in T$. Thus, $f_n(\sigma) \sim_{\{v\}} f_n(\sigma')$, which gives the desired $f_\infty(\sigma) \sim_{\{v\}} f_\infty(\sigma')$.

It remains to show the last requirement, i.e., the closure under F. Pick an arbitrary $f \in T$. Consider a variable $v \in \mathrm{Var}$ and states $\sigma_0, \sigma_0', \sigma$, and σ' such that

$$\sigma_0 \sim_V \sigma_0', \qquad F(f)(\sigma_0) \in \mathrm{St}, \qquad \sigma \sim_{d(v)} \sigma', \qquad F(f)(\sigma), F(f)(\sigma') \in \mathrm{St}.$$

We will show that $F(f)(\sigma_0') \in \mathrm{St}$ and $F(f)(\sigma) \sim_{\{v\}} F(f)(\sigma')$, while referring to these two desired properties as conditions (i) and (ii), as we have done before.

Let us handle the condition (i) first. If $[\![b]\!]\sigma_0 = \mathit{false}$, we have $[\![b]\!]\sigma_0' = \mathit{false}$, because $\sigma_0 \sim_V \sigma_0'$ and $fv(b) \subseteq V$. Thus, in this case, $F(f)(\sigma_0') = \sigma_0' \in \mathrm{St}$. If $[\![b]\!]\sigma_0 = \mathit{true}$, then $[\![b]\!]\sigma_0'$ is also true. Furthermore, in this case, by induction hypothesis, $[\![c_0]\!]\sigma_0' \in \mathrm{St}$ since $V \supseteq V_0$, $\sigma_0 \sim_V \sigma_0'$, and $[\![c_0]\!]\sigma_0 \in \mathrm{St}$. Also, by induction hypothesis again, $[\![c_0]\!]\sigma_0 \sim_V [\![c_0]\!]\sigma_0'$, since $V \supseteq (d_0)_\cup(V)$ and $\sigma_0 \sim_V \sigma_0'$. Since $f \in T$ and $f([\![c_0]\!]\sigma_0) \in \mathrm{St}$, we have $f([\![c_0]\!]\sigma_0') \in \mathrm{St}$, which implies that $F(f)(\sigma_0') \in \mathrm{St}$, as desired.

Next, we prove the condition (ii). If $[\![b]\!]\sigma = \mathit{false}$, we have $[\![b]\!]\sigma' = \mathit{false}$ since $fv(b) \subseteq d(v)$ and $\sigma \sim_{d(v)} \sigma'$. Thus, in this case, $F(f)(\sigma) = \sigma$ and $F(f)(\sigma') = \sigma'$. Also, $\{v\} \subseteq d(v)$, and so, $\sigma \sim_{d(v)} \sigma'$ implies that $F(f)(\sigma) = \sigma \sim_{\{v\}} \sigma' = F(f)(\sigma')$, as desired. Now assume that $[\![b]\!]\sigma = \mathit{true}$. Then, $[\![b]\!]\sigma' = \mathit{true}$ by the reason that $fv(b) \subseteq d(v)$ and $\sigma \sim_{d(v)} \sigma'$. Also, $[\![c_0]\!]\sigma$ and $[\![c_0]\!]\sigma'$ are in St, so that $F(f)(\sigma) = f([\![c_0]\!]\sigma)$ and $F(f)(\sigma') = f([\![c_0]\!]\sigma')$. Furthermore, since $d(v) \supseteq (d_0)_\cup(d(v))$ and

$\sigma \sim_{d(v)} \sigma'$, we have $[\![c_0]\!]\sigma \sim_{d(v)} [\![c_0]\!]\sigma'$. We then use the fact that $f \in T$ and $f([\![c_0]\!]\sigma), f([\![c_0]\!]\sigma') \in \mathrm{St}$, and conclude that $f([\![c_0]\!]\sigma) \sim_{\{v\}} f([\![c_0]\!]\sigma')$, which gives the desired $F(f)(\sigma) \sim_{\{v\}} F(f)(\sigma')$. □

F.4 Proof of Theorem F.2

Let *seq* be the following operator, which models sequential composition:

$$seq : [\mathrm{St} \to \mathrm{St}_\bot] \times [\mathrm{St} \to \mathrm{St}_\bot] \to [\mathrm{St} \to \mathrm{St}_\bot]$$
$$seq(f, g) \triangleq g^\dagger \circ f.$$

Also, define an operator *cond* for modelling if commands as follows:

$$cond : [\mathrm{St} \to \mathbb{B}] \times [\mathrm{St} \to \mathrm{St}_\bot] \times [\mathrm{St} \to \mathrm{St}_\bot] \to [\mathrm{St} \to \mathrm{St}_\bot]$$
$$cond(h, f, g)(\sigma) \triangleq \begin{cases} f(\sigma) & \text{if } h(\sigma) = \mathit{true}, \\ g(\sigma) & \text{if } h(\sigma) = \mathit{false}. \end{cases}$$

Proof of Theorem F.2. We prove the theorem by induction on the structure of c. Let $(p, d, V) \triangleq [\![c]\!]^\sharp$.

Case $c \equiv \mathtt{skip}$**.** In this case, $[\![c]\!](\sigma) = \sigma$ for all $\sigma \in \mathrm{St}$, and $p(v) = \mathrm{Var}$ for all $v \in \mathrm{Var}$. To prove the conclusion, consider $v \in \mathrm{Var}$ and $\tau \in \mathrm{St}[p(v)^c] = \mathrm{St}[\emptyset]$. We should show $g \in \phi_{p(v),\{v\}}$, where

$$g(\sigma) = \begin{cases} (\pi_{\mathrm{Var},\{v\}} \circ [\![c]\!])(\sigma \oplus \tau) & \text{if } [\![c]\!](\sigma \oplus \tau) \in \mathrm{St} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Since $[\![c]\!](\sigma \oplus \tau) = [\![c]\!](\sigma) = \sigma \in \mathrm{St}$ for all $\sigma \in \mathrm{St}$, we have $g = \pi_{\mathrm{Var},\{v\}}$. Thus, Assumption 3 implies

$$g = \pi_{\mathrm{Var},\{v\}} \in \phi_{\mathrm{Var},\{v\}} = \phi_{p(v),\{v\}}.$$

Case $c \equiv (x := e)$**.** In this case, $[\![c]\!](\sigma) = \sigma[x \mapsto [\![e]\!]\sigma]$ for all $\sigma \in \mathrm{St}$. Also, $p(v) = \mathrm{Var}$ if $v \not\equiv x$, and $(\!|e|\!)^\sharp$ if $v \equiv x$. Consider $v \in \mathrm{Var}$ and $\tau \in \mathrm{St}[p(v)^c]$. We should show $g \in \phi_{p(v),\{v\}}$, where

$$g(\sigma) = \begin{cases} (\pi_{\mathrm{Var},\{v\}} \circ [\![c]\!])(\sigma \oplus \tau) & \text{if } [\![c]\!](\sigma \oplus \tau) \in \mathrm{St} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

If $v \not\equiv x$, then $g(\sigma) = \pi_{\mathrm{Var},\{v\}}([\![c]\!](\sigma)) = \pi_{\mathrm{Var},\{v\}}(\sigma[x \mapsto [\![e]\!]\sigma]) = \pi_{\mathrm{Var},\{v\}}(\sigma)$ for all $\sigma \in \mathrm{St}$, where the first equality uses $p(v) = \mathrm{Var}$, and the last uses $v \not\equiv x$. Hence, Assumption 3 implies

$$g = \pi_{\mathrm{Var},\{v\}} \in \phi_{\mathrm{Var},\{v\}} = \phi_{p(v),\{v\}}.$$

If $v \equiv x$, then $g(\sigma) = (\pi_{\mathrm{Var},\{x\}} \circ [\![c]\!])(\sigma \oplus \tau) = \pi_{\mathrm{Var},\{x\}}((\sigma \oplus \tau)[x \mapsto [\![e]\!](\sigma \oplus \tau)]) = [x \mapsto [\![e]\!](\sigma \oplus \tau)]$ for all $\sigma \in \mathrm{St}$. Since $\tau \in \mathrm{St}[((\!|e|\!)^\sharp)^c]$ and $p(v) = (\!|e|\!)^\sharp$, Assumption 2 implies

$$g = \lambda\sigma.\, [x \mapsto [\![e]\!](\sigma \oplus \tau)] \in \phi_{(\!|e|\!)^\sharp,\{x\}} = \phi_{p(v),\{v\}}.$$

Case $c \equiv (c'; c'')$**.** Let $(p', d', V') \triangleq [\![c']\!]^\sharp$ and $(p'', d'', V'') \triangleq [\![c'']\!]^\sharp$. Then,

$$\begin{aligned} p(v) &= \left(V' \cup (p')_\cap(d''(v))^c \cup (d')_\cup(p''(v)^c)\right)^c \\ &= \left((p')_\cap(d''(v))\right) \setminus \left(V' \cup (d')_\cup(p''(v)^c)\right) \text{ for all } v \in \mathrm{Var}. \end{aligned}$$

Also, we have $[\![c]\!] = seq([\![c']\!], [\![c'']\!])$. To prove the conclusion, let $v \in \mathrm{Var}$. It suffices to apply Lemma F.7 to $f = [\![c']\!], g = [\![c'']\!], K = p(v), L = d''(v) \cap p''(v), L' = d''(v)$, and $M = \{v\}$. What remains is to show the preconditions of the lemma:

(a) $\models \Phi([\![c']\!], p(v), d''(v) \cap p''(v))$.
(b) $\models \Phi([\![c'']\!], d''(v) \cap p''(v), \{v\})$.
(c) $\models \Delta([\![c']\!], p(v)^c, d''(v) \setminus (d''(v) \cap p''(v)))$.

(d) $\models \Delta([\![c'']\!], d''(v), \{v\})$.

We obtain (b) as follows: by induction hypothesis on c'', we have $\models \Phi([\![c'']\!], p''(v), \{v\})$, and by the weakening lemma for Φ (Lemma F.4), we have (b). We obtain (d) directly by Theorem F.1 on c''. For (a), consider induction hypothesis on c', which says that $\models \Phi([\![c']\!], p'(w), \{w\})$ for all $w \in \mathrm{Var}$. By the merging lemma for Φ (Lemma F.6), we have $\models \Phi([\![c']\!], (p')_{\cap}(d''(v) \cap p''(v)), d''(v) \cap p''(v))$. Since

$$p(v) \subseteq (p')_{\cap}(d''(v)) \subseteq (p')_{\cap}(d''(v) \cap p''(v)),$$

we obtain (a) by the weakening lemma for Φ (Lemma F.4). For (c), observe that

$$p(v)^c \supseteq V' \cup (d')_{\cup}(p''(v)^c) \text{ and } d''(v) \setminus (d''(v) \cap p''(v)) = p''(v)^c \tag{23}$$

where the second equality follows from $p''(v) \supseteq d''(v)^c$. By Theorem F.1 on c', we have $\models \Delta([\![c']\!], V', \emptyset)$ and $\models \Delta([\![c']\!], d'(w), \{w\})$ for all $w \in \mathrm{Var}$. If $p''(v)^c = \emptyset$, then $\models \Delta([\![c']\!], V', p''(v)^c)$ holds, and if $p''(v)^c \neq \emptyset$, then $\models \Delta([\![c']\!], (d')_{\cup}(p''(v)^c), p''(v)^c)$ holds by the merging lemma for Δ (Lemma F.5). By Eq. (23) and the weakening lemma for Δ (Lemma F.3), we obtain (c) for both cases. Note that we crucially used $p(v)^c \supseteq V'$ to handle the case $p''(v)^c = \emptyset$.

Case $c \equiv (\texttt{if}\ b\ \{c'\}, \texttt{else}\ \{c''\})$**.** Let $(p', d', V') \triangleq [\![c']\!]^\sharp$ and $(p'', d'', V'') \triangleq [\![c'']\!]^\sharp$. Then,

$$p(v) = fv(b)^c \cap p'(v) \cap p''(v) \text{ for all } v \in \mathrm{Var}. \tag{24}$$

Also, $[\![c]\!] = cond([\![b]\!], [\![c']\!], [\![c'']\!])$. To prove the conclusion, let $v \in \mathrm{Var}$. It suffices to apply Lemma F.8 to $f = [\![c']\!], g = [\![c'']\!], K = p(v), L = \{v\}$, and b. What remains is to show the preconditions of the lemma:

(a) $\models \Phi([\![c']\!], p(v), \{v\})$.
(b) $\models \Phi([\![c'']\!], p(v), \{v\})$.
(c) $p(v)^c \supseteq fv(b)$.

We obtain (a) and (b) as follows: by induction hypothesis on c' and c'', we have $\models \Phi([\![c']\!], p'(v), \{v\})$ and $\models \Phi([\![c'']\!], p''(v), \{v\})$, and by Eq. (24) and the weakening lemma for Φ (Lemma F.4), we have (a) and (b). We obtain (c) directly by Eq. (24).

Case $c \equiv (\texttt{while}\ b\ \{c_0\})$**.** The proof starts by decomposing $[\![c]\!]$ and $[\![c]\!]^\sharp$ into smaller pieces. Let

$$(p_0, d_0, V_0) \triangleq [\![c_0]\!]^\sharp.$$

Define $F : [\mathrm{St} \to \mathrm{St}_\perp] \to [\mathrm{St} \to \mathrm{St}_\perp]$ and $F^\sharp : \mathcal{D}^\sharp \to \mathcal{D}^\sharp$ as in §3 and Fig. 3:

$$F(t)(\sigma) \triangleq \begin{cases} \sigma & \text{if } [\![b]\!]\sigma = \mathit{false} \\ t^\dagger([\![c_0]\!]\sigma) & \text{if } [\![b]\!]\sigma = \mathit{true}, \end{cases}$$

$$F^\sharp(p, d, V) \triangleq \begin{pmatrix} \lambda v.\, fv(b)^c \cap (V_0 \cup (p_0)_{\cap}(d(v))^c \cup (d_0)_{\cup}(p(v)^c))^c, \\ \lambda v.\, fv(b) \cup V_0 \cup (d_0)_{\cup}(d(v)) \cup \{v\}, \\ fv(b) \cup (d_0)_{\cup}(V) \cup V_0 \end{pmatrix}.$$

Define $t'_n \in [\mathrm{St} \to \mathrm{St}_\perp]$ and $(p'_n, d'_n, V'_n) \in \mathcal{D}^\sharp$ for $n \in \mathbb{N} \cup \{\infty\}$ as

$$t'_n \triangleq \begin{cases} \lambda\sigma.\, F^n(t_\perp)(\sigma) & \text{if } n \in \mathbb{N} \\ \bigsqcup_{i \in \mathbb{N}} t'_i & \text{if } n = \infty, \end{cases} \qquad (p'_n, d'_n, V'_n) \triangleq \begin{cases} (F^\sharp)^n(p_\perp, d_\perp, V_\perp), & \text{if } n \in \mathbb{N} \\ \bigsqcup_{i \in \mathbb{N}} (p'_i, d'_i, V'_i) & \text{if } n = \infty, \end{cases}$$

where $t_\perp = \lambda\sigma.\, \perp$ and $(p_\perp, d_\perp, V_\perp) = (\lambda v.\, \mathrm{Var},\ \lambda v.\, \emptyset,\ \emptyset)$. Then, we have

$$[\![c]\!] = t'_\infty \quad \text{and} \quad [\![c]\!]^\sharp = (p'_\infty, d'_\infty, V'_\infty).$$

The proof is organized as follows. Define $T, T' \subseteq [\mathrm{St} \to \mathrm{St}_\perp]$ as

$$T \triangleq \{f \in [\mathrm{St} \to \mathrm{St}_\perp] \mid \forall v \in \mathrm{Var}.\ \models \Delta(f, d'_\infty(v), \{v\}) \wedge \models \Delta(f, V'_\infty, \emptyset)\},$$
$$T' \triangleq \{f \in [\mathrm{St} \to \mathrm{St}_\perp] \mid \forall v \in \mathrm{Var}.\ \models \Phi(f, p'_\infty(v), \{v\})\}.$$

In Theorem F.1, we proved

$$t'_n \in T \text{ for all } n \in \mathbb{N} \cup \{\infty\}. \tag{25}$$

In this theorem, our goal is to show $t'_\infty \in T'$. To do so, we prove the next three statements:

(a) $t'_0 \in T'$.
(b) If $t' \in T' \cap T$, then $F(t') \in T'$.
(c) If $t'_n \in T'$ for all $n \in \mathbb{N}$, then $t'_\infty \in T'$.

It suffices to prove the three because (a), (b), and Eq. (25) imply $t'_n \in T'$ for all $n \in \mathbb{N}$, and this and (c) imply $t'_\infty \in T'$. We now prove (a), (b), and (c) as follows.

First, (a) follows directly from Lemma F.9.

Next, we prove (b). Consider $t' \in T' \cap T$. Our goal is to show $\models \Phi(F(t'), p'_\infty(v), \{v\})$ for all $v \in \mathsf{Var}$. Observe that

$$F(t') = cond([\![b]\!], seq([\![c_0]\!], t'), [\![\mathtt{skip}]\!]).$$

By Theorem F.1 and induction hypothesis on c_0, we have

$$[\![c_0]\!] \in \gamma(p_0, d_0, V_0),$$

and by assumption, we have

$$t' \in \gamma(p'_\infty, d'_\infty, V'_\infty) = T' \cap T.$$

By applying to these the proofs of skip, sequential composition, and conditional cases, we have

$$\models \Phi(F(t'), p''(v), \{v\}) \quad \text{for all } v \in \mathsf{Var}$$

where

$$p''(v) = fv(b)^c \cap \left(V_0 \cup (p_0)_\cap(d'_\infty(v))^c \cup (d_0)_\cup(p'_\infty(v)^c)\right)^c \cap \mathsf{Var}.$$

Since p'' is the p part of $F^\sharp(p'_\infty, d'_\infty, V'_\infty)$ and $(p'_\infty, d'_\infty, V'_\infty)$ is a fixed point of $F^\sharp$, we have $p'' = p'_\infty$. Hence, we obtain $\models \Phi(F(t'), p'_\infty(v), \{v\})$ for all $v \in \mathsf{Var}$. This completes the proof of (b).

Finally, we prove (c). Suppose that $t'_n \in T'$ for all $n \in \mathbb{N}$, and let $v \in \mathsf{Var}$. Our goal is to show

$$\models \Phi(t'_\infty, p'_\infty(v), \{v\}).$$

Observe that Lemma F.10 implies the goal when applied to $K = p'_\infty(v)$, $L = \{v\}$, and $\{f_n\}_{n\in\mathbb{N}} = \{t'_n\}_{n\in\mathbb{N}}$. Hence, it suffices to show the three preconditions of the lemma:

- $\{t'_n\}_{n\in\mathbb{N}}$ is an ω-chain.
- For all $\tau \in \mathrm{St}[p'_\infty(v)^c]$ and $n \in \mathbb{N}$, the set $\{\sigma' \in \mathrm{St}[p'_\infty(v)] \mid t'_n(\sigma' \oplus \tau) \in \mathrm{St}\}$ is $\emptyset$ or $\mathrm{St}[p'_\infty(v)]$.
- For all $n \in \mathbb{N}$, we have $\models \Phi(t'_n, p'_\infty(v), \{v\})$.

The first precondition was already observed in §3. The third one holds by the assumption that $t'_n \in T'$ for all $n \in \mathbb{N}$. For the second one, it is enough to show the next two statements:

(i) For all $U \subseteq \mathsf{Var}$ with $U \supseteq V'_\infty$, and for all $\tau \in \mathrm{St}[U]$ and $n \in \mathbb{N}$, the next set is $\emptyset$ or $\mathrm{St}[U^c]$:

$$\{\sigma' \in \mathrm{St}[U^c] \mid t'_n(\sigma' \oplus \tau) \in \mathrm{St}\}.$$

(ii) For all $v \in \mathsf{Var}$,

$$p'_\infty(v)^c \supseteq V'_\infty.$$

We give the proof of the two statements below. This completes the proof of the while-loop case.

Proof of (ii). We prove a stronger statement: for all $n \in \mathbb{N}$ and $v \in \mathsf{Var}$, $p'_n(v)^c \supseteq V'_n$. This statement implies (ii) because $p'_\infty(v)^c = (\bigcap_{n\in\mathbb{N}} p'_n(v))^c = \bigcup_{n\in\mathbb{N}} p'_n(v)^c \supseteq \bigcup_{n\in\mathbb{N}} V'_n = V'_\infty$. We prove the statement by induction on n. For $n = 0$, we have

$$p'_n(v)^c = \mathsf{Var}^c = \emptyset \supseteq \emptyset = V'_n \quad \text{for all } v \in \mathsf{Var}.$$

For $n > 0$, let $v \in \mathsf{Var}$. By induction hypothesis, $p'_{n-1}(v)^c \supseteq V'_{n-1}$ holds. Using this, we have

$$\begin{aligned} p'_n(v)^c &= fv(b) \cup V_0 \cup (p_0)_\cap(d'_{n-1}(v))^c \cup (d_0)_\cup(p'_{n-1}(v)^c) \\ &\supseteq fv(b) \cup V_0 \cup (d_0)_\cup(V'_{n-1}) = V'_n. \end{aligned}$$

This completes the proof of (ii).

Proof of (i). Consider $U \subseteq \mathsf{Var}$, $\tau \in \mathrm{St}[U]$, and $n \in \mathbb{N}$ such that $U \supseteq V'_\infty$. Let $\Sigma \triangleq \{\sigma' \in \mathrm{St}[U^c] \mid t'_n(\sigma' \oplus \tau) \in \mathrm{St}\}$. If $\Sigma = \emptyset$, there is nothing left to prove. So assume $\Sigma \neq \emptyset$. To prove $\Sigma = \mathrm{St}[U^c]$, we need to show that $t'_n(\sigma' \oplus \tau) \in \mathrm{St}$ for any $\sigma' \in \mathrm{St}[U^c]$. Choose $\sigma' \in \mathrm{St}[U^c]$. We show $t'_n(\sigma' \oplus \tau) \in \mathrm{St}$ using the next two claims:

(iii) For all $n \in \mathbb{N}$ and $\sigma \in \mathrm{St}$,

$$t'_n(\sigma) = \begin{cases} [\![c_0^{(i)}]\!]\sigma & \text{if } i \in I_n(\sigma) \\ \bot & \text{otherwise,} \end{cases} \tag{26}$$

where $c_0^{(i)} \triangleq (\mathtt{skip}; c_0; \cdots; c_0)$ that has i copies of c_0, and

$$I_n(\sigma) \triangleq \{i \in [0, n-1] \mid [\![c_0^{(i)}]\!]\sigma \in \mathrm{St} \wedge [\![b]\!]([\![c_0^{(i)}]\!]\sigma) = \mathit{false} \\ \wedge [\![b]\!]([\![c_0^{(i-1)}]\!]\sigma) = \cdots = [\![b]\!]([\![c_0^{(0)}]\!]\sigma) = \mathit{true}\}.$$

Note that Eq. (26) is well-defined since $I_n(\sigma)$ has at most one element.

(iv) For all $n \in \mathbb{N}$,

$$\models \Delta([\![c_0^{(n)}]\!], V'_\infty, fv(b)).$$

We give the proof of the two claims below, and for now we just assume them.

Since $\Sigma \neq \emptyset$, there is some $\sigma'' \in \mathrm{St}[U^c]$ such that $t'_n(\sigma'' \oplus \tau) \in \mathrm{St}$. Since $t'_n(\sigma'' \oplus \tau) \in \mathrm{St}$, (iii) implies that

$$t'_n(\sigma'' \oplus \tau) = [\![c_0^{(m)}]\!](\sigma'' \oplus \tau) \in \mathrm{St}$$

for some $m \in I_n(\sigma'' \oplus \tau)$. Since $\sigma' \oplus \tau \sim_{V'_\infty} \sigma'' \oplus \tau$ (by $U \supseteq V'_\infty$) and $[\![c_0^{(i)}]\!](\sigma'' \oplus \tau) \in \mathrm{St}$ for all $i \in [0, m]$ (by $[\![c_0^{(m)}]\!](\sigma'' \oplus \tau) \in \mathrm{St}$), (iv) implies that

$$[\![c_0^{(i)}]\!](\sigma' \oplus \tau) \in \mathrm{St} \quad \text{and} \quad [\![c_0^{(i)}]\!](\sigma' \oplus \tau) \sim_{fv(b)} [\![c_0^{(i)}]\!](\sigma'' \oplus \tau) \quad \text{for all } i \in [0, m].$$

By combining these with $m \in I_n(\sigma'' \oplus \tau)$, we get $m \in I_n(\sigma' \oplus \tau)$. Hence, by (iii), we have

$$t'_n(\sigma' \oplus \tau) = [\![c_0^{(m)}]\!](\sigma' \oplus \tau) \in \mathrm{St}.$$

This completes the proof of (i).

Proof of (iii). We prove this by induction on n. For $n = 0$, $t'_n(\sigma) = \bot$ and $I_n(\sigma) = \emptyset$ for all $\sigma \in \mathrm{St}$. Hence, Eq. (26) holds. For $n > 0$, we have

$$\begin{aligned} t'_n(\sigma) &= F(t'_{n-1})(\sigma) \\ &= \begin{cases} \sigma & \text{if } [\![b]\!]\sigma = \mathit{false} \\ (t'_{n-1})^\dagger([\![c_0]\!]\sigma) & \text{if } [\![b]\!]\sigma = \mathit{true} \end{cases} \end{aligned}$$

$$= \begin{cases} \sigma & \text{if } [\![b]\!]\sigma = \mathit{false} \ \cdots (*_1) \\ [\![c_0^{(i)}]\!]([\![c_0]\!]\sigma) & \text{if } [\![b]\!]\sigma = \mathit{true}, [\![c_0]\!]\sigma \in \mathrm{St} \text{ and } i \in I_{n-1}([\![c_0]\!]\sigma) \ \cdots (*_2) \\ \bot & \text{otherwise} \end{cases}$$

$$= \begin{cases} [\![c_0^{(0)}]\!]\sigma & \text{if } 0 \in I_n(\sigma) \ \cdots (*'_1) \\ [\![c_0^{(i+1)}]\!]\sigma & \text{if } i+1 \in I_n(\sigma) \text{ and } i+1 \geq 1 \ \cdots (*'_2) \\ \bot & \text{otherwise.} \end{cases}$$

The second equality is by the definition of F, the third by induction hypothesis, and the last by the following: for any $\sigma \in \mathrm{St}$ and $j \in \{1,2\}$, $(*_j)$ holds iff $(*'_j)$ holds; and for any $i \in \mathbb{N}$, $[\![c_0]\!]\sigma \in \mathrm{St}$ implies $[\![c_0^{(i)}]\!]([\![c_0]\!]\sigma) = [\![c_0^{(i+1)}]\!]\sigma$, and $[\![c_0^{(i+1)}]\!]\sigma \in \mathrm{St}$ implies $[\![c_0]\!]\sigma \in \mathrm{St}$. Hence, Eq. (26) holds. This completes the proof of (iii).

Proof of (iv). To prove this, we prove a stronger statement: for all $n \in \mathbb{N}$, $\models \Delta([\![c_0^{(n)}]\!], V'_{n+1}, \mathit{fv}(b))$. This statement implies (iv) since $V'_n \subseteq V'_\infty$ for all $n \in \mathbb{N}$ and we have the weakening lemma for Δ (Lemma F.3). We prove the statement by induction on n. For $n = 0$, $[\![c_0^{(n)}]\!] = [\![\mathtt{skip}]\!]$ and $V'_{n+1} = \mathit{fv}(b) \cup V_0$. By Theorem F.1 on skip, we have $\models \Delta([\![\mathtt{skip}]\!], \{v\}, \{v\})$ for all $v \in \mathrm{Var}$, and then by the merging lemma for Δ (Lemma F.5), we have

$$\models \Delta([\![\mathtt{skip}]\!], \mathit{fv}(b), \mathit{fv}(b)).$$

Since $V'_{n+1} \supseteq \mathit{fv}(b)$, $\models \Delta([\![c_0^{(n)}]\!], V'_{n+1}, \mathit{fv}(b))$ holds by the weakening lemma for Δ (Lemma F.3). Next, for $n > 0$, $[\![c_0^{(n)}]\!] = [\![c_0; c_0^{(n-1)}]\!]$ and $V'_{n+1} = \mathit{fv}(b) \cup V_0 \cup (d_0)_\cup(V'_n)$. By $[\![c_0]\!]^\sharp = (p_0, d_0, V_0)$, Theorem F.1 on c_0, and induction hypothesis of the theorem (not that of the claim (iv)), we have

$$[\![c_0]\!] \in \gamma(p_0, d_0, V_0).$$

Also, by induction hypothesis of our strengthening of the claim (iv) and the weakening lemma for Δ (Lemma F.3),

$$\models \Delta([\![c_0^{(n-1)}]\!], V'_n, \{v\}) \text{ for all } v \in \mathit{fv}(b).$$

By applying to these the proof of Theorem F.1 (on the sequential composition case), we have

$$\models \Delta([\![c_0; c_0^{(n-1)}]\!], V_0 \cup (d_0)_\cup(V'_n), \{v\}) \ \text{ for all } v \in \mathit{fv}(b).$$

By the merging lemma for Δ (Lemma F.5), $\models \Delta([\![c_0^{(n)}]\!], V_0 \cup (d_0)_\cup(V'_n), \mathit{fv}(b))$ holds. Since V'_{n+1} includes $V_0 \cup (d_0)_\cup(V'_n)$, we get $\models \Delta([\![c_0^{(n)}]\!], V'_{n+1}, \mathit{fv}(b))$ by the weakening lemma for Δ (Lemma F.3). This completes the proof of (iv).

Case $c \equiv (x := \mathtt{sam}(\mathtt{name}(\alpha, e), \mathtt{dist}_{\mathtt{N}}(e_1, e_2), \lambda y.e'))$**.** To prove the conclusion, consider $v \in \mathrm{Var}$ and $\tau \in \mathrm{St}[p(v)^c]$. We should show $g \in \phi_{p(v),\{v\}}$, where

$$g(\sigma) = \pi_{\mathrm{Var},\{v\}}([\![c]\!](\sigma \oplus \tau)) = [v \mapsto [\![c]\!](\sigma \oplus \tau)(v)].$$

We prove this by case analysis on v.

First, suppose $v \notin \{x\} \cup \{\mathit{val}_\mu, \mathit{pr}_\mu, \mathit{cnt}_\mu \mid \mu \in \mathrm{Name}, \mu = (\alpha, \_)\}$. Then, $p(v) = \mathrm{Var}$ and

$$g(\sigma) = [v \mapsto [\![c]\!]\sigma(v)] = [v \mapsto \sigma(v)] = \pi_{\mathrm{Var},\{v\}}(\sigma)$$

for all $\sigma \in \mathrm{St}[p(v)]$. Here the first equality follows from $\tau \in \mathrm{St}[\emptyset]$, and the second equality holds since $[\![c]\!]$ does not change the value of v. Hence, by Assumption 3, $g = \pi_{\mathrm{Var},\{v\}} \in \phi_{\mathrm{Var},\{v\}} = \phi_{p(v),\{v\}}$.

Next, suppose $v \in \{x\} \cup \{\mathit{val}_\mu, \mathit{pr}_\mu, \mathit{cnt}_\mu \mid \mu \in \mathrm{Name}, \mu = (\alpha, \_)\}$. Then, we have $p(v)^c \supseteq \mathit{fv}(e)$: if e is a constant, $\mathit{fv}(e) = \emptyset$ holds, and if e is not a constant, the definition of $p(v)$ ensures this. Thus, there exists $\mu_0 \in \mathrm{Name}$ such that $\mathit{create\_name}(\alpha, [\![e]\!](\sigma \oplus \tau)) = \mu_0$ for all $\sigma \in \mathrm{St}[p(v)]$. We now do refined case analysis on v using μ_0.

- Case $v \in \{val_\mu, pr_\mu, cnt_\mu \mid \mu \in \mathsf{Name}, \mu = (\alpha, \_), \mu \neq \mu_0\}$. In this case,
$$g(\sigma) = [v \mapsto (\sigma \oplus \tau)(v)] = \pi_{\mathsf{Var},\{v\}}(\sigma \oplus \tau)$$
for all $\sigma \in \mathrm{St}[p(v)]$. Here the first equality holds since $[\![c]\!]$ does not change the value of v. By Assumption 3, we have $\pi_{\mathsf{Var},\{v\}} \in \phi_{\mathsf{Var},\{v\}}$. Then, by Assumption 5, we obtain $g \in \phi_{p(v),\{v\}}$. Note that this argument does not depend on the value of $p(v)$ (which can be Var, $fv(e)^c \cap (\!|v+1|\!)^\sharp$, etc., depending on e and v).
- Case $v \in \{x, val_{\mu_0}\}$. Define $K_1 \triangleq (\!|e'[\mu_0/y]|\!)^\sharp$. Then, $p(v)^c \supseteq ((\!|e'[\mu_0/y]|\!)^\sharp)^c = K_1^c$. So, there exist $\tau_1 \in \mathrm{St}[K_1^c]$ and $\tau_2 \in \mathrm{St}[p(v)^c \setminus K_1^c]$ such that $\tau = \tau_1 \oplus \tau_2$. Let $h : \mathrm{St}[K_1] \to \mathrm{St}[\{v\}]$ be a function defined by
$$h(\sigma') \triangleq [v \mapsto [\![e'[\mu_0/y]]\!](\sigma' \oplus \tau_1)].$$
Then,
$$g(\sigma) = \left[v \mapsto [\![e'[\mu_0/y]]\!](\sigma \oplus \tau)\right] = h(\sigma \oplus \tau_2)$$
for all $\sigma \in \mathrm{St}[p(v)]$. By Assumption 2, we have $h \in \phi_{K_1,\{v\}}$, Then, by Assumption 5, we obtain $g \in \phi_{p(v),\{v\}}$.
- Case $v \equiv pr_{\mu_0}$. In this case, $p(v) = fv(e)^c \cap K$ for $K = (\!|\mathtt{pdf}_{\mathsf{N}}(\mu_0; e_1, e_2)|\!)^\sharp$. Since $\tau \in \mathrm{St}[p(v)^c]$ and $p(v)^c = (K \setminus fv(e)^c) \uplus K^c$, there exist $\tau_1 \in \mathrm{St}[K \setminus fv(e)^c]$ and $\tau_2 \in \mathrm{St}[K^c]$ such that $\tau = \tau_1 \oplus \tau_2$. Using τ_1 and τ_2, we have
$$\begin{aligned} g(\sigma) &= \left[v \mapsto [\![\mathtt{dist}_{\mathsf{N}}(e_1, e_2)]\!](\sigma \oplus \tau)\big((\sigma \oplus \tau)(\mu_0)\big)\right] \\ &= \left[v \mapsto [\![\mathtt{pdf}_{\mathsf{N}}(\mu_0; e_1, e_2)]\!](\sigma \oplus \tau)\right] = h(\sigma \oplus \tau_1) \end{aligned}$$
for all $\sigma \in \mathrm{St}[p(v)]$, where $h : \mathrm{St}[K] \to \mathrm{St}[\{v\}]$ is defined by
$$h(\sigma') = [v \mapsto [\![\mathtt{pdf}_{\mathsf{N}}(\mu_0; e_1, e_2)]\!](\sigma' \oplus \tau_2)].$$
Here the first equality follows from the definition of $[\![c]\!]$, the second equality holds because $\mathtt{pdf}_{\mathsf{N}}$ is the density function of a normal distribution, and the third equality comes from $\tau = \tau_1 \oplus \tau_2$. By Assumption 2, we have $h \in \phi_{K,\{v\}}$. Then, by Assumption 5, we obtain $g \in \phi_{p(v),\{v\}}$.
- Case $v \equiv cnt_{\mu_0}$. The proof is similar to the above case $v \equiv pr_{\mu_0}$. In this case, $p(v) = fv(e)^c \cap K$ for $K = (\!|cnt_{\mu_0} + 1|\!)^\sharp$. As in the above case, there exist $\tau_1 \in \mathrm{St}[K \setminus fv(e)^c]$ and $\tau_2 \in \mathrm{St}[K^c]$ such that $\tau = \tau_1 \oplus \tau_2$, and we have
$$\begin{aligned} g(\sigma) &= [v \mapsto (\sigma \oplus \tau)(cnt_{\mu_0}) + 1] \\ &= [v \mapsto [\![cnt_{\mu_0} + 1]\!](\sigma \oplus \tau)] = h(\sigma \oplus \tau_1) \end{aligned}$$
for all $\sigma \in \mathrm{St}[p(v)]$, where $h : \mathrm{St}[K] \to \mathrm{St}[\{v\}]$ is defined by
$$h(\sigma') = [v \mapsto [\![cnt_{\mu_0} + 1]\!](\sigma' \oplus \tau_2)].$$
By Assumption 2, we have $h \in \phi_{K,\{v\}}$. Then, by Assumption 5, we obtain $g \in \phi_{p(v),\{v\}}$.

Case $c \equiv \mathtt{obs}(\mathtt{dist}_{\mathsf{N}}(e_1, e_2), r)$**.** To prove the conclusion, consider $v \in \mathsf{Var}$ and $\tau \in \mathrm{St}[p(v)^c]$. We should show $g \in \phi_{p(v),\{v\}}$, where
$$g(\sigma) = \pi_{\mathsf{Var},\{v\}}([\![c]\!](\sigma \oplus \tau)) = [v \mapsto [\![c]\!](\sigma \oplus \tau)(v)].$$
We prove this by case analysis on v.

First, suppose $v \not\equiv like$. Then, $p(v) = \mathsf{Var}$ and
$$g(\sigma) = [v \mapsto [\![c]\!]\sigma(v)] = [v \mapsto \sigma(v)] = \pi_{\mathsf{Var},\{v\}}(\sigma)$$
for all $\sigma \in \mathrm{St}[p(v)]$. Here the first equality is by $\tau \in \mathrm{St}[\emptyset]$, and the second equality holds since $[\![c]\!]$ does not change the value of v. Hence, by Assumption 3, $g = \pi_{\mathsf{Var},\{v\}} \in \phi_{\mathsf{Var},\{v\}} = \phi_{p(v),\{v\}}$.

Next, suppose $v \equiv \mathit{like}$. Then, $p(v) = (\!|\mathit{like} \times \mathtt{pdf}_{\mathsf{N}}(r; e_1, e_2)|\!)^\sharp$ and

$$\begin{aligned} g(\sigma) &= [v \mapsto (\sigma \oplus \tau)(\mathit{like}) \cdot [\![\mathtt{dist}_{\mathsf{N}}(e_1, e_2)]\!](\sigma \oplus \tau)(r)] \\ &= [v \mapsto [\![\mathit{like} \times \mathtt{pdf}_{\mathsf{N}}(r; e_1; e_2)]\!](\sigma \oplus \tau)] \end{aligned}$$

for all $\sigma \in \mathrm{St}[p(v)]$. Here the first equality is by the definition of $[\![c]\!]$, and the second equality holds because $\mathtt{pdf}_{\mathsf{N}}$ is the density function of a normal distribution. Hence, by Assumption 2, we have $g \in \phi_{p(v),\{v\}}$. □

F.5 Proofs of Lemmas for Theorem F.2

Here are the lemmas used to prove Theorem F.2:

Lemma F.3 (Weakening; Δ). *Let $f \in [\mathrm{St} \to \mathrm{St}_\bot]$ and $K, K', L, L' \subseteq \mathrm{Var}$. Then,*

$$\models \Delta(f, K, L) \wedge (K \subseteq K') \wedge (L \supseteq L') \implies \models \Delta(f, K', L').$$

Proof. Consider $\sigma, \sigma' \in \mathrm{St}$ with $\sigma \sim_{K'} \sigma'$. Then, $\sigma \sim_K \sigma'$ because $K \subseteq K'$. Since $\models \Delta(f, K, L)$,

$$(f(\sigma) \in \mathrm{St} \iff f(\sigma') \in \mathrm{St}) \text{ and } (f(\sigma) \in \mathrm{St} \implies f(\sigma) \sim_L f(\sigma')).$$

Note that the conclusion of the second conjunct implies $f(\sigma) \sim_{L'} f(\sigma')$ since $L' \subseteq L$. From what we have just shown, the desired conclusion $\models \Delta(f, K', L')$ follows. □

Lemma F.4 (Weakening; Φ). *Let $f \in [\mathrm{St} \to \mathrm{St}_\bot]$ and $K, K', L, L' \subseteq \mathrm{Var}$. Then,*

$$\models \Phi(f, K, L) \wedge (K \supseteq K') \wedge (L \supseteq L') \implies \models \Phi(f, K', L').$$

Proof. We prove the lemma using Assumptions 3, 5 and 6. Consider $\tau \in \mathrm{St}[(K')^c]$, and let g be the following partial function:

$$g : \mathrm{St}[K'] \rightharpoonup \mathrm{St}[L'], \qquad g(\sigma') \triangleq \begin{cases} (\pi_{\mathrm{Var},L'} \circ f)(\sigma' \oplus \tau) & \text{if } f(\sigma' \oplus \tau) \in \mathrm{St} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

We should show $g \in \phi_{K',L'}$. Note that $(K')^c \supseteq K^c$. Thus, there exist $\tau_1 \in \mathrm{St}[(K')^c \backslash K^c]$ and $\tau_2 \in \mathrm{St}[K^c]$ such that $\tau = \tau_1 \oplus \tau_2$. Define a partial function $h : \mathrm{St}[K] \to \mathrm{St}[L]$ by

$$h(\sigma'') \triangleq \begin{cases} (\pi_{\mathrm{Var},L} \circ f)(\sigma'' \oplus \tau_2) & \text{if } f(\sigma'' \oplus \tau_2) \in \mathrm{St} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Then, since $\models \Phi(f, K, L)$, we have $h \in \phi_{K,L}$. Note that for all $\sigma' \in \mathrm{St}[K']$,

$$g(\sigma') = (\pi_{L,L'} \circ h)(\sigma' \oplus \tau_1).$$

By Assumptions 3, 5 and 6, the above equation implies $g \in \phi_{K',L'}$, as desired. □

Lemma F.5 (Merging; Δ). *Let $f \in [\mathrm{St} \to \mathrm{St}_\bot]$ and $K, K', L, L' \subseteq \mathrm{Var}$. Then,*

$$\models \Delta(f, K, L) \wedge \models \Delta(f, K', L') \implies \models \Delta(f, K \cup K', L \cup L').$$

Proof. Consider $\sigma, \sigma' \in \mathrm{St}$ with $\sigma \sim_{K \cup K'} \sigma'$. Then, $\sigma \sim_K \sigma'$, and by the assumption that $\models \Delta(f, K, L)$, we have

$$f(\sigma) \in \mathrm{St} \iff f(\sigma') \in \mathrm{St}.$$

It remains to show that if $f(\sigma), f(\sigma') \in \mathrm{St}$, then $f(\sigma) \sim_{L \cup L'} f(\sigma')$. Assume $f(\sigma), f(\sigma') \in \mathrm{St}$. Since $\sigma \sim_{K \cup K'} \sigma'$ and we have $\models \Delta(f, K, L)$ and $\models \Delta(f, K', L')$ by assumption,

$$f(\sigma) \sim_L f(\sigma') \text{ and } f(\sigma) \sim_{L'} f(\sigma').$$

This implies that $f(\sigma) \sim_{L \cup L'} f(\sigma')$, as desired. □

Lemma F.6 (Merging; Φ). *Let* $f \in [\mathrm{St} \to \mathrm{St}_\perp]$ *and* $K, K', L, L' \subseteq \mathrm{Var}$. *Then,*

$$\models \Phi(f, K, L) \wedge \models \Phi(f, K', L') \implies \models \Phi(f, K \cap K', L \cup L').$$

Proof. Uses the weakening lemma for Φ (Lemma F.4), we have

$$\models \Phi(f, K \cap K', L) \text{ and } \models \Phi(f, K \cap K', L').$$

This and Assumption 4 then imply the desired conclusion. Concretely, for all $\tau \in \mathrm{St}[(K \cap K')^c]$, if g, g_1, and g_2 are the following partial functions

$$g : \mathrm{St}[K \cap K'] \rightharpoonup \mathrm{St}[L \cup L'], \qquad g(\sigma') \triangleq \begin{cases} (\pi_{\mathrm{Var}, L \cup L'} \circ f)(\sigma' \oplus \tau) & \text{if } f(\sigma' \oplus \tau) \in \mathrm{St} \\ \text{undefined} & \text{otherwise,} \end{cases}$$

$$g_1 : \mathrm{St}[K \cap K'] \rightharpoonup \mathrm{St}[L], \qquad g_1(\sigma') \triangleq \begin{cases} (\pi_{\mathrm{Var}, L} \circ f)(\sigma' \oplus \tau) & \text{if } f(\sigma' \oplus \tau) \in \mathrm{St} \\ \text{undefined} & \text{otherwise,} \end{cases}$$

$$g_2 : \mathrm{St}[K \cap K'] \rightharpoonup \mathrm{St}[L'], \qquad g_2(\sigma') \triangleq \begin{cases} (\pi_{\mathrm{Var}, L'} \circ f)(\sigma' \oplus \tau) & \text{if } f(\sigma' \oplus \tau) \in \mathrm{St} \\ \text{undefined} & \text{otherwise,} \end{cases}$$

then $g_1 \in \phi_{K \cap K', L}$, $g_2 \in \phi_{K \cap K', L'}$, and $g = \langle g_1, g_2 \rangle$, so that by Assumption 4, we have $g \in \phi_{K \cap K', L \cup L'}$ as desired. □

Lemma F.7 (Sequence). *Let* $f, g \in [\mathrm{St} \to \mathrm{St}_\perp]$ *and* $K, L, L', M \subseteq \mathrm{Var}$. *Then,*

$$\models \Phi(f, K, L) \wedge \models \Phi(g, L, M) \wedge \models \Delta(f, K^c, L' \setminus L) \wedge \models \Delta(g, L', M) \implies \models \Phi(\mathit{seq}(f, g), K, M).$$

Proof. Consider $f, g \in [\mathrm{St} \to \mathrm{St}_\perp]$ and $K, L, L', M \subseteq \mathrm{Var}$ that satisfy the given conditions:

$$\models \Phi(f, K, L), \quad \models \Phi(g, L, M), \quad \models \Delta(f, K^c, L' \setminus L), \quad \text{and} \quad \models \Delta(g, L', M).$$

To prove the conclusion, pick an arbitrary $\tau \in \mathrm{St}[K^c]$. We have to show $h \in \phi_{K,M}$, where

$$h(\sigma) = \begin{cases} \pi_{\mathrm{Var}, M}((g^\dagger \circ f)(\sigma \oplus \tau)) & \text{if } (g^\dagger \circ f)(\sigma \oplus \tau) \in \mathrm{St} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Observe that since $\models \Phi(f, K, L)$ and $\models \Phi(g, L, M)$, we have $h_1 \in \phi_{K,L}$ and $h_2 \in \phi_{L,M}$ for any $\tau_1 \in \mathrm{St}[K^c]$ and $\tau_2 \in \mathrm{St}[L^c]$, where h_1 and h_2 are parameterised by τ_1 and τ_2, and defined by

$$h_1 : \mathrm{St}[K] \rightharpoonup \mathrm{St}[L], \qquad h_1(\sigma) \triangleq \begin{cases} \pi_{\mathrm{Var}, L}(f(\sigma \oplus \tau_1)) & \text{if } f(\sigma \oplus \tau_1) \in \mathrm{St} \\ \text{undefined} & \text{otherwise,} \end{cases}$$

$$h_2 : \mathrm{St}[L] \rightharpoonup \mathrm{St}[M], \qquad h_2(\sigma) \triangleq \begin{cases} \pi_{\mathrm{Var}, M}(g(\sigma \oplus \tau_2)) & \text{if } g(\sigma \oplus \tau_2) \in \mathrm{St} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Given these, it suffices to show the claim that $h = h_2 \circ h_1$ for some τ_1 and τ_2: if the claim holds, then we have $h = h_2 \circ h_1 \in \phi_{K,M}$ by Assumption 6, since $h_1 \in \phi_{K,L}$ and $h_2 \in \phi_{L,M}$. We prove the claim by case analysis on $f(- \oplus \tau)$.

Case $f(\sigma \oplus \tau) \notin \mathrm{St}$ for all $\sigma \in \mathrm{St}[K]$. In this case, we set $\tau_1 \triangleq \tau$ and pick any $\tau_2 \in \mathrm{St}[L^c]$. Then, for all $\sigma \in \mathrm{St}[K]$, $h(\sigma)$ and $(h_2 \circ h_1)(\sigma)$ are both undefined, as desired. Note that the latter term is undefined since $h_1(\sigma)$ is undefined.

Case $f(\sigma' \oplus \tau) \in \mathrm{St}$ for some $\sigma' \in \mathrm{St}[K]$. In this case, we set $\tau_1 \triangleq \tau$ and $\tau_2 \triangleq \pi_{\mathrm{Var},L^c}(f(\sigma' \oplus \tau))$. To show $h = h_2 \circ h_1$, consider any $\sigma \in \mathrm{St}[K]$. If $f(\sigma \oplus \tau) \notin \mathrm{St}$, then by the same argument for the above case, $h(\sigma)$ and $(h_2 \circ h_1)(\sigma)$ are both undefined. So, assume that $f(\sigma \oplus \tau) \in \mathrm{St}$. Then,

$$h(\sigma) = \begin{cases} \pi_{\mathrm{Var},M}(g(\sigma_1)) & \text{if } g(\sigma_1) \in \mathrm{St} \\ \text{undefined} & \text{otherwise,} \end{cases} \qquad (h_2 \circ h_1)(\sigma) = \begin{cases} \pi_{\mathrm{Var},M}(g(\sigma_2)) & \text{if } g(\sigma_2) \in \mathrm{St} \\ \text{undefined} & \text{otherwise,} \end{cases} \tag{27}$$

where

$$\sigma_1 = f(\sigma \oplus \tau) \in \mathrm{St}, \qquad \sigma_2 = \pi_{\mathrm{Var},L}(f(\sigma \oplus \tau)) \oplus \pi_{\mathrm{Var},L^c}(f(\sigma' \oplus \tau)) \in \mathrm{St}.$$

Our goal is to show that $h(\sigma)$ and $(h_2 \circ h_1)(\sigma)$ are both undefined, or they are both defined and are the same. By $\models \Delta(g, L', M)$ and Eq. (27), it suffices to show $\sigma_1 \sim_{L'} \sigma_2$. To prove this, we show a stronger statement: $\sigma_1 \sim_L \sigma_2$ and $\sigma_1 \sim_{L' \setminus L} \sigma_2$. The former relation holds since $\pi_{\mathrm{Var},L}(\sigma_1) = \pi_{\mathrm{Var},L}(f(\sigma \oplus \tau)) = \pi_{\mathrm{Var},L}(\sigma_2)$. The latter relation is equivalent to $f(\sigma \oplus \tau) \sim_{L' \setminus L} f(\sigma' \oplus \tau)$, and this holds by $\models \Delta(f, K^c, L' \setminus L)$ and $\sigma \oplus \tau \sim_{K^c} \sigma' \oplus \tau$. Hence, $h = h_2 \circ h_1$ as desired. □

Lemma F.8 (Conditional). *Let $f, f' \in [\mathrm{St} \to \mathrm{St}_\perp]$ and $K, L \subseteq \mathrm{Var}$. Then, for any boolean expression b,*

$$\models \Phi(f, K, L) \wedge \models \Phi(f', K, L) \wedge (K^c \supseteq fv(b)) \implies \models \Phi(\mathit{cond}(\llbracket b \rrbracket, f, f'), K, L).$$

Proof. Let f, f', K, L, and b be the functions, sets and a boolean expression such that

$$\models \Phi(f, K, L), \quad \models \Phi(f', K, L), \quad \text{and} \quad K^c \supseteq fv(b).$$

Consider $\tau \in \mathrm{St}[K^c]$. Define $f'' \triangleq \mathit{cond}(\llbracket b \rrbracket, f, f')$, and also partial functions g, g', and g'' as follows:

$$g : \mathrm{St}[K] \rightharpoonup \mathrm{St}[L], \qquad g(\sigma) \triangleq \begin{cases} (\pi_{\mathrm{Var},L} \circ f)(\sigma \oplus \tau) & \text{if } f(\sigma \oplus \tau) \in \mathrm{St} \\ \text{undefined} & \text{otherwise,} \end{cases}$$

$$g' : \mathrm{St}[K] \rightharpoonup \mathrm{St}[L], \qquad g'(\sigma) \triangleq \begin{cases} (\pi_{\mathrm{Var},L} \circ f')(\sigma \oplus \tau) & \text{if } f'(\sigma \oplus \tau) \in \mathrm{St} \\ \text{undefined} & \text{otherwise,} \end{cases}$$

$$g'' : \mathrm{St}[K] \rightharpoonup \mathrm{St}[L], \qquad g''(\sigma) \triangleq \begin{cases} (\pi_{\mathrm{Var},L} \circ f)(\sigma \oplus \tau) & \text{if } f(\sigma \oplus \tau) \in \mathrm{St} \\ \text{undefined} & \text{otherwise,} \end{cases}$$

We should show $g'' \in \phi_{K,L}$. Since $K^c \supseteq fv(b)$, either $\llbracket b \rrbracket(\sigma \oplus \tau) = \mathit{true}$ for all $\sigma \in \mathrm{St}[K]$ or $\llbracket b \rrbracket(\sigma \oplus \tau) = \mathit{false}$ for all $\sigma \in \mathrm{St}[K]$. In the former case, $g'' = g$, and in the latter case, $g'' = g'$. Since both g and g' are in $\phi_{K,L}$, we have the desired $g'' \in \phi_{K,L}$ in both cases. □

Lemma F.9 (Loop; base). *Let $K, L \subseteq \mathrm{Var}$. Then,*

$$\models \Phi((\lambda\sigma \in \mathrm{St}.\, \perp), K, L).$$

Proof. Consider $\tau \in \mathrm{St}[K^c]$. Define a partial function $g : \mathrm{St}[K] \rightharpoonup \mathrm{St}[L]$ by

$$g(\sigma) \triangleq \begin{cases} (\pi_{\mathrm{Var},L} \circ (\lambda\sigma \in \mathrm{St}.\perp))(\sigma) & \text{if } (\lambda\sigma \in \mathrm{St}.\perp)(\sigma) \in \mathrm{St} \\ \text{undefined} & \text{otherwise} \end{cases}$$
$$= \text{undefined}.$$

Then, $g \in \phi_{K,L}$ by Assumption 7. □

Lemma F.10 (Loop; limit). *Let $K, L \subseteq \mathrm{Var}$ and $\{f_n \in [\mathrm{St} \to \mathrm{St}_\perp]\}_{n\in\mathbb{N}}$ be an ω-chain (i.e., $f_n \sqsubseteq f_{n+1}$ for all $n \in \mathbb{N}$). Here we write $f \sqsubseteq g$ if $f(\sigma) \sqsubseteq g(\sigma)$ for all $\sigma \in \mathrm{St}$. Suppose that for any $\tau \in \mathrm{St}[K^c]$ and $n \in \mathbb{N}$, the set $\{\sigma \in \mathrm{St}[K] \mid f_n(\sigma \oplus \tau) \in \mathrm{St}\}$ is either $\emptyset$ or $\mathrm{St}[K]$. Then,*

$$\bigwedge_{n\in\mathbb{N}} \models \Phi(f_n, K, L) \implies \models \Phi(\bigsqcup_{n\in\mathbb{N}} f_n, K, L).$$

Proof. Consider an ω-chain $\{f_n \in [\mathrm{St} \to \mathrm{St}_\perp]\}_{n\in\mathbb{N}}$ such that $\models \Phi(f_n, K, L)$ for all n. Pick an arbitrary $\tau \in \mathrm{St}[K^c]$. Let $f_\infty \triangleq \bigsqcup_{n\in\mathbb{N}} f_n$, and define partial functions g_∞ and g_n for all n as follows:

$$g_\infty : \mathrm{St}[K] \rightharpoonup \mathrm{St}[L], \qquad g_\infty(\sigma) \triangleq \begin{cases} (\pi_{\mathrm{Var},L} \circ f_\infty)(\sigma \oplus \tau) & \text{if } f_\infty(\sigma \oplus \tau) \in \mathrm{St} \\ \text{undefined} & \text{otherwise,} \end{cases}$$

$$g_n : \mathrm{St}[K] \rightharpoonup \mathrm{St}[L], \qquad g_n(\sigma) \triangleq \begin{cases} (\pi_{\mathrm{Var},L} \circ f_n)(\sigma \oplus \tau) & \text{if } f_n(\sigma \oplus \tau) \in \mathrm{St} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Then, $\{g_n\}_{n\in\mathbb{N}}$ is an ω-chain when we order g_n's by graph inclusion, and g_∞ is the least upper bound of this chain for the same order. We will show that $g_\infty = g_n$ for some $n \in \mathbb{N}$ by case analysis on $\Sigma_n \triangleq \{\sigma \in \mathrm{St}[K] \mid f_n(\sigma \oplus \tau) \in \mathrm{St}\}$. Note that this implies the desired $g_\infty \in \phi_{K,L}$ because $g_n \in \phi_{K,L}$ for all $n \in \mathbb{N}$. If $\Sigma_n = \emptyset$ for all $n \in \mathbb{N}$, then $g = g_n$ for any n, since both g and g_n are the same empty partial function. Otherwise, by the assumption of the lemma, $\Sigma_n = \mathrm{St}[K]$ for some n. This means that g_n is the total function, and so $g_n = g_m$ for all $m \geq n$, which implies that $g = g_n$, as desired. □

G DEFERRED RESULTS IN §5.3

G.1 Proof of Theorem 5.10

Proof of Theorem 5.10. We go through the assumptions, and show that they are satisfied by $\phi^{(d)}$ and $\phi^{(l)}$.

Case of Assumption 3. Let $K, L \subseteq \mathrm{Var}$ such that $L \subseteq K$. The projection $\pi_{K,L}$ is total and has an open set as its domain. Furthermore, the projection $\pi_{K,L}$ is differentiable and 1-Lipschitz continuous. Since Lipschitz continuity implies local Lipschitz continuity, we have both $\pi_{K,L} \in \phi^{(d)}_{K,L}$ and $\pi_{K,L} \in \phi^{(l)}_{K,L}$, as desired.

Case of Assumption 4. Let $K, L_0, L_1 \subseteq \mathrm{Var}$ such that $L_0 \cap L_1 = \emptyset$. Consider $f_0, g_0 \in [\mathrm{St}[K] \rightharpoonup \mathrm{St}[L_0]]$ and $f_1, g_1 \in [\mathrm{St}[K] \rightharpoonup \mathrm{St}[L_1]]$ such that all of the following hold:

$$f_0 \in \phi^{(d)}_{K,L_0}, \qquad f_1 \in \phi^{(d)}_{K,L_1}, \qquad g_0 \in \phi^{(l)}_{K,L_0}, \qquad \text{and} \qquad g_1 \in \phi^{(l)}_{K,L_1}.$$

Let

$$L \triangleq L_0 \cup L_1;$$

$$f : \mathrm{St}[K] \rightharpoonup \mathrm{St}[L], \qquad f(\sigma) \triangleq \begin{cases} f_0(\sigma) \oplus f_1(\sigma) & \text{if } \sigma \in \mathrm{dom}(f_0) \cap \mathrm{dom}(f_1), \\ \text{undefined} & \text{otherwise;} \end{cases}$$

$$g : \mathrm{St}[K] \rightharpoonup \mathrm{St}[L], \qquad g(\sigma) \triangleq \begin{cases} g_0(\sigma) \oplus g_1(\sigma) & \text{if } \sigma \in \mathrm{dom}(g_0) \cap \mathrm{dom}(g_1), \\ \text{undefined} & \text{otherwise.} \end{cases}$$

We should show that f and g satisfy $\phi^{(d)}_{K,L}$ and $\phi^{(l)}_{K,L}$, respectively. In both cases, $\mathrm{dom}(f)$ and $\mathrm{dom}(g)$ are the intersections of two open sets, so that they are open as required.

To prove the differentiability of f, consider $\sigma \in \mathrm{dom}(f)$. Let h_0 and h_1 be the linear functions in $[\mathrm{St}[K] \rightharpoonup \mathrm{St}[L_0]]$ and $[\mathrm{St}[K] \rightharpoonup \mathrm{St}[L_1]]$, respectively, such that their domains are open and contain

0, and for all $i \in \{0, 1\}$,

$$\lim_{\sigma' \to 0} \frac{\|f_i(\sigma + \sigma') - f_i(\sigma) - h_i(\sigma')\|_2}{\|\sigma'\|_2} = 0.$$

Let h be the linear function in $[\mathrm{St}[K] \multimap \mathrm{St}[L]]$ defined by

$$h(\sigma) \triangleq \begin{cases} h_0(\sigma) \oplus h_1(\sigma) & \text{if } \sigma \in \mathrm{dom}(h_0) \cap \mathrm{dom}(h_1); \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Then, $\mathrm{dom}(h)$ is open and contains σ. Furthermore,

$$\begin{aligned}
&\lim_{\sigma' \to 0} \frac{\|f(\sigma + \sigma') - f(\sigma) - h(\sigma')\|_2}{\|\sigma'\|_2} \\
&= \lim_{\sigma' \to 0} \frac{\sqrt{\|f_0(\sigma + \sigma') - f_0(\sigma) - h_0(\sigma')\|_2^2 + \|f_1(\sigma + \sigma') - f_1(\sigma) - h_1(\sigma')\|_2^2}}{\|\sigma'\|_2} \\
&= \lim_{\sigma' \to 0} \sqrt{\left(\frac{\|f_0(\sigma + \sigma') - f_0(\sigma) - h_0(\sigma')\|_2}{\|\sigma'\|_2}\right)^2 + \left(\frac{\|f_1(\sigma + \sigma') - f_1(\sigma) - h_1(\sigma')\|_2}{\|\sigma'\|_2}\right)^2} \\
&= \sqrt{\left(\lim_{\sigma' \to 0} \frac{\|f_0(\sigma + \sigma') - f_0(\sigma) - h_0(\sigma')\|_2}{\|\sigma'\|_2}\right)^2 + \left(\lim_{\sigma' \to 0} \frac{\|f_1(\sigma + \sigma') - f_1(\sigma) - h_1(\sigma')\|_2}{\|\sigma'\|_2}\right)^2} \\
&= 0.
\end{aligned}$$

Thus, f is differentiable at σ, as desired.

It remains to prove the local Lipschitzness of g. Pick $\sigma \in \mathrm{dom}(g)$. Then, g_0 and g_1 are defined at σ and they are locally Lipschitz. Thus, there exist open sets $O_0 \subseteq \mathrm{dom}(g_0)$ and $O_1 \subseteq \mathrm{dom}(g_1)$ and constants $B_0, B_1 > 0$ such that σ belongs to both O_0 and O_1, and for all $\sigma_0, \sigma_0' \in O_0$ and $\sigma_1, \sigma_1' \in O_1$,

$$\|g_0(\sigma_0) - g_0(\sigma_0')\|_2 \leq B_0 \|\sigma_0 - \sigma_0'\|_2 \quad \text{and} \quad \|g_1(\sigma_1) - g_1(\sigma_1')\|_2 \leq B_1 \|\sigma_1 - \sigma_1'\|_2.$$

Let $O \triangleq O_0 \cap O_1$. The set O is open, and contains σ. Furthermore, for all $\sigma', \sigma'' \in O$,

$$\begin{aligned}
\|g(\sigma') - g(\sigma'')\|_2 &= \sqrt{\|g_0(\sigma') - g_0(\sigma'')\|_2^2 + \|g_1(\sigma') - g_1(\sigma'')\|_2^2} \\
&\leq \sqrt{B_0^2 \|\sigma' - \sigma''\|_2^2 + B_1^2 \|\sigma' - \sigma''\|_2^2} \\
&= \sqrt{B_0^2 + B_1^2} \cdot \|\sigma' - \sigma''\|_2.
\end{aligned}$$

Thus, g is Lipschitz in O, as desired.

Case of Assumption 5. Consider $K, K', L \subseteq \mathrm{Var}$ with $K \subseteq K'$, and $\tau \in \mathrm{St}[K' \setminus K]$. Let f and g be partial functions in $[\mathrm{St}[K'] \multimap \mathrm{St}[L]]$ such that $f \in \phi^{(d)}_{K',L}$ and $g \in \phi^{(l)}_{K',L}$. Let

$$f_1 : \mathrm{St}[K] \multimap \mathrm{St}[L], \qquad f_1(\sigma) \triangleq \begin{cases} f(\sigma \oplus \tau) & \text{if } \sigma \oplus \tau \in \mathrm{dom}(f) \\ \text{undefined} & \text{otherwise,} \end{cases}$$

$$g_1 : \mathrm{St}[K] \multimap \mathrm{St}[L], \qquad g_1(\sigma) \triangleq \begin{cases} g(\sigma \oplus \tau) & \text{if } \sigma \oplus \tau \in \mathrm{dom}(g) \\ \text{undefined} & \text{otherwise.} \end{cases}$$

We should show that f_1 and g_1 satisfy $\phi^{(d)}_{K,L}$ and $\phi^{(l)}_{K,L}$, respectively. Note that

$$\mathrm{dom}(f_1) = \{\sigma \mid \sigma \oplus \tau \in \mathrm{dom}(f)\} \quad \text{and} \quad \mathrm{dom}(g_1) = \{\sigma \mid \sigma \oplus \tau \in \mathrm{dom}(g)\}.$$

These two sets are open since $\mathrm{dom}(f)$ and $\mathrm{dom}(g)$ are open and for any open O, the slice $\{\sigma \in \mathrm{St}[K] \mid \sigma \oplus \tau \in O\}$ is open. Let $\sigma_0 \in \mathrm{dom}(f_1)$ and $\sigma_1 \in \mathrm{dom}(g_1)$. We will show that f_1 is differentiable at σ_0, and g_1 is Lipschitz in an open neighbourhood of σ_1.

Since f is differentiable and $\sigma_0 \oplus \tau \in \mathrm{dom}(f)$, there exists a linear map $h : \mathrm{St}[K'] \multimap \mathrm{St}[L]$ such that $\mathrm{dom}(h)$ is open and contains 0, and

$$\lim_{\sigma' \to 0} \frac{\|f(\sigma_0 \oplus \tau + \sigma') - f(\sigma_0 \oplus \tau) - h(\sigma')\|_2}{\|\sigma'\|_2} = 0.$$

Let $\tau_0 \triangleq \lambda v \in K' \setminus K.\, 0$, and h_1 be the partial function from $\mathrm{St}[K]$ to $\mathrm{St}[L]$ defined by

$$h_1(\sigma) \triangleq \begin{cases} h(\sigma \oplus \tau_0) & \text{if } \sigma \oplus \tau_0 \in \mathrm{dom}(h), \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Then, h_1 is linear, its domain is open (since taking a slice of an open set in $\mathrm{St}[K] \cong \mathbb{R}^{|K'|}$ by fixing some coordinate variables gives an open set), and

$$\begin{aligned} \lim_{\sigma'' \to 0} \frac{\|f_1(\sigma_0 + \sigma'') - f_1(\sigma_0) - h_1(\sigma'')\|_2}{\|\sigma''\|_2} &= \lim_{\sigma'' \to 0} \frac{\|f(\sigma_0 \oplus \tau + \sigma'' \oplus \tau_0) - f(\sigma_0 \oplus \tau) - h(\sigma'' \oplus \tau_0)\|_2}{\|\sigma'' \oplus \tau_0\|_2} \\ &= 0. \end{aligned}$$

This means that f_1 is differentiable at σ_0.

Since g is locally Lipschitz and $\sigma_1 \oplus \tau \in \mathrm{dom}(g)$, there exists an open subset O of $\mathrm{dom}(g)$ such that O contains $\sigma_1 \oplus \tau$ and g is Lipschitz in O, that is, there exists a real number $B > 0$ such that

$$\|g(\sigma) - g(\sigma')\|_2 \le B \cdot \|\sigma - \sigma'\|_2$$

for all $\sigma, \sigma' \in O$. Let

$$O' \triangleq \{\sigma \in \mathrm{St}[K] \mid \sigma \oplus \tau \in O\}.$$

Then, O' is open, and it contains σ_1. Furthermore, for all $\sigma, \sigma' \in O'$,

$$\|g_1(\sigma) - g_1(\sigma')\|_2 = \|g(\sigma \oplus \tau) - g(\sigma' \oplus \tau)\|_2 \le B \cdot \|\sigma \oplus \tau - \sigma' \oplus \tau\|_2 = B \cdot \|\sigma - \sigma'\|_2.$$

Thus, g_1 is Lipschitz in O', as desired.

Case of Assumption 6. For the composition condition, we handle the differentiability case only. The other case can be proved similarly. Consider

$$K, L, M \subseteq \mathrm{Var}, \quad f \in [\mathrm{St}[K] \multimap \mathrm{St}[L]], \quad \text{and} \quad g \in [\mathrm{St}[L] \multimap \mathrm{St}[M]].$$

Assume that $f \in \phi^{(d)}_{K,L}$ and $g \in \phi^{(d)}_{L,M}$. Let h be the standard composition of partial functions g and f. We should show that $h \in \phi^{(d)}_{K,M}$ as well, that is, $\mathrm{dom}(h)$ is open and h is differentiable on its domain. Note that

$$\mathrm{dom}(h) = \mathrm{dom}(f) \cap f^{-1}(\mathrm{dom}(g)).$$

Since $g \in \phi^{(d)}_{L,M}$, the set $\mathrm{dom}(g)$ is open. Because $f \in \phi^{(d)}_{K,L}$, $\mathrm{dom}(f)$ is open and f is continuous on its domain. The latter implies that $f^{-1}(\mathrm{dom}(g))$ is open as well. Thus, the intersection of $\mathrm{dom}(f)$ and $f^{-1}(\mathrm{dom}(g))$ is open as desired. The differentiability of h on its domain holds since the restriction of f to $\mathrm{dom}(h)$ gives a differentiable total function from $\mathrm{dom}(h)$ to $\mathrm{dom}(g)$, that of g to $\mathrm{dom}(g)$ is also a differentiable total function, and the composition of two differentiable functions is differentiable.

Case of Assumption 7. The empty set is open, and the empty function is jointly differentiable and locally Lipschitz continuous. Thus, the strictness assumption holds for both predicate families. □

H DEFERRED RESULTS IN §6

H.1 Proof of Theorem 6.2

Proof of Theorem 6.2. Suppose that the algorithm returns π (without an error message). We should show the conclusion that π is simple and satisfies (R2) and (R3).

We make several observations before proving the conclusion. Let

$$(\mathbb{p}_m, \mathbb{d}_m, \mathbb{V}_m) \triangleq [\![c_m]\!]^\sharp, \qquad (\mathbb{p}_g, \mathbb{d}_g, \mathbb{V}_g) \triangleq [\![c_g]\!]^\sharp, \qquad (\overline{\mathbb{p}_g}, \overline{\mathbb{d}_g}, \overline{\mathbb{V}_g}) \triangleq [\![\overline{c_g}^{\pi}]\!]^\sharp,$$

and $K \subseteq \mathsf{Var}$ be the set defined in Eq. (12). Also, let S_r be the set of names that the algorithm uses to construct the returned reparameterisation plan π. Then, by the algorithm, we have the two inclusions in Eq. (12) and Eq. (13), and also $\pi = \pi_0[S_r]$. In addition, $S_r \subseteq K$ since

$$S_r \subseteq \{(\alpha, i) \in \mathsf{Name} \mid \text{for all } i' \in \mathbb{N}, (\alpha, i') \in \mathsf{Name} \implies (\alpha, i') \in K\} \subseteq K.$$

We now prove the conclusion in three parts.

First part: We show that π is simple. To show this, consider $(n, d, l), (n', d', l') \in \mathsf{NameEx} \times \mathsf{DistEx} \times \mathsf{LamEx}$ such that $n = \mathsf{name}(\alpha, e)$ and $n' = \mathsf{name}(\alpha, e')$ for some $\alpha \in \mathsf{Str}, e$, and e'. Suppose that $(n, d, l) \in \mathrm{dom}(\pi)$. We should show $(n', d', l') \in \mathrm{dom}(\pi)$. Since $(n, d, l) \in \mathrm{dom}(\pi) = \mathrm{dom}(\pi_0[S_r])$, we have $(n, d, l) \in \mathrm{dom}(\pi_0)$ and $(\alpha, \_) \in S_r$. This implies that $(n', d', l') \in \mathrm{dom}(\pi_0)$ because π_0 is simple. Since $n' = \mathsf{name}(\alpha, \_)$ and $(\alpha, \_) \in S_r$, we have $(n', d', l') \in \mathrm{dom}(\pi)$ as desired.

Second part: We show that π satisfies (R2) in three steps.

First step: We prove $\theta \cup rv(\pi) \subseteq K$. To do so, observe that the S in the algorithm always satisfies the following property:

$$(\alpha, i) \in S \implies (\alpha, i') \in S \quad \text{for any } (\alpha, i), (\alpha, i') \in \mathsf{Name}. \tag{28}$$

We can prove this by induction: the initial S (i.e., $S = \{(\alpha, i) \in \mathsf{Name} \mid \text{for all } i' \in \mathbb{N}, (\alpha, i') \in \mathsf{Name} \implies (\alpha, i') \in K\}$) satisfies the property, and each update of S (i.e., $S \leftarrow S \setminus \{(\alpha, i) \in \mathsf{Name}\}$ for some $(\alpha, \_) \in S$) preserves the property. From this, we obtain $rv(\pi) = rv(\pi_0) \cap S$:

$$\begin{aligned}
rv(\pi) &= rv(\pi_0[S]) \\
&= \{(\alpha, i) \in \mathsf{Name} \mid \exists e, d, l.\, (\mathsf{name}(\alpha, e), d, l) \in \mathrm{dom}(\pi_0[S])\} \\
&= \left\{(\alpha, i) \in \mathsf{Name} \mid \exists e, d, l.\, \left((\mathsf{name}(\alpha, e), d, l) \in \mathrm{dom}(\pi_0) \wedge \exists i'.\, (\alpha, i') \in S\right)\right\} \\
&= \left\{(\alpha, i) \in \mathsf{Name} \mid \left(\exists e, d, l.\, \left((\mathsf{name}(\alpha, e), d, l) \in \mathrm{dom}(\pi_0)\right)\right) \wedge \left(\exists i'.\, (\alpha, i') \in S\right)\right\} \\
&= \{(\alpha, i) \in \mathsf{Name} \mid (\mathsf{name}(\alpha, _), _, _) \in \mathrm{dom}(\pi_0)\} \cap \{(\alpha, i) \in \mathsf{Name} \mid (\alpha, _) \in S\} \\
&= rv(\pi_0) \cap S,
\end{aligned}$$

where the second and third equalities use the definitions of $rv(-)$ and $\pi_0[S]$, respectively, and the last equality uses Eq. (28). Since $rv(\pi) \subseteq S \subseteq K$ and $\theta \subseteq K$ by Eq. (12) (the inclusion of θ), we get $\theta \cup rv(\pi) \subseteq K$ as desired.

Second step: We prove that for all $u \in \{like\} \cup \{pr_\mu \mid \mu \in \mathsf{Name}\}$ and $v \in \{pr_\mu \mid \mu \in \mathsf{Name}\}$, the following functions (which are total since c_m and c_g always terminate) are differentiable with respect to the variables in $\theta \cup rv(\pi)$ jointly:

$$\begin{aligned}
(\sigma_\theta, \sigma_n) \in \mathsf{St}[\theta] \times \mathsf{St}[\mathsf{Name}] &\longmapsto [\![c_m]\!](\sigma_{p \setminus \theta} \oplus \sigma_\theta \oplus \sigma_n \oplus g(\sigma_n))(u), \\
(\sigma_\theta, \sigma_n) \in \mathsf{St}[\theta] \times \mathsf{St}[\mathsf{Name}] &\longmapsto [\![c_g]\!](\sigma_{p \setminus \theta} \oplus \sigma_\theta \oplus \sigma_n \oplus g(\sigma_n))(v),
\end{aligned} \tag{29}$$

where $\sigma_{p \setminus \theta} \triangleq (\lambda v \in \mathsf{PVar} \setminus \theta.\, 0)$ and the function $g : \mathsf{St}[\mathsf{Name}] \to \mathsf{St}[\mathsf{AVar}]$ takes σ_n and returns σ_a such that σ_a maps $like$ to 1, pr_μ to $\mathcal{N}(\sigma_n(\mu); 0, 1)$, val_μ to $\sigma_n(\mu)$, and all the other variables to 0. The state $\sigma_{p \setminus \theta} \oplus g(\sigma_n)$ is the very initialisation used in Eq. (3). This step consists of two substeps.

First substep: We first show that for all $u \in \{like\} \cup \{pr_\mu \mid \mu \in \mathsf{Name}\}$ and $v \in \{pr_\mu \mid \mu \in \mathsf{Name}\}$, the functions $f_m, f_g : \mathrm{St}[\theta] \times \mathrm{St}[\mathsf{Name}] \times \mathrm{St}[\mathsf{AVar}] \to \mathbb{R}$ are differentiable with respect to the variables in $\theta \cup rv(\pi)$ jointly:

$$\begin{aligned} f_m(\sigma_\theta, \sigma_n, \sigma_a) &\triangleq [\![c_m]\!](\sigma_{p\setminus\theta} \oplus \sigma_\theta \oplus \sigma_n \oplus \sigma_a)(u), \\ f_g(\sigma_\theta, \sigma_n, \sigma_a) &\triangleq [\![c_g]\!](\sigma_{p\setminus\theta} \oplus \sigma_\theta \oplus \sigma_n \oplus \sigma_a)(v). \end{aligned} \tag{30}$$

Note that in f_m and f_g, the AVar part does not depend on the Name part (unlike in Eq. (29)). For the proof, pick arbitrary $u \in \{like\} \cup \{pr_\mu \mid \mu \in \mathsf{Name}\}$ and $v \in \{pr_\mu \mid \mu \in \mathsf{Name}\}$. Then, $\theta \cup rv(\pi) \subseteq K \subseteq \mathbb{p}_m(u) \cap \mathbb{p}_g(v)$, where the first inclusion is from the above result and the second from Eq. (12) (the definition of K). By the soundness of differentiability analysis (Theorem 5.8), $\models \Phi([\![c_m]\!], \mathbb{p}_m(u), \{u\})$ and $\models \Phi([\![c_g]\!], \mathbb{p}_g(v), \{v\})$. From this, and by the weakening lemma of Φ with $\theta \cup rv(\pi) \subseteq \mathbb{p}_m(u) \cap \mathbb{p}_g(v)$ (Lemma F.4), we have $\models \Phi([\![c_m]\!], \theta \cup rv(\pi), \{u\})$ and $\models \Phi([\![c_g]\!], \theta \cup rv(\pi), \{v\})$. Hence, the functions in Eq. (30) are differentiable with respect to $\theta \cup rv(\pi)$ jointly as desired, by the definition of Φ (§5.1) and the definition of "$f : \mathrm{St}[L] \to \mathbb{R}$ for $L \subseteq \mathsf{Var}$ is differentiable with respect to $L' \subseteq L$ jointly" (§4.2).

Second substep: We now prove that the claim of the second step follows from the first substep just proved. Pick any $u \in \{like\} \cup \{pr_\mu \mid \mu \in \mathsf{Name}\}$. We should show that the first function in Eq. (29) is differentiable with respect to $\theta \cup rv(\pi)$. Note that we should also show the same for the second function (for any v), but the proof is similar to the first function so we omit this case. To prove the claim for the first function, pick any $\xi'_{n,0} \in \mathrm{St}[\mathsf{Name} \setminus rv(\pi)]$ and $\sigma_{a,0} \in \mathrm{St}[\mathsf{AVar}]$. Define $f' : \mathrm{St}[\theta] \times \mathrm{St}[rv(\pi)] \times \mathrm{St}[\mathsf{AVar}]$ as

$$f'(\sigma_\theta, \xi_n, \sigma_a) \triangleq f(\sigma_\theta, \xi_n \oplus \xi'_{n,0}, \sigma_a).$$

Then, by Lemma C.6-(2) and Lemma C.6-(3),

$$f'(\sigma_\theta, \xi_n, \sigma_a) \triangleq \begin{cases} f(\sigma_\theta, \xi_n \oplus \xi'_{n,0}, \sigma_{a,0}) & \text{if } (\sigma_\theta, \xi_n) \in U \\ proj(\sigma_a) & \text{if } (\sigma_\theta, \xi_n) \notin U \end{cases}$$

for some $U \subseteq \mathrm{St}[\theta] \times \mathrm{St}[rv(\pi)]$ and some projection map $proj : \mathrm{St}[\mathsf{AVar}] \to \mathbb{R}$. Also, since f is differentiable with respect to $\theta \cup rv(\pi)$, $f'(-, -, \sigma_a) : \mathrm{St}[\theta] \times \mathrm{St}[rv(\pi)]$ is differentiable and thus continuous for all $\sigma_a \in \mathrm{St}[\mathsf{AVar}]$. From these, Lemma H.1 is applicable to f', implying that U should be either $\emptyset$ or $\mathrm{St}[\theta] \times \mathrm{St}[rv(\pi)]$. We now consider $f'' : \mathrm{St}[\theta] \times \mathrm{St}[rv(\pi)] \to \mathbb{R}$ defined by

$$f''(\sigma_\theta, \xi_n) \triangleq f'(\sigma_\theta, \xi_n, g(\xi_n) \oplus g(\xi'_{n,0})),$$

where g is extended to accept a substate in $\mathrm{St}_\square[\mathsf{Name}]$ and return a substate for the corresponding auxiliary part. Then, to prove the claim, it suffices to show that f'' is differentiable (since $\xi'_{n,0}$ was chosen arbitrarily). We do case analysis on U. If $U = \mathrm{St}[\theta] \times \mathrm{St}[rv(\pi)]$, then

$$f''(\sigma_\theta, \xi_n) = f(\sigma_\theta, \xi_n \oplus \xi'_{n,0}, \sigma_{a,0}) \quad \text{for all } \sigma_\theta \text{ and } \xi_n;$$

since f is differentiable with respect to $\theta \cup rv(\pi)$, f'' is differentiable. If $U = \emptyset$, then

$$f''(\sigma_\theta, \xi_n) = proj(g(\xi_n) \oplus g(\xi'_{n,0})) \quad \text{for all } \sigma_\theta \text{ and } \xi_n;$$

since g, $\oplus$, and $proj$ are all differentiable (because g only uses projection and the density of the standard normal distribution), f'' is differentiable. Hence, f'' is differentiable in both cases, and this shows the claim of the second step.

Third step: We prove that π satisfies (R2), i.e., the functions in (R2) are differentiable with respect to $\theta \cup rv(\pi)$ jointly. This holds because: c_m and c_g do not have a double-sampling error, so each function in (R2) is a multiplication of some of the functions in Eq. (29) (for different u and v); the functions in Eq. (29) are differentiable with respect to $\theta \cup rv(\pi)$ jointly (by the above result); and multiplication preserves differentiability.

Third part: We show that π satisfies (R3), i.e., the functions in (R3) are differentiable with respect to θ jointly. For this, it suffices to show the claim that for all $v \in \{pr_\mu, val_\mu \mid \mu \in \mathsf{Name}\}$ and $\sigma_n \in \mathsf{St}[\mathsf{Name}]$, the following function is differentiable with respect to θ jointly:

$$\sigma_\theta \in \mathsf{St}[\theta] \longmapsto [\![\overline{c_g}^{\pi}]\!](\sigma_{p\setminus\theta} \oplus \sigma_\theta \oplus \sigma_n \oplus g(\sigma_n))(v). \tag{31}$$

This implies (R3) because: $\overline{c_g}^{\pi}$ does not have a double-sampling error, so each function in (R2) is either a multiplication or a pairing of the function in Eq. (31) (for different v); and multiplication and pairing preserve differentiability. To show the claim, consider any $v \in \{pr_\mu, val_\mu \mid \mu \in \mathsf{Name}\}$. Then, $\theta \subseteq \overline{\mathbb{P}_g}(v)$ by Eq. (13). By the soundness of differentiability analysis (Theorem 5.8), $\models \Phi([\![\overline{c_g}^{\pi}]\!], \overline{\mathbb{P}_g}(v), \{v\})$. From this, and by the weakening lemma of Φ with $\theta \subseteq \overline{\mathbb{P}_g}(v)$ (Lemma F.4), we have $\models \Phi([\![\overline{c_g}^{\pi}]\!], \theta, \{v\})$. Hence, the function in Eq. (31) is differentiable with respect to θ jointly. This completes the overall proof. □

Lemma H.1. *Let $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ be a function such that*

$$f(x,y) = \begin{cases} f_1(x) & \text{if } x \in U \\ f_2(y) & \text{if } x \notin U \end{cases}$$

for some $f_1 : \mathbb{R}^n \to \mathbb{R}$, $f_2 : \mathbb{R}^m \to \mathbb{R}$, and $U \subseteq R^n$. Suppose that $f_2(\mathbb{R}^m) = \mathbb{R}$ and $f(-,y) : \mathbb{R}^n \to \mathbb{R}$ is continuous for all $y \in \mathbb{R}^m$. Then, U is either $\emptyset$ or $\mathbb{R}^n$.

Proof. Here is a sketch of the proof. We prove the lemma by contradiction. Suppose that U is neither $\emptyset$ nor $\mathbb{R}^n$. Then, the boundary of U (i.e., $\mathrm{bd}(U) \subseteq \mathbb{R}^n$) is nonempty, since the boundary of a set is empty if and only if the set is both open and closed, and since $\emptyset$ and $\mathbb{R}^n$ are the only subsets of $\mathbb{R}^n$ that are both open and closed. Let $x \in \mathrm{bd}(U)$ and consider two cases: $x \in U$ or $x \notin U$. In each of the two cases, we can show that there exists $y \in \mathbb{R}^m$ such that $f(-,y)$ is not continuous at x. When showing the discontinuity, we use the following: $x \in \mathrm{bd}(U)$; the specific way that f is defined (in terms of U, f_1, and f_2); and the assumption that $f_2(\mathbb{R}^m) = \mathbb{R}$. By assumption, $f(-,y)$ should be continuous over the entire $\mathbb{R}^n$, so we get contradiction. □

I DEFERRED RESULTS IN §7

I.1 Deferred Experiment Details and Results

Table 7. Pyro examples used in experiments and their key features (continued from Table 3). The last five columns show the total number of code lines (excluding comments), loops, sample commands, observe commands, and learnable parameters (declared explicitly by `pyro.param` or implicitly by a neural network module). Each number is the sum of the counts in the model and guide.

Name	Probabilistic model	LoC	`while`	`sam`	`obs`	`param`
`dpmm`	Dirichlet process mixture models	27	0	6	1	4
`vae`	Variational autoencoder (VAE)	35	0	2	1	5
`csis`	Inference compilation	38	0	2	2	5
`br`	Bayesian regression	42	0	10	1	5
`lda`	Amortised latent Dirichlet allocation	57	0	8	1	5
`prodlda`	Probabilistic topic modelling	58	0	2	1	5
`ssvae`	Semi-supervised VAE	60	0	4	1	7

Table 8. Results of smoothness analyses (continued from Table 4). "Manual" and "Ours" denote the number of continuous random variables and learnable parameters in which the density of the program is smooth, computed by hand and by our analyser. "Time" denotes the runtime of our analyser in seconds. "#CRP" denotes the total number of continuous random variables and learnable parameters in the program. `-m` and `-g` denote model and guide. We consider $\{(\alpha, i) \in \text{Name}\}$ as one random variable for each $\alpha \in \text{Name}$.

	Differentiable			Locally Lipschitz			
Name	Manual	Ours	Time	Manual	Ours	Time	#CRP
`dpmm-m`	2	2	0.002	2	2	0.002	2
`dpmm-g`	6	6	0.003	6	6	0.003	6
`vae-m`	3	3	0.002	3	3	0.003	3
`vae-g`	4	4	0.002	4	4	0.002	4
`csis-m`	1	1	0.001	1	1	0.001	1
`csis-g`	2	2	0.004	6	6	0.004	6
`br-m`	5	5	0.002	5	5	0.002	5
`br-g`	10	10	0.004	10	10	0.004	10
`lda-m`	3	3	0.002	3	3	0.002	3
`lda-g`	7	7	0.007	7	7	0.007	7
`prodlda-m`	2	2	0.008	2	2	0.007	2
`prodlda-g`	5	5	0.007	5	5	0.006	5
`ssvae-m`	3	3	0.004	3	3	0.003	3
`ssvae-g`	6	6	0.007	6	6	0.009	6

Table 9. Results of variable selections (continued from Table 5). "Ours-Time" denote the runtime of our variable selector in seconds. "Ours-Sound" and "Pyro \ Ours" denote the number of random variables in the example that are in π_{ours}, and that are in π_0 but not in π_{ours}, respectively, where π_{ours} and π_0 denote the reparameterisation plans given by our variable selector and by Pyro. "Pyro \ Ours" is partitioned into "Sound" and "Unsound": the latter denotes the number of random variables that make (R2') or (R3') violated when added to π_{ours}, and the former denotes the number of the rest. "#CR" and "#DR" denote the total number of continuous and discrete random variables in the example. We consider $\{(\alpha, i) \in \text{Name}\}$ as one random variable for each $\alpha \in \text{Name}$.

	Ours		Pyro \ Ours			
Name	Time	Sound	Sound	Unsound	#CR	#DR
`dpmm`	0.007	2	0	0	2	1
`vae`	0.004	1	0	0	1	0
`csis`	0.014	1	0	0	1	0
`br`	0.009	5	0	0	5	0
`lda`	0.011	3	0	0	3	1
`prodlda`	0.018	1	0	0	1	0
`ssvae`	0.013	1	0	0	1	1