

Synthesizing Guide Programs for Sound, Effective Deep Amortized Inference

JIANLIN LI, University of Waterloo, Canada

LENI VEN, University of Waterloo, Canada

PENGYUAN SHI, University of Waterloo, Canada

YIZHOU ZHANG, University of Waterloo, Canada

In probabilistic programming languages (PPLs), a critical step in optimization-based inference methods is constructing, for a given *model* program, a trainable *guide* program. Soundness and effectiveness of inference rely on constructing good guides, but the expressive power of a universal PPL poses challenges. This paper introduces an approach to automatically generating guides for deep amortized inference in a universal PPL. Guides are generated using a type-directed translation per a novel behavioral type system. Guide generation extracts and exploits independence structures using a syntactic approach to conditional independence, with a semantic account left to further work. Despite the control-flow expressiveness allowed by the universal PPL, generated guides are guaranteed to satisfy a critical soundness condition and, moreover, consistently improve training and inference over state-of-the-art baselines for a suite of benchmarks.

This technical report accompanies Li et al. [2023].

1 INTRODUCTION

A Bayesian probabilistic model denotes a distribution $p(\mathbf{Z}, \mathbf{X})$, where random variables \mathbf{Z} are called *latent* and \mathbf{X} *observed*. Bayesian inference is aimed at calculating $p(\mathbf{Z}|\mathbf{X} = \mathbf{x})$, the posterior distribution of latents \mathbf{Z} given observed data \mathbf{x} .

Probabilistic programming languages (PPLs) are powerful tools for modeling Bayesian-inference problems. *Universal PPLs*, in particular, offer linguistic features including stochastic branching and general recursion, making it possible to express mixture models, probabilistic context-free grammars, kernel induction models [Le et al. 2019; Saad et al. 2019], and so forth.

PPLs also provide inference methods for solving inference problems. *Optimization-based* methods, such as variational inference [Jordan et al. 1999], are taking hold in some PPLs empowered by deep neural networks [Bingham et al. 2019; Tran et al. 2018]. They consist in approximating the true posterior $p(\mathbf{Z}|\mathbf{X} = \mathbf{x})$ of a *model* program using a neural *guide* program $q_\phi(\mathbf{Z})$ where ϕ stands for neural-network parameters: they optimize an objective function via gradient descent, searching for ϕ that makes $q_\phi(\mathbf{Z})$ close to $p(\mathbf{Z}|\mathbf{X} = \mathbf{x})$. Notice that guide programs must be free of observed random variables.

Deep amortized inference [Paige and Wood 2016; Ritchie et al. 2016; Le et al. 2017; van de Meent et al. 2021, §8] is based on optimization and profits from learning amortized guides. Amortization is an ingrained idea in machine learning literature [Hinton et al. 1995; Kingma and Welling 2014]. Rather than training neural networks every time a new observation is given, amortized inference trains *ahead of time* a guide program $q_\phi(\mathbf{Z}; \mathbf{x})$ that takes an observation \mathbf{x} as input. Then, at *run time*, inferring $p(\mathbf{Z}|\mathbf{X} = \mathbf{x}_0)$ for actual observations \mathbf{x}_0 is cheap using $q_\phi(\mathbf{Z}; \mathbf{x}_0)$, amortizing the upfront cost of training ϕ . Optimization and amortization help inference scale for probabilistic programs that need to be applied repeatedly to different observations.

Constructing good guides is critical. Ahead-of-time training and run-time inference are (1) sound only when model–guide pairs satisfy *absolute continuity* and (2) are most effective when guides are *faithful yet parsimonious* in terms of the conditional dependences they encode:

- (1) Absolute continuity, informally speaking, is a soundness condition requiring two distributions to have compatible supports. A guide with an incompatible support causes inference to crash or produce incorrect results [Lee et al. 2019].

- (2) Faithful guides do not contain conditional independences not found in models—missing correlations make guides unable to express true posteriors even if they had unbounded neural-network capacities (i.e., number of trainable parameters). Parsimonious guides do not encode more conditional dependences than necessary—excessive correlations introduce computational burdens as ahead-of-time training has to unlearn them to uncover true posteriors [Webb et al. 2018].

Mainstream PPLs currently have limited support for emitting guide programs. In Pyro, for example, the `autoguide` [2022] library works only for non-universal probabilistic programs, does not guarantee absolute continuity, and ignores faithfulness altogether. Creating good guide programs remains a demanding, error-prone task.

This paper introduces an approach to automatically generating guide programs for a universal PPL. Generated guides enjoy strong soundness guarantees and consistently improve training and inference over state-of-the-art approaches, despite the expressiveness allowed by the universal PPL. We first review relevant background in Section 2. We then identify technical challenges and our core contributions in Section 3.

2 BACKGROUND

Bayesian Networks (BN). Absent of branching and recursion, straight-line probabilistic programs are essentially BNs [Pearl 1988]. An example is `model` in Figure 1. In the BN next to it, nodes `a`, `b`, and `obs` are variables in the model. Shaded node `observe(...)` stands for the observe statement. Edges signal dataflow: data in `a`, `b`, and `obs` flow to the observe statement.

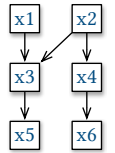
`a` and `b` are *conditionally dependent* on each other, because the observe statement *conditions* the model on `a+b` being close to `obs`. This correlation is visualized by the ground-truth plot labeled $p(a, b | \text{obs})$, which is obtained by the asymptotically exact inference method MCMC.

BN Active trails. Conditional independence can sometimes be read off from BNs. This reasoning is via a notion of active trails [Pearl 1988]. Let X_1, X_2, \dots, X_n be a *trail* (i.e., an undirected, acyclic path) in a BN. The trail is *active given* a (possibly empty) set Z of nodes in the BN when

- (i) for any chain $X_{i-1} \rightarrow X_i \leftarrow X_{i+1}$ (called a *collider*), X_i or one of its descendants is either in Z or a conditioning node (namely `observe(...)`); and
- (ii) no other nodes in the trail is in Z or is a conditioning node.

Whenever there is no active trail between any $X \in X$ and $Y \in Y$ given Z , we have that X and Y are independent given Z (notated $X \perp\!\!\!\perp Y \mid Z$).

Consider the BN to the right. We can assert $x3 \perp\!\!\!\perp x4 \mid x2$: knowing `x2` falsifies condition (ii), deactivating the trail `x3` ← `x2` → `x4`. Similarly, we have $x1 \perp\!\!\!\perp x4 \mid x2$. But we cannot assert $x1 \perp\!\!\!\perp x4 \mid x3$ or $x1 \perp\!\!\!\perp x4 \mid x5$ due to condition (i): knowing `x3` or its descendant `x5` activates the collider `x1` → `x3` ← `x2`. In Figure 1, we cannot assert $a \perp\!\!\!\perp b$: the collider `a` → `observe(...)` ← `b` is active because of the conditioning.



Traces and universal probabilistic programming. A *universal PPL* equips a **Turing**-complete language with constructs for sampling and conditioning [Goodman et al. 2008]. It supports linguistic features including stochastic branching and general recursion, so it allows a stochastic set of random variables (RVs) to be sampled every time a program is run. BNs assume a fixed set of RVs, so in general, BNs cannot express programs in universal PPLs.

Traces are a common semantic notion underlying many universal PPLs [Wood et al. 2014; Siddharth et al. 2017; Bingham et al. 2019; Mansinghka et al. 2018; Cusumano-Towner et al. 2019; Tran et al. 2018]. Running a probabilistic program generates a trace: the trace records RVs and their values sampled in that run. The semantics of a program is characterized by a distribution over the measure space of all possible traces.

Addresses are unique names identifying RVs in a trace. Consider the recursive function `pcfg` (written in Pyro) in Figure 2a, which models a probabilistic context-free grammar. A possible trace of this program is

$$["_A": .7, "_D1_A": .3, "_D1_C": 2, "_D2_A": .1, "_D2_C": 9], \quad (2.1)$$

which will result in the expression `Add(Const(2), Const(9))` in the context-free language.

Deep amortized inference. The ahead-of-time training stage of amortized inference searches for ϕ that makes $q_\phi(\mathbf{Z}; \mathbf{x})$ similar to $p(\mathbf{Z}|\mathbf{X} = \mathbf{x})$ on average for most \mathbf{x} . More precisely, it solves the following optimization problem:

$$\operatorname{argmin}_\phi \mathbb{E}_{\mathbf{x} \sim p(\mathbf{X})} [\text{KL}(p(\mathbf{Z}|\mathbf{x}) \parallel q_\phi(\mathbf{Z}; \mathbf{x}))], \quad (2.2)$$

where $p(\mathbf{Z}|\mathbf{x})$ abbreviates $p(\mathbf{Z}|\mathbf{X} = \mathbf{x})$, and the Kullback–Leibler divergence quantifies how similar two distributions are. Objective (2.2) can be simplified to

$$\operatorname{argmin}_\phi \mathbb{E}_{\mathbf{z}, \mathbf{x} \sim p(\mathbf{Z}, \mathbf{X})} [-\log q_\phi(\mathbf{z}; \mathbf{x})], \quad (2.3)$$

which is convenient: training data $\mathbf{z}, \mathbf{x} \sim p(\mathbf{Z}, \mathbf{X})$ can be generated simply by running the model program $p(\mathbf{Z}, \mathbf{X})$ repeatedly. We note that a variant of amortized inference [Ritchie et al. 2016] optimizes a different objective, known as the evidence lower bound (ELBo):

$$\operatorname{argmax}_\phi \sum_{\mathbf{x}_i} \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{Z}; \mathbf{x}_i)} [\log p(\mathbf{z}, \mathbf{x}_i) - \log q_\phi(\mathbf{z}; \mathbf{x}_i)]. \quad (2.4)$$

Objective (2.4) requires a training dataset $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ be provided, as indicated by the summation over \mathbf{x}_i in (2.4). In this paper, we focus on the first type of amortized inference that uses objective (2.3), but our technical contributions can in principle be applied to the ELBo objective (2.4).

The run-time inference stage performs importance sampling (IS), per the equation

$$p(\mathbf{Z}|\mathbf{x}_0) = p(\mathbf{Z}, \mathbf{x}_0) \left/ \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{Z}; \mathbf{x}_0)} \left[\frac{p(\mathbf{z}, \mathbf{x}_0)}{q_\phi(\mathbf{z}; \mathbf{x}_0)} \right] \right. . \quad (2.5)$$

Given observation \mathbf{x}_0 , IS approximates $p(\mathbf{Z}|\mathbf{x}_0)$ by sampling \mathbf{z} from the trained *proposal* distribution $q_\phi(\mathbf{Z}; \mathbf{x}_0)$ and weighting samples \mathbf{z} by their importance ratio $p(\mathbf{z}, \mathbf{x}_0)/q_\phi(\mathbf{z}; \mathbf{x}_0)$.

3 PROBLEMS AND CONTRIBUTIONS

3.1 Absolute Continuity (AC)

Crucially, soundness of both training and inference relies on AC. First, per formula (2.5), IS is unbiased only when $p(\mathbf{Z}, \mathbf{x}_0)$ is *absolutely continuous* with respect to $q_\phi(\mathbf{Z}; \mathbf{x}_0)$, written as $p(\mathbf{Z}, \mathbf{x}_0) \ll q_\phi(\mathbf{Z}; \mathbf{x}_0)$. That is, in a trace-based PPL, for samples from $q_\phi(\mathbf{Z}; \mathbf{x}_0)$ to asymptotically approximate the posterior, $q_\phi(\mathbf{Z}; \mathbf{x}_0)$ is required to have non-null probability on any measurable set of traces on which $p(\mathbf{Z}, \mathbf{x}_0)$ has non-null probability. Second, objective (2.3) is defined only when $p(\mathbf{Z}, \mathbf{x}) \ll q_\phi(\mathbf{Z}; \mathbf{x})$, lest logarithm-of-zero errors.¹ Furthermore, the ELBo objective (2.4) is defined only when AC holds in the other direction: $q_\phi(\mathbf{Z}; \mathbf{x}_i) \ll p(\mathbf{Z}, \mathbf{x}_i)$. Indeed, AC is a pervasive soundness condition required by many inference methods (e.g., MCMC).

It is simple to verify AC when probabilistic programs can be described as BNs: it suffices to check that two programs contain the same set of RVs. For example, in Figure 1, `model` is absolutely continuous with respect to both guides, even though the guides reverse the order `a` and `b` are sampled. All three programs have the same support made up of two RVs: `a` over \mathbb{R} and `b` over \mathbb{R}_+ .

Control-flow expressiveness of universal PPLs pose challenges to a formal argument, however: branching and recursion allow a stochastic set of RVs to be sampled every time a program is run.

¹Gradient descent further requires differentiability, which is studied in other work (e.g., Mak et al. [2021]).

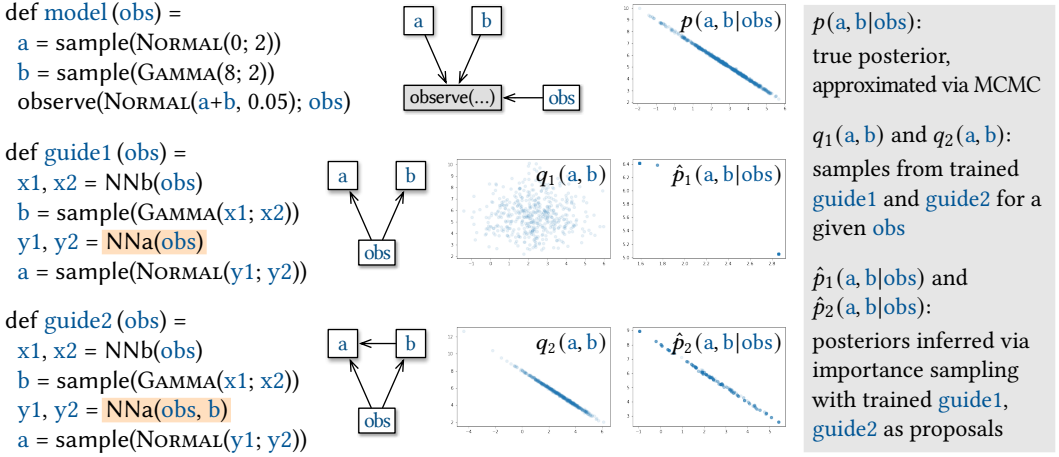


Figure 1. Soundness: `model` is (trivially) absolutely continuous with both `guide1` and `guide2`, despite that the guides reorder `a` and `b`. Faithfulness: `guide2` faithfully captures conditional dependence between `a` and `b` given `obs`, whereas `guide1` considers `a` and `b` independent given `obs`. The payoff of being faithful is that `guide2` leads to higher-quality posterior samples than `guide1`.

Consider the guide in Figure 2b, manually created for the PCFG model in Figure 2a. It might appear that the guide is compatible with the model: half the time it samples a leaf node of the syntax tree, and the other half the time it recursively generates subtrees. But it has two sources of unsoundness:

- Its branching condition $a > .5$, different from the model’s, causes the guide to execute different branches—and thus sample different RVs—than the model. The support mismatch causes accesses to unavailable addresses, crashing a typical inference engine. Notice, however, that reordering of the subtrees by the guide does not change the distribution’s support and is thus not unsoundness.
- Second, RVs at addresses `L + "_C"` have a smaller support (\mathbb{R}_+) than they do in the model (\mathbb{R}). The mismatch can lead to logarithm-of-zero errors, NaNs, or biased estimates.

The run-time errors are frustrating when they happen deep into the gradient-descent loop. Even worse is when programs return normally but with incorrect results.

State-of-the-art approaches. The challenge has led to recent work on type systems for ensuring AC statically [Lew et al. 2019; Wang et al. 2021]: a model and a guide are guaranteed to satisfy AC if they have compatible types. Unfortunately, these systems are not geared to automatic guide generation, and more critically, they restrict expressible models or guides. Lew et al.’s type system supports only limited forms of branching and loops, and it disallows general recursion. Wang et al. address specifically these limitations, but their coroutine-based semantics requires a guide to perform computations in exactly the same order as its model does (thus failing to type-check trivially sound guides such as `guide1` and `guide2` in Figure 1).

These restrictions have implications to modeling power and to performance. Without general recursion, it is difficult to express useful models including probabilistic context-free grammars [Manning and Schütze 1999]. Without the possibility to reorder RVs, faithful guides may require denser dependence structures than otherwise avoidable (see Figure 16, for example), which can reduce training performance as we show later.

3.2 Faithfulness and Parsimony

Faithfulness matters. Effectiveness of IS is highly dependent on having good proposals. In amortized inference, IS yields low-variance estimates if ahead-of-time training makes the proposal distribution

```
a = pcfg("") # main program
x = tensorize(a)
sample("obs", Normal(x,1), obs=obs)
```

```
def pcfg(L): # L is address prefix
  a = sample(L+"_A", Uniform)
  if a < .5:
    c = sample(L+"_C", Normal(0,1))
    b = Const(c)
  else:
    d1 = pcfg(L+"_D1") # left subtree
    d2 = pcfg(L+"_D2") # right subtree
    b = Add(d1, d2)
  return b
```

(a) PCFG model, written in Pyro

```
a = pcfg("", obs) # main program
```

```
def pcfg(L, obs):
  a = sample(L+"_A", Uniform)
  if a > .5:
    C1, C2 = NNC(obs)
    c = sample(L+"_C", Gamma(C1, C2))
    b = Const(c)
  else:
    d2 = pcfg(L+"_D2", obs) # right subtree
    d1 = pcfg(L+"_D1", obs) # left subtree
    b = Add(d1, d2)
  return b
```

(b) PCFG guide, written in Pyro

Figure 2. Implementation of a probabilistic context-free grammar (PCFG). The guide **breaks AC** as required by formulas (2.2) and (2.5). It is **unfaithful**, as it introduces conditional independences not found in the model.

$q_\phi(\mathbf{Z}; \mathbf{x}_0)$ similar in shape to the true posterior $p(\mathbf{Z}|\mathbf{x}_0)$. But an unfaithful guide could not represent the true posterior even if it had infinite neural-network capacities, because it cannot encode some conditional dependences present in the model.

In Figure 1, **guide2** faithfully expresses that **a** and **b** are conditionally dependent given **obs** using the highlighted code. By contrast, **guide1** considers **a** and **b** independent. Thus, after training neural networks **NNa** and **NNb**, **guide2** can approximate true posteriors much better than **guide1**, as shown by plots $q_2(\mathbf{a}, \mathbf{b})$ and $q_1(\mathbf{a}, \mathbf{b})$. In turn, at run time, IS with **guide2** leads to much higher-quality samples than IS with **guide1**, as shown by plots $\hat{p}_2(\mathbf{a}, \mathbf{b}|\mathbf{obs})$ and $\hat{p}_1(\mathbf{a}, \mathbf{b}|\mathbf{obs})$.

Parsimony matters too. Parsimonious guides do not encode more conditional dependences than necessary. Given that training budgets and network capacities are finite, it would be wasteful to expend resources on unlearning spurious correlations that could otherwise be ruled out from the get-go. As we show later, parsimonious guides can indeed lead to better convergence.

Notice that parsimony of dependences is not in conflict with overparameterization of neural networks. Overparameterized networks have many more trainable parameters than there are training examples. It has been observed that they can make training easier while improving generalizability [Zhang et al. 2017]. A parsimonious guide can choose to overparameterize its neural networks.

State-of-the-art approaches. Methods for automating the design of faithful guides exist in the machine learning literature. Webb et al. [2018] construct faithful, parsimonious guides for BN models, but the approach is not applicable to universal PPLs with branching or recursion. For instance, the approach cannot handle the PCFG model in Figure 2a.

In fact, it is not entirely clear how to even manually create a faithful, parsimonious guide for the PCFG model, because of general recursion. Unsoundness aside, the guide in Figure 2b is unfaithful. It is based on the simplifying *mean-field* assumption that RVs are conditionally independent (given **obs**). Mean-field guides are a standard default in practice for variational inference [Zhang et al. 2019], but as our experiments confirm, they may lead to unsatisfying results for PCFG-like models that exhibit strong correlations between RVs.

To handle branching and recursion, Le et al. [2017] construct guides using advanced recurrent neural networks (RNNs) such as an LSTM, effectively treating any two random variables as correlated. Thus, RNN guides may be non-parsimonious, and so ahead-of-time training must work extra hard to uncover independences. As we also show, RNNs' tendency to forget long-range

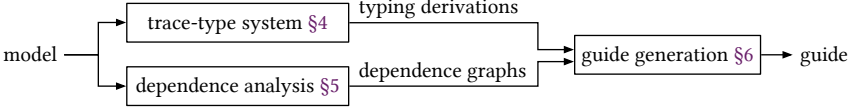


Figure 3. Workflow of the Fidelio guide generator

dependencies may weaken its correlation-preserving guarantee for RVs sampled far apart. What is needed of a guide generator is the ability to take into account structures of probabilistic programs.

3.3 Contributions

Figure 3 shows how our guide generator (called Fidelio) works. In Section 4, we design a behavioral trace-type system capturing probabilistic effects and control flow—compatible typing implies absolute continuity. In Section 5, we construct dependence graphs for probabilistic programs (PDGs) and adapt the notion of active trails from BNs to PDGs—active trails suggest possible correlation. Guide generation is then a type-guided, dependence-aware translation (Section 6).

A unique combination of features in the Fidelio semantics enables it to offer soundness guarantees while permitting expressiveness. First, traces are *tree-structured* rather than lists. So a guide can reorder computations that are subtrees of a parent, while generating the same set of traces as its model. Second, trace types are lightweight dependent types that can track control flow through what we call *checkpoint expressions*. So a model is free to use stochastic branching and general recursion, while being well-typed.

We prove that our type system is sound: compatibly typed model–guide pairs satisfy AC (Theorem 4.5). We prove that generated guides preserve typing (Theorem 6.1). As a result, AC is guaranteed by construction (Theorem 6.2). We also show that our type system allows more models and guides to be typed using a collection of programs from literature (Section 8.1).

Incorporating program dependence information in guide generation is new in itself; existing approaches assume either independence or correlation altogether. Our use of PDGs, augmented with hidden states and empowered by the idea of active trails, enables modularly capturing control dependences that cannot otherwise be expressed by BNs. We note that this treatment of conditional independence is syntactic and not proven sound; we leave a semantic account to future work.

Our evaluation of the dependence analysis focuses on assessing whether it benefits training and inference in practice (Section 8.2). Experiments show that being dependence-aware in guide generation pays off: Fidelio-generated guides consistently improve on state-of-the-art baselines for a diverse set of inference problems.

4 ABSOLUTE CONTINUITY BY TYPING

In Section 4, we establish a reasoning principle (Theorem 4.5) for proving absolute continuity of models and guides, based on typing. Section 6 then uses the reasoning principle to derive the static guarantee of absolute continuity for automatically generated guides.

As alluded to in Section 3, our system addresses AC while permitting expressiveness, by using *tree-structured traces* and *checkpoint-dependent trace types*. We formalize this semantics using the Fidelio Core Calculus (FCC), which captures core aspects of the language supported by Fidelio.

4.1 Syntax of FCC

Figure 4 defines the syntax for FCC programs. Identifiers (i.e., variables and global names) are notated in blue. Capture-free substitution is notated $\cdot\{ \cdot / \cdot \}$: for example, $m\{v/a\}$ substitutes a value v for a term variable a in command m .

| | | | |
|------|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------|
| §4.1 | expressions | $e ::= x \mid a \mid \text{unit} \mid \text{true} \mid \text{false} \mid r \mid n \mid \lambda x. e \mid \text{op}_\diamond(e_1, \dots, e_n) \mid \text{app}(e_1; e_2) \mid \text{let}(e_1; x. e_2) \mid \text{if}(e; e_1; e_2) \mid D(e_1; \dots; e_n)$ | |
| | distributions | $D ::= \text{BERN} \mid \text{UNIF} \mid \text{BETA} \mid \text{GAMMA} \mid \text{NORMAL} \mid \text{CAT} \mid \text{GEO} \mid \text{POIS}$ | |
| | terms | $t ::= \text{sample}(e) \mid m \mid \text{ite}(e; m_1; m_2) \mid \text{call}(f; e_1; e_2)$ | |
| | commands | $m ::= \text{ret}(e) \mid \text{observe}(e_1; e_2); m \mid x = e; m \mid a = t; m$ | |
| | function definitions | $\mathcal{F} ::= \text{def } f\langle a \rangle(x) = m$ | |
| §4.2 | values | $v, d ::= \text{unit} \mid \text{true} \mid \text{false} \mid r \mid n \mid \lambda x. e \mid D(v_1; \dots; v_n)$ | |
| | events | $s ::= \text{atom } v \mid \text{trace } \sigma \mid \text{inj } \sigma \mid \text{fold } \sigma$ | |
| | traces | $\sigma ::= \{\overline{a : s}\}$ | |
| §4.3 | data types | $\tau ::= \mathbb{1} \mid \mathbb{2} \mid \mathbb{R}_{[0,1]} \mid \mathbb{R}_+ \mid \mathbb{R} \mid \mathbb{N}_n \mid \mathbb{N} \mid \tau_1 \rightarrow \tau_2 \mid \text{dist}(\tau)$ | |
| | event types | $S ::= \text{atom } \tau \mid \text{trace } \Sigma \mid \Sigma_1 +_e \Sigma_2 \mid F\langle e \rangle$ | expression variables x, y, h |
| | trace types | $\Sigma ::= \{\overline{a : S}\}_e$ | term variables a, b, c |
| | function signatures | $\mathcal{S} ::= f : \langle \tau_1 \rangle(\tau_2) \rightsquigarrow \tau_3 \# F$ | function names f |
| | trace-type definitions | $\mathcal{T} ::= \text{typedef } F = \forall a : \tau. \Sigma$ | trace-type names F |

Figure 4. Syntax of the Fidelio core calculus (FCC)

FCC has a pure, deterministic fragment and a monadic, probabilistic fragment. The pure fragment consists of *expressions*. It is a simply typed lambda calculus equipped with booleans, natural numbers n , real numbers r , n -ary operations, and various primitive distributions. n -ary operators op_\diamond can be neural networks as well as arithmetic and logical operators. The monadic fragment consists of *terms* and *commands*. They have mutually recursive syntax:

- A term can sample from a distribution $\text{sample}(e)$; be a nested command m ; perform branching $\text{ite}(e; m_1; m_2)$, where the branches are commands; or invoke a function $\text{call}(f; e_1; e_2)$. We shall write $\text{ite}(e; m_1; m_2)$ using if-then-else in examples.
- A command sequences computations, ending in $\text{ret}(e)$. Three forms can be sequenced with a command m to form a larger command: (i) $x = e$ binds an *expression variable* x to expression e in m , (ii) $a = t$ binds a *term variable* a to term t in m , and (iii) $\text{observe}(e_1; e_2)$ conditions on observed data e_2 being drawn from distribution e_1 .

An FCC program is a global set of possibly mutually recursive function definitions $\overline{\mathcal{F}}$, as well as a “main” command. The syntax $\text{def } f\langle a \rangle(x) = m$ assumes for simplicity that f takes two arguments as input (one for a and one for x), but we shall relax this restriction in examples.

Figure 5a defines the same PCFG model program as Figure 2a, using FCC syntax. The program is composed of a main command (top) and a recursive function (bottom). Figure 5b is the automatically generated guide program, also in FCC syntax. By the end of this section, we will be able to show that these two programs have the same trace type.

Sections 4.2–4.4 give an operational semantics, a static semantics, and a measure semantics, respectively, allowing a formal notion of AC for FCC programs.

4.2 Operational Semantics of FCC

This section defines a trace-based, big-step operational semantics, in a style similar to prior trace-based semantics for universal PPLs (e.g., Borgström et al. [2016]). But unlike the prior semantics, traces in FCC are tree-structured rather than lists, and they can record not just sampling but also control-flow events while avoiding imposing a total ordering on these events.

Expressions evaluate to *values*. Judgments have form $e \Downarrow v$. Rules in this pure fragment are standard and omitted herein. A complete semantics can be found in Appendix A.

```

1 a = call(pcfg)
2 x = embed(a)
3 observe(NORMAL(x; 1); obs)
4 ret(unit)

```

```

5 def pcfg () =
6   a = sample(UNIF)
7   b = if a < .5 then
8     c = sample(NORMAL(0; 1))
9     ret(Const(c))
10  else
11    d1 = call(pcfg) // recursive call
12    d2 = call(pcfg) // recursive call
13    ret(Add(d1, d2))
14  ret(b)

```

(a) PCFG model, written in FCC

```

1 h = obs // set hidden state
2 a = call(pcfg; h) // call
3 ret(unit)

```

```

4 def pcfg (h) =
5   x1, x2 = NNa(h)
6   a = sample(BETA(x1; x2))
7   b = if a < .5 then
8     y1, y2 = NNc(h)
9     c = sample(NORMAL(y1; y2))
10    ret(Const(c))
11  else
12    h2 = h // set hidden state
13    d2 = call(pcfg; h2) // recursive call
14    h1 = NNd1(h, embed(d2)) // set hidden state
15    d1 = call(pcfg; h1) // recursive call
16    ret(Add(d1, d2))
17  ret(b)

```

(b) PCFG guide, generated

Figure 5. (a) PCFG model as in Figure 2a, in FCC syntax. (b) Generated guide, with the same trace type.

$$\begin{array}{c}
\boxed{s \vdash_t t \Downarrow^w v} \quad \boxed{\sigma \vdash_m m \Downarrow^w v} \\
\\
\text{(E:SAMPLE)} \quad \frac{e \Downarrow d \quad v \in d.\text{support} \quad w = d.\text{density}(v)}{\text{atom } v \vdash_t \text{sample}(e) \Downarrow^w v} \quad \text{(E:CMD)} \quad \frac{\sigma \vdash_m m \Downarrow^w v}{\text{trace } \sigma \vdash_t m \Downarrow^w v} \quad \text{(E:BRANCH)} \quad \frac{e \Downarrow b \quad \sigma \vdash_m m_b \Downarrow^w v}{\text{inj } \sigma \vdash_t \text{ite}(e; m_{\text{true}}; m_{\text{false}}) \Downarrow^w v} \\
\\
\text{(E:CALL)} \quad \frac{\text{def } f(a)(x) = m \quad e_1 \Downarrow v_a \quad e_2 \Downarrow v_x \quad \sigma \vdash_t m \{v_a/a\} \{v_x/x\} \Downarrow^w v}{\text{fold } \sigma \vdash_t \text{call}(f; e_1; e_2) \Downarrow^w v} \quad \text{(E:OBSERVE)} \quad \frac{v \in d.\text{support} \quad e_1 \Downarrow d \quad e_2 \Downarrow v_2 \quad w_1 = d.\text{density}(v_2) \quad \sigma \vdash_m m \Downarrow^{w_2} v}{\sigma \vdash_m \text{observe}(e_1; e_2); m \Downarrow^{w_1 \cdot w_2} v} \\
\\
\text{(E:BND:TERM)} \quad \frac{s \vdash_t t \Downarrow^{w_1} v_1 \quad \{\overline{a'} : s'\} \vdash_m m \{v_1/a\} \Downarrow^{w_2} v_2}{\{\overline{a} : s, \overline{a'} : s'\} \vdash_m a = t; m \Downarrow^{w_1 \cdot w_2} v_2} \quad \text{(E:BND:EXPR)} \quad \frac{e \Downarrow v_e \quad \sigma \vdash_m m \{v_e/x\} \Downarrow^w v}{\sigma \vdash_m x = e; m \Downarrow^w v} \quad \text{(E:RET)} \quad \frac{e \Downarrow v}{\{\} \vdash_m \text{ret}(e) \Downarrow^1 v}
\end{array}$$

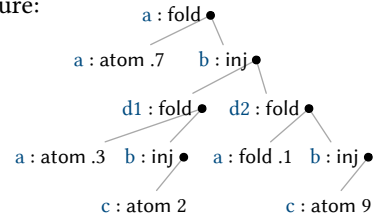
Figure 6. Operational semantics for the monadic fragment of FCC: terms and commands.

Terms are evaluated on *events*, and commands are evaluated on *traces*. As Figure 4 shows, events s and traces σ have mutually recursive syntax.

A trace $\{\overline{a} : s\}$ is an *unordered* map from term variables to events. Events have four forms, corresponding to the four forms of terms: (1) $\text{atom } v$ stands for sampling a value v ; (2) $\text{trace } \sigma$ stands for a nested trace σ ; (3) $\text{inj } \sigma$ stands for branching, where σ traces the chosen branch; and (4) $\text{fold } \sigma$ stands for a function call, where σ traces the function body.

For example, consider trace σ_{main} in (4.1) and its tree structure:

$$\begin{aligned}
\sigma_{\text{main}} &\stackrel{\text{def}}{=} \{a : \text{fold } \sigma_0\} \\
\sigma_0 &\stackrel{\text{def}}{=} \{a : \text{atom } .7, b : \text{inj } \{d1 : \text{fold } \sigma_1, d2 : \text{fold } \sigma_2\}\} \\
\sigma_1 &\stackrel{\text{def}}{=} \{a : \text{atom } .3, b : \text{inj } \{c : \text{atom } 2\}\} \\
\sigma_2 &\stackrel{\text{def}}{=} \{a : \text{atom } .1, b : \text{inj } \{c : \text{atom } 9\}\}
\end{aligned} \quad (4.1)$$



| | | |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|
| $\Delta; \Gamma \vdash_t t : \tau \# S$ | $\Delta; \Gamma \vdash_m m : \tau \# \Sigma$ | $\vdash_{\mathcal{F}} \mathcal{F} : \langle \tau_1 \rangle (\tau_2) \rightsquigarrow \tau_3 \# F$ |
| <p>(S:SAMPLE)</p> $\frac{\Delta; \Gamma \vdash_e e : \text{dist}(\tau)}{\Delta; \Gamma \vdash_t \text{sample}(e) : \tau \# \text{atom } \tau}$ <p>(S:CMD)</p> $\frac{\Delta; \Gamma \vdash_m m : \tau \# \Sigma}{\Delta; \Gamma \vdash_t m : \tau \# \text{trace } \Sigma}$ <p>(S:OBSERVE)</p> $\frac{\Delta; \Gamma \vdash_e e_1 : \text{dist}(\tau) \quad \Delta; \Gamma \vdash_e e_2 : \tau \quad \Delta; \Gamma \vdash_m m : \tau \# \Sigma}{\Delta; \Gamma \vdash_m \text{observe}(e_1; e_2); m : \tau \# \Sigma}$ <p>(S:BND:EXPR)</p> $\frac{\Delta; \Gamma \vdash_e e : \tau_1 \quad \Delta; \Gamma, \mathbf{x} : \tau_1 \vdash_m m : \tau_2 \# \Sigma}{\Delta; \Gamma \vdash_m \mathbf{x} = e; m : \tau_2 \# \Sigma}$ | <p>(S:BRANCH)</p> $\frac{\Delta; \bullet \vdash_e e : \mathcal{2} \quad \Delta; \Gamma \vdash_m m_1 : \tau \# \Sigma_1 \quad \Delta; \Gamma \vdash_m m_2 : \tau \# \Sigma_2}{\Delta; \Gamma \vdash_t \text{ite}(e; m_1; m_2) : \tau \# \Sigma_1 +_e \Sigma_2}$ <p>(S:CALL)</p> $\frac{f : \langle \tau_1 \rangle (\tau_2) \rightsquigarrow \tau_3 \# F \quad \Delta; \bullet \vdash_e e_1 : \tau_1 \quad \Delta; \Gamma \vdash_e e_2 : \tau_2}{\Delta; \Gamma \vdash_t \text{call}(f; e_1; e_2) : \tau_3 \# F(e_1)}$ <p>(S:BND:TERM)</p> $\frac{\Delta; \Gamma \vdash_t t : \tau_1 \# S \quad \Delta, \mathbf{a} : \tau_1; \Gamma \vdash_m m : \tau_2 \# \{\overline{\mathbf{a}' : S'}\}_e}{\Delta; \Gamma \vdash_m \mathbf{a} = t; m : \tau_2 \# \{\mathbf{a} : S, \overline{\mathbf{a}' : S'}\}_e}$ <p>(S:DEF)</p> $\frac{f : \langle \tau_1 \rangle (\tau_2) \rightsquigarrow \tau_3 \# F \quad \text{typedef } F = \forall \mathbf{a} : \tau_1. \Sigma \quad \mathbf{a} : \tau_1; \mathbf{x} : \tau_2 \vdash_m m : \tau_3 \# \Sigma}{\vdash_{\mathcal{F}} \text{def } f(\mathbf{a})(\mathbf{x}) = m : \langle \tau_1 \rangle (\tau_2) \rightsquigarrow \tau_3 \# F}$ | <p>(S:RETURN)</p> $\frac{\Delta; \bullet \vdash_e e : \tau}{\Delta; \Gamma \vdash_m \text{ret}(e) : \tau \# \{\}_e}$ |

Figure 7. Typing rules for the monadic fragment of FCC: terms, commands, and functions.

It is a trace of the PCFG model (Figure 5a). It is also a trace of the PCFG guide (Figure 5b), although the guide reorders **d1** and **d2**. Evaluating either program on this trace generates $\text{Add}(\text{Const}(2), \text{Const}(9))$ in the context-free language.

Contrast (4.1) with (2.1). Unlike the trace in (2.1), FCC traces do not stipulate a *total* ordering on events. An added benefit is that unsafe manual name mangling of the kind found in Figure 2a is avoided. That Pyro program would lead to name collisions and crash the runtime if not for manual name mangling such as $L + _C$. By contrast, in FCC, RVs in trace σ_{main} are uniquely identified even if they may correspond to the same term variable in the program text.

Figure 6 defines how terms and commands evaluate. Rules have forms $s \vdash_t t \Downarrow^w v$ and $\sigma \vdash_m m \Downarrow^w v$, meaning that on trace σ (resp. event s), command m (resp. term t) evaluates to v and produces *weight* w . A command denotes a distribution over traces; weight w is the (unnormalized) density of the trace. In **E:SAMPLE**, event $\text{atom } v$ specifies the value v sampled, and the weight is adjusted by the distribution's density at v . In **E:BRANCH**, event $\text{inj } \sigma$ specifies the nested trace σ on which to evaluate the chosen branch. In **E:CALL**, event $\text{fold } \sigma$ specifies the nested trace σ on which to evaluate the function body. **E:BND:TERM** evaluates $\mathbf{a} = t; m$ by splitting the trace into an event s and a smaller trace $\{\overline{\mathbf{a}' : S'}\}$ under which t and m are respectively evaluated. **E:BND:EXPR** need not split the trace, as expression variables are not captured in traces. **E:OBSERVE** performs conditioning, multiplying the weight by the density of the observed data.

4.3 Static Semantics of FCC

Typing judgments of the pure fragment of FCC have form $\Delta; \Gamma \vdash_e e : \tau$. The rules are standard and omitted herein. Δ are Γ are environments of term variables and expression variables, respectively:

$$\Delta ::= \bullet \mid \Delta, \mathbf{a} : \tau \quad \Gamma ::= \bullet \mid \Gamma, \mathbf{x} : \tau$$

A distribution has type $\text{dist}(\tau)$; samples from the distribution have type τ . For example, a Gamma distribution $\text{GAMMA}(e_1; e_2)$ has type $\text{dist}(\mathbb{R}_+)$; the support is positive reals.

Typing commands and terms. *Trace types* (resp. *event types*) describe control-flow behaviors and probabilistic effects of commands (resp. *terms*). As Figure 4 shows, trace types Σ and event types S

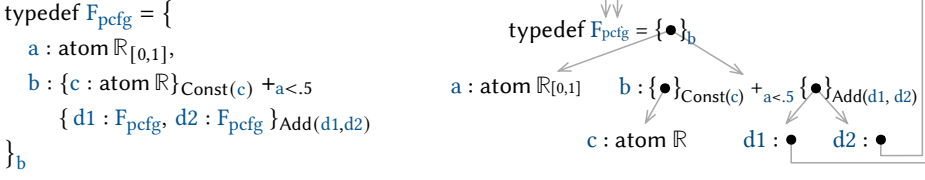


Figure 8. Trace type of the `pcfg` functions in Figure 5.

have mutually recursive syntax. The trace type of a command has form $\{\overline{a : S}\}_e$, where $\overline{a : S}$ is an *unordered* map from term variables to event types, and e is the command’s return. Event types have four forms, corresponding to the four forms of events (or terms): `atom` τ , trace Σ , $\Sigma_1 +_e \Sigma_2$, and $F\langle e \rangle$.

Figure 7 defines the typing rules for commands and terms, which have forms $\Delta; \Gamma \vdash_m m : \tau \# \Sigma$ and $\Delta; \Gamma \vdash_t t : \tau \# S$. They make sure that sampling, branching, and invocation of (possibly) recursive functions are recorded in trace types and event types. Figure 7 also gives the typing rule for global function definitions (`S:DEF`). Metavariable F ranges over names of functions’ trace types. For example, function `pcfg` in Figure 5a has trace type F_{pcfg} , defined recursively in Figure 8. The main command in Figure 5a then has trace type $\{a : F_{\text{pcfg}}\}_{\text{unit}}$.

Two compatibly typed programs need not agree on the orders computations take place. In particular, it is possible for a compatibly typed guide to reorder computations under the same parent in a tree-structured trace type. For instance, the guide `pcfg` function in Figure 5b reorders subtrees `d1` and `d2`, but it can still be typed as F_{pcfg} . The guide also does extra computations such as running neural networks. These computations are pure; they are not part of the guide’s trace type.

Checkpoints are part of trace types. Two compatibly typed programs must agree on control-flow decisions. F_{pcfg} captures the branching condition $a < .5$. So for the guide `pcfg` function to also have type F_{pcfg} , it should use $a < .5$ as the branching condition. More generally, two compatibly typed programs must agree on *checkpoint expressions*, which include branching condition e in `ite`($e; m_1; m_2$), command return e in `ret`(e), and term argument e_1 in `call`($f; e_1; e_2$). For example, trace type F_{pcfg} captures not only the branching condition $a < .5$ but also command returns such as $\{\dots\}_{\text{Const}(c)}$ and $\{\dots\}_b$. Consider the following command:

$$a = \text{call}(\text{pcfg}); \quad b = \text{ite}(\text{height}(a) < 6; m_1; m_2); \quad \dots$$

If a model `pcfg` and its guide were allowed to disagree on command returns, then even though two programs use the same branching condition `height(a) < 6`, they could end up making different branching choices because they would disagree on a . Hence, to use trace typing to justify AC, the type system must keep track of checkpoints as part of trace types.

A consequence is that checkpoints can use only term variables—expression variables are not bound in trace types. In `S:BRANCH`, `S:RETURN`, and `S:CALL`, checkpoint expressions are type-checked under an empty environment \bullet of expression variables. The requirement does not restrict the class of model programs that can be expressed, because the programmer can always use term variables and monadic returns to fix a program that would otherwise not type-check with expression variables. For example, the program below does not type-check because z , an expression variable, is used in the checkpoint $z > 0$:

$$a = \text{sample}(\text{NORMAL}(x; y)); \quad z = a + 1; \quad c = \text{ite}(z > 0; m_1; m_2); \quad \dots$$

This program is easily fixed by replacing z with a term variable b and replacing $a + 1$ with `ret(a + 1)`.

It may appear restrictive that a guide must have syntactically identical checkpoints as its model. We found that in practice it is often trivial to modify manually created guides to meet this requirement. More importantly, it fits in a compiler pipeline where guides are automatically generated. Another benefit is that it allows typing to be entirely syntax-directed, thus striking a good balance between simplicity and expressivity.

There exist distributions whose supports depend on parameters. Like prior systems, the expressive power of FCC is limited by the lack of such dependently supported distributions in general. However, we believe that checkpoints offer a viable approach to incorporating them in the future. In particular, sampling from a delta distribution whose support is concentrated on e can be encoded as $\mathbf{a} = \text{ret}(e)$: because this checkpoint is part of the command’s trace type, two programs are required to agree on e to be compatibly typed.

Term variables are more than binders. Term variables in FCC are unusual, in the sense that they are not just binders, which have local scopes, but also labels, which can occur in trace types. Thus, renaming the term variables in a command will cause it to have a different trace type. In a trace type $\{\mathbf{a}_1 : S_1, \dots, \mathbf{a}_N : S_N\}_e$, a term variable \mathbf{a}_i can occur in e and the event types of its siblings.

We note that it is possible to adapt the FCC design so that binders and labels are decoupled, using a syntax like $\mathbf{x} = \text{sample}(\ell; \text{NORMAL}(0; 1))$, where \mathbf{x} is a binder and ℓ a label. We choose to represent the binder and the label as one single term variable, for less verbiage.

4.4 Measure Semantics of FCC

Definition 4.1. A closed command m of trace type $\Sigma (\bullet; \bullet \vdash_m m : \tau \# \Sigma)$ induces a measure $\llbracket m \rrbracket$ over a measurable space $\llbracket \Sigma \rrbracket$ of traces:

$$\mathbf{P}_m(\sigma) \stackrel{\text{def}}{=} \begin{cases} w, & \text{if } \sigma \vdash_m m \Downarrow^w v \\ 0, & \text{otherwise} \end{cases} \quad \llbracket m \rrbracket(A) \stackrel{\text{def}}{=} \int_A \mathbf{P}_m(\sigma) \mu_\Sigma(d\sigma)$$

Measure $\llbracket m \rrbracket$ is defined by integrating a density function $\mathbf{P}_m(\cdot)$ over a measurable set A of traces. As is usual, only terminating executions have a positive density. Measure $\llbracket m \rrbracket$ is in general unnormalized as m may perform conditioning. When the normalization constant is in $(0, \infty)$, the normalized measure exists and is given by $\int_A \mathbf{P}_m(\sigma) d\sigma / \int \mathbf{P}_m(\sigma) d\sigma$.

The construction of the measurable space $\llbracket \Sigma \rrbracket$ of traces, as well as that of its stock measure μ_Σ , is by a double induction, first on a “step index” for handling recursive trace types, and then on the structure of trace types and event types. The definition can be found in Appendix A.2. We omit showing the measurability of $\mathbf{P}_m(\cdot)$; we expect a proof to follow the technique found in Borgström et al. [2016] and Szymczak and Katoen [2019].

With the measure semantics for commands defined, we can formalize the notion of absolute continuity, which says the measure induced by one command is “supported” by that of the other:

Definition 4.2. A command m_1 is *absolutely continuous* with respect to another m_2 , if $\llbracket m_2 \rrbracket(A) \neq 0$ on every measurable set A of traces for which $\llbracket m_1 \rrbracket(A) \neq 0$.

4.5 Theoretical Results

Before we can establish the theorems, we introduce a notion of typing for traces and events, namely $s \vdash_s S \Downarrow v$ and $\sigma \vdash_\sigma \Sigma \Downarrow v$ defined in Figure 9. The typing rules evaluate checkpoint expressions in types to make sure that they agree with what happens in the trace. The evaluation happens in the pure fragment of FCC. Using the rules, we can derive that trace σ_{main} from (4.1) is well-typed: $\sigma_{\text{main}} \vdash_\sigma \{ \mathbf{a} : \mathbf{F}_{\text{pcfg}} \}_{\text{unit}} \Downarrow \text{unit}$.

$$\begin{array}{c}
\boxed{s \vdash_S S \Downarrow v} \quad \boxed{\sigma \vdash_\sigma \Sigma \Downarrow v} \\
\text{(W:EVENT:ATOM)} \quad \text{(W:EVENT:TRACE)} \quad \text{(W:EVENT:BRANCH)} \\
\frac{v : \tau}{\text{atom } v \vdash_S \text{atom } \tau \Downarrow v} \quad \frac{\sigma \vdash_\sigma \Sigma \Downarrow v}{\text{trace } \sigma \vdash_S \text{trace } \Sigma \Downarrow v} \quad \frac{e \Downarrow b \quad \sigma \vdash_\sigma \Sigma_b \Downarrow v}{\text{inj } \sigma \vdash_S \Sigma_{\text{true}} +_e \Sigma_{\text{false}} \Downarrow v} \\
\text{(W:EVENT:CALL)} \quad \text{(W:TRACE:RET)} \quad \text{(W:TRACE:BNB)} \\
\frac{\text{typedef } \mathbf{F} = \forall \mathbf{a} : \tau. \Sigma \quad e \Downarrow v_a \quad \sigma \vdash_S \Sigma \{v_a/a\} \Downarrow v}{\text{fold } \sigma \vdash_S \mathbf{F}\langle e \rangle \Downarrow v} \quad \frac{e \Downarrow v}{\{\} \vdash_\sigma \{\} e \Downarrow v} \quad \frac{s \vdash_S S \Downarrow v_1 \quad \overline{\{a' : s'\}} \vdash_\sigma \overline{\{a' : S'\}}_e \{v_1/a\} \Downarrow v}{\{a : s, \overline{a' : s'}\} \vdash_\sigma \{a : S, \overline{a' : S'}\}_e \Downarrow v}
\end{array}$$

Figure 9. Well-formedness of events and traces with respect to event types and trace types.

Typing of traces is needed solely as a theoretical device to establish Theorem 4.5 through Theorems 4.3 and 4.4: a command m of trace type Σ has positive density $\mathbf{P}_m(\sigma) > 0$ on a trace σ if and only if σ has type Σ . The guide generator need not type-check traces.

We first present type safety results for the operational semantics. The PRESERVATION theorem states that if a closed, well-typed command (resp. term) evaluates to some value under some trace (resp. event), then the value is well-typed and so is the trace (resp. event):

THEOREM 4.3 (PRESERVATION).

- (1) If $\bullet; \bullet \vdash_m m : \tau \# \Sigma$, and $\sigma \vdash_m m \Downarrow^w v$, then $\bullet; \bullet \vdash_e v : \tau$ and $\sigma \vdash_\sigma \Sigma \Downarrow v$.
- (2) If $\bullet; \bullet \vdash_t t : \tau \# S$ and $s \vdash_t t \Downarrow^w v$, then $\bullet; \bullet \vdash_e v : \tau$ and $s \vdash_S S \Downarrow v$.

The NORMALIZATION theorem states that if a command and a trace (resp. a term and an event) can be typed using the same trace type (resp. event type), then evaluation of the command (resp. term) under the trace (resp. event) terminates to some value:

THEOREM 4.4 (NORMALIZATION).

- (1) If $\bullet; \bullet \vdash_m m : \tau \# \Sigma$ and $\sigma \vdash_\sigma \Sigma \Downarrow v$, then there exist w such that $\sigma \vdash_m m \Downarrow^w v$.
- (2) If $\bullet; \bullet \vdash_t t : \tau \# S$ and $s \vdash_S S \Downarrow v$, then there exist w such that $s \vdash_t t \Downarrow^w v$.

Normalization of the pure fragment of FCC follows from the well-known result for the simply typed lambda calculus, and normalization of the monadic fragment is a result of traces being finite.

Using PRESERVATION and NORMALIZATION, we can show that two compatibly typed commands are mutually absolutely continuous:

THEOREM 4.5 (ABSOLUTE CONTINUITY BY TYPING). *If $\bullet; \bullet \vdash_m m_1 : \tau \# \Sigma$ and $\bullet; \bullet \vdash_m m_2 : \tau \# \Sigma$, then for any measurable set A of traces, $\llbracket m_1 \rrbracket(A) \neq 0$ if and only if $\llbracket m_2 \rrbracket(A) \neq 0$.*

Proofs of the theorems can be found in Appendix A

We now specialize Theorem 4.5 to amortized inference of the PCFG model in Figure 5. There, amortization is over a free variable **obs** in the model m_m and in the guide m_g . As discussed in Section 3.1, both training and importance sampling require that $m_g \{v_{\text{obs}}/\text{obs}\}$ be absolutely continuous with respect to $m_m \{v_{\text{obs}}/\text{obs}\}$. Let $\Sigma \stackrel{\text{def}}{=} \{a : \mathbf{F}_{\text{pcfg}}\}_{\text{unit}}$. Because $\bullet; \bullet \vdash_m m_m \{v_{\text{obs}}/\text{obs}\} : \mathbb{1} \# \Sigma$ and $\bullet; \bullet \vdash_m m_g \{v_{\text{obs}}/\text{obs}\} : \mathbb{1} \# \Sigma$, we know that AC is guaranteed to hold for m_m and m_g by Theorem 4.5. What remains to be shown is that m_g can be automatically generated and that generated guides have the right trace types (Section 6).

5 IDENTIFYING CORRELATIONS AND INDEPENDENCES

We want a generated guide to faithfully yet parsimoniously respect conditional dependences. However, the control-flow expressiveness of a universal PPL poses challenges.

Consider conditional dependences in the PCFG model in Figure 5a. The model uses stochastic branching and general recursion, so it is impossible to express it as a BN. But given any of its traces, we can unfold recursion for that trace, as Figure 10 does. Nodes \square stand for returns of recursive calls to `pcfg`, and arrows signify dataflow.

Absent of control flow, the dataflow graph is essentially a BN. Per the notion of BN active trails, colliders in Figure 10 mean that leaf nodes \square are correlated with each other and with obs . But we had to unroll recursion to identify such correlations for a given trace. Yet it is infeasible to unroll recursion for every possible trace, as general recursion yields traces of infinitely many shapes.

Adding to the complexities is that *control dependences* may or may not transmit correlation.

- In Figure 5a, we know that `a` (line 6) and `pcfg`'s return (line 14) must be *correlated*. We also know that this correlation is due to control dependence: `a` determines which branch is taken and thus influences `pcfg`'s return value.
- In contrast, we know that `a`'s value and `c`'s value (line 8) must be *independent given $a < .5$* . Yet `c` is control-dependent on `a`: `a` determines if `c` is sampled in the first place.

How can a guide generator identify such correlations and independences that involve control flow?

Fidelio extracts program dependences into a graph representation (Section 5.1). It uses a notion of active trails for the graph representation as an indicator of correlation (Sections 5.2 and 6).

5.1 A Graph Representation of Dependences

Program dependence graphs (PDGs) [Ferrante et al. 1987], employed by many optimizing compilers, are a classic idea to represent dependences between program elements. PDGs are appealing because they represent not only data dependences (as BNs do) but also control dependences. The exact definition of PDGs are tailored to specific languages and problems being considered.

Figure 11a shows two PDGs, one for the main command and the other for the `pcfg` function in Figure 5a. They are generated by the construction in Figure 12. While the presentation of Figure 12 is dense, the construction is mostly standard. It defines three sets $\text{Nodes}_{\mathcal{F}}(\mathcal{F})$, $\text{CDE}_{\mathcal{F}}(\mathcal{F})$, and $\text{DDE}_{\mathcal{F}}(\mathcal{F})$ by structural induction. The sets contain nodes, control-dependence edges (CDEs), and data-dependence edges (DDEs) of the PDG.

There are two types of nodes in a PDG: *control nodes* and *data nodes*. Control nodes are drawn as rounded rectangles \bigcirc . They include ENTRY , one per function (or main command), and e , one per branching term $\text{ite}(e; m_1; m_2)$. Metavariable $C ::= \text{ENTRY} \mid \text{e}$ ranges over control nodes. Control nodes are sources of CDEs, which are drawn as gray, dashed arrows.

CDEs can have labels $\ell ::= \top \mid \text{F} \mid \varepsilon$. CDEs from a branching node e are labeled either \top or F , signifying the branch taken. For example, in Figure 11a, all CDEs from $(a < .5)$ are labeled. CDEs from ENTRY are not labeled (i.e., they have the null label ε). In Figure 12, the construction $\text{CDE}_m(m) \ell C$ (resp. $\text{CDE}_t(t) \ell C$) defines CDEs induced by a command (resp. a term), where C is the current *control parent*, and ℓ labels the CDE from the control parent.

Data nodes are drawn as rectangles \square . They are created for variable bindings a and obs , monadic returns $\text{ret}(e)$, call arguments e , and conditionings $\text{observe}(e_1; e_2)$. Data nodes are sources of DDEs, which are drawn as black, solid arrows.

In Figure 12, $\text{DDE}_m(m) \gamma N$ (resp. $\text{DDE}_t(t) \gamma N$) defines DDEs induced by a command (resp. a term). To focus on terms in the monadic fragment, $\text{Nodes}_m(m)$ does not create data nodes for expression variables. However, expression variables can transmit data dependence too. Thus, $\text{DDE}_m(m) \gamma N$ takes as input a substitution $\gamma ::= \bullet \mid \gamma, x \mapsto e$ for expression variables occurring

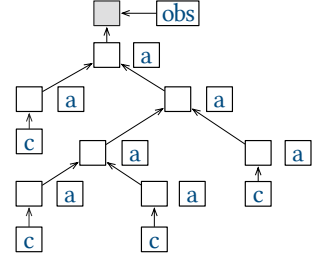
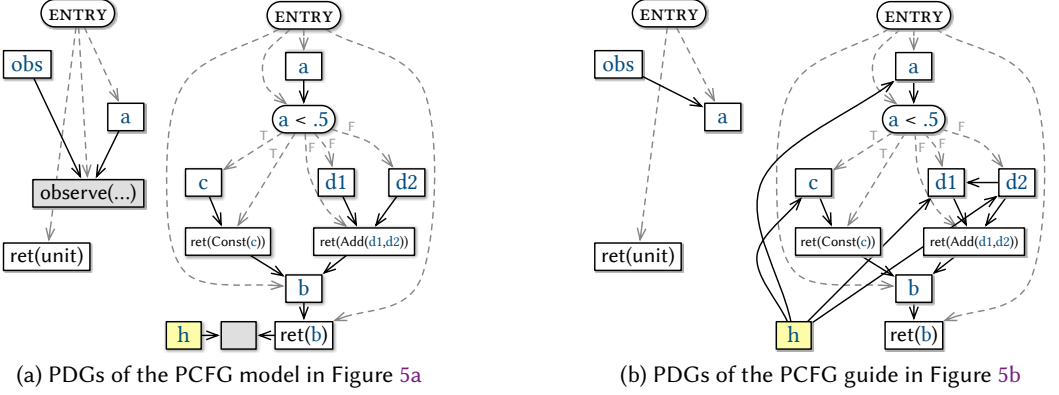


Figure 10. Dataflow graph for a single trace of the PCFG model.



(a) PDGs of the PCFG model in Figure 5a

(b) PDGs of the PCFG guide in Figure 5b

Figure 11. For the PCFG model program, Fidelio uses the construction in Figure 12 to build the PDGs in (a) before generating the guide program. For comparison, guide PDGs are shown in (b). The PDG construction inserts the collider $\boxed{h} \rightarrow \square \leftarrow \text{ret}(b)$, creating active trails between \boxed{h} and nodes in the PDG. Informed by the active trails, the guide generator makes sample and call terms depend on h (e.g., $\boxed{h} \rightarrow a$ in (b)), thus faithfully expressing their correlation to historical information stored in h (Section 6).

free in m , and it uses an auxiliary function $\text{TV}_\gamma(e)$ to obtain the term variables on which an expression e transitively depends through expression variables.

The other parameter N in $\text{DDE}_m(m) \gamma N$ is the data node to which the result of evaluating m flows. It can be either a term variable \boxed{a} or a synthetic conditioning sink \square to which a function’s return flows. It is assumed that *stochastically* recursive functions like `pcfg` do not perform conditioning themselves, since in amortized inference, amortization is always over a *statically known* set of observe sites.

Besides the synthetic \square , construction $\text{Nodes}_\mathcal{F}(\mathcal{F})$ also introduces a synthetic \boxed{h} , and $\text{DDE}_\mathcal{F}(\mathcal{F})$ adds $\boxed{h} \rightarrow \square$. For example, see collider $\boxed{h} \rightarrow \square \leftarrow \text{ret}(b)$ in Figure 11a. As Section 6 discusses, Fidelio makes \boxed{h} store historical information to which RVs in the current function are correlated.

5.2 A Notion of Active Trails for PDGs

As Section 2 reviews, in BNs, absence of active trails implies conditional independence. We adapt the notion of active trails to PDGs. We do not claim soundness or completeness of this adapted notion in the general case, though it is sound and almost complete for BN-like programs. Soundness and completeness are theoretically appealing properties, but they require much more measure-theoretic sophistication and we leave them to future work. We satisfy ourselves with experimental evidence that absence (resp. presence) of active trails is a good indicator of independence (resp. correlation) (Figure 13), and with an extensive evaluation of how dependence awareness impacts the effectiveness of deep amortized inference (Section 8.2).

A definition of PDG active trails must take into consideration control dependences. Our definition uses a few terminologies. A *control path*, starting from $\overline{\text{ENTRY}}$, is a directed path consisting solely of CDEs. A control node C is a *control ancestor* of a PDG node N if C is on the control path to N . Two nodes *control-contradict* each other if their control paths travel through mutually exclusive branches of the same branching node.

Definition 5.1 (PDG active trails). Let \mathcal{G} be a PDG. Let \mathbf{M} be a set of nodes in \mathcal{G} . Let N_1, N_2, \dots, N_n be a trail in \mathcal{G} (i.e., an undirected, acyclic path composed of CDEs and DDEs). The trail is said to be *active given* \mathbf{M} when all the following conditions hold:

- (1) Control ancestors of N_1 and those of N_n are in \mathbf{M} .
- (2) \mathbf{M} is closed under control ancestry. That is, control ancestors of any node $M \in \mathbf{M}$ are in \mathbf{M} , too.

$$\begin{aligned}
\text{Nodes}_{\mathcal{F}}(\text{def } f\langle a \rangle = m) &\stackrel{\text{def}}{=} \{\overline{\text{ENTRY}}, \boxed{a}, \boxed{h}, \boxed{}\} \cup \text{Nodes}_m(m) && \# \boxed{h} \text{ and } \boxed{} \text{ are synthetic nodes} \\
\text{Nodes}_m(\text{ret}(e)) &\stackrel{\text{def}}{=} \{\overline{\text{ret}(e)}\} \\
\text{Nodes}_m(\text{observe}(e_1; e_2); m) &\stackrel{\text{def}}{=} \{\overline{\text{observe}(e_1; e_2)}\} \cup \text{Nodes}_m(m) \\
\text{Nodes}_m(x = e; m) &\stackrel{\text{def}}{=} \text{Nodes}_m(m) \\
\text{Nodes}_m(a = t; m) &\stackrel{\text{def}}{=} \{\boxed{a}\} \cup \text{Nodes}_t(t) \cup \text{Nodes}_m(m) \\
\text{Nodes}_t(m) &\stackrel{\text{def}}{=} \text{Nodes}_m(m) \\
\text{Nodes}_t(\text{ite}(e; m_1; m_2)) &\stackrel{\text{def}}{=} \{\overline{e}\} \cup \text{Nodes}_m(m_1) \cup \text{Nodes}_m(m_2) \\
\text{Nodes}_t(\text{sample}(e)) &\stackrel{\text{def}}{=} \emptyset \\
\text{Nodes}_t(\text{call}(f; e)) &\stackrel{\text{def}}{=} \boxed{e} \\
\\
\text{CDE}_{\mathcal{F}}(\text{def } f\langle a \rangle = m) &\stackrel{\text{def}}{=} \text{CDE}_m(m) \varepsilon \overline{\text{ENTRY}} && \# m \text{ has control parent } \overline{\text{ENTRY}} \\
\text{CDE}_m(\text{ret}(e)) \ell C &\stackrel{\text{def}}{=} \{C \xrightarrow{\ell} \overline{\text{ret}(e)}\} \\
\text{CDE}_m(\text{observe}(e_1; e_2); m) \ell C &\stackrel{\text{def}}{=} \{C \xrightarrow{\ell} \overline{\text{observe}(e_1; e_2)}\} \cup \text{CDE}_m(m) \ell C \\
\text{CDE}_m(x = e; m) \ell C &\stackrel{\text{def}}{=} \text{CDE}_m(m) \ell C \\
\text{CDE}_m(a = t; m) \ell C &\stackrel{\text{def}}{=} \{C \xrightarrow{\ell} \boxed{a}\} \cup \text{CDE}_t(t) \ell C \cup \text{CDE}_m(m) \ell C \\
\text{CDE}_t(m) \ell C &\stackrel{\text{def}}{=} \text{CDE}_m(m) \ell C \\
\text{CDE}_t(\text{ite}(e; m_1; m_2)) \ell C &\stackrel{\text{def}}{=} \{C \xrightarrow{\ell} \overline{e}\} \cup \text{CDE}_m(m_1) \top \overline{e} \cup \text{CDE}_m(m_2) \text{F} \overline{e} && \# m_i \text{ has control parent } \overline{e} \\
\text{CDE}_t(\text{sample}(e)) \ell C &\stackrel{\text{def}}{=} \emptyset \\
\text{CDE}_t(\text{call}(f; e)) \ell C &\stackrel{\text{def}}{=} \{C \xrightarrow{\ell} \boxed{e}\} \\
\\
\text{DDE}_{\mathcal{F}}(\text{def } f\langle a \rangle = m) &\stackrel{\text{def}}{=} \{\boxed{h} \rightarrow \boxed{}\} \cup \text{DDE}_m(m) \bullet \boxed{} && \# \text{function's return flows into } \boxed{} \\
\text{DDE}_m(\text{ret}(e)) \gamma N &\stackrel{\text{def}}{=} \{\boxed{a} \rightarrow \overline{\text{ret}(e)} \mid a \in \text{TV}_{\gamma}(e)\} \cup \{\overline{\text{ret}(e)} \rightarrow N\} \\
\text{DDE}_m(\text{observe}(e_1; e_2); m) \gamma N &\stackrel{\text{def}}{=} \{\boxed{a} \rightarrow \overline{\text{observe}(e_1; e_2)} \mid a \in \text{TV}_{\gamma}(e_1) \cup \text{TV}_{\gamma}(e_2)\} \cup \text{DDE}_m(m) \gamma N \\
\text{DDE}_m(x = e; m) \gamma N &\stackrel{\text{def}}{=} \text{DDE}_m(m) (\gamma, x \mapsto e) N && \# \text{substitution } \gamma \text{ is extended} \\
\text{DDE}_m(a = t; m) \gamma N &\stackrel{\text{def}}{=} \text{DDE}_t(t) \gamma \boxed{a} \cup \text{DDE}_m(m) \gamma N && \# \text{evaluation result of } t \text{ flows into } \boxed{a} \\
\text{DDE}_t(m) \gamma N &\stackrel{\text{def}}{=} \text{DDE}_m(m) \gamma N \\
\text{DDE}_t(\text{ite}(e; m_1; m_2)) \gamma N &\stackrel{\text{def}}{=} \{\boxed{a} \rightarrow \overline{e} \mid a \in \text{TV}_{\gamma}(e)\} \cup \text{DDE}_m(m_1) \gamma N \cup \text{DDE}_m(m_2) \gamma N \\
\text{DDE}_t(\text{sample}(e)) \gamma N &\stackrel{\text{def}}{=} \{\boxed{a} \rightarrow N \mid a \in \text{TV}_{\gamma}(e)\} \\
\text{DDE}_t(\text{call}(f; e)) \gamma N &\stackrel{\text{def}}{=} \{\boxed{a} \rightarrow \boxed{e} \mid a \in \text{TV}_{\gamma}(e)\} \cup \{\boxed{e} \rightarrow N\} \\
\text{TV}_{\gamma}(e) &\stackrel{\text{def}}{=} \{a \mid a \in \text{FV}(e)\} \cup \{a \mid x \in \text{FV}(e) \wedge a \in \text{TV}_{\gamma, x}(\gamma(x))\}
\end{aligned}$$

Figure 12. Constructing PDGs for model programs.

- (3) No node in the trail control-contradicts N_1 , N_n , or M .
- (4) The trail is active given M , in the sense of Bayesian networks. That is,
 - (i) for any collider $N_{i-1} \rightarrow N_i \leftarrow N_{i+1}$ in the trail, N_i or one of its descendants is either in M or a conditioning node (namely $\overline{\text{observe}(\dots)}$ or $\boxed{}$);
 - (ii) no other node in the trail is in M or is a conditioning node.

Only when two nodes are both reached in the same execution does it make sense to speak of correlation between them. Condition (1) ensures that when we speak of conditional dependence between N_1 and N_n , we do condition on the branching choices that caused N_1 and N_n to be reached in the first place. Condition (2) further ensures that the branching nodes themselves are reached. Condition (3) says that correlation cannot be transmitted through nodes impossible to be reached when N_1 , N_n , and M are all reached. Finally, condition (4) requires that a PDG active trail be active in the BN sense.

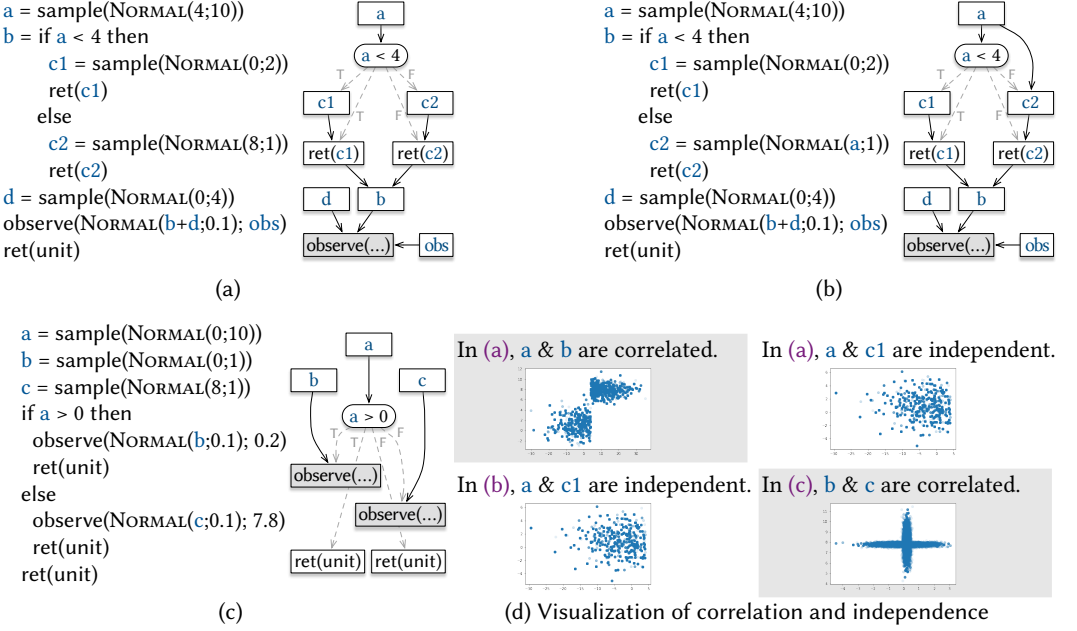


Figure 13. Independence (resp. correlation) as visualized in (d) agrees with the absence (resp. presence) of PDG active trails per Definition 5.1. (ENTRY nodes have been elided in PDGs.)

By conditions (1) and (2), $\overline{\text{ENTRY}}$ is always in M , which by condition (4) implies that $\overline{\text{ENTRY}}$ cannot be on an active trail. So we elide $\overline{\text{ENTRY}}$ (and CDEs from it) in PDG drawings hereafter. Also notice that conditions (1)–(3) are automatically satisfied when the PDG does not have branching nodes—that is, when the PDG can be reduced to a BN. In this case, results about soundness [Verma and Pearl 1988] and (almost sure) completeness of active trails [Meek 1995] carry over.

We use examples in Figure 13 to further explain and justify Definition 5.1:

- *An active trail can contain CDEs.* In program (a), a and b are correlated, as is visualized by the top-left plot in (d). The correlation agrees with the presence of active trails between a and b in the PDG. All the active trails contain a CDE (e.g., $\boxed{a} \rightarrow \boxed{a < 4} \xrightarrow{F} \boxed{\text{ret}(c2)} \rightarrow \boxed{b}$).
- *Control ancestors of end nodes are always considered given.* That is, they are in M per condition (1). Independence between a and $c1$ in (a) is visualized by the top-right plot in (d). In the PDG, because $c1$'s dependences are in question, the value of its control parent $\boxed{a < 4}$ is considered known. Thus, given $\boxed{a < 4}$, the trail $\boxed{a} \rightarrow \boxed{a < 4} \xrightarrow{T} \boxed{c1}$ is inactive in the BN sense, and so is the trail $\boxed{a} \rightarrow \boxed{a < 4} \xrightarrow{F} \boxed{c2} \rightarrow \boxed{\text{ret}(c2)} \rightarrow \boxed{b} \leftarrow \boxed{\text{ret}(c1)} \leftarrow \boxed{c1}$.
- *An active trail cannot contain nodes that control-contradict any end node.* Program (b) differs from (a) in the line $c2 = \text{sample}(\text{NORMAL}(a;1))$. Hence, its PDG has an extra DDE $\boxed{a} \rightarrow \boxed{c2}$. But still, a and $c1$ are independent, as visualized by the bottom-left plot in (d). The trail $\boxed{a} \rightarrow \boxed{c2} \rightarrow \boxed{\text{ret}(c2)} \rightarrow \boxed{b} \leftarrow \boxed{\text{ret}(c1)} \leftarrow \boxed{c1}$ is inactive per condition (3): $\boxed{c2}$ and $\boxed{\text{ret}(c2)}$ in the trail are on the false branch, whereas end node $\boxed{c1}$ is on the true branch.
- *An active trail can contain intermediary nodes that control-contradict each other.* In program (c), b and c are correlated, as visualized by the bottom-right plot in (d). The correlation agrees with an active trail in the PDG: $\boxed{b} \rightarrow \boxed{\text{observe}(\dots)} \xleftarrow{T} \boxed{a > 0} \xrightarrow{F} \boxed{\text{observe}(\dots)} \leftarrow \boxed{c}$. Although the trail contains nodes on mutually exclusive branches, it is still active per Definition 5.1.

Active trails are compositional across call boundaries. Definition 5.1 does not unroll recursive functions. A call can create correlations internal to the callee; they are addressed by the callee’s PDG. A call can also create correlations across call boundaries. We can show that when a call is unrolled, a trail across the call boundary is active if and only if (1) the caller part of the trail is active and (2) the callee part of the trail is active. Below we illustrate this compositionality using PDGs in Figure 11a:

- `obs` in the main command is correlated with `c` in the callee `pcfg`. The correlation is indicated by two active trails: $\boxed{\text{obs}} \rightarrow \boxed{\text{observe(...)}} \leftarrow \boxed{\text{a}}$ in the caller, and $\boxed{\text{ret(b)}} \leftarrow \boxed{\text{b}} \leftarrow \boxed{\text{ret(Const(c))}} \leftarrow \boxed{\text{c}}$ in the callee, with the call’s return value flowing from $\boxed{\text{ret(b)}}$ to $\boxed{\text{a}}$.
- Right subtree `d2` is correlated with the `c` sampled by the left subtree `d1 = call(pcfg)`, as indicated by two active trails: $\boxed{\text{d2}} \rightarrow \boxed{\text{ret(Add(d1,d2))}} \leftarrow \boxed{\text{d1}}$ in the caller, and $\boxed{\text{ret(b)}} \leftarrow \boxed{\text{b}} \leftarrow \boxed{\text{ret(Const(c))}} \leftarrow \boxed{\text{c}}$ in the callee, with the recursive call’s return value flowing from $\boxed{\text{ret(b)}}$ to $\boxed{\text{d1}}$.

6 GUIDE GENERATION AS TYPE-DIRECTED, DEPENDENCE-AWARE TRANSLATION

We formalize guide generation as a translation of model programs. The translation makes sure that generated guides have the same trace type as their models, thereby guaranteeing absolute continuity. The translation can reorder computations under the same control parent and reconfigure data dependences using the notion of active trails, thereby offering faithfulness and parsimony.

Figure 14 presents the translation rules. The translation is by structural induction on *typing derivations* of commands, terms, and function definitions.

Translating function definitions. Compositionality of active trails (Section 5.2) implies that guide generation is intra-procedural and thus modular. A key insight is that guide functions can use an extra *hidden-state* parameter `h` as a proxy for transmitting correlations across call boundaries. Rule `GG:DEF` defines the translation of a function, informed by its PDG \mathcal{G} . The translated function has an extra parameter `h`.

At a call site, the guide generator gathers into a hidden state the historical information correlated with the call’s return, and passes the hidden state to the callee (`GG:TM:CALL`). In turn, inside the callee,² if a sample or call term has an active trail to parameter `h` in the PDG—indicating correlation with history through the return value—then the guide generator makes that term depend on `h` (`GG:TM:SAMPLE` and `GG:TM:CALL`). We expand on this translation below.

Translating terms. Translation of a term t has form $\mathcal{G}; C; \Delta'; \mathbf{h} : \mathbb{R}; \mathbf{a} \vdash_t \langle \Delta; \Gamma \vdash_t t : \tau \# S \rangle \rightsquigarrow t'$, where \mathcal{G} is the PDG of the current function \mathcal{F} , C is the set of control ancestors, Δ' is the term variables in scope after reordering (done by `GG:CMD`), `h` is the hidden-state parameter added for \mathcal{F} , and `a` is the term variable to which t and t' are bound. `GG:TM:BRANCH` and `GG:TM:CMD` translate branching terms and nested commands, by recursively invoking command translation. `GG:TM:SAMPLE` and `GG:TM:CALL` translate sample and call terms, and they must rewire data dependences.

Translating sample terms by reconfiguring dependences. `GG:TM:SAMPLE` translates `sample(e)`. It first chooses a learnable distribution D_{ϕ_a} having the same type as e . For instance, `a = sample(UNIF)` in line 6 of Figure 5a is translated using `BETA(NNa(·))` as the learnable distribution over $\mathbb{R}_{[0,1]}$ (lines 5–6 of Figure 5b). It is also possible to choose a universal density estimator using recent advances in normalizing flows [Huang et al. 2018].

`GG:TM:SAMPLE` then passes relevant data dependences $\alpha_1, \dots, \alpha_k$ as input to the learnable distribution via $D_{\phi_a}(\alpha_1, \dots, \alpha_k)$. Data dependences $\alpha_i \in \{\mathbf{b}_1, \dots, \mathbf{b}_n, \mathbf{h}\}$ are chosen from variables lexically

²Recall that in Figure 12, the PDG construction adds a synthetic node $\boxed{\text{h}}$ and connects it to the function’s return via a collider $\boxed{\text{h}} \rightarrow \boxed{\phantom{\text{h}}} \leftarrow \boxed{\text{ret(...)}}$. See, for instance, collider $\boxed{\text{h}} \rightarrow \boxed{\phantom{\text{h}}} \leftarrow \boxed{\text{ret(b)}}$ in Figure 11a.

$$\begin{array}{c}
\text{(REORDER)} \\
\frac{(l_i)_{i=1}^N \text{ is a permutation of } (i)_{i=1}^N \quad (\exists N \in \text{Nodes}_t(t_k). \boxed{a_l} \rightarrow N \in \mathcal{G}.\text{edges}) \Rightarrow j < k}{(a_{l_i} = t_{l_i})_{i=1}^N \text{ reorders}_{\mathcal{G}} (a_i = t_i)_{i=1}^N} \\
\boxed{\mathcal{G}; C; \Delta'; \mathbf{h} : \mathbb{R} \vdash_m \langle \Delta; \Gamma \vdash_m m : \tau \# \Sigma \rangle \rightsquigarrow m'} \\
\text{(GG:CMD)} \\
\frac{\text{TermBindings}(m) = (a_i = t_i)_{i=1}^N \quad (a_{l_i} = t_{l_i})_{i=1}^N \text{ reorders}_{\mathcal{G}} (a_i = t_i)_{i=1}^N}{\forall i. \mathcal{G}; C; \Delta', \mathbf{a}_{l_i} : \tau_{l_i}, \dots, \mathbf{a}_{l_{i-1}} : \tau_{l_{i-1}}; \mathbf{h} : \mathbb{R}; \mathbf{a}_{l_i} \vdash_t \langle \Delta_{l_i}; \Gamma_{l_i} \vdash_t t_{l_i} : \tau_{l_i} \# S_{l_i} \rangle \rightsquigarrow t'_{l_i}} \\
\mathcal{G}; C; \Delta'; \mathbf{h} : \mathbb{R} \vdash_m \langle \Delta; \Gamma \vdash_m m : \tau \# \{\mathbf{a}_1 : S_1, \dots, \mathbf{a}_N : S_N\}_e \rangle \rightsquigarrow \mathbf{a}_{l_1} = t'_{l_1}; \dots; \mathbf{a}_{l_N} = t'_{l_N}; \text{ret}(e) \\
\boxed{\mathcal{G}; C; \Delta'; \mathbf{h} : \mathbb{R}; \mathbf{a} \vdash_t \langle \Delta; \Gamma \vdash_t t : \tau \# S \rangle \rightsquigarrow t'} \\
\text{(GG:TM:SAMPLE)} \\
\frac{\mathbf{b}_1 : \tau_1, \dots, \mathbf{b}_n : \tau_n; \mathbf{h} : \mathbb{R} \vdash_e D_{\phi_a}(\alpha_1, \dots, \alpha_k) : \text{dist}(\tau) \quad \mathbf{M} \stackrel{\text{def}}{=} C \cup \{\boxed{\mathbf{b}_1}, \dots, \boxed{\mathbf{b}_n}, \boxed{\mathbf{h}}\}}{\{\alpha_1, \dots, \alpha_k\} = \{\alpha \in \{\mathbf{b}_1, \dots, \mathbf{b}_n, \mathbf{h}\} \mid \text{there is an active trail in } \mathcal{G} \text{ between } \boxed{\mathbf{a}} \text{ and } \boxed{\alpha} \text{ given } \mathbf{M} \setminus \{\boxed{\alpha}\}\}} \\
\mathcal{G}; C; \mathbf{b}_1 : \tau_1, \dots, \mathbf{b}_n : \tau_n; \mathbf{h} : \mathbb{R}; \mathbf{a} \vdash_t \langle \Delta; \Gamma \vdash_t \text{sample}(e) : \tau \# \text{atom } \tau \rangle \rightsquigarrow \text{sample}(D_{\phi_a}(\alpha_1, \dots, \alpha_k)) \\
\text{(GG:TM:CALL)} \\
\frac{\mathbf{M} \stackrel{\text{def}}{=} C \cup \{\boxed{\mathbf{b}_1}, \dots, \boxed{\mathbf{b}_n}, \boxed{\mathbf{h}}, \boxed{e}\}}{\{\alpha_1, \dots, \alpha_k\} = \{\alpha \in \{\mathbf{b}_1, \dots, \mathbf{b}_n, \mathbf{h}\} \mid \text{there is an active trail in } \mathcal{G} \text{ between } \boxed{\mathbf{a}} \text{ and } \boxed{\alpha} \text{ given } \mathbf{M} \setminus \{\boxed{\alpha}\}\}} \\
\mathcal{G}; C; \mathbf{b}_1 : \tau_1, \dots, \mathbf{b}_n : \tau_n; \mathbf{h} : \mathbb{R}; \mathbf{a} \vdash_t \langle \Delta; \Gamma \vdash_t \text{call}(\mathbf{f}; e) : \tau \# \mathbf{F}(e) \rangle \rightsquigarrow \text{call}(\mathbf{f}; e; \text{op}_{\phi_a}(\alpha_1, \dots, \alpha_k)) \\
\text{(GG:TM:BRANCH)} \\
\frac{\forall i. \mathcal{G}; C \cup \{\boxed{e}\}; \Delta'; \mathbf{h} : \mathbb{R} \vdash_m \langle \Delta; \Gamma \vdash_m m_i : \tau \# \Sigma_i \rangle \rightsquigarrow m'_i}{\mathcal{G}; C; \Delta'; \mathbf{h} : \mathbb{R}; \mathbf{a} \vdash_t \langle \Delta; \Gamma \vdash_t \text{ite}(e; m_1; m_2) : \tau \# \Sigma_1 +_e \Sigma_2 \rangle \rightsquigarrow \text{ite}(e; m'_1; m'_2)} \\
\text{(GG:TM:CMD)} \\
\frac{\mathcal{G}; C; \Delta'; \mathbf{h} : \mathbb{R} \vdash_m \langle \Delta; \Gamma \vdash_m m : \tau \# \Sigma \rangle \rightsquigarrow m'}{\mathcal{G}; C; \Delta'; \mathbf{h} : \mathbb{R}; \mathbf{a} \vdash_t \langle \Delta; \Gamma \vdash_t m : \tau \# \Sigma \rangle \rightsquigarrow m'} \\
\boxed{\mathcal{G}; \mathbf{h} : \mathbb{R} \vdash_{\mathcal{F}} \langle \vdash_{\mathcal{F}} \mathcal{F} : \langle \tau_1 \rangle \rightsquigarrow \tau_2 \# \mathbf{F} \rangle \rightsquigarrow \mathcal{F}'} \\
\text{(GG:DEF)} \\
\frac{\mathbf{f} : \langle \tau_1 \rangle \rightsquigarrow \tau_2 \# \mathbf{F} \quad \text{typedef } \mathbf{F} = \forall \mathbf{a} : \tau_1. \Sigma}{\mathcal{G}; \{\overline{\text{ENTRY}}\}; \mathbf{a} : \tau_1; \mathbf{h} : \mathbb{R} \vdash_m \langle \mathbf{a} : \tau_1; \bullet \vdash_m m : \tau_2 \# \Sigma \rangle \rightsquigarrow m'} \\
\mathcal{G}; \mathbf{h} : \mathbb{R} \vdash_{\mathcal{F}} \langle \vdash_{\mathcal{F}} \text{def } \mathbf{f}(\mathbf{a}) = m : \langle \tau_1 \rangle \rightsquigarrow \tau_2 \# \mathbf{F} \rangle \rightsquigarrow \text{def } \mathbf{f}(\mathbf{a})(\mathbf{h}) = m'
\end{array}$$

Figure 14. Guide generation as translation of typing derivations

in scope, including the hidden-state parameter \mathbf{h} . The translation uses existence of PDG active trails as an indicator of correlation. That is, α_i is considered a dependency of $\mathbf{a} = \text{sample}(D_{\phi_a}(\dots))$ if an active trail exists in \mathcal{G} between $\boxed{\alpha_i}$ and $\boxed{\mathbf{a}}$ given all other variables in scope (as well as all control ancestors).

For instance, consider translating $\mathbf{c} = \text{sample}(\text{NORMAL}(0; 1))$ in line 8 of Figure 5a. In the PDG (Figure 11a), trail $\boxed{\mathbf{c}} \rightarrow \boxed{\text{ret}(\text{Const}(\mathbf{c}))} \rightarrow \boxed{\mathbf{b}} \rightarrow \boxed{\text{ret}(\mathbf{b})} \rightarrow \boxed{\phantom{\mathbf{a}}} \leftarrow \boxed{\mathbf{h}}$ is active given $\{\overline{\text{ENTRY}}, \boxed{\mathbf{a} < .5}, \boxed{\mathbf{a}}\}$. So \mathbf{h} is input to the learnable distribution $\text{NORMAL}(\text{NN}_c(\cdot))$ for \mathbf{c} in lines 8–9 of Figure 5b:

$$\mathcal{G}; \{\overline{\text{ENTRY}}, \boxed{\mathbf{a} < .5}\}; \mathbf{a} : \mathbb{R}_{[0,1]}; \mathbf{h} : \mathbb{R}; \mathbf{c} \vdash_t \langle \mathbf{a} : \mathbb{R}_{[0,1]}; \bullet \vdash_t \text{sample}(\text{NORMAL}(0; 1)) : \mathbb{R} \# \text{atom } \mathbb{R} \rangle \rightsquigarrow \text{sample}(\text{NORMAL}(\text{NN}_c(\mathbf{h})))$$

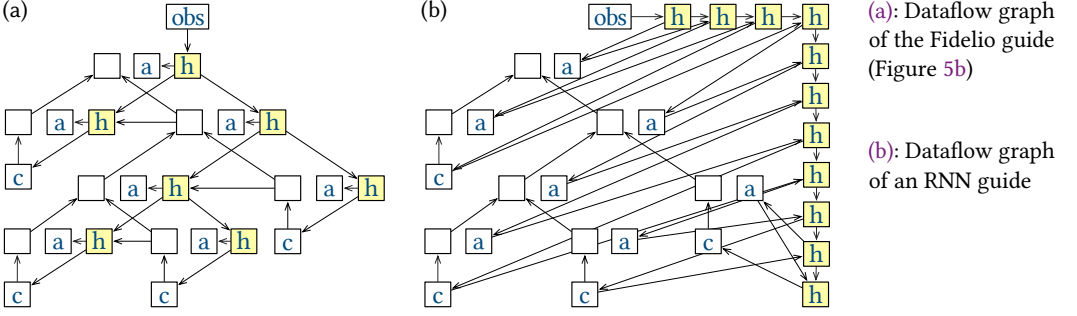


Figure 15. Dataflow in two guides, for the same trace of the PCFG model (cf. Figure 10).

Translating call terms by evolving hidden states. `GG:TM:CALL` translates function calls. As in `GG:TM:SAMPLE`, dependencies $\alpha_1, \dots, \alpha_k$ are those in-scope variables to which active trails exist in the PDG. They are input to a learnable function $\text{op}_{\phi_a}(\cdot)$, whose output is then passed to the callee as its hidden-state parameter.

For instance, consider translating `d1 = call(pcfg)` in line 11 of Figure 5a. In the PDG,

- $\boxed{d1} \rightarrow \boxed{\text{ret(Add(d1,d2))}} \leftarrow \boxed{d2}$ is an active trail given $\{\text{ENTRY}, \boxed{a < .5}, \boxed{h}, \boxed{a}\}$, and
- $\boxed{d1} \rightarrow \boxed{\text{ret(Const(c))}} \rightarrow \boxed{b} \rightarrow \boxed{\text{ret(b)}} \rightarrow \square \leftarrow \boxed{h}$ is an active trail given $\{\text{ENTRY}, \boxed{a < .5}, \boxed{d2}, \boxed{a}\}$.

So both `h` and `d2` are merged into the hidden state passed to the callee, in lines 14–15 of Figure 5b:

$$\mathcal{G}: \{\text{ENTRY}, \boxed{a < .5}\}; a : \mathbb{R}_{[0,1]}, d2 : \text{Expr}; h : \mathbb{R}; d1 \vdash_t \langle a : \mathbb{R}_{[0,1]}; \bullet \vdash_t \text{call}(\text{pcfg}) : \text{Expr} \# F_{\text{pcfg}} \rangle \\ \rightsquigarrow \text{call}(\text{pcfg}; \text{NN}_{d1}(h, \text{embed}(d2)))$$

`d2` needs an embedding layer because it is a syntax tree (of type `Expr`) rather than a scalar value.

If we unfold recursion in the PCFG guide for the same trace of execution as in Figure 10, we get Figure 15a, which makes it explicit how the hidden state is evolved to include history. Initially, the hidden state contains only information about `obs`. Visiting a right subtree simply uses the parent’s hidden state. Before visiting a left subtree `d1`, the hidden state is updated to include information about the right subtree `d2` just visited (see the leftward horizontal arrows $\boxed{h} \leftarrow \square$).

While information about historically sampled `c`’s flows into the hidden state via returns of recursive calls \square , information about historically sampled `a`’s is never merged into `h`. As the beginning of Section 5 analyzes, an `a` is independent of all RVs to be sampled later in the current subtree, given the branching condition $\boxed{a < .5}$. An `a` is also independent of all RVs to be sampled after the current call returns, given the call’s return value \square .

Contrast that with Figure 15b, which visualizes dataflow for the same trace but in a guide that uses an RNN to create correlations [Le et al. 2017]. Unaware of the program dependence structure, the RNN inevitably renders all RVs correlated. As the syntax tree is traversed, the RNN’s hidden state accretes information about every RV visited thus far, causing a later RV to treat all earlier RVs as dependencies. As a result of the to-and-fro dataflow, hidden states contain unneeded information, diluting learning signals.

Translating commands and reordering terms. Rule `GG:CMD` translates a model command, which has the trace type $\{a_1 : S_1, \dots, a_N : S_N\}_e$, to a guide command of form $a_{I_1} = t'_{I_1}; \dots; a_{I_N} = t'_{I_N}; \text{ret}(e)$.

Importantly, the guide command can choose a different ordering $(a_{I_i} = t_{I_i})_{i=1}^N$ of term bindings than the original ordering $(a_i = t_i)_{i=1}^N$. Rule `REORDER` specifies the constraint that reordering must satisfy. In particular, if a_{I_j} is a free variable in a checkpoint node in term t_{I_k} , then $a_{I_j} = t_{I_j}$ must occur before $a_{I_k} = t_{I_k}$. Recall from Section 4.3 that checkpoints are part of trace types, so satisfaction of

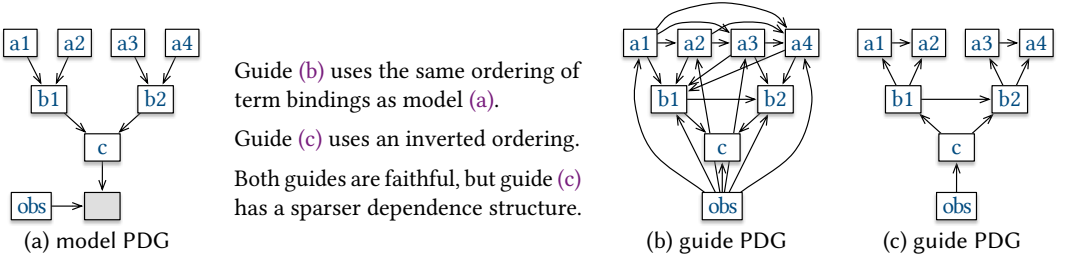


Figure 16. Ordering of term bindings has implications for the data-dependence sparsity of faithful guides.

the constraint helps ensure that the translated guide command can use the same checkpoints and thus be compatibly typed.

Each subterm of the model command is translated by recursively invoking the term translation rules. The term-variable environment for subterm t'_{i_i} is extended with $a_{i_1}, \dots, a_{i_{i-1}}$, the term variables in scope for the guide term after reordering.

Reordering of term bindings is often desirable because it can lead to fewer dependences while maintaining faithfulness. Fidelio follows the heuristic of [Stuhlmüller et al. \[2013\]](#) and [Paige and Wood \[2016\]](#) to invert topological orderings (while respecting the reordering constraint).

Consider a model that has a PDG as shown in Figure 16a. There, $a_1, a_2, a_3, a_4, b_1, b_2, c$ is a valid topological ordering of the term bindings. In (b) is the PDG of a faithful guide, using the same ordering. (Notice that removing any DDE makes it unfaithful by introducing conditional independences not found in the model.) In (c) is the PDG of another faithful guide, using an inverted topological ordering $c, b_1, b_2, a_1, a_2, a_3, a_4$. Whereas (b) is almost fully connected, guide (c) attains a sparse dependence structure using available conditional independences—e.g., given b_2, a_3 is independent of all variables in $\{\text{obs}, c, b_1, a_1, a_2\}$. As we show in Section 8, for the same total number of neural-network parameters, guide (c) leads to better performance than guide (b), both for training and for inference.

AC by construction. We prove that the translation preserves trace typing:

THEOREM 6.1 (TYPE-PRESERVING TRANSLATION).

If $\mathcal{G}; \{\overline{\text{ENTRY}}\}; \bullet; \text{obs} : \mathbb{R} \vdash_m \langle \bullet; \text{obs} : \mathbb{R} \vdash_m m_m : \tau \# \Sigma \rangle \rightsquigarrow m_g$, then $\bullet; \text{obs} : \mathbb{R} \vdash_m m_g : \tau \# \Sigma$.

It follows from Theorems 4.5 and 6.1 that for a well-typed model, the translation is guaranteed to generate a guide that is mutually absolutely continuous with the model:

THEOREM 6.2 (ABSOLUTE CONTINUITY BY CONSTRUCTION).

If $\mathcal{G}; \{\overline{\text{ENTRY}}\}; \bullet; \text{obs} : \mathbb{R} \vdash_m \langle \bullet; \text{obs} : \mathbb{R} \vdash_m m_m : \tau \# \Sigma \rangle \rightsquigarrow m_g$ and $v_{\text{obs}} : \mathbb{R}$, then for any measurable set A of traces, $\llbracket m_m \{v_{\text{obs}}/\text{obs}\} \rrbracket (A) \neq 0$ if and only if $\llbracket m_g \{v_{\text{obs}}/\text{obs}\} \rrbracket (A) \neq 0$.

7 IMPLEMENTATION

Fidelio uses Pyro [[Bingham et al. 2019](#)] as the underlying inference engine. A Fidelio model is compiled to a model and a guide in Pyro with explicit name mangling. The user can specify as arguments hyperparameters including the number of layers in a network, the number of neurons in each layer, and the activation function used. It is also easy to modify neural networks in generated guides, as we do in setting up our experiments (Section 8.2).

A few surface-syntax features help streamline guide generation. (1) Fidelio supports directives using which the programmer can hint shapes of tensors. (2) Fidelio supports directives for specifying embedding functions for data that warrant special embedding layers (e.g., convolutional networks

Table 1. ● model and guide have compatible types. ○ model and guide fail to be compatibly typed. Last seven programs appeared in the expressiveness evaluation conducted by Wang et al. [2021].

| Program | Description | Lew et al. [2019] | Wang et al. [2021] | Fidelio |
|-----------|------------------------------------------|-------------------|--------------------|---------|
| collider | Figure 1 (model & guide2) | ● | ○ | ● |
| pcfg | Figure 5, (a) & (b) | ○ | ○ | ● |
| treebn | Figure 16, (a) & (c) | ● | ○ | ● |
| curvefit | Curve fitting, Lew et al. [2019, Fig. 4] | ● | ○ | ● |
| kalman | Kalman Smoother | ● | ● | ● |
| branching | Random Control Flow | ○ | ● | ● |
| marsaglia | Marsaglia Algorithm | ○ | ● | ● |
| aircraft | Aircraft Detection | ● | ● | ● |
| gmm | Gaussian Mixture Model | ● | ● | ● |
| sprinkler | Bayesian Network | ● | ● | ● |
| dp | Dirichlet Process | ○ | ○ | ○ |

as image embeddings). (3) Fidelio provides an option that generalizes MADE [Germain et al. 2015] to allow weight sharing among local networks.

8 EVALUATION

8.1 Expressiveness of the Trace-Type System

Table 1 evaluates the expressiveness of Fidelio’s type system, comparing it with state-of-the-art type systems for AC [Lew et al. 2019; Wang et al. 2021]. The benchmark programs include examples in this paper and also those taken from prior work [Lew et al. 2019; Wang et al. 2021], Anglican [Wood et al. 2014], and Pyro, with reasonable modifications so that they conform to our syntax.

The type system of Lew et al. [2019] cannot type-check *models* of pcfg, branching, and marsaglia, because they use branching and recursion in their general forms. The type system of Wang et al. [2021] cannot type-check *guides* of collider, pcfg, treebn, and curvefit, because they reorder computations. dp uses stochastic memoization, a PPL feature not yet supported by these systems.

8.2 Performance Implication of Dependence-Awareness for Training and Inference

We assess the implications of adopting dependence-aware guide generation for both ahead-of-time training and run-time inference. Table 2 summarizes our benchmarks, which include a variety of applications, program sizes, dependence structures, and training setups.

We compare Fidelio mainly with two methods for guide generation in a universal PPL: mean-field guides, which make strong independence assertions, and RNN guides, which make all variables correlated. Generated Fidelio guides were modified so that they have the same number of trainable parameters (i.e., capacity) as the other two guides. Networks in the Fidelio guides and mean-field guides typically have no more than four linear layers and use ReLU activation. All guides for a benchmark program are trained for the same number of iterations. An appendix in the technical report documents experimental setups and additional statistics. The RNN we use in all experiments is LSTM [Hochreiter and Schmidhuber 1997], which deals with the vanishing gradient problem and is thus better at remembering correlation than vanilla RNNs.

Table 2 shows validation loss as a metric for evaluating training performance. Fidelio consistently leads to lower validation loss as computed by (2.3)—trained Fidelio guides are closer approximations to true posteriors than mean-field or LSTM guides, averaged on the validation sample set.

Validation losses alone do not tell the whole story, however. Below we explain our benchmarks and findings in more detail.

Table 2. Benchmarks for evaluating different approaches to guide generation. Column “#RVs”: expected number of RVs sampled in a trace. Validation loss is computed per (2.3) on a validation sample set.

| Program | Structures | #RVs | LoC (model) | Capacity | Validation Loss | | |
|------------------------|------------|-------|----------------|----------|-----------------|--------|---------|
| | | | | | Mean-Field | LSTM | Fidelio |
| treebn | BN | 31 | 37 | 2K | 51.88 | 45.76 | 41.07 |
| | | | | 5K | 51.88 | 43.96 | 41.04 |
| ecoli70 | BN | 46 | 54 | 10K | 11.89 | 15.28 | 16.56 |
| | | | | 15K | 11.94 | 15.31 | 16.60 |
| longrange ($k = 5$) | loop | 8 | 16 | 1.4K | 25.39 | 24.49 | 21.88 |
| longrange ($k = 10$) | loop | 13 | 16 | 1.4K | 44.05 | 43.69 | 40.38 |
| longrange ($k = 20$) | loop | 23 | 16 | 1.4K | 81.23 | 81.32 | 77.59 |
| longrange ($k = 40$) | loop | 43 | 16 | 1.4K | 155.8 | 156.0 | 152.1 |
| captcha | recursion | 6 | 43 | 4.4M | 1.644 | -1.325 | -1.212 |
| | | | | 100K | 6.226 | 2.352 | 1.980 |
| astropcfg | recursion | 15.0 | 159 | 175K | 6.217 | 2.362 | 2.035 |
| | | | | 250K | 6.216 | 2.387 | 2.114 |
| | | | | 1.1M | 7.962 | 14.84 | 3.603 |
| mathcaptcha | recursion | 14.5 | 237 | 1.25M | 8.150 | 14.78 | 2.242 |
| | | | | 1.5M | 8.329 | 14.79 | 1.950 |
| | | | | 3M | 8.380 | 9.531 | 2.511 |
| | | | | 20K | 111.1 | 257.8 | 16.22 |
| gmmcc | recursion | 217.4 | 49 | 50K | 111.2 | 170.1 | 15.16 |
| | | | | 80K | 111.1 | 152.4 | 14.55 |

Long-range correlation. Benchmark longrange [Harvey et al. 2019] tests the ability for guides to capture correlation between variables sampled far apart. The model samples two Gaussian variables a and b —and also k noise variables in between—before observing the sum of all $k + 2$ variables.

We assess how well the inferred posteriors capture the strong correlation between a and b sampled k variables apart, with $k \in \{5, 10, 20, 40\}$. We draw 2,000 samples after training each guide for 20,000 steps and plot them in Figure 17. In columns one and three, samples are drawn from the trained guides. In columns two and four, samples are drawn using IS, with the guides as proposals.

The LSTM guide manages to capture the correlation to some degree when $k = 10$. However, the LSTM guide fails to remember correlation when $k = 20$. Quality of the learned proposal distributions has a knock-on effect on IS. When $k = 20$, the LSTM guide results in visibly lower-quality IS samples than the Fidelio guide of equal capacity. We also compute the effective sample size (ESS) for each guide, as a quantitative (though not comprehensive) metric of sample quality [Kong 1992]; Fidelio guides lead to significantly higher effective sample sizes.

Computation reordering. Benchmark treebn assesses the impact of computation reordering on training and inference performance. The model is linear Gaussian BN, similar to Figure 16a in structure. In addition to the mean-field and LSTM guides, we compare two Fidelio guides that either retain or invert the model’s topological ordering of variables (cf. Figures 16b and 16c).

Figure 18 plots the training loss (2.3) as training progresses. All four guides have the same capacity (5K). The sparser Fidelio guide converges to the best solution. The table in Figure 18 shows the Fidelio guide with an inverted topological ordering results in significantly higher ESS than all the other guides: a sparser yet faithful dependence structure makes inference more effective.

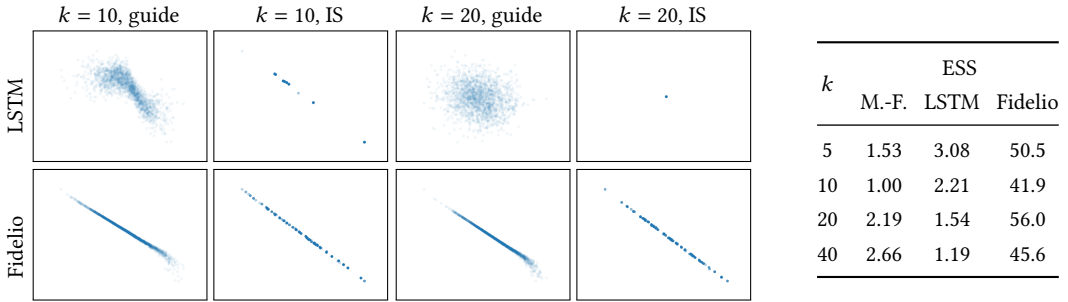


Figure 17. Drawing samples from LSTM and Fidelio guides trained for the **longrange** benchmark. Plots visualize correlation between two variables that are sampled k variables apart in the model.

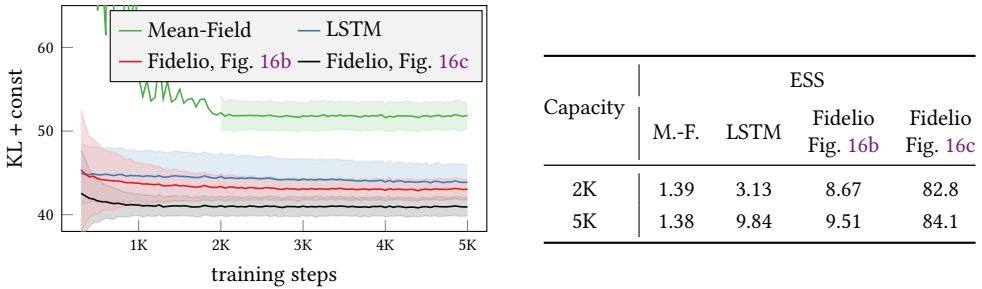


Figure 18. Experimental results for **treebn**. Left: Training steps vs. training loss profile, with all guides having capacity 5K. Right: Quality of IS samples as measured by ESS.

The sparser guide is rejected by a prior type system [Wang et al. 2021], which disallows reordering computations. Trace-type systems have performance implications.

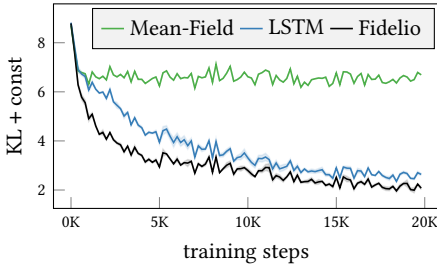
ECOLI70: a large Bayesian network. We use benchmark *ecoli70* [Schäfer and Strimmer 2005] to evaluate the implication of dependence-aware guide generation for large BN models. The program is a linear Gaussian BN with 46 variables and 70 arcs. Results can be found in an appendix; they show (1) that the mean-field guide and the LSTM guide did equally poorly on *ecoli70* and (2) that Fidelio significantly improves on these dependence-agnostic approaches.

Captcha: a regular grammar. In benchmark *captcha* [Mansinghka et al. 2013], the model generates a captcha image by sampling a probabilistic regular grammar, and the inference problem is to recognize captcha text given an image. Guide programs use convolutional networks as image embeddings (Section 7). Experimental results can be found in an appendix.

While the mean-field guide did poorly on *captcha*, the LSTM guide and the Fidelio guide did almost equally well, in terms of convergence and recognition rates. A possible explanation is given by the structure of the model. Because *captcha* samples from a regular—rather than context-free—language, the dataflow exhibits a linear-recursive pattern (shown in an appendix). As a result, the model has almost no conditional independence for Fidelio to exploit, and the unfolded dataflow graph for the Fidelio guide is almost identical to that induced by an RNN.

Astronomer: a context-free grammar. In benchmark *astropcfg* [Manning and Schütze 1999], the model samples sentences such as “astronomers saw stars with telescopes”. We consider the inference task of regenerating observed sentences. The program consists of mutually recursive functions corresponding to nonterminal symbols in the PCFG.

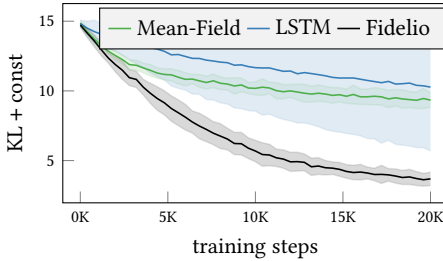
Figure 19 shows the training-loss profile for all guides of the same capacity (175K). It also shows inference accuracy, as measured by sentence regeneration rates and Levenshtein edit distances for



Accuracy of sentence regeneration

| Capacity | Regeneration Rate | | | Levenshtein Distance | | |
|----------|-------------------|-------|--------------|----------------------|------|-------------|
| | M.-F. | LSTM | Fidelio | M.-F. | LSTM | Fidelio |
| 100K | 40.0% | 57.4% | 71.7% | 4.36 | 2.99 | 2.72 |
| 175K | 39.8% | 60.8% | 71.3% | 4.41 | 3.08 | 2.63 |
| 250K | 39.2% | 62.4% | 69.8% | 4.36 | 3.04 | 2.75 |

Figure 19. Experimental results for **astropcfg**. Left: Training steps vs. training loss profile, with all guides having capacity 175K. Right: Inference accuracy, at three capacities.



Accuracy of captcha recognition

| Capacity | Recognition Rate | | | Levenshtein Distance | | |
|----------|------------------|-------|--------------|----------------------|------|--------------|
| | M.-F. | LSTM | Fidelio | M.-F. | LSTM | Fidelio |
| 100K+CNN | 35.8% | 8.94% | 70.5% | 2.50 | 4.25 | 0.809 |
| 250K+CNN | 36.8% | 9.82% | 79.5% | 2.45 | 4.21 | 0.579 |
| 500K+CNN | 35.3% | 9.67% | 83.1% | 2.59 | 4.22 | 0.463 |
| 2M+CNN | 34.9% | 39.4% | 77.6% | 2.62 | 2.66 | 0.650 |

Figure 20. Experimental results for **mathcaptcha**. Left: Training steps vs. training loss profile, with all guides having capacity 3M. Right: Inference accuracy, at four capacities. All guides use a CNN of capacity 1M.

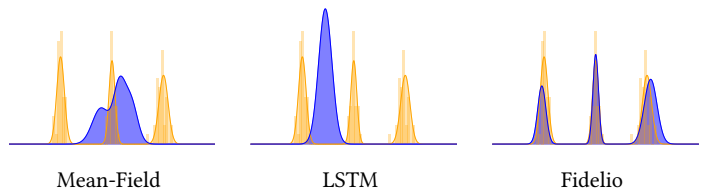
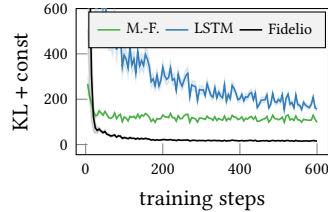


Figure 21. Experimental results for **gmmcc**. Left: training steps vs. training-loss profile. Right: ground-truth clusters (yellow) and clusters inferred using trained guides (blue). All guides have the same capacity.

all guides and for three capacities. Fidelio demonstrates a clear advantage over the LSTM guide. In particular, it increases regeneration rate by 10% on average. The results suggest that it pays off to exploit the tree-recursive dataflow pattern and the resulting conditional independences in the **astropcfg** model (Figures 10 and 15).

Math captcha. We design a second captcha benchmark, **mathcaptcha**, that samples simple arithmetic expressions and renders them as captcha images (e.g., $9+2*6+1$). Unlike captcha, **mathcaptcha** samples from a context-free language, so we expect Fidelio to improve convergence and captcha recognition rate over the LSTM guide.

Figure 20 confirms our belief: Fidelio outperforms mean-field and LSTM guides, by a large margin. The LSTM guide did not start to pick up until network capacity was raised to 3M (including an invariant 1M for a CNN embedding). In contrast, with limited network capacity and training budget, Fidelio can do better. Conditional independences inherent in PCFG-like models make them favored applications for Fidelio.

Gaussian mixture model for clustering and classification. In benchmark **gmmcc** [Le et al. 2017], the model samples a stochastic number of Gaussians and further samples data points from

them. Given an observed dataset, the inference task is to simultaneously identify clusters and classify data points into clusters. This model is challenging because it samples a large number of RVs (~200). Nonetheless, since data points are independently distributed, it offers conditional independences that Fidelio can potentially tap into.

Figure 21 plots the training-loss profile and kernel density estimations: with the same number of particles, the Fidelio guide is the only guide that correctly inferred the number of clusters.

Summary. We observe that Fidelio consistently improves over LSTM and mean-field guides, both on convergence of training and on accuracy of inference. Our experiments suggest that compared with LSTM, it is most favorable to use Fidelio for models that contain conditional independences and contain ~10 RVs or more.

For models containing ~20 RVs or more, a mean-field guide may be competitive against LSTM; see `ecoli70`, `longrange` ($k = 20, 40$), `mathcaptcha`, and `gmmcc`. Unaware of program dependence structures, LSTM has to accrete information about all RVs in its hidden state. Unlike Fidelio or mean-field guides, LSTM guides are unable to directly express any conditional independence, which leads to diluted learning signals especially when the number of RVs is large. In contrast, Fidelio guides capitalize on available conditional independences, so training can focus on learning the real correlations.

9 RELATED WORK

Language-based solutions to absolute continuity. Lew et al. [2019] and Wang et al. [2021] design type systems to check AC statically. Lew et al. [2019] formulate AC in terms of a denotational measure semantics for a trace-based PPL. Their type system allows some forms of stochastic control flow, including loops, but disallows general recursion or the kind of stochastic branching needed for models such as PCFGs. It allows guides to reorder computations in some way.

Wang et al. [2021] propose a coroutine-based PPL, where a model and a guide send and receive messages over communication channels to synchronize on sampled values, branching choices, and function calls. AC is formulated in terms of a measure semantics induced by this operational semantics. The approach is inspired by session types [Honda et al. 1999]: their *guide types* describe communication protocols of model–guide pairs. Unlike Lew et al. [2019], the type system allows branching and recursion in their general forms. However, it requires that computations happen in exactly the same (total) order as prescribed by guide types.

In both prior approaches, it is the programmer who authors guides and makes sure that guides have compatible types with models. In our approach, guides are automatically generated and are guaranteed to have compatible types with models.

Lee et al. [2019] introduce a static analysis for Pyro’s stochastic variational inference. The analysis aims to prove that a guide has the same support as the model and satisfies differentiability conditions. It deals with Pyro features including tensors and plating, but it does not support stochastic branching or general recursion.

Generating guides automatically. A recent spate of work in machine learning has focused on generating guide programs [Stuhlmüller et al. 2013; Paige and Wood 2016; Le et al. 2017; Webb et al. 2018, 2019; Baudart and Mandel 2021]. Except for Le et al. [2017], the prior approaches deal with graphical models or non-universal PPLs; stochastic branching and general recursion are not supported. Le et al. [2017] address the challenge by having a guide program talk to an LSTM network, leading to faithful but “fully connected”, non-parsimonious guides. Webb et al. [2018] create faithful, parsimonious guides, but only for BNs. Webb et al. [2019] and Weilbach et al. [2020] integrate normalizing flows [Rezende and Mohamed 2015; Huang et al. 2018] to increase the

representational power of learnable continuous distributions; it is an orthogonal extension that can be incorporated into Fidelio.

Dependence analysis for PPLs. Hur et al. [2014] study program slicing for an imperative PPL with loops. Like guide generation, slicing takes into account conditional dependences as indicated by active trails. Unlike guide generation, the slicer is concerned with conditional dependencies of a program’s return value, which exists in all finite executions, whereas a guide generator is concerned with conditional dependencies of sample and call terms, which are reached only stochastically and subject to reordering in guides. Gorinova et al. [2021] use an information-flow type system to identify conditional independences in a non-universal PPL. It is interesting to study if the information-flow perspective can be applied to a universal PPL, which could serve as a good basis for studying soundness of PDG active trails as proposed in Section 5.2.

10 CONCLUSION

We have presented Fidelio, a framework that can automatically generate guide programs for deep amortized inference in a universal PPL. It addresses challenges posed by the control-flow expressiveness of a universal PPL. Fidelio uses a novel trace-type system and generates guides via a type-guided, dependence-aware translation of models. Theoretical results show that the translation is type-preserving and thus ensures absolute continuity by construction. Experimental results show that *symbolic* dependence information in probabilistic programs can be used effectively to aid in *neural-network-based* inference.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable feedback. We thank Nada Amin, Hongfei Fu, Edward Lee, Yingzao Li, Di Wang, and Rob Zinkov for discussions and help. This work was supported by the Natural Sciences and Engineering Research Council of Canada. The views and opinions expressed are those of the authors and do not necessarily reflect the position of any funding agency.

REFERENCES

- autoguide 2022. Automatic guide generation (Pyro documentation). <https://docs.pyro.ai/en/1.8.0/infer.autoguide.html>.
- Guillaume Baudart and Louis Mandel. 2021. Automatic guide generation for Stan via NumPyro. In *Int’l Conf. on Probabilistic Programming (PROBPROG)*. arXiv:2110.11790
- Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D. Goodman. 2019. Pyro: Deep universal probabilistic programming. *Journal of Machine Learning Research (JMLR)* 20, 1 (2019). arXiv:1810.09538
- Johannes Borgström, Ugo Dal Lago, Andrew D. Gordon, and Marcin Szymczak. 2016. A lambda-calculus foundation for universal probabilistic programming. In *ACM SIGPLAN Conf. on Functional Programming (ICFP)*. <https://doi.org/10.1145/2951913.2951942>
- Marco F. Cusumano-Towner, Feras A. Saad, Alexander K. Lew, and Vikash K. Mansinghka. 2019. Gen: A general-purpose probabilistic programming system with programmable inference. In *ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI)*. <https://doi.org/10.1145/3314221.3314642>
- Jeanne Ferrante, Karl J. Ottenstein, and Joe D. Warren. 1987. The program dependence graph and its use in optimization. *ACM Tran. on Programming Languages and Systems (TOPLAS)* 9, 3 (July 1987). <https://doi.org/10.1145/24039.24041>
- Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. 2015. MADE: Masked autoencoder for distribution estimation. In *Int’l Conf. on Machine Learning (ICML)*. <http://proceedings.mlr.press/v37/germain15.pdf>
- Noah Goodman, Vikash K. Mansinghka, Daniel M Roy, Keith Bonawitz, and Joshua B. Tenenbaum. 2008. Church: A language for generative models. In *Conf. on Uncertainty in Artificial Intelligence (UAI)*. arXiv:1206.3255

- Maria I. Gorinova, Andrew D. Gordon, Charles Sutton, and Matthijs Vákár. 2021. Conditional independence by typing. *ACM Tran. on Programming Languages and Systems (TOPLAS)* 44, 1 (Dec. 2021). <https://doi.org/10.1145/3490421> arXiv:2010.11887
- William Harvey, Andreas Munk, Atılım Güneş Baydin, Alexander Bergholm, and Frank Wood. 2019. Attention for inference compilation. arXiv:1910.11961
- Geoffrey E. Hinton, Peter Dayan, Brendan J. Frey, and Radford M. Neal. 1995. The “wake-sleep” algorithm for unsupervised neural networks. *Science* 268, 5214 (1995). <https://doi.org/10.1126/science.7761831>
- Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8 (Nov. 1997). <https://doi.org/10.1162/neco.1997.9.8.1735>
- Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. 1999. Language primitives and type discipline for structured communication-based programming. In *European Symp. on Programming (ESOP)*. <https://doi.org/10.1007/BFb0053567>
- Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron Courville. 2018. Neural autoregressive flows. In *Int’l Conf. on Machine Learning (ICML)*. <http://proceedings.mlr.press/v80/huang18d/huang18d.pdf>
- Chung-Kil Hur, Aditya V. Nori, Sriram K. Rajamani, and Selva Samuel. 2014. Slicing probabilistic programs. In *ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI)*. <https://doi.org/10.1145/2594291.2594303>
- Michael I. Jordan, Zoubin Ghahramani, Tommi S. Jaakkola, and Lawrence K. Saul. 1999. An introduction to variational methods for graphical models. *Machine learning* 37, 2 (1999). <https://doi.org/10.1023/A:1007665907178>
- Diederik P. Kingma and Max Welling. 2014. Auto-encoding variational Bayes. In *Int’l Conf. on Learning Representations (ICLR)*. arXiv:1312.6114
- Augustine Kong. 1992. *A Note on Importance Sampling Using Standardized Weights*. Technical Report 348. Department of Statistics, University of Chicago. <https://d3qi0qp55mx5f5.cloudfront.net/stat/docs/tech-rpts/tr348.pdf>
- Tuan Anh Le, Atılım Güneş Baydin, and Frank Wood. 2017. Inference compilation and universal probabilistic programming. In *Int’l Conf. on Artificial Intelligence and Statistics (AISTATS)*. arXiv:1610.09900
- Tuan Anh Le, Adam R. Kosiorek, N. Siddharth, Yee Whye Teh, and Frank Wood. 2019. Revisiting reweighted wake-sleep for models with stochastic control flow. In *Conf. on Uncertainty in Artificial Intelligence (UAI)*. arXiv:1805.10469
- Wonyeol Lee, Hangyeol Yu, Xavier Rival, and Hongseok Yang. 2019. Towards verified stochastic variational inference for probabilistic programs. *Proc. of the ACM on Programming Languages (PACMPL)* 4, POPL (Dec. 2019). <https://doi.org/10.1145/3371084>
- Alexander K. Lew, Marco F. Cusumano-Towner, Benjamin Sherman, Michael Carbin, and Vikash K. Mansinghka. 2019. Trace types and denotational semantics for sound programmable inference in probabilistic languages. *Proc. of the ACM on Programming Languages (PACMPL)* 4, POPL (Dec. 2019). <https://doi.org/10.1145/3371087>
- Jianlin Li, Leni Ven, Pengyuan Shi, and Yizhou Zhang. 2023. Type-preserving, dependence-aware guide generation for sound, effective amortized probabilistic inference. *Proc. of the ACM on Programming Languages (PACMPL)* 7, POPL (2023). <https://doi.org/10.1145/3571243>
- Carol Mak, C.-H. Luke Ong, Hugo Paquet, and Dominik Wagner. 2021. Densities of almost surely terminating probabilistic programs are differentiable almost everywhere. In *European Symp. on Programming (ESOP)*. https://doi.org/10.1007/978-3-030-72019-3_16 arXiv:2004.03924
- Christopher Manning and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press.
- Vikash K. Mansinghka, Tejas D. Kulkarni, Yura N. Perov, and Joshua B. Tenenbaum. 2013. Approximate Bayesian image interpretation using generative probabilistic graphics programs. In *Conf. on Neural Information Processing Systems (NIPS)*. arXiv:1307.0060
- Vikash K. Mansinghka, Ulrich Schaechtle, Shivam Handa, Alexey Radul, Yutian Chen, and Martin Rinard. 2018. Probabilistic programming with programmable inference. In *ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI)*. <https://doi.org/10.1145/3192366.3192409>
- Christopher Meek. 1995. Strong completeness and faithfulness in Bayesian networks. In *Conf. on Uncertainty in Artificial Intelligence (UAI)*. arXiv:1302.4973
- Brooks Paige and Frank Wood. 2016. Inference networks for sequential Monte Carlo in graphical models. In *Int’l Conf. on Machine Learning (ICML)*. arXiv:1602.06701
- Judea Pearl. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann. <https://doi.org/10.1016/B978-0-08-051489-5.50001-1>

- Danilo Rezende and Shakir Mohamed. 2015. Variational inference with normalizing flows. In *Int'l Conf. on Machine Learning (ICML)*. arXiv:1505.05770
- Daniel Ritchie, Paul Horsfall, and Noah D. Goodman. 2016. Deep amortized inference for probabilistic programs. arXiv:1610.05735
- Feras A. Saad, Marco F. Cusumano-Towner, Ulrich Schaechtle, Martin C. Rinard, and Vikash K. Mansinghka. 2019. Bayesian synthesis of probabilistic programs for automatic data modeling. *Proc. of the ACM on Programming Languages (PACMPL)* 3, POPL (2019). <https://doi.org/10.1145/3290350> arXiv:1907.06249
- Juliane Schäfer and Korbinian Strimmer. 2005. A shrinkage approach to large-scale covariance matrix estimation and implications for functional genomics. *Statistical Applications in Genetics and Molecular Biology* 4 (2005). <https://doi.org/10.2202/1544-6115.1175>
- 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 *Conf. on Neural Information Processing Systems (NIPS)*. arXiv:1706.00400
- Andreas Stuhlmüller, Jessica Taylor, and Noah D. Goodman. 2013. Learning stochastic inverses. In *Conf. on Neural Information Processing Systems (NIPS)*. <https://proceedings.nips.cc/paper/2013/file/7f53f8c6c730af6aeb52e66eb74d8507-Paper.pdf>
- Marcin Szymczak and Joost-Pieter Katoen. 2019. Weakest preexpectation semantics for Bayesian inference. In *Int'l School on Engineering Trustworthy Software Systems (SETSS)*. https://doi.org/10.1007/978-3-030-55089-9_3
- Dustin Tran, Matthew D. Hoffman, Dave Moore, Christopher Suter, Srinivas Vasudevan, Alexey Radul, Matthew Johnson, and Rif A. Saurous. 2018. Simple, distributed, and accelerated probabilistic programming. In *Conf. on Neural Information Processing Systems (NeurIPS)*. arXiv:1811.02091
- Alan M. Turing. 1937. On computable numbers, with an application to the entscheidungsproblem. *Proc. of the London mathematical society* 2, 1 (1937). <https://doi.org/10.1112/plms/s2-42.1.230>
- Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. 2021. *An Introduction to Probabilistic Programming*. arXiv:1809.10756
- Thomas Verma and Judea Pearl. 1988. Causal networks: Semantics and expressiveness. In *Conf. on Uncertainty in Artificial Intelligence (UAI)*. arXiv:1304.2379
- Di Wang, Jan Hoffmann, and Thomas Reps. 2021. Sound probabilistic inference via guide types. In *ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI)*. arXiv:2104.03598
- Stefan Webb, Jonathan P. Chen, Martin Jankowiak, and Noah Goodman. 2019. Improving automated variational inference with normalizing flows. In *6th ICML Workshop on Automated Machine Learning*. https://www.automl.org/wp-content/uploads/2019/06/automlws2019_Paper23.pdf
- Stefan Webb, Adam Goliński, Robert Zinkov, N. Siddharth, Tom Rainforth, Yee Whye Teh, and Frank Wood. 2018. Faithful inversion of generative models for effective amortized inference. In *Conf. on Neural Information Processing Systems (NIPS)*. arXiv:1712.00287
- Christian Weilbach, Boyan Beronov, William Harvey, and Frank Wood. 2020. Structured conditional continuous normalizing flows for efficient amortized inference in graphical models. In *Int'l Conf. on Artificial Intelligence and Statistics (AISTATS)*. <http://proceedings.mlr.press/v108/weilbach20a/weilbach20a.pdf>
- Frank D. Wood, Jan-Willem van de Meent, and Vikash Mansinghka. 2014. A new approach to probabilistic programming inference. In *Int'l Conf. on Artificial Intelligence and Statistics (AISTATS)*. arXiv:1507.00996
- Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. 2017. Understanding deep learning requires rethinking generalization. In *Int'l Conf. on Learning Representations (ICLR)*. arXiv:1611.03530
- Cheng Zhang, Judith Bütetage, Hedvig Kjellström, and Stephan Mandt. 2019. Advances in variational inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 41, 8 (2019). <https://doi.org/10.1109/TPAMI.2018.2889774> arXiv:1711.05597

A COMPLETE SEMANTICS OF FCC

In this appendix, we present the complete semantics of FCC and proofs of the main theorems.

A.1 Operational and Static Semantics

See Figures 22–26.

| | | | | |
|-------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------|
| $e \Downarrow v$ | | | | |
| (EE:TRIV) $\frac{}{\text{unit} \Downarrow \text{unit}}$ | (EE:TRUE) $\frac{}{\text{true} \Downarrow \text{true}}$ | (EE:FALSE) $\frac{}{\text{false} \Downarrow \text{false}}$ | (EE:COND:TRUE) $\frac{e \Downarrow \text{true} \quad e_1 \Downarrow v}{\text{if}(e; e_1; e_2) \Downarrow v}$ | (EE:COND:FALSE) $\frac{e \Downarrow \text{false} \quad e_2 \Downarrow v}{\text{if}(e; e_1; e_2) \Downarrow v}$ |
| (EE:REAL) $\frac{}{r \Downarrow r}$ | (EE:NAT) $\frac{}{n \Downarrow n}$ | (EE:OP) $\frac{e_1 \Downarrow v_1 \quad e_2 \Downarrow v_2 \quad v = v_1 \diamond v_2}{\text{op}_{\diamond}(e_1; e_2) \Downarrow v}$ | (EE:ABS) $\frac{}{\lambda x. e \Downarrow \lambda x. e}$ | |
| (EE:APP) $\frac{e_1 \Downarrow \lambda x. e_0 \quad e_2 \Downarrow v_2 \quad e_0 \{v_2/x\} \Downarrow v}{\text{app}(e_1; e_2) \Downarrow v}$ | | (EE:LET) $\frac{e_1 \Downarrow v_1 \quad e_2 \{v_1/x\} \Downarrow v_2}{\text{let}(e_1; x.e_2) \Downarrow v_2}$ | (EE:BER) $\frac{e \Downarrow v}{\text{BERN}(e) \Downarrow \text{BERN}(v)}$ | |
| (EE:UNIF) $\frac{}{\text{UNIF} \Downarrow \text{UNIF}}$ | (EE:BETA) $\frac{e_1 \Downarrow v_1 \quad e_2 \Downarrow v_2}{\text{BETA}(e_1; e_2) \Downarrow \text{BETA}(v_1; v_2)}$ | (EE:GAMMA) $\frac{e_1 \Downarrow v_1 \quad e_2 \Downarrow v_2}{\text{GAMMA}(e_1; e_2) \Downarrow \text{GAMMA}(v_1; v_2)}$ | | |
| (EE:NORMAL) $\frac{e_1 \Downarrow v_1 \quad e_2 \Downarrow v_2}{\text{NORMAL}(e_1; e_2) \Downarrow \text{NORMAL}(v_1; v_2)}$ | (EE:CAT) $\frac{\forall i \in \{1, \dots, n\}. e_i \Downarrow v_i}{\text{CAT}(e_1, \dots, e_n) \Downarrow \text{CAT}(v_1, \dots, v_n)}$ | | (EE:GEO) $\frac{e \Downarrow v}{\text{GEO}(e) \Downarrow \text{GEO}(v)}$ | |
| (EE:POIS) $\frac{e \Downarrow v}{\text{POIS}(e) \Downarrow \text{POIS}(v)}$ | | | | |

Figure 22. Evaluation rules for expressions.

$$\begin{array}{c}
\boxed{s \vdash t \Downarrow^w v} \\
\\
\text{(E:CMD)} \quad \frac{\sigma \vdash_m m \Downarrow^w v}{\text{trace } \sigma \vdash_t m \Downarrow^w v} \qquad \text{(E:SAMPLE)} \quad \frac{e \Downarrow d \quad v \in d.\text{support} \quad w = d.\text{density}(v)}{\text{atom } v \vdash_t \text{sample}(e) \Downarrow^w v} \qquad \text{(E:BRANCH)} \quad \frac{e \Downarrow b \quad \sigma \vdash_m m_b \Downarrow^w v}{\text{inj } \sigma \vdash_t \text{ite}(e; m_{\text{true}}; m_{\text{false}}) \Downarrow^w v} \\
\\
\text{(E:CALL)} \quad \frac{\text{def } f(\mathbf{a})(\mathbf{x}) = m \quad e_1 \Downarrow v_a \quad e_2 \Downarrow v_x \quad \sigma \vdash_t m \{v_a/\mathbf{a}\} \{v_x/\mathbf{x}\} \Downarrow^w v}{\text{fold } \sigma \vdash_t \text{call}(f; e_1; e_2) \Downarrow^w v} \\
\\
\boxed{\sigma \vdash_m m \Downarrow^w v} \\
\\
\text{(E:RET)} \quad \frac{e \Downarrow v}{\{\} \vdash_m \text{ret}(e) \Downarrow^1 v} \qquad \text{(E:OBSERVE)} \quad \frac{e_1 \Downarrow d \quad e_2 \Downarrow v_2 \quad v \in d.\text{support} \quad w_1 = d.\text{density}(v_2) \quad \sigma \vdash_m m \Downarrow^{w_2} v}{\sigma \vdash_m \text{observe}(e_1; e_2); m \Downarrow^{w_1 \cdot w_2} v} \\
\\
\text{(E:BND:EXPR)} \quad \frac{e \Downarrow v_e \quad \sigma \vdash_m m \{v_e/\mathbf{x}\} \Downarrow^w v}{\sigma \vdash_m \mathbf{x} = e; m \Downarrow^w v} \qquad \text{(E:BND:TERM)} \quad \frac{s \vdash_t t \Downarrow^{w_1} v_1 \quad \{\overline{\mathbf{a}' : s'}\} \vdash_m m \{v_1/\mathbf{a}\} \Downarrow^{w_2} v_2}{\{\mathbf{a} : s, \overline{\mathbf{a}' : s'}\} \vdash_m \mathbf{a} = t; m \Downarrow^{w_1 \cdot w_2} v_2}
\end{array}$$

Figure 23. Evaluation rules for terms and commands.

| | | | | |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| $\Delta; \Gamma \vdash e : \tau$ | | | | |
| (TE:EVAR) $\Delta; \Gamma, \mathbf{x} : \tau \vdash \mathbf{x} : \tau$ | (TE:TVAR) $\Delta, \mathbf{a} : \tau; \Gamma \vdash \mathbf{a} : \tau$ | (TE:TRIV) $\Delta; \Gamma \vdash \text{unit} : \mathbb{1}$ | (TE:TRUE) $\Delta; \Gamma \vdash \text{true} : \mathbb{2}$ | (TE:FALSE) $\Delta; \Gamma \vdash \text{false} : \mathbb{2}$ |
| (TE:COND) $\Delta; \Gamma \vdash e : \mathbb{2} \quad \Delta; \Gamma \vdash e_1 : \tau \quad \Delta; \Gamma \vdash e_2 : \tau$ $\Delta; \Gamma \vdash \text{if}(e; e_1; e_2) : \tau$ | | (TE:UREAL) $r \in [0, 1]$ $\Delta; \Gamma \vdash r : \mathbb{R}_{[0,1]}$ | (TE:PREAL) $r > 0$ $\Delta; \Gamma \vdash r : \mathbb{R}_+$ | (TE:REAL) $\Delta; \Gamma \vdash r : \mathbb{R}$ |
| (TE:FNAT) $n < m$ $\Delta; \Gamma \vdash n : \mathbb{N}_m$ | (TE:NAT) $\Delta; \Gamma \vdash n : \mathbb{N}$ | (TE:OP) $\Delta; \Gamma \vdash e_1 : \diamond.\text{arg}_1 \quad \Delta; \Gamma \vdash e_2 : \diamond.\text{arg}_2$ $\Delta; \Gamma \vdash \text{op}_\diamond(e_1; e_2) : \diamond.\text{res}$ | | (TE:ABS) $\Delta; \Gamma, \mathbf{x} : \tau \vdash e : \tau'$ $\Delta; \Gamma \vdash \lambda \mathbf{x}. e : \tau \rightarrow \tau'$ |
| (TE:APP) $\Delta; \Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Delta; \Gamma \vdash e_2 : \tau_1$ $\Delta; \Gamma \vdash \text{app}(e_1; e_2) : \tau_2$ | | (TE:LET) $\Delta; \Gamma \vdash e_1 : \tau_1 \quad \Delta; \Gamma, \mathbf{x} : \tau_1 \vdash e_2 : \tau_2$ $\Delta; \Gamma \vdash \text{let}(e_1; \mathbf{x}. e_2) : \tau_2$ | | |
| (TE:BER) $\Delta; \Gamma \vdash e : \mathbb{R}_{[0,1]}$ $\Delta; \Gamma \vdash \text{BERN}(e) : \text{dist}(\mathbb{2})$ | | (TE:UNIF) $\Delta; \Gamma \vdash \text{UNIF} : \text{dist}(\mathbb{R}_{[0,1]})$ | (TE:BETA) $\Delta; \Gamma \vdash e_1 : \mathbb{R}_+ \quad \Delta; \Gamma \vdash e_2 : \mathbb{R}_+$ $\Delta; \Gamma \vdash \text{BETA}(e_1; e_2) : \text{dist}(\mathbb{R}_{[0,1]})$ | |
| (TE:GAMMA) $\Delta; \Gamma \vdash e_1 : \mathbb{R}_+ \quad \Delta; \Gamma \vdash e_2 : \mathbb{R}_+$ $\Delta; \Gamma \vdash \text{GAMMA}(e_1; e_2) : \text{dist}(\mathbb{R}_+)$ | | (TE:NORMAL) $\Delta; \Gamma \vdash e_1 : \mathbb{R} \quad \Delta; \Gamma \vdash e_2 : \mathbb{R}_+$ $\Delta; \Gamma \vdash \text{NORMAL}(e_1; e_2) : \text{dist}(\mathbb{R})$ | | |
| (TE:CAT) $\forall i \in \{1, \dots, n\}. \Delta; \Gamma \vdash e_i : \mathbb{R}_+$ $\Delta; \Gamma \vdash \text{CAT}(e_1, \dots, e_n) : \text{dist}(\mathbb{N}_n)$ | | (TE:GEO) $\Delta; \Gamma \vdash e : \mathbb{R}_{[0,1]}$ $\Delta; \Gamma \vdash \text{GEO}(e) : \text{dist}(\mathbb{N})$ | (TE:POIS) $\Delta; \Gamma \vdash e : \mathbb{R}_+$ $\Delta; \Gamma \vdash \text{POIS}(e) : \text{dist}(\mathbb{N})$ | |
| $v : \tau$ | | | | |
| (TV:TRIV) $\text{unit} : \mathbb{1}$ | (TV:TRUE) $\text{true} : \mathbb{2}$ | (TV:FALSE) $\text{false} : \mathbb{2}$ | (TV:UREAL) $r \in (0, 1)$ $r : \mathbb{R}_{[0,1]}$ | (TV:PREAL) $r > 0$ $r : \mathbb{R}_+$ |
| (TV:REAL) $r : \mathbb{R}$ | (TV:FNAT) $n < m$ $n : \mathbb{N}_m$ | (TV:NAT) $n : \mathbb{N}$ | (TV:CLO) $V : \Gamma \quad \Gamma \vdash \lambda \mathbf{x}. e : \tau \rightarrow \tau'$ $\text{clo}(V, \lambda \mathbf{x}. e) : \tau \rightarrow \tau'$ | |
| (TV:UNIF) $\text{UNIF} : \text{dist}(\mathbb{R}_{[0,1]})$ | | (TV:BETA) $v_1 : \mathbb{R}_+ \quad v_2 : \mathbb{R}_+$ $\text{BETA}(v_1; v_2) : \text{dist}(\mathbb{R}_{[0,1]})$ | (TV:GAMMA) $v_1 : \mathbb{R}_+ \quad v_2 : \mathbb{R}_+$ $\text{GAMMA}(v_1; v_2) : \text{dist}(\mathbb{R}_+)$ | |
| (TV:NORMAL) $v_1 : \mathbb{R} \quad v_2 : \mathbb{R}_+$ $\text{NORMAL}(v_1; v_2) : \text{dist}(\mathbb{R})$ | | (TV:CAT) $\forall i \in \{1, \dots, n\}. v_i : \mathbb{R}_+$ $\text{CAT}(v_1, \dots, v_n) : \text{dist}(\mathbb{N}_n)$ | (TV:GEO) $v : \mathbb{R}_{[0,1]}$ $\text{GEO}(v) : \text{dist}(\mathbb{N})$ | (TV:POIS) $v : \mathbb{R}_+$ $\text{POIS}(v) : \text{dist}(\mathbb{N})$ |

Figure 24. Typing rules for expressions and values.

$$\begin{array}{c}
\boxed{\Delta; \Gamma \vdash_t t : \tau \# S} \\
\\
\text{(S:CMD)} \quad \frac{\Delta; \Gamma \vdash_m m : \tau \# \Sigma}{\Delta; \Gamma \vdash_t m : \tau \# \text{trace } \Sigma} \qquad \text{(S:SAMPLE)} \quad \frac{\Delta; \Gamma \vdash_e e : \text{dist}(\tau)}{\Delta; \Gamma \vdash_t \text{sample}(e) : \tau \# \text{atom } \tau} \\
\\
\text{(S:BRANCH)} \quad \frac{\Delta; \bullet \vdash_e e : \mathcal{Z} \quad \Delta; \Gamma \vdash_m m_1 : \tau \# \Sigma_1 \quad \Delta; \Gamma \vdash_m m_2 : \tau \# \Sigma_2}{\Delta; \Gamma \vdash_t \text{ite}(e; m_1; m_2) : \tau \# \Sigma_1 +_e \Sigma_2} \\
\\
\text{(S:CALL)} \quad \frac{\mathbf{f} : \langle \tau_1 \rangle (\tau_2) \rightsquigarrow \tau_3 \# \mathbf{F} \quad \Delta; \bullet \vdash_e e_1 : \tau_1 \quad \Delta; \Gamma \vdash_e e_2 : \tau_2}{\Delta; \Gamma \vdash_t \text{call}(\mathbf{f}; e_1; e_2) : \tau_3 \# \mathbf{F}(e_1)} \\
\\
\boxed{\Delta; \Gamma \vdash_m m : \tau \# \Sigma} \\
\\
\text{(S:RETURN)} \quad \frac{\Delta; \bullet \vdash_e e : \tau}{\Delta; \Gamma \vdash_m \text{ret}(e) : \tau \# \{e\}} \qquad \text{(S:OBSERVE)} \quad \frac{\Delta; \Gamma \vdash_e e_1 : \text{dist}(\tau) \quad \Delta; \Gamma \vdash_e e_2 : \tau \quad \Delta; \Gamma \vdash_m m : \tau \# \Sigma}{\Delta; \Gamma \vdash_m \text{observe}(e_1; e_2); m : \tau \# \Sigma} \qquad \text{(S:BND:EXPR)} \quad \frac{\Delta; \Gamma \vdash_e e : \tau_1}{\Delta; \Gamma, \mathbf{x} : \tau_1 \vdash_m m : \tau_2 \# \Sigma} \\
\\
\text{(S:BND:TERM)} \quad \frac{\Delta; \Gamma \vdash_t t : \tau_1 \# S \quad \Delta, \mathbf{a} : \tau_1; \Gamma \vdash_m m : \tau_2 \# \left\{ \overline{\mathbf{a}'} : S' \right\}_e}{\Delta; \Gamma \vdash_m \mathbf{a} = t; m : \tau_2 \# \left\{ \mathbf{a} : S, \overline{\mathbf{a}'} : S' \right\}_e} \\
\\
\boxed{\vdash_{\mathcal{F}} \mathcal{F} : \langle \tau_1 \rangle (\tau_2) \rightsquigarrow \tau_3 \# \mathbf{F}} \\
\\
\text{(S:DEF)} \quad \frac{\mathbf{f} : \langle \tau_1 \rangle (\tau_2) \rightsquigarrow \tau_3 \# \mathbf{F} \quad \text{typedef } \mathbf{F} = \forall \mathbf{a} : \tau_1. \Sigma \quad \mathbf{a} : \tau_1; \mathbf{x} : \tau_2 \vdash_m m : \tau_3 \# \Sigma}{\vdash_{\mathcal{F}} \text{def } \mathbf{f} \langle \mathbf{a} \rangle (\mathbf{x}) = m : \langle \tau_1 \rangle (\tau_2) \rightsquigarrow \tau_3 \# \mathbf{F}}
\end{array}$$

Figure 25. Typing rules for terms, commands, and function definitions.

$$\begin{array}{c}
\boxed{s \vdash_s S \Downarrow v} \quad \boxed{\sigma \vdash_\sigma \Sigma \Downarrow v} \\
\\
\text{(W:EVENT:ATOM)} \quad \frac{v : \tau}{\text{atom } v \vdash_s \text{atom } \tau \Downarrow v} \qquad \text{(W:EVENT:TRACE)} \quad \frac{\sigma \vdash_\sigma \Sigma \Downarrow v}{\text{trace } \sigma \vdash_s \text{trace } \Sigma \Downarrow v} \qquad \text{(W:EVENT:BRANCH)} \quad \frac{e \Downarrow b \quad \sigma \vdash_\sigma \Sigma_b \Downarrow v}{\text{inj } \sigma \vdash_s \Sigma_{\text{true}} +_e \Sigma_{\text{false}} \Downarrow v} \\
\\
\text{(W:EVENT:CALL)} \quad \frac{\text{typedef } \mathbf{F} = \forall \mathbf{a} : \tau. \Sigma \quad e \Downarrow v_a \quad \sigma \vdash_s \Sigma \{v_a/\mathbf{a}\} \Downarrow v}{\text{fold } \sigma \vdash_s \mathbf{F}(e) \Downarrow v} \qquad \text{(W:TRACE:RET)} \quad \frac{e \Downarrow v}{\{\} \vdash_\sigma \{e\} \Downarrow v} \qquad \text{(W:TRACE:BND)} \quad \frac{s \vdash_s S \Downarrow v_1 \quad \left\{ \overline{\mathbf{a}'} : S' \right\} \vdash_\sigma \left\{ \overline{\mathbf{a}'} : S' \right\}_e \{v_1/\mathbf{a}\} \Downarrow v}{\left\{ \mathbf{a} : s, \overline{\mathbf{a}'} : S' \right\} \vdash_\sigma \left\{ \mathbf{a} : S, \overline{\mathbf{a}'} : S' \right\}_e \Downarrow v}
\end{array}$$

Figure 26. Typing rules for traces and events.

A.2 Measure semantics

Before developing the measure semantics for FCC programs, we first review the basics of measure theory briefly.

σ -algebras A σ -algebra on a set X is a collection Φ of subsets of X that includes X itself and is closed under complement and under countable unions. Let F be an arbitrary family of subsets of X , then there exists a unique smallest σ -algebra that is a superset of F . This σ -algebra is said to be generated by F , denoted by $\sigma(F)$.

Measurable spaces A measurable space (X, Φ) consists of an underlying set and its σ -algebra.

Extended nonnegative reals \mathbb{R}^+ is the set of nonnegative reals extended with infinity ∞ .

Measures Let (X, Φ) be a measurable space. A function $\mu : \Phi \rightarrow \mathbb{R}_+$ is called a measure if it satisfies nonnegativity, null empty set, and countable additivity.

Measure spaces The triple (X, Φ, μ) is called a measure space.

σ -finite measures The measure μ of a measure space (X, Φ, μ) is σ -finite if X can be decomposed into a countable union of sets with finite measure.

Lebesgue measure The Lebesgue measure is a σ -finite measure for n -dimensional Euclidean space, corresponding to length, area and volume for $n = 1, 2, 3$.

Borel spaces A set is a Borel set if and only if it is obtained by countable union, countable intersection, and relative complement from open sets in a topological space. For a topological space (X, τ) , the collection of all Borel sets on X forms a σ -algebra, known as Borel σ -algebra, notated $\mathcal{B}(X)$. A measurable space (X, Φ) is said to be a standard Borel space, if Φ is a Borel σ -algebra generated by a complete metric space on X . A measurable space (X, Φ) is a standard Borel space if and only if it is isomorphic to \mathbb{R}_+ or a subset of \mathbb{N} .

Product measure space Given two measure spaces (X_1, Φ_1, μ_1) and (X_2, Φ_2, μ_2) , the product measurable space is defined as

$$(X_1, \Phi_1) \otimes (X_2, \Phi_2) \stackrel{\text{def}}{=} (X_1 \times X_2, \sigma(\{B_1 \times B_2 \mid B_1 \in \Phi_1 \text{ and } B_2 \in \Phi_2\})).$$

The product measure is defined as $(\mu_1 \otimes \mu_2)(B_1 \times B_2) \stackrel{\text{def}}{=} \mu_1(B_1) \cdot \mu_2(B_2)$. The measure $\mu_1 \otimes \mu_2$ is σ -finite when both μ_1 and μ_2 are σ -finite.

Coproduct measure space Given two measure spaces (X_1, Φ_1, μ_1) and (X_2, Φ_2, μ_2) , the coproduct measure space is defined as

$$(X_1, \Phi_1) \amalg (X_2, \Phi_2) \stackrel{\text{def}}{=} (X_1 \uplus X_2, \sigma(\{B_1 \uplus B_2 \mid B_1 \in \Phi_1 \text{ and } B_2 \in \Phi_2\})).$$

The coproduct measure is defined as $(\mu_1 \amalg \mu_2)(B_1 \uplus B_2) \stackrel{\text{def}}{=} \mu_1(B_1) + \mu_2(B_2)$. Countable coproduct preserves σ -finiteness.

Measurable spaces denoted by types. The definition of $\llbracket \tau \rrbracket$ for each data type τ as a standard Borel space is standard:

$$\begin{aligned} \llbracket \mathbb{1} \rrbracket &\stackrel{\text{def}}{=} (\{\text{unit}\}, \wp(\{\text{unit}\})) \\ \llbracket \mathbb{2} \rrbracket &\stackrel{\text{def}}{=} (\{\text{true}, \text{false}\}, \wp(\{\text{true}, \text{false}\})) \\ \llbracket \mathbb{R}_{[0,1]} \rrbracket &\stackrel{\text{def}}{=} ((0, 1), \mathcal{B}(0, 1)) \\ \llbracket \mathbb{R}_+ \rrbracket &\stackrel{\text{def}}{=} ((0, \infty), \mathcal{B}(0, \infty)) \\ \llbracket \mathbb{R} \rrbracket &\stackrel{\text{def}}{=} ((-\infty, \infty), \mathcal{B}(-\infty, \infty)) \\ \llbracket \mathbb{N}_n \rrbracket &\stackrel{\text{def}}{=} (\{0, 1, \dots, n-1\}, \wp(\{0, 1, \dots, n-1\})) \end{aligned}$$

$$\llbracket \mathbb{N} \rrbracket \stackrel{\text{def}}{=} (\{0, 1, \dots\}, \wp(\{0, 1, \dots\}))$$

Then for each trace type Σ (resp. event type S), we construct a standard Borel space $\llbracket \Sigma \rrbracket$ (resp. $\llbracket S \rrbracket$). To handle recursive trace types, we first define a step-indexed version $\llbracket \Sigma \rrbracket^n$ (resp. $\llbracket S \rrbracket^n$). The definition uses helper injections that map sets formed by the substructures of traces and events into the corresponding sets of traces and events:

$$\begin{aligned} I_{\mathbf{a}_1, \dots, \mathbf{a}_N}(A) &\stackrel{\text{def}}{=} \{ \{ \mathbf{a}_1 : s_1, \dots, \mathbf{a}_N : s_N \} \mid (s_1, \dots, s_N) \in A \} \quad \text{where } \mathbf{a}_1, \dots, \mathbf{a}_N \text{ are lexicographically ordered}^3 \\ I_{\text{atom}}(A) &\stackrel{\text{def}}{=} \{ \text{atom } v \mid v \in A \} \\ I_{\text{trace}}(A) &\stackrel{\text{def}}{=} \{ \text{trace } \sigma \mid \sigma \in A \} \\ I_{\text{inj}}(A) &\stackrel{\text{def}}{=} \{ \text{inj } \sigma \mid \langle _, \sigma \rangle \in A \} \\ I_{\text{fold}}(A) &\stackrel{\text{def}}{=} \{ \text{fold } \sigma \mid \langle _, \sigma \rangle \in A \} \end{aligned}$$

We can now define the step-indexed measurable spaces $\llbracket \Sigma \rrbracket^n$ and $\llbracket S \rrbracket^n$, by double induction, first on the step index, and then on the structure of types:

$$\begin{aligned} \llbracket \{ \mathbf{a}_1 : S_1, \dots, \mathbf{a}_N : S_N \}_e \rrbracket^n &\stackrel{\text{def}}{=} (I_{\mathbf{a}_1, \dots, \mathbf{a}_N}(X), \{ I_{\mathbf{a}_1, \dots, \mathbf{a}_N}(B) \mid B \in \Phi \}) \quad \text{where } (X, \Phi) \stackrel{\text{def}}{=} \bigotimes_{i=1}^N \llbracket S_{i_i} \rrbracket^n \text{ and } (\mathbf{a}_i)_{i=1}^N \text{ is} \\ &\quad \text{the lexicographical reordering of } (\mathbf{a}_i)_{i=1}^N \\ \llbracket \text{atom } \tau \rrbracket^n &\stackrel{\text{def}}{=} (I_{\text{atom}}(X), \{ I_{\text{atom}}(B) \mid B \in \Phi \}) \quad \text{where } (X, \Phi) \stackrel{\text{def}}{=} \llbracket \tau \rrbracket \\ \llbracket \text{trace } \Sigma \rrbracket^n &\stackrel{\text{def}}{=} (I_{\text{trace}}(X), \{ I_{\text{trace}}(B) \mid B \in \Phi \}) \quad \text{where } (X, \Phi) \stackrel{\text{def}}{=} \llbracket \Sigma \rrbracket^n \\ \llbracket \Sigma_1 +_e \Sigma_2 \rrbracket^n &\stackrel{\text{def}}{=} (I_{\text{inj}}(X), \{ I_{\text{inj}}(B) \mid B \in \Phi \}) \quad \text{where } (X, \Phi) \stackrel{\text{def}}{=} \llbracket \Sigma_1 \rrbracket^n \amalg \llbracket \Sigma_2 \rrbracket^n \\ \llbracket \mathbf{F}\langle e \rangle \rrbracket^0 &\stackrel{\text{def}}{=} (\emptyset, \wp(\emptyset)) \\ \llbracket \mathbf{F}\langle e \rangle \rrbracket^{n+1} &\stackrel{\text{def}}{=} (I_{\text{fold}}(X), \{ I_{\text{fold}}(B) \mid B \in \Phi \}) \quad \text{where } (X, \Phi) \stackrel{\text{def}}{=} \prod_{i=0}^n \llbracket \Sigma \{e/\mathbf{a}_i\} \rrbracket^i \text{ and} \\ &\quad \text{typedef } \mathbf{F} = \forall \mathbf{a} : \tau. \Sigma \end{aligned}$$

Measurable spaces $\llbracket \Sigma \rrbracket$ and $\llbracket S \rrbracket$ can then be defined by induction on the structure of types, with the $\llbracket \mathbf{F}\langle e \rangle \rrbracket$ case taking the fixed point:

$$\begin{aligned} \llbracket \{ \mathbf{a}_1 : S_1, \dots, \mathbf{a}_N : S_N \}_e \rrbracket &\stackrel{\text{def}}{=} (I_{\mathbf{a}_1, \dots, \mathbf{a}_N}(X), \{ I_{\mathbf{a}_1, \dots, \mathbf{a}_N}(B) \mid B \in \Phi \}) \quad \text{where } (X, \Phi) \stackrel{\text{def}}{=} \bigotimes_{i=1}^N \llbracket S_{i_i} \rrbracket \text{ and } (\mathbf{a}_i)_{i=1}^N \text{ is} \\ &\quad \text{the lexicographical reordering of } (\mathbf{a}_i)_{i=1}^N \\ \llbracket \text{atom } \tau \rrbracket &\stackrel{\text{def}}{=} (I_{\text{atom}}(X), \{ I_{\text{atom}}(B) \mid B \in \Phi \}) \quad \text{where } (X, \Phi) \stackrel{\text{def}}{=} \llbracket \tau \rrbracket \\ \llbracket \text{trace } \Sigma \rrbracket &\stackrel{\text{def}}{=} (I_{\text{trace}}(X), \{ I_{\text{trace}}(B) \mid B \in \Phi \}) \quad \text{where } (X, \Phi) \stackrel{\text{def}}{=} \llbracket \Sigma \rrbracket \\ \llbracket \Sigma_1 +_e \Sigma_2 \rrbracket &\stackrel{\text{def}}{=} (I_{\text{inj}}(X), \{ I_{\text{inj}}(B) \mid B \in \Phi \}) \quad \text{where } (X, \Phi) \stackrel{\text{def}}{=} \llbracket \Sigma_1 \rrbracket \amalg \llbracket \Sigma_2 \rrbracket \\ \llbracket \mathbf{F}\langle e \rangle \rrbracket &\stackrel{\text{def}}{=} (I'_{\text{fold}}(X), \{ I'_{\text{fold}}(B) \mid B \in \Phi \}) \quad \text{where } (X, \Phi) \stackrel{\text{def}}{=} \prod_{n=0}^{\infty} \llbracket \mathbf{F}\langle e \rangle \rrbracket^n \end{aligned}$$

where $I'_{\text{fold}}(A) \stackrel{\text{def}}{=} \{ \text{fold } \sigma \mid \langle _, \text{fold } \sigma \rangle \in A \}$.

³The lexicographical ordering is on variable names. A canonical ordering of the term variables is needed, because traces are unordered maps but the construction of N -tuples requires an ordering.

Stock measures. The stock measure μ_τ on the measurable space $\llbracket \tau \rrbracket$ denoted by a data type τ is standard, with $\mu_{\mathbb{1}}$, $\mu_{\mathbb{2}}$, $\mu_{\mathbb{N}_n}$ and $\mu_{\mathbb{N}}$ being the counting measures on their respective spaces, and $\mu_{\mathbb{R}_{[0,1]}}$, $\mu_{\mathbb{R}_+}$ and $\mu_{\mathbb{R}}$ the Lebesgue measure.

For each trace type Σ (resp. event type S), we define a stock measure μ_Σ (resp. μ_S) on its semantic domain $\llbracket \Sigma \rrbracket$ (resp. $\llbracket S \rrbracket$). Again, we first define a step-indexed version, μ_Σ^n and μ_S^n , via double induction on the index and the structure of types:

$$\begin{aligned} \mu_{\{a_1:S_1, \dots, a_N:S_N\}_e}^n (I_{a_1, \dots, a_N}(B)) &\stackrel{\text{def}}{=} \left(\bigotimes_{i=1}^N \mu_{S_i}^n \right) (B) \quad \text{where } (a_i)_{i=1}^N \text{ is the lexicographical reordering of } (a_i)_{i=1}^N \\ \mu_{\text{atom } \tau}^n (I_{\text{atom}}(B)) &\stackrel{\text{def}}{=} \mu_\tau(B) \\ \mu_{\text{trace } \Sigma}^n (I_{\text{trace}}(B)) &\stackrel{\text{def}}{=} \mu_\Sigma^n(B) \\ \mu_{\Sigma_1 +_e \Sigma_2}^n (I_{\text{inj}}(B)) &\stackrel{\text{def}}{=} \left(\mu_{\Sigma_1}^n \amalg \mu_{\Sigma_2}^n \right) (B) \\ \mu_{\mathbb{F}\langle e \rangle}^0 (\emptyset) &\stackrel{\text{def}}{=} 0 \\ \mu_{\mathbb{F}\langle e \rangle}^{n+1} (I_{\text{fold}}(B)) &\stackrel{\text{def}}{=} \left(\prod_{i=0}^n \mu_{\Sigma\{e/a\}}^i \right) (B) \quad \text{where } \text{typedef } \mathbb{F} = \forall a : \tau. \Sigma \end{aligned}$$

Measures μ_Σ^n and μ_S^n are then defined by induction on the structure of types:

$$\begin{aligned} \mu_{\{a_1:S_1, \dots, a_N:S_N\}_e} (I_{a_1, \dots, a_N}(B)) &\stackrel{\text{def}}{=} \left(\bigotimes_{i=1}^N \mu_{S_i} \right) (B) \quad \text{where } (a_i)_{i=1}^N \text{ is the lexicographical reordering of } (a_i)_{i=1}^N \\ \mu_{\text{atom } \tau} (I_{\text{atom}}(B)) &\stackrel{\text{def}}{=} \mu_\tau(B) \\ \mu_{\text{trace } \Sigma} (I_{\text{trace}}(B)) &\stackrel{\text{def}}{=} \mu_\Sigma(B) \\ \mu_{\Sigma_1 +_e \Sigma_2} (I_{\text{inj}}(B)) &\stackrel{\text{def}}{=} \left(\mu_{\Sigma_1} \amalg \mu_{\Sigma_2} \right) (B) \\ \mu_{\mathbb{F}\langle e \rangle} (I'_{\text{fold}}(B)) &\stackrel{\text{def}}{=} \left(\prod_{n=0}^{\infty} \mu_{\mathbb{F}\langle e \rangle}^n \right) (B) \end{aligned}$$

The measures are σ -finite, because the measures μ_τ are σ -finite, and because σ -finiteness is preserved by finite products and countable coproducts, which are used to construct μ_Σ and μ_S .

Definition A.1 (Restatement of Definition 4.1). A closed command m of trace type Σ ($\bullet; \bullet \vdash_m m : \tau \# \Sigma$) induces a measure $\llbracket m \rrbracket$ over a measurable space $\llbracket \Sigma \rrbracket$ of traces:

$$\mathbf{P}_m(\sigma) \stackrel{\text{def}}{=} \begin{cases} w, & \text{if } \sigma \vdash_m m \Downarrow^w v \\ 0, & \text{otherwise} \end{cases} \quad \llbracket m \rrbracket (A) \stackrel{\text{def}}{=} \int_A \mathbf{P}_m(\sigma) \mu_\Sigma(d\sigma)$$

B METATHEORETICAL DEVELOPMENT FOR FCC

B.1 Theorems in Section 4

PROPOSITION B.1. *If $d : \text{dist}(\tau)$ and v is a value, then $v : \tau$ if and only if $v \in d.\text{support}$ (i.e., $d.\text{density}(v) > 0$).*

PROOF. Appeal to mathematical properties of primitive distributions. \square

PROPOSITION B.2 (PRESERVATION AND STRONG NORMALIZATION OF STLC).

- If $\Delta; \Gamma \vdash_e e : \tau$, and $e \Downarrow v$, then $\Delta; \Gamma \vdash_e v : \tau$.
- If $\Delta; \Gamma \vdash_e e : \tau$, then there exists a value v such that $e \Downarrow v$.

PROOF. Appeal to type soundness and strong normalization of the simply-typed lambda calculus. \square

LEMMA B.3.

- (1) If $\sigma \vdash_m m \Downarrow^w v$, then $w > 0$.
- (2) If $s \vdash_t t \Downarrow^w v$, then $w > 0$.

PROOF. By mutual structural induction on evaluation derivations. \square

LEMMA B.4 (SUBSTITUTION OF EXPRESSIONS).

- (1) If $\Delta; \Gamma, \mathbf{x} : \tau_1 \vdash_e e : \tau_2$ and $\Delta; \Gamma \vdash_e v : \tau_1$, then $\Delta; \Gamma \vdash_e e \{v/\mathbf{x}\} : \tau_2$.
- (2) If $\Delta, \mathbf{a} : \tau_1; \Gamma \vdash_e e : \tau_2$ and $\Delta; \Gamma \vdash_e v : \tau_1$, then $\Delta; \Gamma \vdash_e e \{v/\mathbf{a}\} : \tau_2$.

PROOF. Appeals to the substitution lemma of the simply-typed lambda calculus. \square

LEMMA B.5 (SUBSTITUTION FOR COMMANDS).

- (1) If $\Delta; \Gamma, \mathbf{x} : \tau_1 \vdash_t t : \tau_2 \# S$ and $\Delta; \Gamma \vdash_e v : \tau_1$, then $\Delta; \Gamma \vdash_t t \{v/\mathbf{x}\} : \tau_2 \# S$.
- (2) If $\Delta, \mathbf{a} : \tau_1; \Gamma \vdash_t t : \tau_2 \# S$ and $\Delta; \Gamma \vdash_e v : \tau_1$, then $\Delta; \Gamma \vdash_t t \{v/\mathbf{a}\} : \tau_2 \# S \{v/\mathbf{a}\}$.
- (3) If $\Delta; \Gamma, \mathbf{x} : \tau_1 \vdash_m m : \tau_2 \# \Sigma$ and $\Delta; \Gamma \vdash_e v : \tau_1$, then $\Delta; \Gamma \vdash_m m \{v/\mathbf{x}\} : \tau_2 \# \Sigma$.
- (4) If $\Delta, \mathbf{a} : \tau_1; \Gamma \vdash_m m : \tau_2 \# \Sigma$ and $\Delta; \Gamma \vdash_e v : \tau_1$, then $\Delta; \Gamma \vdash_m m \{v/\mathbf{a}\} : \tau_2 \# \Sigma \{v/\mathbf{a}\}$.

PROOF. We show several nontrivial cases for (item 4) and (item 2).

Proof by mutual induction.

Case: (S:RETURN)

- (1) $\Delta, \mathbf{a} : \tau_1; \Gamma \vdash_m \text{ret}(e) : \tau_2 \# \{e\}$ and $\Delta; \Gamma \vdash_e v : \tau_1$ (ASSUMPTION)
- (2) $\Delta, \mathbf{a} : \tau_1; \bullet \vdash_e e : \tau_2$ (INVERSION) $\hat{\circ}$ (1)
- (3) $\Delta; \bullet \vdash_e e \{v/\mathbf{a}\} : \tau_2$ (LEMMA B.4) $\hat{\circ}$ (1) $\hat{\circ}$ (2)
- (4) $\Delta; \Gamma \vdash_m \text{ret}(e \{v/\mathbf{a}\}) : \tau_2 \# \{e \{v/\mathbf{a}\}\}$ (S:RETURN) $\hat{\circ}$ (3)

Case: (S:BND:EXPR)

- (1) $\Delta, \mathbf{a} : \tau_1; \Gamma \vdash_m \mathbf{x} = e; m : \tau_2 \# \Sigma$ and $\Delta; \Gamma \vdash_e v : \tau_1$ (ASSUMPTION)
- (2) $\Delta, \mathbf{a} : \tau_1; \Gamma \vdash_e e : \tau_x$ and $\Delta, \mathbf{a} : \tau_1; \Gamma, \mathbf{x} : \tau_x \vdash_m m : \tau_2 \# \Sigma$ (INVERSION) $\hat{\circ}$ (1)
- (3) $\Delta; \Gamma \vdash_e e \{v/\mathbf{a}\} : \tau_x$ (LEMMA B.4) $\hat{\circ}$ (1) $\hat{\circ}$ (2)
- (4) $\Delta; \Gamma, \mathbf{x} : \tau_x \vdash_e v : \tau_1$ (WEAKENING) $\hat{\circ}$ (1)
- (5) $\Delta; \Gamma, \mathbf{x} : \tau_x \vdash_m m \{v/\mathbf{a}\} : \tau_2 \# \Sigma \{v/\mathbf{a}\}$ (I.H.) $\hat{\circ}$ (2) $\hat{\circ}$ (4)
- (6) $\Delta; \Gamma \vdash_m \mathbf{x} = e \{v/\mathbf{a}\}; m \{v/\mathbf{a}\} : \tau_2 \# \Sigma \{v/\mathbf{a}\}$ (S:BND:EXPR) $\hat{\circ}$ (3) $\hat{\circ}$ (5)

Case: (S:BND:TERM)

- (1) $\Delta, \mathbf{a} : \tau_a; \Gamma \vdash_m \mathbf{b} = t; m : \tau_2 \# \left\{ \mathbf{b} : S, \overline{\mathbf{a}' : S'} \right\}_e$ and $\Delta; \Gamma \vdash_e v : \tau_v$ (ASSUMPTION)
- (2) $\Delta, \mathbf{a} : \tau_1; \Gamma \vdash_t t : \tau_b \# S$ and $\Delta, \mathbf{a} : \tau_1, \mathbf{b} : \tau_b; \Gamma \vdash_m m : \tau_2 \# \left\{ \overline{\mathbf{a}' : S'} \right\}_e$ (INVERSION) $\hat{\circ}$ (1)
- (3) $\Delta; \Gamma \vdash_t t \{v/\mathbf{a}\} : \tau_b \# S \{v/\mathbf{a}\}$ (I.H.) $\hat{\circ}$ (1) $\hat{\circ}$ (2)
- (4) $\Delta, \mathbf{b} : \tau_b; \Gamma \vdash_e v : \tau_1$ (WEAKENING) $\hat{\circ}$ (1)

$$(5) \Delta, \mathbf{b} : \tau_b; \Gamma \vdash_m m \{v/a\} : \tau_2 \# \overline{\{a' : S' \{v/a\}\}}_{e\{v/a\}} \quad (\text{I.H.}) \dot{\circ} (2) \dot{\circ} (4)$$

$$(6) \Delta; \Gamma \vdash_m \mathbf{b} = t \{v/a\}; m \{v/a\} : \tau_2 \# \overline{\{\mathbf{b} : S \{v/a\}, a' : S' \{v/a\}\}}_{e\{v/a\}} \quad \text{S:BND:TERM} \dot{\circ} (3) \dot{\circ} (5)$$

Case: (S:CMD)

$$(1) \Delta, \mathbf{a} : \tau_1; \Gamma \vdash_t m : \tau_2 \# \text{trace } \Sigma \text{ and } \Delta; \Gamma \vdash_e v : \tau_1 \quad (\text{ASSUMPTION})$$

$$(2) \Delta, \mathbf{a} : \tau_1; \Gamma \vdash_m m : \tau_2 \# \Sigma \quad (\text{INVERSION}) \dot{\circ} (1)$$

$$(3) \Delta; \Gamma \vdash_m m \{v/a\} : \tau_2 \# \Sigma \{v/a\} \quad (\text{I.H.}) \dot{\circ} (1) \dot{\circ} (2)$$

$$(4) \Delta; \Gamma \vdash_t m \{v/a\} : \tau_2 \# \text{trace } \Sigma \{v/a\} \quad \text{S:CMD} \dot{\circ} (3)$$

Case: (S:SAMPLE)

$$(1) \Delta, \mathbf{a} : \tau_1; \Gamma \vdash_t \text{sample}(e) : \tau_2 \# \text{atom } \tau_2 \text{ and } \Delta; \Gamma \vdash_e v : \tau_1 \quad (\text{ASSUMPTION})$$

$$(2) \Delta, \mathbf{a} : \tau_1; \Gamma \vdash_e e : \text{dist}(\tau_2) \quad (\text{INVERSION}) \dot{\circ} (1)$$

$$(3) \Delta; \Gamma \vdash_e e \{v/a\} : \text{dist}(\tau_2) \quad (\text{LEMMA B.4}) \dot{\circ} (1) \dot{\circ} (2)$$

$$(4) \Delta; \Gamma \vdash_t \text{sample}(e \{v/a\}) : \tau_2 \# \text{atom } \tau_2 \quad \text{S:SAMPLE} \dot{\circ} (3)$$

Case: (S:BRANCH)

$$(1) \Delta, \mathbf{a} : \tau_1; \Gamma \vdash_t \text{ite}(e; m_1; m_2) : \tau_2 \# \Sigma_1 +_e \Sigma_2 \text{ and } \Delta; \Gamma \vdash_e v : \tau_1 \quad (\text{ASSUMPTION})$$

$$(2) \Delta, \mathbf{a} : \tau_1; \bullet \vdash_e e : 2,$$

$$\Delta, \mathbf{a} : \tau_1; \Gamma \vdash_m m_1 : \tau_2 \# \Sigma_1 \text{ and}$$

$$\Delta, \mathbf{a} : \tau_1; \Gamma \vdash_m m_2 : \tau_2 \# \Sigma_2 \quad (\text{INVERSION}) \dot{\circ} (1)$$

$$(3) \Delta; \bullet \vdash_e e \{v/a\} : 2,$$

$$\Delta; \Gamma \vdash_m m_1 \{v/a\} : \tau_2 \# \Sigma_1 \{v/a\} \text{ and}$$

$$\Delta; \Gamma \vdash_m m_2 \{v/a\} : \tau_2 \# \Sigma_1 \{v/a\}$$

$$(\text{PROPOSITION B.2}) \dot{\circ} (\text{I.H.}) \dot{\circ} (1) \dot{\circ} (2)$$

$$(4) \Delta; \Gamma \vdash_t \text{ite}(e \{v/a\}; m_1 \{v/a\}; m_2 \{v/a\}) : \tau_2 \# \Sigma_1 \{v/a\} e \{v/a\} \Sigma_2 \{v/a\} \quad \text{S:BRANCH} \dot{\circ} (3)$$

Case: (S:CALL)

$$(1) \Delta, \mathbf{a} : \tau_1; \Gamma \vdash_t \text{call}(\mathbf{f}; e_1; e_2) : \tau_3 \# \mathbf{F}\langle e_1 \rangle \text{ and } \Delta; \Gamma \vdash_e v : \tau_1 \quad (\text{ASSUMPTION})$$

$$(2) \mathbf{f} : \langle \tau_1 \rangle (\tau_2) \rightsquigarrow \tau_3 \# \mathbf{F}$$

$$\Delta, \mathbf{a} : \tau_a; \bullet \vdash_e e_1 : \tau_1$$

$$\Delta, \mathbf{a} : \tau_a; \Gamma \vdash_e e_2 : \tau_2 \quad (\text{INVERSION}) \dot{\circ} (1)$$

$$(3) \Delta; \bullet \vdash_e e_1 \{v/a\} : \tau_1$$

$$\Delta; \Gamma \vdash_e e_2 \{v/a\} : \tau_2$$

$$(\text{LEMMA B.4}) \dot{\circ} (1) \dot{\circ} (2)$$

$$(4) \Delta; \Gamma \vdash_t \text{call}(\mathbf{f}; e_1 \{v/a\}; e_2 \{v/a\}) : \tau_3 \# \mathbf{F}\langle e_1 \{v/a\} \rangle \quad \text{S:CALL} \dot{\circ} (2) \dot{\circ} (3)$$

□

THEOREM B.6 (RESTATEMENT OF THEOREM 4.3).

(1) If $\bullet; \bullet \vdash_m m : \tau \# \Sigma$, and $\sigma \vdash_m m \Downarrow^w v$, then $\bullet; \bullet \vdash_e v : \tau$ and $\sigma \vdash_\sigma \Sigma \Downarrow v$.

(2) If $\bullet; \bullet \vdash_t t : \tau \# S$ and $s \vdash_t t \Downarrow^w v$, then $\bullet; \bullet \vdash_e v : \tau$ and $s \vdash_s S \Downarrow v$.

PROOF. By mutual induction on the derivation of $\sigma \vdash_m m \Downarrow^w v$, and $s \vdash_t t \Downarrow^w v$ followed by inversion on $\bullet; \bullet \vdash_m m : \tau \# \Sigma$ and $\bullet; \bullet \vdash_t t : \tau \# S$ respectively. We show several nontrivial cases; others are similar to one of these cases.

Case: (E:RET)

$$\frac{(\text{E:RET}) \quad e \Downarrow v}{\{\} \vdash_m \text{ret}(e) \Downarrow^1 v}$$

$$\frac{(\text{S:RETURN}) \quad \Delta; \bullet \vdash_e e : \tau}{\Delta; \Gamma \vdash_m \text{ret}(e) : \tau \# \{\} e}$$

$$(1) \bullet; \bullet \vdash_e e : \tau \text{ and } e \Downarrow v \quad (\text{ASSUMPTION})$$

$$(2) \bullet; \bullet \vdash_e v : \tau \quad (\text{PROPOSITION B.2}) \dot{\circ} (1)$$

$$(3) \{\} \vdash_\sigma \{\} e \Downarrow v \quad \text{W:TRACE:RET} \dot{\circ} (1)$$

Case: (E:BND:EXPR)

$$\frac{\text{(E:BND:EXPR)} \quad e \Downarrow v_e \quad \sigma \vdash_m m \{v_e/x\} \Downarrow^w v}{\sigma \vdash_m x = e; m \Downarrow^w v}$$

- (1) $\bullet; \bullet \vdash_e e : \tau_1$ and $e \Downarrow v_e$
- (2) $\bullet; \bullet \vdash_e v_e : \tau_1$
- (3) $\bullet; \bullet; x : \tau_1 \vdash_m m : \tau_2 \# \Sigma$
- (4) $\bullet; \bullet; \bullet \vdash_m m \{v_e/x\} : \tau_2 \# \Sigma$
- (5) $\sigma \vdash_m m \{v_e/x\} \Downarrow^w v$
- (6) $\bullet; \bullet \vdash_e v : \tau_2$
- (7) $\sigma \vdash_\sigma \Sigma \Downarrow v$

(S:BND:EXPR)

$$\frac{\Delta; \Gamma \vdash_e e : \tau_1 \quad \Delta; \Gamma, x : \tau_1 \vdash_m m : \tau_2 \# \Sigma}{\Delta; \Gamma \vdash_m x = e; m : \tau_2 \# \Sigma}$$

- (ASSUMPTION)
 (PROPOSITION B.2) $\hat{\sigma}$ (1)
 (ASSUMPTION)
 (LEMMA B.5) $\hat{\sigma}$ (2) $\hat{\sigma}$ (3)
 (ASSUMPTION)
 (I.H.) $\hat{\sigma}$ (4) $\hat{\sigma}$ (5)
 (I.H.) $\hat{\sigma}$ (4) $\hat{\sigma}$ (5)

Case: (E:BND:TERM)

$$\frac{\text{(E:BND:TERM)} \quad s \vdash_t t \Downarrow^{w_1} v_1 \quad \{\overline{a'} : s'\} \vdash_m m \{v_1/a\} \Downarrow^{w_2} v_2}{\{\overline{a} : s, \overline{a'} : s'\} \vdash_m a = t; m \Downarrow^{w_1 \cdot w_2} v_2}$$

(S:BND:TERM)

$$\frac{\Delta; \Gamma \vdash_t t : \tau_1 \# S \quad \Delta, \mathbf{a} : \tau_1; \Gamma \vdash_m m : \tau_2 \# \{\overline{a'} : S'\}_e}{\Delta; \Gamma \vdash_m a = t; m : \tau_2 \# \{\overline{a} : S, \overline{a'} : S'\}_e}$$

- (1) $\bullet; \bullet \vdash_t t : \tau_1 \# S$ and $s \vdash_t t \Downarrow^{w_1} v_1$ (ASSUMPTION)
- (2) $\bullet; \bullet \vdash_e v_1 : \tau_1$ (I.H.) $\hat{\sigma}$ (1)
- (3) $\mathbf{a} : \tau_1; \bullet \vdash_m m : \tau_2 \# \{\overline{a'} : S'\}_e$ (ASSUMPTION)
- (4) $\bullet; \bullet \vdash_m m \{v_1/a\} : \tau_2 \# \{\overline{a'} : S'\}_e \{v_1/a\}$ (LEMMA B.5) $\hat{\sigma}$ (2) $\hat{\sigma}$ (3)
- (5) $\{\overline{a'} : s'\} \vdash_m m \{v_1/a\} \Downarrow^{w_2} v_2$ (ASSUMPTION)
- (6) $\bullet; \bullet \vdash_e v_2 : \tau_2$ (I.H.) $\hat{\sigma}$ (4) $\hat{\sigma}$ (5)
- (7) $\{\overline{a'} : s'\} \vdash_\sigma \{\overline{a'} : S'\}_e \{v_1/a\} \Downarrow v_2$ (I.H.) $\hat{\sigma}$ (4) $\hat{\sigma}$ (5)
- (8) $s \vdash_\sigma S \Downarrow v_1$ (I.H.) $\hat{\sigma}$ (1)
- (9) $\{\overline{a} : s, \overline{a'} : s'\} \vdash_\sigma \{\overline{a} : S, \overline{a'} : S'\}_e \Downarrow v_2$ (W:TRACE:BND) $\hat{\sigma}$ (7) $\hat{\sigma}$ (8)

Case: (E:OBSERVE)

(E:OBSERVE)

$$\frac{v \in d.\text{support} \quad e_1 \Downarrow d \quad e_2 \Downarrow v_2 \quad w_1 = d.\text{density}(v_2) \quad \sigma \vdash_m m \Downarrow^{w_2} v}{\sigma \vdash_m \text{observe}(e_1; e_2); m \Downarrow^{w_1 \cdot w_2} v}$$

- (1) $\bullet; \bullet \vdash_m m : \tau \# \Sigma$ and $\sigma \vdash_t m \Downarrow^{w_2} v$
- (2) $\bullet; \bullet \vdash_e v : \tau$ and $\sigma \vdash_\sigma \Sigma \Downarrow v$

(S:OBSERVE)

$$\frac{\Delta; \Gamma \vdash_e e_1 : \text{dist}(\tau) \quad \Delta; \Gamma \vdash_e e_2 : \tau \quad \Delta; \Gamma \vdash_m m : \tau \# \Sigma}{\Delta; \Gamma \vdash_m \text{observe}(e_1; e_2); m : \tau \# \Sigma}$$

- (ASSUMPTION)
 (I.H.) $\hat{\sigma}$ (1)

Case: (E:CMD)

(E:CMD)

$$\frac{\sigma \vdash_m m \Downarrow^w v}{\text{trace } \sigma \vdash_t m \Downarrow^w v}$$

- (1) $\bullet; \bullet \vdash_m m : \tau \# \Sigma$ and $\sigma \vdash_t m \Downarrow^w v$
- (2) $\bullet; \bullet \vdash_e v : \tau$ and $\sigma \vdash_\sigma \Sigma \Downarrow v$
- (3) $\text{trace } \sigma \vdash_s \text{trace } \Sigma \Downarrow v$

(S:CMD)

$$\frac{\Delta; \Gamma \vdash_m m : \tau \# \Sigma}{\Delta; \Gamma \vdash_t m : \tau \# \text{trace } \Sigma}$$

- (ASSUMPTION)
 (I.H.) $\hat{\sigma}$ (1)
 (W:EVENT:TRACE) $\hat{\sigma}$ (2)

Case: (E:SAMPLE)

$$\frac{\text{(E:SAMPLE)} \quad \begin{array}{l} e \Downarrow d \quad v \in d.\text{support} \\ w = d.\text{density}(v) \end{array}}{\text{atom } v \vdash_t \text{sample}(e) \Downarrow^w v} \quad \text{(S:SAMPLE)} \quad \frac{\Delta; \Gamma \vdash_e e : \text{dist}(\tau)}{\Delta; \Gamma \vdash_t \text{sample}(e) : \tau \# \text{atom } \tau}$$

(1) $\bullet; \bullet \vdash_e e : \text{dist}(\tau)$ and $v \in d.\text{support}$ (ASSUMPTION)

(2) $\bullet; \bullet \vdash_e v : \tau$ (PROPOSITION B.1) $\hat{\circ}$ (1)

(3) $\text{atom } v \vdash_\sigma \text{atom } \tau \Downarrow v$ (W:EVENT:ATOM) $\hat{\circ}$ (2)

Case: (E:BRANCH)

$$\frac{\text{(E:BRANCH)} \quad \frac{e \Downarrow b \quad \sigma \vdash_m m_b \Downarrow^w v}{\text{inj } \sigma \vdash_t \text{ite}(e; m_{\text{true}}; m_{\text{false}}) \Downarrow^w v}}{\text{(S:BRANCH)} \quad \frac{\Delta; \bullet \vdash_e e : 2 \quad \Delta; \Gamma \vdash_m m_1 : \tau \# \Sigma_1 \quad \Delta; \Gamma \vdash_m m_2 : \tau \# \Sigma_2}{\Delta; \Gamma \vdash_t \text{ite}(e; m_1; m_2) : \tau \# \Sigma_1 +_e \Sigma_2}}$$

(1) $\bullet; \bullet \vdash_m m_i : \tau \# \Sigma_i$ and $\sigma \vdash_t m_i \Downarrow^w v$ where $i = \text{ite}(v_b, 1, 2)$ (ASSUMPTION)

(2) $\bullet; \bullet \vdash_e v : \tau$ (I.H.) $\hat{\circ}$ (1)

(3) $\sigma \vdash_\sigma \Sigma_i \Downarrow v$ (I.H.) $\hat{\circ}$ (1)

(4) $\bullet; \bullet \vdash_e e : 2$ and $e \Downarrow v_b$ (ASSUMPTION)

(5) $\text{inj } \sigma \vdash_\sigma \Sigma_1 +_e \Sigma_2 \Downarrow v_b$ (W:EVENT:BRANCH) $\hat{\circ}$ (3) $\hat{\circ}$ (4)

Case: (E:CALL)

$$\frac{\text{(E:CALL)} \quad \frac{\text{def } f\langle a \rangle(x) = m \quad e_1 \Downarrow v_a \quad e_2 \Downarrow v_x}{\sigma \vdash_t m \{v_a/a\} \{v_x/x\} \Downarrow^w v}}{\text{fold } \sigma \vdash_t \text{call}(f; e_1; e_2) \Downarrow^w v} \quad \text{(S:CALL)} \quad \frac{f : \langle \tau_1 \rangle(\tau_2) \rightsquigarrow \tau_3 \# F \quad \Delta; \bullet \vdash_e e_1 : \tau_1 \quad \Delta; \Gamma \vdash_e e_2 : \tau_2}{\Delta; \Gamma \vdash_t \text{call}(f; e_1; e_2) : \tau_3 \# F\langle e_1 \rangle} \quad \text{(S:DEF)} \quad \frac{f : \langle \tau_1 \rangle(\tau_2) \rightsquigarrow \tau_3 \# F \quad \text{typedef } F = \forall a : \tau_1. \Sigma}{a : \tau_1; x : \tau_2 \vdash_m m : \tau_3 \# \Sigma}}{\vdash_{\mathcal{F}} \text{def } f\langle a \rangle(x) = m : \langle \tau_1 \rangle(\tau_2) \rightsquigarrow \tau_3 \# F}$$

(1) $a : \tau_1; x : \tau_2 \vdash_m m : \tau_3 \# \Sigma$ (ASSUMPTION)

(2) $\bullet; \bullet \vdash_e e_1 : \tau_1, \bullet; \bullet \vdash_e e_2 : \tau_2, e_1 \Downarrow v_a$ and $e_2 \Downarrow v_x$ (ASSUMPTION)

(3) $\bullet; \bullet \vdash_e v_a : \tau_1$ and $\bullet; \bullet \vdash_e v_x : \tau_2$ (PROPOSITION B.2) $\hat{\circ}$ (2)

(4) $\bullet; \bullet \vdash_m m \{v_a/a\} \{v_x/x\} : \tau_3 \# \Sigma \{v_a/a\}$ (LEMMA B.5) $\hat{\circ}$ (3)

(5) $\sigma \vdash_t m \{v_a/a\} \{v_x/x\} \Downarrow^w v$ (ASSUMPTION)

(6) $\bullet; \bullet \vdash_e v : \tau_3$ (I.H.) $\hat{\circ}$ (4) $\hat{\circ}$ (5)

(7) $\sigma \vdash_\sigma \Sigma \{v_a/a\} \Downarrow v$ (I.H.) $\hat{\circ}$ (4) $\hat{\circ}$ (5)

(8) $\text{typedef } F = \forall a : \tau_1. \Sigma$ (ASSUMPTION)

(9) $\text{fold } \sigma \vdash_s : F(e) \Downarrow v$ (W:EVENT:CALL) $\hat{\circ}$ (2) $\hat{\circ}$ (7) $\hat{\circ}$ (8)

□

THEOREM B.7 (RESTATEMENT OF THEOREM 4.4).

- (1) If $\bullet; \bullet \vdash_m m : \tau \# \Sigma$ and $\sigma \vdash_\sigma \Sigma \Downarrow v$, then there exist w such that $\sigma \vdash_m m \Downarrow^w v$.
- (2) If $\bullet; \bullet \vdash_t t : \tau \# S$ and $s \vdash_s S \Downarrow v$, then there exist w such that $s \vdash_t t \Downarrow^w v$.

PROOF. By mutual induction on the derivation of $\bullet; \bullet \vdash_m m : \tau \# \Sigma$, and $\bullet; \bullet \vdash_t t : \tau \# S$, followed by nested induction on $\sigma \vdash_\sigma \Sigma \Downarrow v$ and $s \vdash_s S \Downarrow v$ respectively.

Case: (S:RETURN)

$$\begin{array}{c} \text{(S:RETURN)} \\ \Delta; \bullet \vdash_e e : \tau \\ \hline \Delta; \Gamma \vdash_m \text{ret}(e) : \tau \# \{e\} \end{array} \qquad \begin{array}{c} \text{(W:TRACE:RET)} \\ e \Downarrow v \\ \hline \{ \} \vdash_\sigma \{e\} \Downarrow v \end{array}$$

- (1) $\bullet; \bullet \vdash_e e : \tau$ and $e \Downarrow v$
(2) $\{ \} \vdash_m \text{ret}(e) \Downarrow^1 v$

(ASSUMPTION)
E:RET $\hat{\circ}$ (1)

Case: (S:BND:EXPR)

$$\begin{array}{c} \text{(S:BND:EXPR)} \\ \Delta; \Gamma \vdash_e e : \tau_1 \\ \Delta; \Gamma, \mathbf{x} : \tau_1 \vdash_m m : \tau_2 \# \Sigma \\ \hline \Delta; \Gamma \vdash_m \mathbf{x} = e; m : \tau_2 \# \Sigma \end{array}$$

- (1) $\bullet; \bullet \vdash_e e : \tau_1$
(2) there exists a value v_e s.t. $e \Downarrow v_e$ and $\bullet; \bullet \vdash_e v_e : \tau_1$
(3) $\bullet; \mathbf{x} : \tau_1 \vdash_m m : \tau_2 \# \Sigma$
(4) $\bullet; \bullet \vdash_m m \{v_e/\mathbf{x}\} : \tau_2 \# \Sigma$
(5) $\sigma \vdash_\sigma \Sigma \Downarrow v$
(6) there exists w and v s.t. $\sigma \vdash_m m \{v_e/\mathbf{x}\} \Downarrow^w v$
(7) $\sigma \vdash_m \mathbf{x} = e; m \Downarrow^w v$

(ASSUMPTION)
(PROPOSITION B.2) $\hat{\circ}$ (1)
(ASSUMPTION)
(LEMMA B.5) $\hat{\circ}$ (2)
(ASSUMPTION)
(I.H.) $\hat{\circ}$ (4) $\hat{\circ}$ (5)
E:BND:EXPR $\hat{\circ}$ (2) $\hat{\circ}$ (6)

Case: (S:BND:TERM)

$$\begin{array}{c} \text{(S:BND:TERM)} \\ \Delta; \Gamma \vdash_t t : \tau_1 \# S \quad \Delta, \mathbf{a} : \tau_1; \Gamma \vdash_m m : \tau_2 \# \left\{ \overline{\mathbf{a}' : S'} \right\}_e \\ \hline \Delta; \Gamma \vdash_m \mathbf{a} = t; m : \tau_2 \# \left\{ \mathbf{a} : S, \overline{\mathbf{a}' : S'} \right\}_e \\ \text{(W:TRACE:BND)} \\ s \vdash_s S \Downarrow v_1 \quad \left\{ \overline{\mathbf{a}' : s'} \right\} \vdash_\sigma \left\{ \overline{\mathbf{a}' : S'} \right\}_e \{v_1/\mathbf{a}\} \Downarrow v \\ \hline \left\{ \mathbf{a} : s, \overline{\mathbf{a}' : s'} \right\} \vdash_\sigma \left\{ \mathbf{a} : S, \overline{\mathbf{a}' : S'} \right\}_e \Downarrow v \end{array}$$

- (1) $\bullet; \bullet \vdash_t t : \tau_1 \# S$ and $s \vdash_s S \Downarrow v_1$
(2) there exists w_1 s.t. $s \vdash_t t \Downarrow^{w_1} v_1$
(3) $\mathbf{a} : \tau_1; \bullet \vdash_m m : \tau_2 \# \left\{ \overline{\mathbf{a}' : S'} \right\}_e$
(4) $\bullet; \bullet \vdash_e v_1 : \tau_1$
(5) $\bullet; \bullet \vdash_m m \{v_1/\mathbf{a}\} : \tau_2 \# \left\{ \overline{\mathbf{a}' : S'} \right\}_e \{v_1/\mathbf{a}\}$
(6) $\left\{ \mathbf{a} : s, \overline{\mathbf{a}' : s'} \right\} \vdash_\sigma \left\{ \mathbf{a} : S, \overline{\mathbf{a}' : S'} \right\}_e \Downarrow v$
(7) $\left\{ \overline{\mathbf{a}' : s'} \right\} \vdash_\sigma \left\{ \overline{\mathbf{a}' : S'} \right\}_e \{v_1/\mathbf{a}\} \Downarrow v$
(8) there exists w_2 s.t. $\left\{ \overline{\mathbf{a}' : s'} \right\} \vdash_m m \{v_1/\mathbf{a}\} \Downarrow^{w_2} v_2$
(9) $\left\{ \mathbf{a} : s, \overline{\mathbf{a}' : s'} \right\} \vdash_m \mathbf{a} = t; m \Downarrow^{w_1 \cdot w_2} v_2$

(ASSUMPTION)
(I.H.) $\hat{\circ}$ (1)
(ASSUMPTION)
(THEOREM B.6) $\hat{\circ}$ (1) $\hat{\circ}$ (2)
(LEMMA B.5) $\hat{\circ}$ (3) $\hat{\circ}$ (4)
(ASSUMPTION)
W:TRACE:BND $\hat{\circ}$ (1) $\hat{\circ}$ (6)
(I.H.) $\hat{\circ}$ (5) $\hat{\circ}$ (7)
E:BND:TERM $\hat{\circ}$ (2) $\hat{\circ}$ (8)

Case: (S:OBSERVE)

$$\begin{array}{c} \text{(S:OBSERVE)} \\ \Delta; \Gamma \vdash_e e_1 : \text{dist}(\tau) \\ \Delta; \Gamma \vdash_e e_2 : \tau \quad \Delta; \Gamma \vdash_m m : \tau \# \Sigma \\ \hline \Delta; \Gamma \vdash_m \text{observe}(e_1; e_2); m : \tau \# \Sigma \end{array}$$

- (1) $\bullet; \bullet \vdash_m m : \tau \# \Sigma$ and $\sigma \vdash_\sigma \Sigma \Downarrow v$

(ASSUMPTION)

- (2) there exists w_2 s.t. $\sigma \vdash_t m \Downarrow^{w_2} v$ (I.H.) \hat{c} (1)
- (3) $\bullet; \bullet \vdash_e e_1 : \text{dist}(\tau)$ and $\bullet; \bullet \vdash_e e_2 : \tau$ (ASSUMPTION)
- (4) $e_1 \Downarrow d, e_2 \Downarrow v$ and $\bullet; \bullet \vdash_e d : \text{dist}(\tau)$ and $\bullet; \bullet \vdash_e v : \tau$ (PROPOSITION B.2) \hat{c} (3)
- (5) $v \in d.\text{support}$ and $w_1 = d.\text{density}(v)$ and $w_1 > 0$ (PROPOSITION B.1) \hat{c} (4)
- (6) $\sigma \vdash_m \text{observe}(e_1; e_2); m \Downarrow^{w_1 \cdot w_2} v$ E:OBSERVE \hat{c} (2) \hat{c} (4) \hat{c} (5)

Case: (S:CMD)

$$\frac{\text{(S:CMD)} \quad \Delta; \Gamma \vdash_m m : \tau \# \Sigma}{\Delta; \Gamma \vdash_t m : \tau \# \text{trace } \Sigma} \quad \frac{\text{(W:EVENT:TRACE)} \quad \sigma \vdash_\sigma \Sigma \Downarrow v}{\text{trace } \sigma \vdash_s \text{trace } \Sigma \Downarrow v}$$

- (1) $\bullet; \bullet \vdash_m m : \tau \# \Sigma$ and $\sigma \vdash_\sigma \Sigma \Downarrow v$ (ASSUMPTION)
- (2) there exists w s.t. $\sigma \vdash_t m \Downarrow^w v$ (I.H.) \hat{c} (1)
- (3) $\text{trace } \sigma \vdash_t m \Downarrow^w v$ E:CMD \hat{c} (2)

Case: (S:SAMPLE)

$$\frac{\text{(S:SAMPLE)} \quad \Delta; \Gamma \vdash_e e : \text{dist}(\tau)}{\Delta; \Gamma \vdash_t \text{sample}(e) : \tau \# \text{atom } \tau} \quad \frac{\text{(W:EVENT:ATOM)} \quad v : \tau}{\text{atom } v \vdash_s \text{atom } \tau \Downarrow v}$$

- (1) $\bullet; \bullet \vdash_e e : \text{dist}(\tau)$ (ASSUMPTION)
- (2) $e \Downarrow d$ (PROPOSITION B.2) \hat{c} (1)
- (3) $\bullet; \bullet \vdash_e d : \text{dist}(\tau)$ (THEOREM 4.3) \hat{c} (1) \hat{c} (2)
- (4) $\bullet; \bullet \vdash_e v : \tau$ (ASSUMPTION)
- (5) $v \in d.\text{support}$ and $w = d.\text{density}(v)$ and $w > 0$ (PROPOSITION B.1) \hat{c} (3) \hat{c} (4)
- (6) $\text{atom } v \vdash_t \text{sample}(e) \Downarrow^w v$ E:SAMPLE \hat{c} (2) \hat{c} (5)

Case: (S:BRANCH)

$$\frac{\text{(S:BRANCH)} \quad \Delta; \bullet \vdash_e e : \mathcal{2} \quad \Delta; \Gamma \vdash_m m_1 : \tau \# \Sigma_1 \quad \Delta; \Gamma \vdash_m m_2 : \tau \# \Sigma_2}{\Delta; \Gamma \vdash_t \text{ite}(e; m_1; m_2) : \tau \# \Sigma_1 +_e \Sigma_2} \quad \frac{\text{(W:EVENT:BRANCH)} \quad e \Downarrow b \quad \sigma \vdash_\sigma \Sigma_b \Downarrow v}{\text{inj } \sigma \vdash_s \Sigma_{\text{true}} +_e \Sigma_{\text{false}} \Downarrow v}$$

- (1) $\bullet; \bullet \vdash_m m_i : \tau \# \Sigma_i$ and $\sigma \vdash_\sigma \Sigma_i \Downarrow v$ where $i = \text{ite}(v_b, 1, 2)$ (ASSUMPTION)
- (2) $e \Downarrow v_b$ (ASSUMPTION)
- (3) there exists w s.t. $\sigma \vdash_m m_i \Downarrow^w v$ (I.H.) \hat{c} (1)
- (4) $\text{inj } \sigma \vdash_t \text{ite}(e; m_1; m_2) \Downarrow^w v$ E:BRANCH \hat{c} (2) \hat{c} (3)

Case: (S:CALL)

$$\frac{\text{(S:CALL)} \quad \mathbf{f} : \langle \tau_1 \rangle (\tau_2) \rightsquigarrow \tau_3 \# \mathbf{F} \quad \Delta; \bullet \vdash_e e_1 : \tau_1 \quad \Delta; \Gamma \vdash_e e_2 : \tau_2}{\Delta; \Gamma \vdash_t \text{call}(\mathbf{f}; e_1; e_2) : \tau_3 \# \mathbf{F}\langle e_1 \rangle} \quad \frac{\text{(W:EVENT:CALL)} \quad \text{typedef } \mathbf{F} = \forall \mathbf{a} : \tau_1. \Sigma \quad e \Downarrow v_a \quad \sigma \vdash_s \Sigma \{v_a/\mathbf{a}\} \Downarrow v}{\text{fold } \sigma \vdash_s \mathbf{F}\langle e \rangle \Downarrow v}$$

$$\frac{\text{(S:DEF)} \quad \mathbf{f} : \langle \tau_1 \rangle (\tau_2) \rightsquigarrow \tau_3 \# \mathbf{F} \quad \text{typedef } \mathbf{F} = \forall \mathbf{a} : \tau_1. \Sigma \quad \mathbf{a} : \tau_1; \mathbf{x} : \tau_2 \vdash_m m : \tau_3 \# \Sigma}{\vdash_{\mathcal{F}} \text{def } \mathbf{f}\langle \mathbf{a} \rangle (\mathbf{x}) = m : \langle \tau_1 \rangle (\tau_2) \rightsquigarrow \tau_3 \# \mathbf{F}}$$

- (1) $\bullet; \bullet \vdash_e e_2 : \tau_2$ (ASSUMPTION)
- (2) there exists a value v_x s.t. $e_2 \Downarrow v_x$ (PROPOSITION B.2) \hat{c} (1)
- (3) $\bullet; \bullet \vdash_e e_1 : \tau_1$ and $e_1 \Downarrow v_a$ (ASSUMPTION)
- (4) $\bullet; \bullet \vdash_e v_a : \tau_1$ and $\bullet; \bullet \vdash_e v_x : \tau_2$ PROPOSITION B.2 \hat{c} (1) \hat{c} (2) \hat{c} (3)
- (5) $\mathbf{a} : \tau_1; \mathbf{x} : \tau_2 \vdash_m m : \tau_3 \# \Sigma$ (ASSUMPTION)
- (6) $\bullet; \bullet \vdash_m m \{v_a/\mathbf{a}\} \{v_x/\mathbf{x}\} : \tau_3 \# \Sigma \{v/\mathbf{a}\}$ LEMMA B.5 \hat{c} (4)
- (7) $\sigma \vdash_\sigma \Sigma \{v/\mathbf{a}\} \Downarrow v$ (ASSUMPTION)
- (8) there exists w s.t. $\sigma \vdash_t m \{v_a/\mathbf{a}\} \{v_x/\mathbf{x}\} \Downarrow^w v$ (I.H.) \hat{c} (6) \hat{c} (7)
- (9) $\text{typedef } \mathbf{F} = \forall \mathbf{a} : \tau_1. \Sigma$ (ASSUMPTION)

$$(10) \text{ fold } \sigma \vdash_t \text{ call}(\mathbf{f}; e_1; e_2) \Downarrow^w v \quad \text{E:CALL } \dot{\Leftarrow} (2) \dot{\Leftarrow} (3) \dot{\Leftarrow} (8) \dot{\Leftarrow} (9) \quad \square$$

LEMMA B.8. *If $\bullet; \bullet \vdash_m m_1 : \tau \# \Sigma$ and $\bullet; \bullet \vdash_m m_2 : \tau \# \Sigma$, then it holds that $\mathbf{P}_{m_1}(\sigma) \neq 0$ if and only if $\mathbf{P}_{m_2}(\sigma) \neq 0$.*

PROOF. The “only if” direction mirrors the “if” direction. Below we show the “if” direction.

- | | | |
|-----|--------------------------------------------------|--------------------------------------------------------------------|
| (1) | $\mathbf{P}_{m_1}(\sigma) > 0$ | ASSUMPTION |
| (2) | $\bullet; \bullet \vdash_m m_1 : \tau \# \Sigma$ | ASSUMPTION |
| (3) | $\bullet; \bullet \vdash_m m_2 : \tau \# \Sigma$ | ASSUMPTION |
| (4) | $\sigma \vdash_m m_1 \Downarrow^{w_1} v$ | DEFINITION A.1 $\dot{\Leftarrow}$ (1) |
| (5) | $\sigma \vdash_\sigma \Sigma \Downarrow v$ | THEOREM B.6 $\dot{\Leftarrow}$ (2) $\dot{\Leftarrow}$ (4) |
| (6) | $\sigma \vdash_m m_2 \Downarrow^{w_2} v$ | THEOREM B.7 $\dot{\Leftarrow}$ (3) $\dot{\Leftarrow}$ (5) |
| (7) | $\mathbf{P}_{m_2}(\sigma) = w_2 > 0$ | DEFINITION A.1 $\dot{\Leftarrow}$ LEMMA B.3 $\dot{\Leftarrow}$ (6) |
-

THEOREM B.9 (RESTATEMENT OF THEOREM 4.5). *If $\bullet; \bullet \vdash_m m_1 : \tau \# \Sigma$ and $\bullet; \bullet \vdash_m m_2 : \tau \# \Sigma$, then for any measurable set A of traces, $\llbracket m_1 \rrbracket(A) \neq 0$ if and only if $\llbracket m_2 \rrbracket(A) \neq 0$.*

PROOF. The proof uses the fact that a Lebesgue integral is zero if and only if the integrand is zero almost everywhere.

$$\begin{aligned} \llbracket m_2 \rrbracket(A) &= \int_E \mathbf{P}_{m_2}(\sigma) d\sigma = 0 \\ \iff \lambda_\sigma(\{\sigma \mid \mathbf{P}_{m_2}(\sigma) \cdot \mathbf{1}_E(\sigma) > 0\}) &= 0 \\ \iff \lambda_\sigma(\{\sigma \mid \mathbf{P}_{m_1}(\sigma) \cdot \mathbf{1}_E(\sigma) > 0\}) &= 0 \quad \text{Lemma B.8} \\ \iff \llbracket m_1 \rrbracket(A) = \int_E \mathbf{P}_{m_1}(\sigma) d\sigma &= 0 \end{aligned}$$

□

B.2 Theorems in Section 6

Define functions $\text{CPFV}_m(\cdot)$ and $\text{CPFV}_t(\cdot)$ that compute free term variables in checkpoints within a command or a term:

$$\begin{aligned} \text{CPFV}_m(\text{ret}(e)) &\stackrel{\text{def}}{=} \text{FV}(e) \\ \text{CPFV}_m(x = e; m) &\stackrel{\text{def}}{=} \text{CPFV}_m(m) \\ \text{CPFV}_m(\mathbf{a} = t; m) &\stackrel{\text{def}}{=} \text{CPFV}_t(t) \cup \text{CPFV}_m(m) \\ \text{CPFV}_m(\text{observe}(e_1; e_2); m) &\stackrel{\text{def}}{=} \text{CPFV}_m(m) \\ \text{CPFV}_t(\text{sample}(e)) &\stackrel{\text{def}}{=} \emptyset \\ \text{CPFV}_t(\text{ite}(e; m_1; m_2)) &\stackrel{\text{def}}{=} \text{FV}(e) \cup \text{CPFV}_m(m_1) \cup \text{CPFV}_m(m_2) \\ \text{CPFV}_t(\text{call}(\mathbf{f}; e)) &\stackrel{\text{def}}{=} \text{FV}(e) \\ \text{CPFV}_t(m) &\stackrel{\text{def}}{=} \text{CPFV}_m(m) \end{aligned}$$

LEMMA B.10.

- (1) *If $\mathcal{G}; \mathbf{C}; \Delta'; \mathbf{h} : \mathbb{R} \vdash_m \langle \Delta; \Gamma \vdash_m m : \tau \# \Sigma \rangle \rightsquigarrow m'$ and $\text{CPFV}_m(m) \subseteq \text{domain}(\Delta')$, then $\Delta'; \mathbf{h} : \mathbb{R} \vdash_m m' : \tau \# \Sigma$.*
- (2) *If $\mathcal{G}; \mathbf{C}; \Delta'; \mathbf{h} : \mathbb{R}; \mathbf{a} \vdash_t \langle \Delta; \Gamma \vdash_t t : \tau \# S \rangle \rightsquigarrow t'$ and $\text{CPFV}_t(t) \subseteq \text{domain}(\Delta')$, then $\Delta'; \mathbf{h} : \mathbb{R} \vdash_t t' : \tau \# S$.*

PROOF. By mutual structural induction on derivations of command translation and term translation. Below we show the most interesting case, namely the translation of a command.

Case: GG:CMD

(GG:CMD)

$$\frac{\text{TermBindings}(m) = (a_i = t_i)_{i=1}^N \quad (a_{l_i} = t_{l_i})_{i=1}^N \text{ reorders}_{\mathcal{G}} (a_i = t_i)_{i=1}^N}{\forall i. \mathcal{G}; C; \Delta', a_{l_i} : \tau_{l_i}, \dots, a_{l_{i-1}} : \tau_{l_{i-1}}; h : \mathbb{R}; a_{l_i} \vdash_t \langle \Delta_{l_i}; \Gamma_{l_i} \vdash_t t_{l_i} : \tau_{l_i} \# S_{l_i} \rangle \rightsquigarrow t'_{l_i}}{\mathcal{G}; C; \Delta'; h : \mathbb{R} \vdash_m \langle \Delta; \Gamma \vdash_m m : \tau \# \{a_1 : S_1, \dots, a_N : S_N\}_e \rangle \rightsquigarrow a_{l_1} = t'_{l_1}; \dots; a_{l_N} = t'_{l_N}; \text{ret}(e)}$$

- (1) $\Delta; \Gamma \vdash_m m : \tau \# \{a_1 : S_1, \dots, a_N : S_N\}_e$ ASSUMPTION
- (2) $(a_{l_i} = t_{l_i})_{i=1}^N \text{ reorders}_{\mathcal{G}} (a_i = t_i)_{i=1}^N$ ASSUMPTION
- (3) $\forall i. \Delta_i; \Gamma_i \vdash_t t_i : \tau_i \# S_i$ (1) $\hat{\circ}$ S:BND:TERM
- (4) $\forall i. \Delta_{l_i}; \Gamma_{l_i} \vdash_t t_{l_i} : \tau_{l_i} \# S_{l_i}$ (2) $\hat{\circ}$ (3)
- (5) $\Delta, a_1 : \tau_1, \dots, a_N : \tau_N; \bullet \vdash_e e : \tau$ (1) $\hat{\circ}$ S:RETURN
- (6) $\Delta, a_{l_1} : \tau_{l_1}, \dots, a_{l_N} : \tau_{l_N}; \bullet \vdash_e e : \tau$ (2) $\hat{\circ}$ (5)
- (7) $\forall i. \text{CPFV}_i(t_{l_i}) \subseteq \text{domain}(\Delta') \cup \{a_{l_1}, \dots, a_{l_{i-1}}\}$ (2) $\hat{\circ}$ REORDER
- (8) $\forall i. \Delta', a_{l_1} : \tau_{l_1}, \dots, a_{l_{i-1}} : \tau_{l_{i-1}}; h : \mathbb{R} \vdash_t t'_{l_i} : \tau_{l_i} \# S_{l_i}$ I.H. $\hat{\circ}$ (4) $\hat{\circ}$ (7)
- (9) $\Delta'; h : \mathbb{R} \vdash_m a_{l_1} = t'_{l_1}; \dots; a_{l_N} = t'_{l_N}; \text{ret}(e) : \tau \# \{a_{l_1} : S_{l_1}, \dots, a_{l_N} : S_{l_N}\}_e$ (6) $\hat{\circ}$ S:RETURN $\hat{\circ}$ (8) $\hat{\circ}$ S:BND:TERM
- (10) $(l_i)_{i=1}^N$ is a permutation of $(i)_{i=1}^N$ INVERSION OF (2)
- (11) $\{a_{l_1} : S_{l_1}, \dots, a_{l_N} : S_{l_N}\}_e = \{a_1 : S_1, \dots, a_N : S_N\}_e$ (10)
- (12) $\Delta'; h : \mathbb{R} \vdash_m a_{l_1} = t'_{l_1}; \dots; a_{l_N} = t'_{l_N}; \text{ret}(e) : \tau \# \{a_1 : S_1, \dots, a_N : S_N\}_e$ (9) $\hat{\circ}$ (11)

□

THEOREM B.11 (RESTATEMENT OF THEOREM 6.1).

If $\mathcal{G}; \{\overline{\text{ENTRY}}\}; \bullet; \text{obs} : \mathbb{R} \vdash_m \langle \bullet; \text{obs} : \mathbb{R} \vdash_m m_m : \tau \# \Sigma \rangle \rightsquigarrow m_g$, then $\bullet; \text{obs} : \mathbb{R} \vdash_m m_g : \tau \# \Sigma$.

PROOF. By Lemma B.10 and by command m being well-typed in an empty term-variable environment. □

THEOREM B.12 (RESTATEMENT OF THEOREM 6.2).

If $\mathcal{G}; \{\overline{\text{ENTRY}}\}; \bullet; \text{obs} : \mathbb{R} \vdash_m \langle \bullet; \text{obs} : \mathbb{R} \vdash_m m_m : \tau \# \Sigma \rangle \rightsquigarrow m_g$ and $v_{\text{obs}} : \mathbb{R}$, then for any measurable set A of traces, $\llbracket m_m \{v_{\text{obs}}/\text{obs}\} \rrbracket (A) \neq 0$ if and only if $\llbracket m_g \{v_{\text{obs}}/\text{obs}\} \rrbracket (A) \neq 0$.

PROOF. By Theorems B.9 and B.11. □

C DATAFLOW GRAPHS FOR CAPTCHA MODEL AND GUIDES

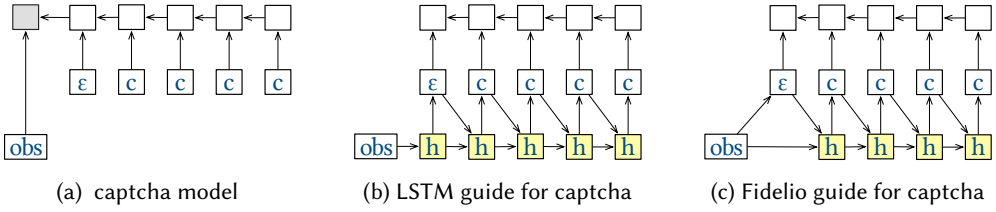


Figure 27. Dataflow for a trace of the captcha model and two guides. In the model (a), RVs c (captcha characters) are all correlated with each other. As a result, the model contains no conditional independence for Fidelio to exploit.

D EXPERIMENTAL SETUPS AND SUPPLEMENTAL STATISTICS

| name | structure | #RV | #LOC (model) | #iteration | guide | capacity | validation loss |
|------------------|---------------------|------------|--------------|------------|------------------------|-----------|-----------------|
| longrange (k=5) | loop | 8 | 16 | 10000 | Fidelia | 1398 | 21.87529387 |
| | | | | | LSTM | 1382 | 24.49491011 |
| | | | | | mean-field | 1386 | 25.38885483 |
| longrange (k=10) | loop | 13 | 16 | 10000 | Fidelia | 1398 | 40.37852367 |
| | | | | | LSTM | 1382 | 43.68544903 |
| | | | | | mean-field | 1386 | 44.04598006 |
| longrange (k=20) | loop | 23 | 16 | 10000 | Fidelia | 1398 | 77.59334972 |
| | | | | | LSTM | 1382 | 81.31816754 |
| | | | | | mean-field | 1386 | 81.24487837 |
| longrange (k=40) | loop | 43 | 16 | 10000 | Fidelia | 1398 | 152.1430431 |
| | | | | | LSTM | 1382 | 156.0312375 |
| | | | | | mean-field | 1386 | 155.7906566 |
| treebn (h=3) | BN | 15 | 20 | 5000 | sparse guide (Fidelia) | 994 | 18.77669632 |
| | | | | | dense guide (Fidelia) | 982 | 20.46032267 |
| | | | | | LSTM | 996 | 20.82039981 |
| | | | | | mean-field | 1078 | 22.88487139 |
| | | | | 5000 | sparse guide (Fidelia) | 1540 | 18.74158391 |
| | | | | | dense guide (Fidelia) | 1533 | 18.99891774 |
| | | | | | LSTM | 1588 | 20.48320966 |
| | | | | | mean-field | 1484 | 22.75920135 |
| | | | | 5000 | sparse guide (Fidelia) | 2023 | 18.72428359 |
| | | | | | dense guide (Fidelia) | 2047 | 19.00601651 |
| | | | | | LSTM | 2116 | 19.97572785 |
| | | | | | mean-field | 2128 | 22.75885654 |
| treebn (h=4) | BN | 31 | 37 | 5000 | sparse guide (Fidelia) | 2055 | 41.06646845 |
| | | | | | dense guide (Fidelia) | 2154 | 43.5052426 |
| | | | | | LSTM | 2196 | 45.76123151 |
| | | | | | mean-field | 2236 | 51.87590573 |
| | | | | 5000 | sparse guide (Fidelia) | 3642 | 41.04653035 |
| | | | | | dense guide (Fidelia) | 3594 | 43.11532713 |
| | | | | | LSTM | 3552 | 44.85451233 |
| | | | | | mean-field | 3498 | 52.33009364 |
| | | | | 5000 | sparse guide (Fidelia) | 5196 | 41.04030352 |
| | | | | | dense guide (Fidelia) | 5298 | 43.11325543 |
| | | | | | LSTM | 5498 | 43.95748757 |
| | | | | | mean-field | 5412 | 51.88407577 |
| PCFG-Captcha | branching recursion | stochastic | 237 | 10000 | Fidelia | 52 838 | 7.536001405 |
| | | | | | LSTM | 50 159 | 14.76856677 |
| | | | | | mean-field | 52 105 | 10.29323038 |
| | | | | 10000 | Fidelia | 104 951 | 3.6027033 |
| | | | | | LSTM | 102 437 | 14.83619706 |
| | | | | | mean-field | 100 217 | 7.961722895 |
| | | | | 10000 | Fidelia | 254 555 | 2.241801572 |
| | | | | | LSTM | 250 209 | 14.78410727 |
| | | | | | mean-field | 250 687 | 8.149919943 |
| | | | | 10000 | Fidelia | 500 603 | 2.918338221 |
| | | | | | LSTM | 500 049 | 14.44708732 |
| | | | | | mean-field | 501 999 | 10.2873026 |
| | | | | 10000 | Fidelia | 2 000 177 | 2.511201136 |
| | | | | | LSTM | 2 000 705 | 9.530939491 |
| | | | | | mean-field | 2 000 321 | 8.379732159 |

Table 3 continued from previous page

| name | structure | #RV | #LOC (model) | #iteration | guide | capacity | validation loss |
|------------|---------------------|-------------|--------------|------------|------------|----------|---------------------|
| ECOLI70 | BN | 46 | 54 | 2000 | Fidelia | 1154 | 20.47216836 |
| | | | | | LSTM | 1112 | 19.46052641 |
| | | | | | mean-field | 1496 | 25.98369968 |
| | | | | 2000 | Fidelia | 5469 | 13.36186348 |
| | | | | | LSTM | 5042 | 15.65118018 |
| | | | | | mean-field | 5456 | 17.55795286 |
| | | | | 2000 | Fidelia | 10505 | 11.89387623 |
| | | | | | LSTM | 10334 | 15.28437446 |
| | | | | | mean-field | 10824 | 16.56617211 |
| | | | | 2000 | Fidelia | 15206 | 11.93564912 |
| | | | | | LSTM | 15074 | 15.31487038 |
| | | | | | mean-field | 15774 | 16.6019876 |
| | | | | 2000 | Fidelia | 50591 | 11.82502852 |
| | | | | | LSTM | 50774 | 15.26491375 |
| | | | | | mean-field | 52404 | 16.76144694 |
| 2000 | Fidelia | 101481 | 12.03494057 | | | | |
| | LSTM | 100004 | 14.9902177 | | | | |
| | mean-field | 100166 | 16.42537214 | | | | |
| 2000 | Fidelia | 153566 | 12.58098554 | | | | |
| | LSTM | 151052 | 15.17583296 | | | | |
| | mean-field | 152196 | 17.51372583 | | | | |
| captcha | branching recursion | 6 | 43 | 200000 | Fidelia | 4416271 | -1.21193114347507 |
| | | | | | LSTM | 4416081 | -1.3246571659603206 |
| | | | | | mean-field | 4423701 | 1.6435429854854278 |
| Astro-PCFG | branching recursion | stochastic | 159 | 20000 | Fidelia | 25028 | 2.45345686 |
| | | | | | LSTM | 25006 | 2.570013847 |
| | | | | 20000 | Fidelia | 50038 | 1.83395378 |
| | | | | | LSTM | 50022 | 2.470131403 |
| | | | | 20000 | Fidelia | 74986 | 1.792654934 |
| | | | | | LSTM | 75035 | 2.356314919 |
| | | | | 20000 | Fidelia | 100023 | 1.979825721 |
| | | | | | LSTM | 100037 | 2.351797449 |
| | | | | 20000 | Fidelia | 125018 | 1.785649728 |
| | | | | | LSTM | 124972 | 2.261098404 |
| | | | | 20000 | Fidelia | 150002 | 2.035392191 |
| | | | | | LSTM | 149928 | 2.361578974 |
| | | | | 20000 | Fidelia | 175040 | 1.605737646 |
| | | | | | LSTM | 175048 | 2.338465666 |
| | | | | 20000 | Fidelia | 199992 | 1.591854952 |
| LSTM | 200009 | 2.297783281 | | | | | |
| 20000 | Fidelia | 225010 | 1.709513105 | | | | |
| | LSTM | 225078 | 2.241109376 | | | | |
| 20000 | Fidelia | 250013 | 2.113929184 | | | | |
| | LSTM | 250078 | 2.386709259 | | | | |
| 20000 | mean-field | 175094 | 6.303524799 | | | | |
| gmm | branching recursion | 116 | 49 | 5000 | Fidelia | 20332 | 16.21922317 |
| | | | | | LSTM | 19794 | 257.8196954 |
| | | | | | mean-field | 20454 | 111.1146122 |
| | | | | 5000 | Fidelia | 49872 | 15.15900177 |
| | | | | | LSTM | 49782 | 170.0629188 |
| | | | | | mean-field | 50262 | 111.164391 |
| 5000 | Fidelia | 80222 | 14.55081707 | | | | |
| | LSTM | 80184 | 152.3689971 | | | | |
| | mean-field | 81018 | 111.0644756 | | | | |

Table 4. Table 2, with standard deviations.

| Program | Structures | #RVs | LoC (model) | Capacity | Validation Loss | | |
|------------------------|------------|-------|----------------|--------------|-------------------|--------------------|-------------------|
| | | | | | Mean-Field | LSTM | Fidelio |
| treebn | BN | 31 | 37 | 2K | 51.88 ± 0.17 | 45.76 ± 1.90 | 41.07 ± 0.14 |
| | | | | 5K | 51.88 ± 0.17 | 43.96 ± 1.64 | 41.04 ± 0.11 |
| ecoli70 | BN | 46 | 54 | 10K | 11.89 ± 0.13 | 15.28 ± 0.33 | 16.56 ± 0.21 |
| | | | | 15K | 11.94 ± 0.13 | 15.31 ± 0.29 | 16.60 ± 0.17 |
| longrange ($k = 5$) | loop | 8 | 16 | 1.4K | 25.39 ± 0.00 | 24.49 ± 0.77 | 21.88 ± 0.06 |
| longrange ($k = 10$) | loop | 13 | 16 | 1.4K | 44.05 ± 0.00 | 43.69 ± 0.43 | 40.38 ± 0.06 |
| longrange ($k = 20$) | loop | 23 | 16 | 1.4K | 81.23 ± 0.01 | 81.32 ± 0.13 | 77.59 ± 0.05 |
| longrange ($k = 40$) | loop | 43 | 16 | 1.4K | 155.79 ± 0.01 | 156.02 ± 0.09 | 152.12 ± 0.06 |
| captcha | recursion | 6 | 43 | 4.4M | 1.644 ± 0.02 | -1.325 ± 0.14 | -1.212 ± 0.13 |
| astropcfg | recursion | 15.0 | 159 | 100K | 6.23 ± 0.02 | 2.35 ± 0.05 | 1.98 ± 0.24 |
| | | | | 175K | 6.22 ± 0.00 | 2.36 ± 0.08 | 2.04 ± 0.02 |
| | | | | 250K | 6.22 ± 0.00 | 2.39 ± 0.04 | 2.11 ± 0.25 |
| mathcaptcha | recursion | 14.5 | 237 | 100K+1M(CNN) | 7.96 ± 0.46 | 14.84 ± 0.21 | 3.60 ± 1.58 |
| | | | | 250K+1M(CNN) | 8.15 ± 0.37 | 14.78 ± 0.32 | 2.24 ± 0.57 |
| | | | | 500K+1M(CNN) | 8.33 ± 0.68 | 14.80 ± 0.21 | 1.95 ± 0.39 |
| | | | | 2M+1M(CNN) | 8.39 ± 0.60 | 9.53 ± 5.32 | 2.51 ± 0.43 |
| gmmcc | recursion | 217.4 | 49 | 20K | 111.11 ± 0.15 | 257.82 ± 46.72 | 16.22 ± 0.63 |
| | | | | 50K | 111.16 ± 0.34 | 170.06 ± 12.03 | 15.16 ± 0.79 |
| | | | | 80K | 111.06 ± 0.32 | 152.37 ± 6.37 | 14.55 ± 0.79 |

Table 5. Table in Figure 17, with standard deviations.

| k | ESS | | |
|-----|-----------------|-----------------|------------------|
| | M.-F. | LSTM | Fidelio |
| 5 | 1.53 ± 0.69 | 3.08 ± 2.44 | 50.45 ± 5.56 |
| 10 | 1.00 ± 0.00 | 2.21 ± 1.62 | 41.88 ± 7.76 |
| 20 | 2.19 ± 0.54 | 1.54 ± 0.69 | 55.99 ± 6.55 |
| 40 | 2.66 ± 0.73 | 1.19 ± 0.24 | 45.58 ± 5.75 |

Table 6. Table in Figure 18, with standard deviations.

| Capacity | Effective Sample Size | | | |
|----------|-----------------------|-----------------|---------------------|---------------------|
| | M.-F. | LSTM | Fidelio Fig. 16b | Fidelio Fig. 16c |
| 2K | 1.39 ± 0.56 | 3.13 ± 2.02 | 8.67 ± 4.22 | 82.81 ± 9.56 |
| 5K | 1.38 ± 0.57 | 9.84 ± 6.53 | 9.51 ± 3.41 | 84.13 ± 3.63 |

Table 7. Table in Figure 19, with standard deviations.

| Capacity | Regeneration Rate | | | Levenshtein Distance | | |
|----------|----------------------|----------------------|----------------------|----------------------|-----------------|-----------------|
| | M.-F. | LSTM | Fidelio | M.-F. | LSTM | Fidelio |
| 100K | $40.00\% \pm 0.45\%$ | $57.37\% \pm 2.46\%$ | $71.67\% \pm 1.89\%$ | 4.36 ± 0.03 | 2.99 ± 0.05 | 2.72 ± 0.17 |
| 175K | $39.83\% \pm 0.31\%$ | $60.83\% \pm 0.73\%$ | $71.32\% \pm 1.14\%$ | 4.41 ± 0.01 | 3.08 ± 0.10 | 2.63 ± 0.07 |
| 250K | $39.17\% \pm 0.12\%$ | $62.42\% \pm 4.16\%$ | $69.83\% \pm 1.03\%$ | 4.36 ± 0.03 | 3.04 ± 0.07 | 2.75 ± 0.01 |

Table 8. Table in Figure 20, with standard deviations.

| Capacity | Recognition Rate | | | Levenshtein Distance | | |
|--------------|------------------|----------------|-----------------------|----------------------|-------------|--------------------|
| | M.-F. | LSTM | Fidelio | M.-F. | LSTM | Fidelio |
| 100K+1M(CNN) | 35.81% ± 2.66% | 8.94% ± 0.96% | 70.49% ± 16.52% | 2.50 ± 0.17 | 4.25 ± 0.11 | 0.81 ± 0.35 |
| 250K+1M(CNN) | 36.75% ± 1.99% | 9.82% ± 0.73% | 79.46% ± 5.00% | 2.45 ± 0.11 | 4.21 ± 0.09 | 0.58 ± 0.20 |
| 500K+1M(CNN) | 35.26% ± 3.42% | 9.67% ± 0.59% | 83.09% ± 3.02% | 2.59 ± 0.257 | 4.22 ± 0.12 | 0.46 ± 0.10 |
| 2M+1M(CNN) | 34.85% ± 3.72% | 39.35 ± 29.93% | 77.56% ± 3.63% | 2.62 ± 0.27 | 2.66 ± 1.56 | 0.65 ± 0.14 |

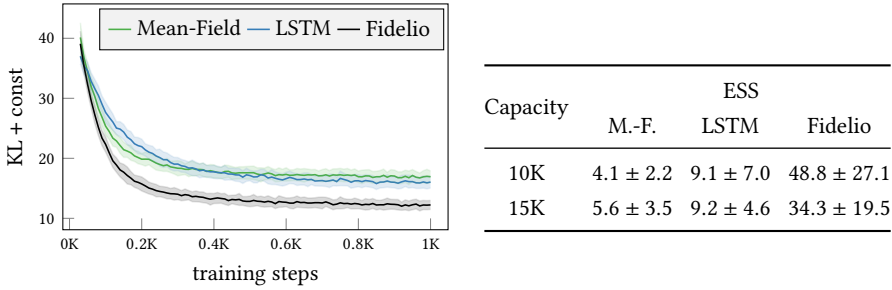


Figure 28. Experimental results for **ecoli70**. Left: Training steps vs. training loss profile, with all guides having capacity 10K. Right: Quality of IS samples as measured by ESS.

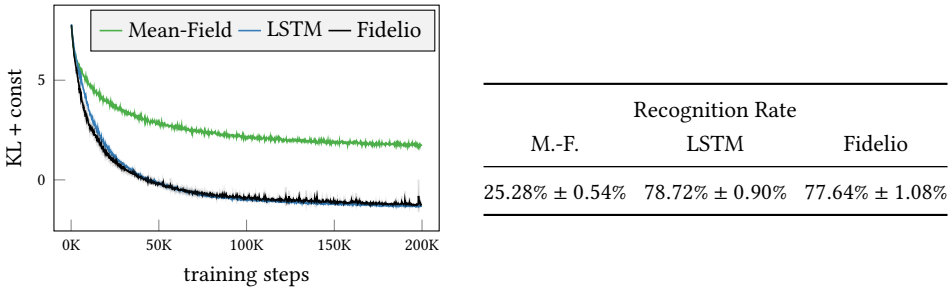


Figure 29. Experimental results for **captcha**. All guides have capacity 4.4M.