Technical Report CS-2021-02, School of Computer Science, University of Waterloo

Semantics and Contextual Equivalence for Probabilistic Programs with Nested Queries and Recursion

YIZHOU ZHANG, University of Waterloo, Canada
NADA AMIN, Harvard University, USA

Metareasoning can be achieved in probabilistic programming languages (PPLs) using agent models that *recursively nest* inference queries inside inference queries. However, the semantics of this powerful, reflection-like language feature has defied an operational treatment, much less reasoning principles for contextual equivalence.

We give formal semantics to a core PPL with continuous distributions, scoring, general recursion, and nested queries. Unlike prior work, the presence of nested queries and general recursion makes it impossible to stratify the definition of a sampling-based operational semantics and that of a measure-theoretic semantics—the two semantics must be defined mutually recursively. A key yet challenging property we establish is that probabilistic programs have well-defined meanings: limits exist for the step-indexed measures they induce.

Beyond a semantics, we offer relational reasoning principles for probabilistic programs making nested queries. We construct a step-indexed, biorthogonal logical-relations model. A soundness theorem establishes that logical relatedness implies contextual equivalence. We demonstrate the usefulness of the reasoning principles by proving novel equivalences of practical relevance—in particular, game-playing and decision-making agents. We mechanize our technical developments leading to the soundness proof using the Coq proof assistant. Nested queries are an important yet theoretically underdeveloped linguistic feature in PPLs; we are first to give them semantics in the presence of general recursion and to provide them with sound reasoning principles for contextual equivalence.

1 INTRODUCTION

Probabilistic programming languages (PPLs) are, in essence, programming languages that support making random choices (i.e., sampling) and scoring random choices (i.e., conditioning).

PPLs have been used to model and solve planning and decision-making problems. Intelligent agents are modeled as probabilistic programs that make random choices and condition on those choices leading to desired outcomes. Their intelligent behavior arises from applying Bayesian inference (also referred to as *queries* in PPLs) to the probabilistic programs. Therefore, in broad strokes, querying a probabilistic program gives rise to the ability to reason.

Nested queries. Querying a program that further makes queries—hence *nested queries*—gives rise to the ability to reason about reasoning.

Stuhlmüller and Goodman [2014] present applications of nested queries in metareasoning. A first example they give is a coordination game of the kind discussed by Schelling [1980]: two agents, Alice and Bob, want to meet up in a café but have no way to contact each other; however, they know each other's preference about the two cafés in town. The probabilistic program in Figure 1 models this problem using two mutually recursive functions, *Alice* and *Bob*. Agents' preferences are expressed by invoking function *SampleLoc* with their prior biases.

Alice reasons as follows: she samples a location from the prior (line 7), reasons about where Bob would like to go (line 8), and conditions her choice on Bob's choosing the same location (line 9). This conditioning is done in *ScoreLoc*, which assigns a zero score when agents choose different locations. Bob reasons in a similar way.

Notice that *Alice* and *Bob* are mutually recursive, with the recursive calls made inside queries (lines 8 and 13). On line 13, Bob uses the expression **query** *Alice* (*depth*) to infer a distribution over Alice's choices. The top-level *Main* function (line 16) makes an outermost (hence non-nested) query, unleashing the nested reasoning "Alice thinks that Bob thinks that Alice thinks that ...".

```
# Sample a location from the prior
def SampleLoc (bias) =
  if sample Bernoulli(bias)
  then "Café A" else "Café B"

# Condition on the locations coinciding
def ScoreLoc (loc1, loc2) =
  if loc1 = loc2 then score 1 else score 0

# Outermost query, inferring Alice's choice
def Main () = sample (query Alice (8))

# Agents make nested queries to reason about each other
def Alice (depth) =
  let aliceLoc = SampleLoc (0.8) in
  let bobLoc = sample (query Bob (depth − 1)) in
  let _ = ScoreLoc (aliceLoc, bobLoc) in aliceLoc

def Bob (depth) =
  let bobLoc = SampleLoc (0.45) in
  if depth > 0 then
    let aliceLoc = sample (query Alice (depth)) in
    let _ = ScoreLoc (aliceLoc, bobLoc) in bobLoc
  else bobLoc
```

Figure 1. Schelling's coordination game via recursively nested queries [Stuhlmüller and Goodman 2014]

The *depth* parameter bounds the depth of recursion, echoing that agents have bounded rationality. Because Alice prefers Café A strongly (line 7) while Bob is roughly indifferent (line 11), the result of inference will converge on them choosing Café A as the depth of recursion increases.

More uses of recursively nested queries abound [Evans et al. 2017], for example, in sequential-decision-making applications: an agent wants to learn actions that, given the state it is in, minimize its chance of incurring high cost over time. The agent's action will influence its future states, which will in turn influence the agent's future actions. Thus, to choose good actions, the agent must reason recursively about how itself will reason in the future. Beyond this single-agent setting, the powerful combination of nested queries and recursion has been used to model metareasoning in multi-agent systems [Seaman et al. 2020], where an agent must reason about how its opponents will react to its action and how it will further respond to its opponents' reactions.

Formalization challenges. Nested queries, along with recursion, are a powerful linguistic construct for expressing cognitive models and artificial intelligence problems. However, their formal semantics—in particular, operational meanings—is underdeveloped.

Bayesian inference $p(z|x) = p(z,x)/p(x)$ introduces a normalization factor $p(x) = \int p(z,x)\,\mathrm{d}z$ that integrates over all possible executions of the program being queried. This integral, termed *model evidence*, is the same for all executions if inference exists only at the top level and queries cannot be nested. Hence, model evidence is often referred to as the *normalization constant*, and it makes sense for a PPL semantics to concern itself with only unnormalized probability $p(z,x)$ in the absence of nested queries [Culpepper and Cobb 2017; Wand et al. 2018].

However, the model evidence $p(x)$ of a *nested* query is in general different for each execution of the *nesting* query, because the nested query itself may be dependent on random choices made in the nesting query. Therefore, the model evidence of a nested Bayesian-inference problem can no longer be considered a constant from the perspective of a nesting query. Normalization must not be ignored, and a semantics must deal with it head-on.

The possibility to nest queries inside queries implies the semantics must perform integration inside integration—the semantics must be self-referential. Tying the knot of this fixpoint definition is an open challenge in operational-semantics-based language models. In prior work [Borgström et al. 2016; Culpepper and Cobb 2017; Wand et al. 2018; Szymczak and Katoen 2019], a measure-theoretic semantics is defined by integrating a standalone, sampling-based operational semantics. Unfortunately, nested queries preclude such a stratified approach: the sampling-based operational semantics needs to invoke the measure semantics to obtain normalization factors, with the measure semantics in turn invoking the sampling-based operational semantics to perform integration. In

a different approach, the distribution-based operational treatment of Staton et al. [2016] readily supports nested queries (though not recursion), but the meaning of nested queries is defined only by assuming a normalization function.

General recursion—hence possible nontermination—further complicates the matter. Almost surely (a.s.) terminating programs naturally exist in practice—e.g., geometric distributions and infinite-horizon Markov decision processes with absorbing transitions. Such programs admit nonterminating executions, albeit the measure of these executions is zero in the limit (i.e., when the number of allowable evaluation steps tends to infinity). Icard [2017] and Szymczak and Katoen [2019] further show that probabilistic programs with even positive measure of nontermination offer useful modeling power. Unfortunately, nontermination poses challenges due to the self-referential nature of the semantics of nested queries: we are not licensed to take the limit of the semantics (to give meaning to a nested, possibly nonterminating program) while the semantics is being defined.

Finally, when we do take the limit of the measure semantics, we must make sure that the limit exists. Because the measure semantics is essentially a fixpoint definition that allows integration to happen recursively inside integration, proving the existence of limit measures becomes a nontrivial undertaking that necessitates care in posing induction hypotheses and organizing proofs.

Equational reasoning. Reasoning about program equivalence is a fundamental problem in semantics research. The gold standard of equivalence is contextual equivalence [Morris 1968] (a.k.a. observational equivalence). It considers two terms equivalent if composing them with *any* well-formed program context yields two programs with indistinguishable behaviors.

Intriguing equivalence questions arise with probabilistic programs. For example, consider the following two terms, e_1 and e_2:

$$e_1 \stackrel{\text{def}}{=} \textbf{let } x = \textbf{sample } \mathit{Bernoulli}(0.2) \textbf{ in } \textbf{let } \_ = \textbf{score } (\textbf{if } x = y \textbf{ then } 1 \textbf{ else } 0) \textbf{ in } x$$

$$e_2 \stackrel{\text{def}}{=} f(y)$$

```
# Function f used in e2
def f (y) =
  let x = sample Bernoulli(0.2) in
  if x = y then x
  else f(y)  # recursive call
```

At first sight, they appear equivalent. Both terms return x, the value of a Bernoulli random variable. And when they return, both satisfy the constraint $x = y$, where y is a free, boolean variable defined in a surrounding program context. Term e_1 enforces $x = y$ using **score**: executions in which $x \neq y$ has zero probability in the posterior distribution. Term e_2 enforces $x = y$ using a recursive function f that does not return until the constraint is satisfied. Notice that while e_1 terminates certainly for any y, e_2 terminates almost surely: the probability of e_2 not terminating is 0 ($\lim_{n\to\infty} 0.8^n$ or $\lim_{n\to\infty} 0.2^n$, depending on the value y takes). So e_1 and e_2 cannot be distinguished on the grounds of probability of nontermination, either.

Despite our intuition, these two terms cannot be considered contextually equivalent: there exist program contexts that can distinguish them. One such program context C is given below:

$$C \stackrel{\text{def}}{=} \textbf{let } y = \textbf{sample } \mathit{Bernoulli}(0.5) \textbf{ in } \textbf{let } \_ = [\cdot] \textbf{ in } y \quad \text{\# To compose a term with C, place it in the hole } [\cdot]$$

It binds y to a fair coin flip, executes the term placed in the hole, and returns y. The observable behavior of the resulting program $C[e_1]$ or $C[e_2]$ is the distribution over its return value y. Program $C[e_1]$ involves conditioning, which affects the posterior distribution of y: *a priori* $x = \textsc{False}$ is more likely than $x = \textsc{True}$, so *a posteriori* $y = \textsc{False}$ is more likely than $y = \textsc{True}$. By contrast, program $C[e_2]$ does not involve conditioning, so $y = \textsc{True}$ and $y = \textsc{False}$ are equally likely.

Nested queries add an important twist. Place e_1 inside a nested inference query and consider the equivalence question again:

$$e_1' \stackrel{\text{def}}{=} \textbf{sample}\ (\textbf{query}\ e_1)$$

Terms e_1' and e_2 *are* contextually equivalent. In particular, when they are composed with the program context C from above, the resulting distribution of y in each program remains a fair coin flip. The reason is that a query denotes a conditional distribution, and the influence of conditioning on posterior probabilities stops at the boundary of the distribution.

Another way to look at it is that $C\left[e_1'\right]$ stands for a different (possibly unnormalized) joint distribution than $C[e_1]$ does. Program $C[e_1]$ stands for the joint distribution $p(x, y)$, defined below, where $\mathit{BernoulliPMF}(b, \cdot)$ is the probability mass function of a Bernoulli distribution with parameter b, and $\mathbb{1}_P$ is the indicator function that has value 1 when proposition P holds and 0 otherwise:

$$p(x,y) \stackrel{\text{def}}{=} p(y)\,p(x)\,\mathbb{1}_{x=y} = \mathit{BernoulliPMF}(0.5, y)\,\mathit{BernoulliPMF}(0.2, x)\,\mathbb{1}_{x=y} \qquad \#\, C[e_1]\ \textit{joint distr.}$$

$$p'(x,y) \stackrel{\text{def}}{=} p(y)\frac{p(x)\,\mathbb{1}_{x=y}}{\int p(x)\,\mathbb{1}_{x=y}\,\mathrm{d}x} \neq p(x,y) \qquad \#\, C\left[e_1'\right]\ \textit{joint distr.}$$

Program $C\left[e_1'\right]$ stands for the joint distribution $p'(x, y)$, defined above. It differs from $p(x, y)$ in that it normalizes the joint probability $p(x)\,\mathbb{1}_{x=y}$ denoted by e_1. Marginalizing out x, we obtain the distribution over the values to which $C\left[e_1'\right]$ can evaluate:

$$\int p'(x,y)\,\mathrm{d}x = \int p(y)\frac{p(x)\,\mathbb{1}_{x=y}}{\int p(x)\,\mathbb{1}_{x=y}\,\mathrm{d}x}\mathrm{d}x = p(y)\frac{\int p(x)\,\mathbb{1}_{x=y}\,\mathrm{d}x}{\int p(x)\,\mathbb{1}_{x=y}\,\mathrm{d}x} = p(y) = \mathit{BernoulliPMF}(0.5, y)$$

Whereas in the above we argued semi-formally that $C\left[e_1'\right]$ denotes the same distribution as $\mathit{Bernoulli}(0.5)$, the discourse does not constitute a rigorous argument of contextual equivalence between e_1' and e_2. For one thing, contextual equivalence is a universally quantified property—its discriminative power comes from the universal quantification—so we need to show $C\left[e_1'\right]$ and $C[e_2]$ induce identical distributions for *any* program context C. Moreover, PPLs afford a much richer model-specification language than the language of probability can capture. To reason about equivalence in general, we need reasoning principles that operate directly on programming constructs: higher-order functions, general recursion, etc.

Equivalence questions arise also from metareasoning agents, where nesting is induced by recursion. Consider again the coordination game in Figure 1. One would presume that nested queries are indispensable in this example; after all, it is the very first example Stuhlmüller and Goodman [2014] use to introduce nested queries in the Church PPL [Goodman et al. 2008]. Yet we later prove the program in Figure 1 equivalent to a variant in which the recursive calls to *Bob* and to *Alice* (lines 8 and 13) are made without being placed inside queries! Puzzling it may seem, the intuition of this equivalence begins to surface after some contemplation: random choices made by nesting queries (lines 7 and 11) are not used by an agent to reason about the other agent in nested queries—by the problem definition, the agents cannot communicate. The equivalence *happens* to hold.

Equational reasoning is often used to relate programs under program optimization, where one program outperforms the other. Consider, in Figure 2, the trajectories sampled by two path-planning robots implemented as Markov decision processes (MDPs). The robots make a sequence of decisions to reach the goal while minimizing the time taken and avoiding obstacles. The programs use nested queries to reason about robots' own reasoning in the future. They differ in that one program puts code inside a recursively nested query, whereas the same code is outside the query in the other program. Figure 2 hints that the two implementations may be equivalent, as the sampled trajectories are equally good. However, they cause the default inference algorithm of

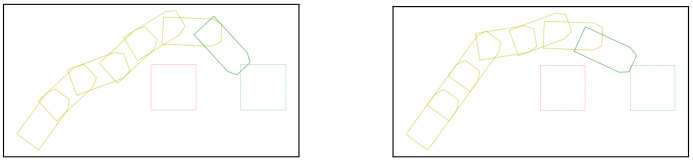


Figure 2. Trajectories of two robots. The objective is to reach the goal region (green square) in the fewest possible steps and avoid the trap (red square). The robots sample equally good trajectories. The programs differ only in that one moves some code from outside a nested query to inside. But inference efficiency differs greatly.

WebPPL [Goodman and Stuhlmüller 2014] to exhibit different performance measured by running time. Formally substantiating the equivalence serves to justify one of the robotics programs as a semantics-preserving optimization of the other.

Contributions. This paper makes the following contributions:

- In §4, we give semantics to a core PPL with nested queries and general recursion, addressing the semantic challenge of mutual reference between an operational semantics and a measure semantics. Step indexing forms a well-founded basis on which we define the semantics. We prove that programs have well-defined meanings: limits of step-indexed measures exist.
- In §6, we approach contextual equivalence (defined in §5) using the powerful technique of logical relations. We construct a step-indexed logical-relations model and prove that logically related terms are contextually equivalent. This theorem offers a sound reasoning principle for proving contextual equivalence concerning nested queries.
- In §7, we demonstrate the usefulness of our theory by using it to prove novel equivalences. The programs are inspired by real-world applications, yet the equivalences have not been suggested before. They reveal the irrelevance of nested queries in a usage previously considered quintessential and bear on the correctness of program optimizations that speed up inference.
- We mechanize our core technical developments using the Coq proof assistant. The mechanization effort is available at https://github.com/yizhouzhang/rrr-popl2022-coq.

We proceed with the statics of the core PPL (§2) and a brief review of measure theory (§3).

2 SYNTAX AND STATIC SEMANTICS

Syntax. Figure 3 defines the syntax of the PPL we work with. It is a statically typed lambda calculus. The usual value forms include variables, real values, the unit value, boolean values, lambda abstractions, fixpoint definitions, and pairs. Program variables are notated in blue. Capture-free substitution of value v for variable x in term e is notated $e\,\{v/x\}$. Real constants c_r have type $\mathbf{R}$, with metavariable r ranging over real numbers on the real line $\mathbb{R}$. General recursion is expressed through the fixpoint construct $\mathbf{fix}\,\mathit{this}.\,e$, where variable this is the self reference bound in e. We will abbreviate $\mathbf{let}\ \_ = e_1\ \mathbf{in}\ e_2$ as $e_1; e_2$. We require pair components to be values; **let**-expressions can be used when pair components need to take evaluation steps. Pair projection is denoted by $\mathbf{fst}\,e$ and $\mathbf{snd}\,e$. Pairs, along with the fixpoint construct, enable mutual recursion.

The calculus builds in the uniform distribution, **Unif**. The **sample** form is used to draw samples from distributions. For example, **sample Unif** produces a real number in the interval $[0, 1]$. Other common distributions are encoded using *inverse transform sampling* that transforms a **Unif** sample into the desired distribution. For example, $\mathit{sample}_{\mathit{Normal}}$ (defined below) is a lambda abstraction returning a sample from a normal distribution. It uses $\mathit{NormalCDF}^{-1}$, a built-in ternary operator that constructs inverse cumulative distribution functions (a.k.a. quantile functions) for normal

$$
\begin{array}{rrcl}
\text{values} & v & ::= & x \mid \mathsf{c}_r \mid () \mid \textsc{True} \mid \textsc{False} \mid \lambda x.\, e \mid \mathbf{fix}\ \mathit{this}.\, e \mid \\
 & & & (v_1, v_2) \mid \mathbf{Unif} \mid \mathbf{query}\ e \\
\text{terms} & e & ::= & v \mid e_1\ e_2 \mid \mathbf{fst}\, e \mid \mathbf{snd}\, e \mid \mathbf{if}\ e_1\ \mathbf{then}\ e_2\ \mathbf{else}\ e_3 \mid \\
 & & & \mathbf{let}\ x = e_1\ \mathbf{in}\ e_2 \mid op^n\ (e_1, ..., e_n) \mid \mathbf{sample}\ e \mid \mathbf{score}\ e \\
\text{unary operators} & op^1 & ::= & \log \mid \exp \mid ... \\
\text{binary operators} & op^2 & ::= & + \mid - \mid \times \mid \div \mid < \mid \leq \mid ... \\
\text{ternary operators} & op^3 & ::= & \mathit{NormalPDF} \mid \mathit{NormalCDF}^{-1} \mid ... \\
\text{types} & \tau & ::= & \mathbf{R} \mid \mathbf{1} \mid \mathbf{2} \mid \tau_1 \to \tau_2 \mid (\tau_1, \tau_2) \mid \mathbf{dist}\ \tau \\
\text{typing environments} & \Gamma & ::= & \cdot \mid \Gamma, x : \tau
\end{array}
$$

$$
\boxed{\Gamma \vdash e : \tau} \qquad \frac{\Gamma, \mathit{this} : \mathbf{1} \to \tau \vdash e : \tau}{\Gamma \vdash \mathbf{fix}\ \mathit{this}.\, e : \mathbf{1} \to \tau} \qquad \frac{op^n : \mathbf{R}^n \to \tau \quad \Gamma \vdash e_1 : \mathbf{R} \quad \cdots \quad \Gamma \vdash e_n : \mathbf{R}}{\Gamma \vdash op^n\ (e_1, ..., e_n) : \tau} \qquad \frac{\Gamma \vdash e : \mathbf{R}}{\Gamma \vdash \mathbf{score}\ e : \mathbf{1}}
$$

$$
\frac{\Gamma \vdash e : \mathbf{dist}\ \tau}{\Gamma \vdash \mathbf{sample}\ e : \tau} \qquad \Gamma \vdash \mathbf{Unif} : \mathbf{dist}\ \mathbf{R} \qquad \frac{\Gamma \vdash e : \mathbf{R}}{\Gamma \vdash \mathbf{query}\ e : \mathbf{dist}\ \mathbf{R}} \qquad \frac{\Gamma \vdash e : \mathbf{2}}{\Gamma \vdash \mathbf{query}\ e : \mathbf{dist}\ \mathbf{2}}
$$

$$
\boxed{op^n : \mathbf{R}^n \to \tau} \qquad \log : \mathbf{R}^1 \to \mathbf{R} \qquad \leq : \mathbf{R}^2 \to \mathbf{2} \qquad \mathit{NormalPDF} : \mathbf{R}^3 \to \mathbf{R} \qquad \mathit{NormalCDF}^{-1} : \mathbf{R}^3 \to \mathbf{R}
$$

Figure 3. Syntax and selected static semantics

distributions. Likewise, $\mathit{sample}_{\mathit{Bern}}$ returns a Bernoulli sample.

$$
\begin{aligned}
\mathit{sample}_{\mathit{Normal}} &\stackrel{\text{def}}{=} \lambda\mu.\, \lambda\sigma.\, \mathbf{let}\ x = \mathbf{sample}\ \mathbf{Unif}\ \mathbf{in}\ \mathit{NormalCDF}^{-1}(\mu, \sigma, x) \\
\mathit{sample}_{\mathit{Bern}} &\stackrel{\text{def}}{=} \lambda p.\, \mathbf{let}\ x = \mathbf{sample}\ \mathbf{Unif}\ \mathbf{in}\ \mathbf{if}\ x \leq p\ \mathbf{then}\ \textsc{True}\ \mathbf{else}\ \textsc{False}
\end{aligned}
$$

The **score** form is for conditioning: it adjusts the probability of the current execution based on its argument, which evaluates to a real value. The higher this real value is, the more probable the current execution is deemed in the posterior distribution denoted by the probabilistic program. For example, MDP agents (Figures 2 and 9) can choose low-cost actions by conditioning as follows:

$$\mathbf{score}\ \exp(-\mathit{cost})$$

The higher a state costs, the exponentially less likely actions leading to the state will be chosen—the MDP agent effectively does Boltzmann exploration [Sutton and Barto 2018], choosing actions with "softmin" cost to balance exploration and exploitation. While this example uses **score** to impose a *soft constraint*, the program in Figure 1 imposes *hard constraints* by assigning 0 scores: zero-scored executions are ruled out from posterior distributions.

The **query** form, as a first-class language construct, enables nested queries—it is our core contribution to have given it meaning. In **query** e, the subterm e may involve sampling, scoring, and further nested queries. The term **query** e, as a whole, is regarded as the *distribution* over evaluation outcomes of e, conditional on the scoring in e.

Typing. Figure 3 shows selected typing rules. The unit type is denoted as **1** and the boolean type **2**. Recursive definitions are functions that take as input at least a unit value. An n-ary operator op^n takes as input n reals. We denote the type of reals by **R**, to differentiate from the real line $\mathbb{R}$.

Term **sample** e has type τ, provided that term e, a distribution, has type **dist** τ. **Unif** is a distribution over reals, so it has type **dist R**. Term **query** e is a distribution over values of type **R** (or **2**), provided e has type **R** (or **2**). Term **score** e has the unit type.

Notice that well-typed programs do not necessarily reduce to values or diverge; they may get stuck. In particular, n-ary operators like log and $\div$ are only partially defined. In this paper we consider the job of preventing such errors to fall on the programmer rather than the type system.

3 BRIEF RECAP OF MEASURE THEORY

We include a cheat sheet on measure theory, and defer to textbooks such as Stein and Shakarchi [2005] and Tao [2011] for further reading.

σ-algebras A σ-algebra on a set X is a collection Σ of subsets of X that includes X itself and is closed under complement and under countable unions. By implication, a σ-algebra includes the empty set and is closed under countable intersections.

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

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

Measures Let (X, Σ) be a measurable space. A function μ from Σ to $\mathbb{R}^+$ is called a measure if it satisfies nonnegativity, null empty set, and countable additivity. We also refer to the quantity $\mu(A)$ that μ assigns to a measurable subset $A \in \Sigma$ as the measure of A.

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

σ-finite measures and (sub)probability measures The measure μ of a measure space (X, Σ, μ) is σ-finite if X can be decomposed into a countable union of sets with finite measure. When $\mu(X) \leq 1$, it is called a subprobability measure. When $\mu(X) = 1$, it is called a probability measure and (X, Σ, μ) a probability space.

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

Indicator functions $\mathbb{1}_P$ takes the value 1 when the proposition P is true; otherwise it is 0.

Measurable functions A measurable function is a function that maps elements in one measure space to elements in another measure space such that the preimage of any measurable subset of the codomain is a measurable subset of the domain.

Lebesgue integration Given a measurable function $f : X \to \mathbb{R}^+$, the Lebesgue integral of f over a measure space (X, Σ, μ) is denoted as $\int f \mathrm{d}\mu$ or $\int f(x)\, \mu(\mathrm{d}x)$. We equip some measurable spaces with a stock measure: the Lebesgue measure $\mu_{\mathbb{R}}$ over reals and the measure $\mu_{\mathbb{S}}$ over an entropy space (§4.1). When integration of f is with respect to a stock measure μ, we write $\int f(x)\, \mathrm{d}x$ to mean $\int f(x)\, \mu(\mathrm{d}x)$. We also use the notation $\int_a^b f(r)\, \mathrm{d}r$ to mean $\int f(r)\, \mathbb{1}_{a \leq r \leq b}\, \mu_{\mathbb{R}}(\mathrm{d}r)$.

Monotone convergence theorem (MCT) (I) Let $\{a_n \in \mathbb{R}^+\}_{n \in \mathbb{N}}$ be a nondecreasing sequence. Then $\lim_{n\to\infty} a_n = \sup_n \{a_n\}$. (II) Let (X, Σ, μ) be a measure space and $\{f_n : X \to \mathbb{R}^+\}_{n \in \mathbb{N}}$ be a pointwise nondecreasing sequence of measurable functions. Then it is legal to interchange limit and integration: $\lim_{n\to\infty} \int f_n \, \mathrm{d}\mu = \int \lim_{n\to\infty} f_n \, \mathrm{d}\mu$. Similar statements to (I) and (II) hold, mutatis mutandis, for nonincreasing sequences and their infima.

4 OPERATIONAL SEMANTICS AND MEASURE SEMANTICS

4.1 Semantics in the Absence of Nested Queries

We introduce the semantics in a phased way by first giving semantics to the core PPL without considering nested queries. The approach is developed in prior work, first defining a deterministic operational semantics and then integrating it over randomness to form a measure semantics. Thus,

the semantics can be viewed as an idealized implementation of importance sampling (or likelihood weighting), conveying the operational intuition of a PPL.

Entropy space. Following prior work [Culpepper and Cobb 2017; Wand et al. 2018; Szymczak and Katoen 2019], we use an *entropy space* $(\mathbb{S}, \Sigma_{\mathbb{S}}, \mu_{\mathbb{S}})$ as the source of randomness. The operational semantics is deterministic in that it carries out evaluation under a single entropy value $\sigma \in \mathbb{S}$. The measure semantics then integrates evaluation over the entropy space. (An alternative, mentioned in §8, is to use a *trace space* [Borgström et al. 2016] as the source of randomness.)

Definition 4.1 (Properties of an entropy space). An entropy space $(\mathbb{S}, \Sigma_{\mathbb{S}}, \mu_{\mathbb{S}})$ is a probability space equipped with a few measurable functions:

(1) $\pi_U : \mathbb{S} \to [0,1]$ interprets an entropy value as a sample from the standard uniform distribution. That is, for any measurable function $f : [0,1] \to \mathbb{R}^+$, $\int f(\pi_U\ \sigma)\, \mu_{\mathbb{S}}(\mathrm{d}\sigma) = \int_0^1 f(r)\, \mathrm{d}r$.

(2) $\pi_L : \mathbb{S} \to \mathbb{S}$ and $\pi_R : \mathbb{S} \to \mathbb{S}$ split an entropy value into a pair of independent ones. That is, for any measurable function $f : \mathbb{S} \times \mathbb{S} \to \mathbb{R}^+$, $\int f(\pi_L\ \sigma,\ \pi_R\ \sigma)\, \mathrm{d}\sigma = \iint f(\sigma_1, \sigma_2)\, \mathrm{d}\sigma_1\, \mathrm{d}\sigma_2$. Therefore, a computation that samples two or more uniform random variables under entropy σ can split σ, use entropy $\pi_L\ \sigma$ to draw the first sample, and perform the rest of the computation under entropy $\pi_R\ \sigma$, thus ensuring that no entropy component is ever reused.

These properties form an abstract definition of an entropy space. We defer to Culpepper and Cobb [2017] for instantiations of these properties on standard probability spaces (e.g., the countably infinite product of unit intervals $[0,1]^\omega$). This paper relies only on the properties specified above.

Operational semantics: small-step reduction. Reduction is of form $\langle \sigma \mid e \rangle \rightsquigarrow \langle \sigma' \mid e' \rangle \bullet w$. A *configuration* $\langle \sigma \mid e \rangle$ consists of an entropy value and an expression. The entropy σ determines the values that uniform samples take in the current execution. A reduction step produces a weight factor $w \in \mathbb{R}^+$ capturing how the reduction step changes the probability of the current execution.

Figure 4 defines the small-step semantics. For now, step indices of reduction rules ($\rightsquigarrow^n$, in gray), as well as the highlighted rules QUERY and QUERY-EXN, should be ignored, since they have to do with nested queries. Rule FIX shows that reducing $(\mathbf{fix}\ \mathit{this}.\ e)\ ()$ essentially unrolls the fixpoint. Rule OP applies an m-ary operation op^m per its built-in interpretation given by $\mathit{interp}\left(op^m, \mathsf{c}_{r_1}, ..., \mathsf{c}_{r_m}\right)$. If $\mathit{interp}\,(op^m, ...)$ is undefined on the real-valued arguments, evaluation is stuck. To reduce **sample Unif**, rule UNIF splits the entropy into two disjoint parts: $\pi_L\ \sigma$ is used to generate the standard uniform sample, while $\pi_R\ \sigma$ is used as the entropy source for the rest of the computation.

Per rule SCORE, **score** c_r multiplies the weight by r, provided $0 < r \leq 1$; otherwise, evaluation is stuck. The requirement of $r \leq 1$ is inherited from Borgström et al. [2016] and Szymczak and Katoen [2019]. Borgström et al. show that **score** may cause probabilistic programs with recursion to have ill-defined model evidence, even when scoring is statically bounded and programs otherwise terminate almost surely. Borgström et al. and Szymczak and Katoen then reconcile this tension between scoring and recursion using 1-bounded **score** (see also §8).

As is usual, the small-step semantics is inductively defined; rule KTX is the basis for structural induction [Felleisen and Hieb 1992]. The small-step semantics is deterministic: a configuration is either irreducible or can be reduced to a unique next configuration and produce a unique weight. Whether a configuration $\langle \sigma \mid e \rangle$ is irreducible depends solely on the term e.

Operational semantics: evaluation. The evaluation relation has form $\langle \sigma \mid e \rangle \searrow^n \langle \sigma' \mid e' \rangle \bullet w$. It states that configuration $\langle \sigma \mid e \rangle$ evaluates to $\langle \sigma' \mid e' \rangle$ under step budget n and produces weight w.

Figure 4 presents evaluation using inference rules, but it can be defined algorithmically as a recursive function, indexed on the step budget. At index 0, no reduction is allowed (EVAL-STOP).

$$\text{evaluation contexts} \quad K ::= [\cdot] \mid K\ e \mid v\ K \mid \mathbf{fst}\ K \mid \mathbf{snd}\ K \mid \mathbf{if}\ K\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2 \mid \mathbf{let}\ x = K\ \mathbf{in}\ e \mid op^n\ (\overline{v}, K, \overline{e}) \mid \mathbf{sample}\ K \mid \mathbf{score}\ K$$

$$\text{terms} \quad e ::= \ldots \mid \mathbf{exn}$$

Small-step reduction: $\langle \sigma \mid e \rangle \rightsquigarrow^n \langle \sigma' \mid e' \rangle \bullet w$

$$\textsc{ktx}\ \frac{\langle \sigma \mid e \rangle \rightsquigarrow^n \langle \sigma' \mid e' \rangle \bullet w}{\langle \sigma \mid K[e] \rangle \rightsquigarrow^n \langle \sigma' \mid K[e'] \rangle \bullet w}$$

$$\textsc{let}\ \langle \sigma \mid \mathbf{let}\ x = v\ \mathbf{in}\ e \rangle \rightsquigarrow^n \langle \sigma \mid e\,\{v/x\} \rangle \bullet 1$$

$$\textsc{op}\ \frac{interp\left(op^m, \mathsf{c}_{r_1}, \ldots, \mathsf{c}_{r_m}\right) = v}{\left\langle \sigma \,\middle|\, op^m\left(\mathsf{c}_{r_1}, \ldots, \mathsf{c}_{r_m}\right) \right\rangle \rightsquigarrow^n \langle \sigma \mid v \rangle \bullet 1}$$

$$\textsc{beta}\ \langle \sigma \mid (\lambda x.\, e)\ v \rangle \rightsquigarrow^n \langle \sigma \mid e\,\{v/x\} \rangle \bullet 1 \qquad \textsc{fix}\ \langle \sigma \mid (\mathbf{fix}\ this.\, e)\ () \rangle \rightsquigarrow^n \langle \sigma \mid e\,\{\mathbf{fix}\ this.\, e/this\} \rangle \bullet 1$$

$$\textsc{fst}\ \langle \sigma \mid \mathbf{fst}\ (v_1, v_2) \rangle \rightsquigarrow^n \langle \sigma \mid v_1 \rangle \bullet 1 \qquad \textsc{then}\ \langle \sigma \mid \mathbf{if}\ \textsc{True}\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2 \rangle \rightsquigarrow^n \langle \sigma \mid e_1 \rangle \bullet 1$$

$$\textsc{snd}\ \langle \sigma \mid \mathbf{snd}\ (v_1, v_2) \rangle \rightsquigarrow^n \langle \sigma \mid v_2 \rangle \bullet 1 \qquad \textsc{else}\ \langle \sigma \mid \mathbf{if}\ \textsc{False}\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2 \rangle \rightsquigarrow^n \langle \sigma \mid e_2 \rangle \bullet 1$$

$$\textsc{unif}\ \frac{r \stackrel{\text{def}}{=} \pi_U\ (\pi_L\ \sigma)}{\langle \sigma \mid \mathbf{sample}\ \mathbf{Unif} \rangle \rightsquigarrow^n \langle \pi_R\ \sigma \mid \mathsf{c}_r \rangle \bullet 1} \qquad \textsc{query}\ \frac{\mu^n_{\mathsf{NS}}(e) > 0}{\langle \sigma \mid \mathbf{sample}\ (\mathbf{query}\ e) \rangle \rightsquigarrow^n \langle \sigma \mid e \rangle \bullet 1 \big/ \mu^n_{\mathsf{NS}}(e)}$$

$$\textsc{score}\ \frac{0 < r \leq 1}{\langle \sigma \mid \mathbf{score}\ \mathsf{c}_r \rangle \rightsquigarrow^n \langle \sigma \mid () \rangle \bullet r} \qquad \textsc{query-exn}\ \frac{\mu^n_{\mathsf{NS}}(e) = 0}{\langle \sigma \mid \mathbf{sample}\ (\mathbf{query}\ e) \rangle \rightsquigarrow^n \langle \sigma \mid \mathbf{exn} \rangle \bullet 1}$$

Evaluation: $\langle \sigma \mid e \rangle \searrow^n \langle \sigma' \mid e' \rangle \bullet w$

$$\textsc{eval-stop}\ \langle \sigma \mid e \rangle \searrow^0 \langle \sigma \mid e \rangle \bullet 1$$

$$\textsc{eval-val}\ \langle \sigma \mid v \rangle \searrow^{n+1} \langle \sigma \mid v \rangle \bullet 1$$

$$\textsc{eval-exn}\ \langle \sigma \mid K[\mathbf{exn}] \rangle \searrow^{n+1} \langle \sigma \mid K[\mathbf{exn}] \rangle \bullet 1$$

$$\textsc{eval-step}\ \frac{\langle \sigma \mid e \rangle \rightsquigarrow^n \langle \sigma' \mid e' \rangle \bullet w \quad \langle \sigma' \mid e' \rangle \searrow^n \langle \sigma'' \mid e'' \rangle \bullet w'}{\langle \sigma \mid e \rangle \searrow^{n+1} \langle \sigma'' \mid e'' \rangle \bullet w \cdot w'}$$

$interp\left(op^m, \mathsf{c}_{r_1}, \ldots, \mathsf{c}_{r_m}\right) = v$ or $\textsc{Undefined}$

$$interp\,(\log, \mathsf{c}_r) = \begin{cases} \mathsf{c}_{\log r} & \text{if } r > 0 \\ \textsc{Undefined} & \text{otherwise} \end{cases} \quad \ldots$$

Figure 4. Operational semantics. In §4.1, highlighted rules and step indices of $\rightsquigarrow^n$ should be disregarded. §4.2 adds nested queries, rendering $\rightsquigarrow^n$, $\searrow^n$, and μ^n_{NS} (Figure 5) mutually recursive.

At index $n+1$, if the term is *terminal*, evaluation is a no-op (EVAL-VAL and EVAL-EXN). Terminal terms include values and exceptions (signaling zero model evidence, §4.2). Otherwise, if it is possible to make a reduction step, then evaluation recurses on the next configuration with the index decremented (EVAL-STEP). The total weight is the product of weights produced by the reduction step and the recursive evaluation. Evaluation is partial: it is undefined when there is step budget left but the term is *stuck*. Notice that being stuck and being terminal are disjoint events.

Step-indexed measures. The measure semantics integrates evaluation with respect to the stock measure $\mu_{\mathbb{S}}$, capturing the aggregate behavior of programs over the entropy space $(\mathbb{S}, \Sigma_{\mathbb{S}}, \mu_{\mathbb{S}})$.

Define $\rho^n_{\mathsf{TV}}(\sigma, e, V)$, as Figure 5 shows. It has value w if evaluation of $\langle \sigma \mid e \rangle$ terminates to a value $v \in V$ within n steps and produces weight w. Integrating $\rho^n_{\mathsf{TV}}(\sigma, e, V)$ with respect to $\mu_{\mathbb{S}}$ induces a measure $\mu^n_{\mathsf{TV}}(e, V)$ over $(\mathbb{V}, \Sigma_{\mathbb{V}})$, the measurable space of syntactic values. The quantity $\mu^n_{\mathsf{TV}}(e, V)$ is the *unnormalized* probability of e terminating to a value under step budget n.

The *normalized* probability is given by $\mu^n_{\mathsf{TV}}(e, V) \big/ \mu^n_{\mathsf{NS}}(e)$. The denominator $\mu^n_{\mathsf{NS}}(e)$ is the model evidence of e when the step budget is n. Defined in Figure 5, it is the integration of $\rho^n_{\mathsf{NS}}(\sigma, e)$ with

$$\rho^n_{\mathsf{TV}}(\sigma, e, V) \stackrel{\text{def}}{=} \begin{cases} w & \text{if } \langle \sigma \mid e \rangle \searrow^n \langle \sigma' \mid v \rangle \bullet w \ \wedge \ v \in V \\ 0 & \text{otherwise} \end{cases} \qquad \mu^n_{\mathsf{TV}}(e, V) \stackrel{\text{def}}{=} \int \rho^n_{\mathsf{TV}}(\sigma, e, V)\, \mu_{\mathbb{S}}(\mathrm{d}\sigma)$$

$$\rho^n_{\mathsf{NS}}(\sigma, e) \stackrel{\text{def}}{=} \begin{cases} w & \text{if } \langle \sigma \mid e \rangle \searrow^n \langle \sigma' \mid e' \rangle \bullet w \\ 0 & \text{otherwise} \end{cases} \qquad \mu^n_{\mathsf{NS}}(e) \stackrel{\text{def}}{=} \int \rho^n_{\mathsf{NS}}(\sigma, e)\, \mu_{\mathbb{S}}(\mathrm{d}\sigma)$$

Figure 5. Step-indexed measure semantics

respect to $\mu_{\mathbb{S}}$. By definition, $\rho^n_{\mathsf{TV}}(\sigma, e, V) \leq \rho^n_{\mathsf{NS}}(\sigma, e)$, since $\rho^n_{\mathsf{NS}}(\sigma, e)$ assigns a positive weight as long as evaluation of $\langle \sigma \mid e \rangle$ does not get stuck within n steps. It follows that $\mu^n_{\mathsf{TV}}(e, V) \leq \mu^n_{\mathsf{NS}}(e)$.

Stuck terms have zero weight. In particular, **score** c_0 is stuck, so we can use **score** c_0 to express hard constraints, as Figure 1 does.

The measurable space of values $(\mathbb{V}, \Sigma_{\mathbb{V}})$, as well as that of terms, is constructed by taking the σ-algebra to be the Borel-measurable sets with respect to a metric. Measurability proofs of the integrand functions are tedious but follow the approach of Borgström et al. [2016, Fig. 5 & §A], Wand et al. [2018, §A], and Szymczak and Katoen [2019, §F].

Limit measures. The meaning of a probabilistic program is characterized, ultimately, by the distribution of its eventual evaluation outcomes—that is, its measure semantics when the step budget n is taken to infinity. Before we can take the limit of the step-indexed approximants, we must make sure that their limits exist.

We can show $\mu^n_{\mathsf{TV}}(e, V)$ are $\mu^n_{\mathsf{NS}}(e)$ are monotonic in n by showing their integrands are monotonic:

- $\rho^n_{\mathsf{TV}}(\sigma, e, V)$ is nondecreasing because it is 0 when evaluation of $\langle \sigma \mid e \rangle$ does not reach a value $v \in V$ within n steps and because it becomes a constant once $\langle \sigma \mid e \rangle$ evaluates to a value.
- $\rho^n_{\mathsf{NS}}(\sigma, e)$ is nonincreasing because no reduction rule other than SCORE changes the weight and because scores are bounded by 1.

So $\mu^n_{\mathsf{TV}}(e, V)$ is nondecreasing and $\mu^n_{\mathsf{NS}}(e)$ is nonincreasing. By the monotone convergence theorem (I), their limits are well-defined:

$$\mu_{\mathsf{NS}}(e) \stackrel{\text{def}}{=} \lim_{n\to\infty} \mu^n_{\mathsf{NS}}(e) = \inf_n \left\{\mu^n_{\mathsf{NS}}(e)\right\} \qquad \mu_{\mathsf{TV}}(e, V) \stackrel{\text{def}}{=} \lim_{n\to\infty} \mu^n_{\mathsf{TV}}(e, V) = \sup_n \left\{\mu^n_{\mathsf{TV}}(e, V)\right\}$$

Notice that the difference $\mu_{\mathsf{NS}}(e) - \mu_{\mathsf{TV}}(e, \mathbb{V})$ is the (unnormalized) measure of *unusual* evaluation outcomes (namely exceptions and divergence); we follow Bichsel et al. [2018] and Szymczak and Katoen [2019] in distinguishing them from stuck computations and giving them measure. If a program e almost surely terminates to values, then $\mu_{\mathsf{TV}}(e, \mathbb{V}) = \mu_{\mathsf{NS}}(e)$. Also notice that to ensure the well-definedness of $\mu_{\mathsf{NS}}(e)$, scores are required to be bounded by 1, a formalization decision inherited from Borgström et al. [2016] and Szymczak and Katoen [2019] (see also §8). Hitherto, the development is standard, in the absence of **query**.

4.2 Semantics in the Presence of Nested Queries

Normalization. If we view **query** as *reifying* a probabilistic program, then sampling it *reflects* the program, as rule QUERY (Figure 4) shows. Importantly, the reduction rule normalizes the weight of a reflective, nesting execution by the model evidence of the reflected, nested term. While it makes sense to consider only unnormalized probabilities in prior semantics [Culpepper and Cobb 2017; Wand et al. 2018], the presence of nested queries makes it impossible to sidestep model evidence.

Mutual recursion between operational semantics and measure semantics. Normalization invokes the measure semantics, breaking the stratification of the two semantics. Whereas §4.1 defines

the measure semantics only after the operational semantics comes into existence, normalization forces the two semantics to be mutually recursive.

The meaning of an inference query is characterized by its measure semantics under a large enough step budget. So a naive way to define rule QUERY would be to evaluate the nested program e under step budget ∞ and use $\lim_{n\to\infty} \mu^n_{\mathsf{NS}}(e)$ as the normalization factor. However, we are not licensed to take the limit of the step-indexed approximants while the very approximants are being defined. We could perhaps avoid taking limits by restricting attention to programs that terminate in finite steps, but the restriction would reject useful programs that terminate a.s. (e.g., the recursive encoding of geometric distributions and infinite-horizon MDPs with absorbing states), let alone programs with positive nontermination measure.

To handle nonterminating executions and tie the recursive knot properly, small-step reduction is step-indexed too: $\langle \sigma \mid e \rangle \rightsquigarrow^n \langle \sigma' \mid e' \rangle \bullet w$, where n means there are at most n allowable steps left *after* this step takes place. Accordingly, to ensure the evaluation function remains well-founded, rule EVAL-STEP (at index $n+1$) invokes small-step reduction at a lower index n: small-step reduction will possibly invoke the measure semantics at index n, which will in turn invoke the evaluation function at index n.

With this indexing method, the index in the operational semantics governs not only the remaining step budget at the current level of evaluation, but also that at lower, reflected levels. Alternatively, we could have used a different index structure consisting of an index governing the remaining step budget at the current level, an index governing the remaining reflection-level budget, and the initial step budget at each level. Going to the reflected level below causes the second index to be decremented but resets the first index. This indexing method could ensure well-foundedness too, through a lexicographic order on the index structure. The two indexing methods result in identical measures when budgets are taken to infinity. We choose the simpler one.

It remains to be shown that limits of measures exist (Theorem 4.2 and §4.3).

Model-evidence exceptions. Normalization is ill-defined when the normalization factor is zero. Hence, rule QUERY requires that the step-indexed model evidence $\mu^n_{\mathsf{NS}}(e)$ be positive. When $\mu^n_{\mathsf{NS}}(e) = 0$, rule QUERY-EXN takes over: it sends the configuration to **exn**, signaling a zero-model-evidence exception. Executions leading to **exn** has positive weight; we consider this evaluation outcome unusual but not unexpected.

Limit measures. An inconvenient consequence of adding nested queries is that integrand $\rho^n_{\mathsf{NS}}(\sigma, e)$ of the measure semantics is no longer monotonic. The loss of this monotonicity makes it much more challenging than in §4.1 (and in prior work) to show that the sequence $\left\{\mu^n_{\mathsf{NS}}(e)\right\}_{n\in\mathbb{N}}$ always converges via the monotone convergence theorem.

It can be seen as follows why $\rho^n_{\mathsf{NS}}(\sigma, e)$ is not necessarily monotonic. We know from §4.1 that the step-indexed $\mu^n_{\mathsf{NS}}(e')$ is nonincreasing if e' does not further nest queries. However, it also means that the weight $\rho^n_{\mathsf{NS}}(\sigma, e)$ of a configuration $\langle \sigma \mid e \rangle$ which steps to **sample** (**query** e') may increase as n increases, since the weight is normalized by the nonincreasing $\mu^n_{\mathsf{NS}}(e')$ per rule QUERY.

In fact, $\rho^n_{\mathsf{NS}}(\sigma, e)$ does not even necessarily converge—let alone not increase—for a nonterminating configuration $\langle \sigma \mid e \rangle$; §B gives an example.

Does the integral $\mu^n_{\mathsf{NS}}(e) = \int \rho^n_{\mathsf{NS}}(\sigma, e)\, \mathrm{d}\sigma$ still converge? We answer this question in the affirmative. In fact, we prove it remains nonincreasing, so the monotone convergence theorem can still be used to show that $\lim_{n\to\infty} \mu^n_{\mathsf{NS}}(e)$ exists. The intuition is that although the normalization factor $\mu^n_{\mathsf{NS}}(e')$ may decrease as n increases, the unnormalized weight of an execution of e' may decrease too, allowing the measure $\mu^n_{\mathsf{NS}}(e)$ of a nesting computation e to not increase. Nevertheless, this intuition is rather equivocal and materializing it necessarily demands that induction hypotheses be chosen

carefully to untangle the convoluted mutual recursion in the definition of the semantics. It is a main goal of §4.3 to prove that $\mu^n_{\mathsf{NS}}(e)$ remains nonincreasing:

Theorem 4.2. *For all e and V, if $m \le n$, then $\mu^m_{\mathsf{TV}}(e,V) \le \mu^n_{\mathsf{TV}}(e,V)$ and $\mu^m_{\mathsf{NS}}(e) \ge \mu^n_{\mathsf{NS}}(e)$. It follows from MCT (I) that the limit measures are well-defined:*

$$\mu_{\mathsf{TV}}(e,V) \stackrel{\text{def}}{=} \lim_{n\to\infty} \mu^n_{\mathsf{TV}}(e,V) = \sup_n \left\{\mu^n_{\mathsf{TV}}(e,V)\right\} \qquad \mu_{\mathsf{NS}}(e) \stackrel{\text{def}}{=} \lim_{n\to\infty} \mu^n_{\mathsf{NS}}(e) = \inf_n \left\{\mu^n_{\mathsf{NS}}(e)\right\}$$

Definition 4.3. Let $\mu^n_{\mathsf{NV}}(e) \stackrel{\text{def}}{=} \mu^n_{\mathsf{NS}}(e) - \mu^n_{\mathsf{TV}}(e,\mathbb{V})$. It follows from Theorem 4.2 that $\mu^n_{\mathsf{NV}}(e)$ is monotonically nonincreasing and has limit $\mu_{\mathsf{NV}}(e) \stackrel{\text{def}}{=} \lim_{n\to\infty} \mu^n_{\mathsf{NV}}(e) = \mu_{\mathsf{NS}}(e) - \mu_{\mathsf{TV}}(e,\mathbb{V})$. $\mu^n_{\mathsf{NV}}(e)$ is the unnormalized measure of those executions of e that evaluate to a non-value under step budget n. As mentioned in §4.1, its limit $\mu_{\mathsf{NV}}(e)$ is the unnormalized measure of unusual evaluation outcomes.

4.3 Properties of the Semantics

We first establish basic facts about the step-indexed semantics defined in §4.2.

Lemma 4.4. $0 \le \mu^n_{\mathsf{TV}}(e,V) \le \mu^n_{\mathsf{NS}}(e) \le 1$.

Lemma 4.5. $\mu^0_{\mathsf{NS}}(e) = 1$.

Lemma 4.6. *If e is terminal, then $\mu^n_{\mathsf{NS}}(e) = 1$.*

Lemma 4.7. *If e is stuck, then $\mu^n_{\mathsf{NS}}(e) = 0$.*

Lemma 4.8. $\mu^n_{\mathsf{NS}}(K[e]) \le \mu^n_{\mathsf{NS}}(e)$.

Lemma 4.9. *If $\langle \sigma \mid e\rangle \rightsquigarrow^n \langle \sigma \mid e'\rangle \bullet w$ for all σ, then $\mu^{n+1}_{\mathsf{NS}}(e) = w \cdot \mu^n_{\mathsf{NS}}(e')$.*

Lemma 4.10. *If $\langle \sigma \mid e\rangle \rightsquigarrow^n \langle \sigma \mid e'\rangle \bullet w$ for all σ, then $\mu^{n+1}_{\mathsf{TV}}(e,V) = w \cdot \mu^n_{\mathsf{TV}}(e',V)$.*

Lemma 4.11. $\mu^{n+1}_{\mathsf{NS}}(K[\textbf{sample Unif}]) = \int_0^1 \mu^n_{\mathsf{NS}}(K[\mathtt{c}_r])\,\mathrm{d}r$.

Lemma 4.12. $\mu^{n+1}_{\mathsf{TV}}(K[\textbf{sample Unif}],V) = \int_0^1 \mu^n_{\mathsf{TV}}(K[\mathtt{c}_r],V)\,\mathrm{d}r$.

Lemmas 4.9–4.12 characterize how measures evolve under small-step reduction. Lemmas 4.9 and 4.10 apply when reduction is independent of entropy—that is, when e is not of form $K[\textbf{sample Unif}]$. Lemmas 4.11 and 4.12 involve integration with respect to the Lebesgue measure over $\mathbb{R}$, because reducing the redex **sample Unif** involves randomness; their proofs rely on Definition 4.1.

The measure monotonicity proof employs a few more definitions.

Definition 4.13. Define a partial, measurable evaluation function of form $\langle \sigma \mid e\rangle \searrow^n_k \langle \sigma' \mid v\rangle \bullet w$:

$$\langle \sigma \mid v\rangle \searrow^n_0 \langle \sigma \mid v\rangle \bullet 1 \qquad \frac{\langle \sigma \mid e\rangle \rightsquigarrow^n \langle \sigma' \mid e'\rangle \bullet w \qquad \langle \sigma' \mid e'\rangle \searrow^n_k \langle \sigma'' \mid v\rangle \bullet w'}{\langle \sigma \mid e\rangle \searrow^{n+1}_{k+1} \langle \sigma'' \mid v\rangle \bullet w \cdot w'}$$

This evaluation function takes as input $\langle \sigma \mid e\rangle$ and n. It differs from the one in Figure 4 in two ways. First, it is defined only on configurations that terminate to a *value* under the given step budget. Second, it additionally returns the number of actual steps taken (i.e., k in $\langle \sigma \mid e\rangle \searrow^n_k \langle \sigma' \mid v\rangle \bullet w$).

Definition 4.14. Define ξ using the evaluation function of Definition 4.13:

$$\xi^n(\sigma,K,e) \stackrel{\text{def}}{=} \begin{cases} w \cdot \mu^{n-k}_{\mathsf{NS}}(K[v]) & \text{if } \langle \sigma \mid e\rangle \searrow^n_k \langle \sigma' \mid v\rangle \bullet w \\ 0 & \text{otherwise} \end{cases}$$

Lemma 4.15. $\mu^n_{\mathsf{NS}}(K[e]) = \mu^n_{\mathsf{NV}}(e) + \int \xi^n(\sigma,K,e)\,\mathrm{d}\sigma$.

The intuition behind Definition 4.14 and Lemma 4.15 is that evaluation of $\langle \sigma \mid K[e]\rangle$ can be divided into two stages: first evaluating $\langle \sigma \mid e\rangle$ to a value $\langle \sigma' \mid v\rangle$ in k steps, and then evaluating $K[v]$ under remaining step budget $n-k$ using fresh entropy independent of the leftover entropy σ'.

Now we are ready to establish measure monotonicity:

Lemma 4.16. *The following statements hold:*

(1) If $\langle \sigma \mid e \rangle \rightsquigarrow^{n+1} \langle \sigma' \mid e_1 \rangle \bullet w_1$, *then there exist* e_0 *and* w_0 *such that* $\langle \sigma \mid e \rangle \rightsquigarrow^{n} \langle \sigma' \mid e_0 \rangle \bullet w_0$.

(2) If $\langle \sigma \mid e \rangle \searrow_{k_0}^{n} \langle \sigma_0 \mid v_0 \rangle \bullet w_0$ *and* $\langle \sigma \mid e \rangle \searrow_{k_1}^{n+1} \langle \sigma_1 \mid v_1 \rangle \bullet w_1$, *then* $\sigma_0 = \sigma_1$, $k_0 = k_1$, $v_0 = v_1$, *and* $w_0 \le w_1$.

(3) $\int \rho_{\mathsf{TV}}^{n}(\sigma, e, \mathbb{V}) \cdot \mathbb{1}_{\rho_{\mathsf{TV}}^{n+1}(\sigma,e,\mathbb{V})=0} \, \mathrm{d}\sigma = 0$.

(4) $\mu_{\mathsf{TV}}^{n}(e, V) \le \mu_{\mathsf{TV}}^{n+1}(e, V)$.

(5) $\mu_{\mathsf{NS}}^{n}(e) \ge \mu_{\mathsf{NS}}^{n+1}(e)$.

The lemma statement is carefully engineered to unravel the mutual recursion among the small-step semantics (1), the evaluation function (2)(3), and the measure semantics (4)(5). As a whole, Lemma 4.16 is proven by

(i) first showing that (1) holds provided that (5) holds, and
(ii) then showing that (2)(3)(4)(5) simultaneously hold by induction on n.

The induction hypotheses of (ii) include the statements of (2)(3)(4)(5) at lower step indices. The proof of (ii) uses (i), instantiating (i) using the induction hypothesis given by (5).

Like $\rho_{\mathsf{NS}}^{n}(\sigma, e)$, $\rho_{\mathsf{TV}}^{n}(\sigma, e, V)$ is in general nonmonotonic in the presence of nested queries. Suppose that at index n, configuration $\langle \sigma \mid K[\textbf{sample}\ (\textbf{query}\ e')] \rangle$ is reduced per rule QUERY. Then at a higher index $n+1$, the configuration may be reduced differently via QUERY-EXN if $\mu_{\mathsf{NS}}^{n}(e') = 0$. Thus, it is not always true that if $\langle \sigma \mid e \rangle \searrow^{n} \langle \sigma' \mid v \rangle \bullet w$ then $\langle \sigma \mid e \rangle$ still terminates to a value under a larger step budget $n+1$. However, we can show that it is *almost* always true, as stated by (3).

Proof Excerpt of Lemma 4.16. Here we show that (5) $\mu_{\mathsf{NS}}^{n}(e) \ge \mu_{\mathsf{NS}}^{n+1}(e)$ holds under the following hypotheses, where H1 corresponds to (i), and H2, H3, H4, and H5 are the induction hypotheses of (ii):

H1. For all m, if $\mu_{\mathsf{NS}}^{m}(e) \ge \mu_{\mathsf{NS}}^{m+1}(e)$ and $\langle \sigma \mid e \rangle \rightsquigarrow^{m+1} \langle \sigma' \mid e_1 \rangle \bullet w_1$, then there exist e_0 and w_0 such that $\langle \sigma \mid e \rangle \rightsquigarrow^{m} \langle \sigma' \mid e_0 \rangle \bullet w_0$.

H2. For all $m < n$, if $\langle \sigma \mid e \rangle \searrow_{k_0}^{m} \langle \sigma_0 \mid v_0 \rangle \bullet w_0$ and $\langle \sigma \mid e \rangle \searrow_{k_1}^{m+1} \langle \sigma_1 \mid v_1 \rangle \bullet w_1$, then $\sigma_0 = \sigma_1$, $k_0 = k_1$, $v_0 = v_1$, and $w_0 \le w_1$.

H3. For all $m < n$, $\int \rho_{\mathsf{TV}}^{m}(\sigma, e, \mathbb{V}) \cdot \mathbb{1}_{\rho_{\mathsf{TV}}^{m+1}(\sigma,e,\mathbb{V})=0} \, \mathrm{d}\sigma = 0$.

H4. For all $m < n$, $\mu_{\mathsf{TV}}^{m}(e, V) \le \mu_{\mathsf{TV}}^{m+1}(e, V)$.

H5. For all $m < n$, $\mu_{\mathsf{NS}}^{m}(e) \ge \mu_{\mathsf{NS}}^{m+1}(e)$.

When $n = 0$, our goal follows from Lemmas 4.4 and 4.5. Below we consider the case $n \ge 1$. There are three situations: e is terminal, e is stuck at index $n-1$, and e is reducible at index $n-1$.

If e is terminal, our goal follows from Lemma 4.6.

If e is stuck at index $n-1$, then $\mu_{\mathsf{NS}}^{n}(e) = 0$ by Lemma 4.7. So we need to show $\mu_{\mathsf{NS}}^{n+1}(e) = 0$, too. By Lemma 4.7, it suffices to show that e is stuck at index n. By induction hypothesis H5, $\mu_{\mathsf{NS}}^{n-1}(e) \ge \mu_{\mathsf{NS}}^{n}(e)$. Using this result to instantiate hypothesis H1, we have that e is reducible at index $n-1$ if e is reducible at index n. Since e is irreducible at index $n-1$, it follows that e is irreducible at index n. Further, since e is not terminal, e must be stuck at index n.

Otherwise, e is reducible at index $n-1$. This reduction step can be categorized into three cases:

(a) e is of form $K[\textbf{sample}\ (\textbf{query}\ e')]$, where reduction depends on the step index;
(b) e is of form $K[\textbf{sample Unif}]$, where reduction depends on the entropy; and
(c) e is of some other form, where reduction is independent of the step index or the entropy.

In case (c), the goal follows from Lemma 4.9. In case (b), the goal follows from Lemma 4.11 and induction hypothesis H5. Case (a) contains two subcases: $\mu_{\mathsf{NS}}^{n-1}(e') = 0$ and $\mu_{\mathsf{NS}}^{n-1}(e') > 0$. If $\mu_{\mathsf{NS}}^{n-1}(e') = 0$, then rule QUERY-EXN is in charge and the goal follows from Lemma 4.9 and hypothesis H5. Below we focus on the second subcase of (a). In this subcase we have

$$\text{for all } \sigma, \langle \sigma \mid K[\textbf{sample}\ (\textbf{query}\ e')] \rangle \rightsquigarrow^{n-1} \langle \sigma \mid K[e'] \rangle \bullet 1 \big/ \mu_{\mathsf{NS}}^{n-1}(e') \tag{4.1}$$

$$\mu_{\mathsf{NS}}^{n}(e) = \mu_{\mathsf{NS}}^{n-1}(K[e']) \big/ \mu_{\mathsf{NS}}^{n-1}(e') \tag{4.2}$$

where (4.2) follows from (4.1) and Lemma 4.9. Now, we have two situations depending on $\mu_{\mathsf{NS}}^{n}(e')$.

- Case $\mu_{\mathsf{NS}}^{n}(e') = 0$. It suffices to show $\mu_{\mathsf{NS}}^{n-1}(K[e']) = \mu_{\mathsf{NS}}^{n-1}(e')$: when the equality holds, $\mu_{\mathsf{NS}}^{n}(e) = 1$ by (4.2), and thus $\mu_{\mathsf{NS}}^{n}(e) \geq \mu_{\mathsf{NS}}^{n+1}(e)$ by Lemma 4.4. Notice that we have $\mu_{\mathsf{TV}}^{n-1}(e', \mathbb{V}) = 0$:

  $$\mu_{\mathsf{TV}}^{n-1}(e', \mathbb{V}) \leq \mu_{\mathsf{TV}}^{n}(e', \mathbb{V}) \leq \mu_{\mathsf{NS}}^{n}(e') = 0$$

  where the first inequality is by induction hypothesis H4 and the second is by Lemma 4.4. So $\mu_{\mathsf{NV}}^{n-1}(e') = \mu_{\mathsf{NS}}^{n-1}(e') - \mu_{\mathsf{TV}}^{n-1}(e', \mathbb{V}) = \mu_{\mathsf{NS}}^{n-1}(e')$. Thus, the goal $\mu_{\mathsf{NS}}^{n-1}(K[e']) = \mu_{\mathsf{NV}}^{n-1}(e')$ follows from

  $$\mu_{\mathsf{NV}}^{n-1}(e') = \mu_{\mathsf{NS}}^{n-1}(e') \geq \mu_{\mathsf{NS}}^{n-1}(K[e']) \geq \mu_{\mathsf{NV}}^{n-1}(e')$$

  where the first inequality is by Lemma 4.8 and the second is by Lemma 4.15.

- Case $\mu_{\mathsf{NS}}^{n}(e') > 0$. This case requires the most ingenuity. In this case, we have

  $$\text{for all } \sigma, \langle \sigma \mid K[\textbf{sample}\ (\textbf{query}\ e')] \rangle \rightsquigarrow^{n} \langle \sigma \mid K[e'] \rangle \bullet 1 \big/ \mu_{\mathsf{NS}}^{n}(e')$$

  $$\mu_{\mathsf{NS}}^{n+1}(e) = \mu_{\mathsf{NS}}^{n}(K[e']) \big/ \mu_{\mathsf{NS}}^{n}(e')$$

  So the goal is to show $\mu_{\mathsf{NS}}^{n-1}(K[e']) \big/ \mu_{\mathsf{NS}}^{n-1}(e') \geq \mu_{\mathsf{NS}}^{n}(K[e']) \big/ \mu_{\mathsf{NS}}^{n}(e')$. By induction hypothesis H5, we have $\mu_{\mathsf{NS}}^{n-1}(K[e']) \geq \mu_{\mathsf{NS}}^{n}(K[e'])$ for the numerators, but we also have $\mu_{\mathsf{NS}}^{n-1}(e') \geq \mu_{\mathsf{NS}}^{n}(e')$ for the denominators—a seeming dilemma. We rewrite the goal using Lemma 4.15, with the evaluation context instantiated by K for the numerators and by $[\cdot]$ for the denominators:

  $$\frac{\mu_{\mathsf{NV}}^{n-1}(e') + \int \xi^{n-1}(\sigma, K, e')\, \mathrm{d}\sigma}{\mu_{\mathsf{NV}}^{n-1}(e') + \int \xi^{n-1}(\sigma, [\cdot], e')\, \mathrm{d}\sigma} \geq \frac{\mu_{\mathsf{NV}}^{n}(e') + \int \xi^{n}(\sigma, K, e')\, \mathrm{d}\sigma}{\mu_{\mathsf{NV}}^{n}(e') + \int \xi^{n}(\sigma, [\cdot], e')\, \mathrm{d}\sigma} \tag{4.3}$$

  The integrals on the RHS of (4.3) can be further decomposed, using $\xi_{<}$ and $\xi_{=}$ defined as follows:

  $$\xi_{<}^{n}(\sigma, K, e) \stackrel{\text{def}}{=} \begin{cases} w \cdot \mu_{\mathsf{NS}}^{n-k}(K[v]) & \text{if } \langle \sigma \mid e \rangle \searrow_{k}^{n} \langle \sigma' \mid v \rangle \bullet w \text{ and } k < n \\ 0 & \text{otherwise} \end{cases}$$

  $$\xi_{=}^{n}(\sigma, e) \stackrel{\text{def}}{=} \begin{cases} w & \text{if } \langle \sigma \mid e \rangle \searrow_{k}^{n} \langle \sigma' \mid v \rangle \bullet w \text{ and } k = n \\ 0 & \text{otherwise} \end{cases}$$

  By Definition 4.14, $\xi^{n}(\sigma, K, e) = \xi_{=}^{n}(\sigma, e) + \xi_{<}^{n}(\sigma, K, e)$: $\xi^{n}(\sigma, K, e)$ is $\xi_{=}^{n}(\sigma, e)$ when $\langle \sigma \mid e \rangle$ terminates to a value in precisely n steps, and $\xi^{n}(\sigma, K, e)$ is $\xi_{<}^{n}(\sigma, K, e)$ when $\langle \sigma \mid e \rangle$ terminates to a value in fewer than n steps. Notice that $\xi_{<}^{n}(\sigma, K, e)$ is closely related to, but subtly different from, $\xi^{n-1}(\sigma, K, e)$: both allow a maximum of $n-1$ evaluation steps at the current level of evaluation, but $\xi_{<}^{n}(\sigma, K, e)$ allows for one more step at lower, reflected levels than $\xi^{n-1}(\sigma, K, e)$ does. Now, rewrite (4.3) by decomposing $\xi^{n}(\sigma, K, e')$ and $\xi^{n}(\sigma, [\cdot], e')$ on its RHS:

  $$\frac{\mu_{\mathsf{NV}}^{n-1}(e') + \int \xi^{n-1}(\sigma, K, e')\, \mathrm{d}\sigma}{\mu_{\mathsf{NV}}^{n-1}(e') + \int \xi^{n-1}(\sigma, [\cdot], e')\, \mathrm{d}\sigma} \geq \frac{\mu_{\mathsf{NV}}^{n}(e') + \int \xi_{=}^{n}(\sigma, e')\, \mathrm{d}\sigma + \int \xi_{<}^{n}(\sigma, K, e')\, \mathrm{d}\sigma}{\mu_{\mathsf{NV}}^{n}(e') + \int \xi_{=}^{n}(\sigma, e')\, \mathrm{d}\sigma + \int \xi_{<}^{n}(\sigma, [\cdot], e')\, \mathrm{d}\sigma} \tag{4.4}$$

  The decomposition allows for connecting the LHS's $\xi^{n-1}(\sigma, K, e')$ and $\xi^{n-1}(\sigma, [\cdot], e')$ to the RHS's $\xi_{<}^{n}(\sigma, K, e')$ and $\xi_{<}^{n}(\sigma, [\cdot], e')$. It also allows for connecting $\mu_{\mathsf{NV}}^{n-1}(e')$ on the LHS to $\mu_{\mathsf{NV}}^{n}(e') +$

$\int \xi^n_{\leq}(\sigma, e')\,\mathrm{d}\sigma$ on the RHS, both of which, roughly speaking, are the measure of e' evaluating to a non-value after $n-1$ steps. Then we can prove (4.4) by applying the following fact

$$\forall \alpha\,\beta\,\gamma\,\alpha'\,\beta'\,\gamma' \in \mathbb{R}^+,\ \ 0 \leq \gamma - \beta \leq \gamma' - \beta' \Rightarrow \alpha' + \gamma' \leq \alpha + \gamma < \infty \Rightarrow \frac{\alpha+\beta}{\alpha+\gamma} \geq \frac{\alpha'+\beta'}{\alpha'+\gamma'} \tag{4.5}$$

with $\alpha = \mu^{n-1}_{\mathsf{NV}}(e')$, $\alpha' = \mu^{n}_{\mathsf{NV}}(e') + \int \xi^n_{\leq}(\sigma, e')\,\mathrm{d}\sigma$, $\beta = \int \xi^{n-1}(\sigma, K, e')\,\mathrm{d}\sigma$, and $\beta' = \int \xi^n_{<}(\sigma, K, e')\,\mathrm{d}\sigma$. This step applying (4.5) is what makes the equivocal intuition mentioned in §4.2 unequivocal. Showing that the antecedents of (4.5) hold is then an exercise of real analysis making use of induction hypotheses H2, H3, and H5; the proof is not shown here to save space. □

The main result of this section, Theorem 4.2, is immediate from Lemma 4.16(4)(5).

It follows from Theorem 4.2 and MCT (I) that index-free versions of Lemmas 4.9 and 4.10 hold:

Lemma 4.17. *If* $\langle \sigma \mid e \rangle \rightsquigarrow^n \langle \sigma \mid e' \rangle \bullet w$ *for all* σ *and* n, *then* $\mu_{\mathsf{NS}}(e) = w \cdot \mu_{\mathsf{NS}}(e')$.

Lemma 4.18. *If* $\langle \sigma \mid e \rangle \rightsquigarrow^n \langle \sigma \mid e' \rangle \bullet w$ *for all* σ *and* n, *then* $\mu_{\mathsf{TV}}(e, V) = w \cdot \mu_{\mathsf{TV}}(e', V)$.

Lemma 4.19. *If* $\mu_{\mathsf{NS}}(e) > 0$, *then* $\mu_{\mathsf{NS}}(K[\mathbf{sample}\ (\mathbf{query}\ e)]) = \mu_{\mathsf{NS}}(K[e]) \,/\, \mu_{\mathsf{NS}}(e)$.

Lemma 4.20. *If* $\mu_{\mathsf{NS}}(e) > 0$, *then* $\mu_{\mathsf{TV}}(K[\mathbf{sample}\ (\mathbf{query}\ e)], V) = \mu_{\mathsf{TV}}(K[e], V) \,/\, \mu_{\mathsf{NS}}(e)$.

Lemmas 4.17 and 4.18 apply when reduction is independent of both entropy and step index. Lemmas 4.19 and 4.20 apply when reduction depends on the step index.

Likewise, it follows from Theorem 4.2 and MCT (II) that index-free versions of Lemmas 4.11 and 4.12 hold too; Lemmas 4.21 and 4.22 apply when reduction depends on the entropy:

Lemma 4.21. $\mu_{\mathsf{NS}}(K[\mathbf{sample\ Unif}]) = \int_0^1 \mu_{\mathsf{NS}}(K[\mathsf{c}_r])\,\mathrm{d}r$.

Lemma 4.22. $\mu_{\mathsf{TV}}(K[\mathbf{sample\ Unif}], V) = \int_0^1 \mu_{\mathsf{TV}}(K[\mathsf{c}_r], V)\,\mathrm{d}r$.

4.4 Implementation in Coq

We formalized the mutually recursive operational semantics and measure semantics and mechanized proofs of the above results using the Coq proof assistant, in about 4,400 lines of Gallina and Ltac code. The development builds on an axiomatization of $\mathbb{R}^+$ and Lebesgue integration, due to Culpepper and Cobb [2017], which further builds on the axiomatization of $\mathbb{R}$ [Mayero 2001] in the standard library of Coq.

5 CONTEXTUAL EQUIVALENCE

The canonical, operational notion of equivalence is *contextual equivalence* [Morris 1968]. A *program context* is a program with a hole $[\cdot]$ in it. Unlike evaluation contexts K, program contexts can bind variables and can be plugged with terms in which the variables occur free. Context typing has form $\vdash C : \Gamma,\ \tau \rightsquigarrow \tau'$, so defined that if term e has typing $\Gamma \vdash e : \tau$, composing C and e yields a program $C[e]$ with typing $\cdot \vdash C[e] : \tau'$. The definition of contexts and context-typing rules are standard and can be found in the Coq development.

Given a notion of indistinguishability, two terms are contextually equivalent if respectively composing them with *any* well-formed program context yields programs that are indistinguishable. For a deterministic, Turing-complete language, indistinguishability is often taken to mean that programs either both terminate or both diverge. If any two programs both terminate but terminate to different values, there must exist a context in which one of the programs terminates and the other diverges—it follows from the universal quantification over contexts that the two programs are not considered contextually equivalent.

For our probabilistic, Turing-complete language, we take indistinguishability to mean (1) that two programs have the same model evidence and (2) that they have the same measure on any measurable set of real values. Formally, contextual equivalence ($\approx_{ctx}$) is defined as follows:

Definition 5.1 (Contextual equivalence).

$$\Gamma \vDash e_1 \approx_{ctx} e_2 : \tau \stackrel{\text{def}}{=} \forall C \ \ \vdash C : \Gamma, \tau \rightsquigarrow \mathbf{R} \Rightarrow \\ \mu_{\mathsf{NS}}(C[e_1]) = \mu_{\mathsf{NS}}(C[e_2]) \ \wedge \ \forall V \ \ \mu_{\mathsf{TV}}(C[e_1], V) = \mu_{\mathsf{TV}}(C[e_2], V)$$

Condition (2) alone is exactly how prior work [Wand et al. 2018] defines indistinguishability. Notice that restricting attention to real-typed programs does not weaken the discriminating power of contextual equivalence, similar to how restricting attention to unit-typed programs in a deterministic setting is not less discriminative. The addition of condition (1) makes our definition of contextual equivalence more discriminative—it more faithfully reflects the meaning of probabilistic programs when exceptions or divergence is possible [Bichsel et al. 2018; Olmedo et al. 2018] (see also §8).

We can define an asymmetric relation called *contextual approximation* ($\preccurlyeq_{ctx}$), where the step-indexed measure semantics of one program approximates the limit measure semantics of the other. It follows from Theorem 4.2 that contextual equivalence has an alternative definition in terms of contextual approximation:

Definition 5.2 (Contextual approximation).

$$\Gamma \vDash e_1 \preccurlyeq_{ctx} e_2 : \tau \stackrel{\text{def}}{=} \forall C \ \ \vdash C : \Gamma, \tau \rightsquigarrow \mathbf{R} \Rightarrow \\ \forall n \ \ \mu^n_{\mathsf{NS}}(C[e_1]) \geq \mu_{\mathsf{NS}}(C[e_2]) \ \wedge \ \forall V \ \ \mu^n_{\mathsf{TV}}(C[e_1], V) \leq \mu_{\mathsf{TV}}(C[e_2], V)$$

Lemma 5.3. $\Gamma \vDash e_1 \approx_{ctx} e_2 : \tau \iff \Gamma \vDash e_1 \preccurlyeq_{ctx} e_2 : \tau \ \wedge \ \Gamma \vDash e_2 \preccurlyeq_{ctx} e_1 : \tau$

6 A SOUND LOGICAL-RELATIONS MODEL

The discriminating power of contextual equivalence arises from the universal quantification over program contexts, which also makes it difficult to prove contextual equivalence directly. A popular method to prove contextual equivalence is by first devising a logical-relations model and then showing that it is sound with respect to contextual equivalence.

6.1 A Biorthogonal, Step-Indexed Definition

Figure 6 defines the logical-relations model. The relation on closed terms is defined using the technique of biorthogonality (also known as $\top\top$-closure) [Pitts and Stark 1998]. A biorthogonal logical relation is automatically complete with respect to contextual equivalence. Two terms are in a biorthogonal term relation if plugging them into related continuations yields related observations. So we need notions of relatedness for observations and for continuations.

Observation relatedness is given by an asymmetric relation O, called *observational approximation*. It relates two programs e_1 and e_2 when the observable behavior (i.e., the measure semantics) of e_1 approximates that of e_2. Observational approximation is indexed on a step budget governing the evaluation of e_1 [Appel and McAllester 2001; Ahmed 2006]. Continuation relatedness is given by $\mathcal{K}$. Two continuations are related if composing them with related values yields related observations.

Value relatedness is given by a semantic interpretation of types $[\![\tau]\!]$, defined as a recursive function that decreases on the size of τ. Two values are related at type τ when the introduction (or elimination) forms of τ are related. In particular, two distribution-typed values are related if sampling the distributions yields related terms.

The $\mathcal{E}$ relation on closed terms is then lifted to logical approximation ($\preccurlyeq_{log}$), a relation on open terms, by quantifying over step indices and over related closing substitutions γ_1 and γ_2 for the free variables bound in Γ. The notation γe means applying substitution function γ to e. Finally, logical equivalence ($\approx_{log}$) is defined as logical approximation in both directions.

Biorthogonality

$$O^n(e_1,\, e_2) \stackrel{\text{def}}{=} \mu^n_{\mathsf{NS}}(e_1) \geq \mu_{\mathsf{NS}}(e_2) \;\wedge\; \forall V\; \mu^n_{\mathsf{TV}}(e_1, V) \leq \mu_{\mathsf{TV}}(e_2, V)$$
$$\mathcal{K}[\![\tau]\!]^n(K_1,\, K_2) \stackrel{\text{def}}{=} \forall m \leq n\;\; \forall v_1\; v_2\;\; [\![\tau]\!]^m(v_1,\, v_2) \Rightarrow O^m(K_1[v_1],\, K_2[v_2])$$
$$\mathcal{E}[\![\tau]\!]^n(e_1,\, e_2) \stackrel{\text{def}}{=} \forall m \leq n\;\; \forall K_1\; K_2\;\; \mathcal{K}[\![\tau]\!]^m(K_1,\, K_2) \Rightarrow O^m(K_1[e_1],\, K_2[e_2])$$

Semantic types

$$[\![\mathbf{R}]\!]^n(v_1,\, v_2) \stackrel{\text{def}}{=} \exists r\;\; v_1 = v_2 = \mathsf{c}_r$$
$$[\![\mathbf{1}]\!]^n(v_1,\, v_2) \stackrel{\text{def}}{=} v_1 = v_2 = ()$$
$$[\![\mathbf{2}]\!]^n(v_1,\, v_2) \stackrel{\text{def}}{=} v_1 = v_2 = \textsc{True} \;\vee\; v_1 = v_2 = \textsc{False}$$
$$[\![\tau' \to \tau]\!]^n(v_1,\, v_2) \stackrel{\text{def}}{=} \cdot \vdash v_{1,2} : \tau' \to \tau \;\wedge\; \forall m \leq n\;\; \forall v_1'\; v_2'\;\; [\![\tau']\!]^m(v_1',\, v_2') \Rightarrow \mathcal{E}[\![\tau]\!]^m(v_1\, v_1',\, v_2\, v_2')$$
$$[\![(\tau,\, \tau')]\!]^n(v_1,\, v_2) \stackrel{\text{def}}{=} \exists u_1\; u_1'\; u_2\; u_2'\;\; v_1 = (u_1,\, u_1') \;\wedge\; v_2 = (u_2,\, u_2') \;\wedge\; [\![\tau]\!]^n(u_1,\, u_2) \;\wedge\; [\![\tau']\!]^n(u_1',\, u_2')$$
$$[\![\mathbf{dist}\;\tau]\!]^n(v_1,\, v_2) \stackrel{\text{def}}{=} \cdot \vdash v_{1,2} : \mathbf{dist}\;\tau \;\wedge\; \mathcal{E}[\![\tau]\!]^n(\mathbf{sample}\;v_1,\, \mathbf{sample}\;v_2)$$

Lifting $\mathcal{E}$ to open terms

$$\gamma \;::=\; \cdot \;\mid\; \gamma,\, x \mapsto v$$
$$[\![\Gamma]\!]^n(\gamma_1,\, \gamma_2) \stackrel{\text{def}}{=} \forall x \in \mathit{domain}(\Gamma)\;\; [\![\Gamma(x)]\!]^n(\gamma_1 x,\, \gamma_2 x)$$
$$\Gamma \vDash e_1 \lesssim_{\mathit{log}} e_2 : \tau \stackrel{\text{def}}{=} \Gamma \vdash e_{1,2} : \tau \;\wedge\; \forall n\;\; \forall \gamma_1\; \gamma_2\;\; [\![\Gamma]\!]^n(\gamma_1,\, \gamma_2) \Rightarrow \mathcal{E}[\![\tau]\!]^n(\gamma_1 e_1,\, \gamma_2 e_2)$$
$$\Gamma \vDash e_1 \approx_{\mathit{log}} e_2 : \tau \stackrel{\text{def}}{=} \Gamma \vDash e_1 \lesssim_{\mathit{log}} e_2 : \tau \;\wedge\; \Gamma \vDash e_2 \lesssim_{\mathit{log}} e_1 : \tau$$

Figure 6. Logical-relations model

$$\frac{\Gamma \vDash e_1 \lesssim_{\mathit{log}} e_2 : \mathbf{dist}\;\tau}{\Gamma \vDash \mathbf{sample}\;e_1 \lesssim_{\mathit{log}} \mathbf{sample}\;e_2 : \tau} \qquad \frac{\Gamma \vDash e_1 \lesssim_{\mathit{log}} e_2 : \mathbf{R}}{\Gamma \vDash \mathbf{score}\;e_1 \lesssim_{\mathit{log}} \mathbf{score}\;e_2 : \mathbf{1}} \qquad \Gamma \vDash \mathbf{Unif} \lesssim_{\mathit{log}} \mathbf{Unif} : \mathbf{dist}\;\mathbf{R}$$

$$\frac{\Gamma,\, \mathit{this} : \mathbf{1} \to \tau \vDash e_1 \lesssim_{\mathit{log}} e_2 : \tau}{\Gamma \vDash \mathbf{fix}\;\mathit{this}.\, e_1 \lesssim_{\mathit{log}} \mathbf{fix}\;\mathit{this}.\, e_2 : \mathbf{1} \to \tau} \qquad \frac{\Gamma \vDash e_1 \lesssim_{\mathit{log}} e_2 : \mathbf{R}}{\Gamma \vDash \mathbf{query}\;e_1 \lesssim_{\mathit{log}} \mathbf{query}\;e_2 : \mathbf{dist}\;\mathbf{R}}$$

Figure 7. Selected compatibility lemmas

6.2 Soundness

Monotonicity of step-indexed logical relations. It follows from Theorem 4.2 that the logical relations are monotone in the step index—what holds in the current world (i.e., at step index n) also holds in future worlds (i.e., at step index $m < n$).

Lemma 6.1. *If $m \leq n$ and $O^n(e_1,\, e_2)$ is true, then $O^m(e_1,\, e_2)$ is true. The same holds, mutatis mutandis, for the step-indexed logical relations $\mathcal{E}[\![\tau]\!]$, $\mathcal{K}[\![\tau]\!]$, $[\![\tau]\!]$, and $[\![\Gamma]\!]$.*

Compatibility lemmas. The building blocks of this section's main theorems are the so-called compatibility lemmas. Figure 7 shows the most relevant ones, presented in the style of inference rules to evoke a structural correspondence to the typing rules in Figure 3. Compatibility proofs for the deterministic fragment of the language use Lemmas 4.9–4.10 (to reduce the LHS term), Lemmas 4.17–4.18 (to reduce the RHS term), and Lemma 6.1. The lemma for **fix** requires induction

on the step index, due to the recursive nature of its operational semantics. The lemma for **Unif** relies on Lemmas 4.11–4.12 and 4.21–4.22. Expectedly, proving the compatibility lemma for **query** requires the most work. It uses Lemma 4.15 and Fact (4.5), as well as Lemmas 4.9–4.10 and 4.19–4.20.

Fundamental property. From the compatibility lemmas follows the so-called fundamental theorem of logical relations [Statman 1985]: a well-typed term is logically related to itself. The proof is by induction on typing derivations and by applying the appropriate compatibility lemma in each case.

Theorem 6.2 (Fundamental Property). *If* $\Gamma \vdash e : \tau$, *then* $\Gamma \vDash e \preccurlyeq_{log} e : \tau$.

Soundness. Our goal is to show that logical relatedness implies contextual equivalence, for which we need Lemmas 6.3 and 6.4. Precongruence is proven by induction on context-typing derivations and by applying the appropriate compatibility lemma in each case.

Lemma 6.3 (Precongruence). *If* $\Gamma \vDash e_1 \preccurlyeq_{log} e_2 : \tau$ *and* $\vdash C : \Gamma, \tau \rightsquigarrow \mathbf{R}$, *then* $\cdot \vDash C[e_1] \preccurlyeq_{log} C[e_2] : \mathbf{R}$.

Lemma 6.4 (Adequacy). *If* $\cdot \vDash e_1 \preccurlyeq_{log} e_2 : \mathbf{R}$, *then* $O^n(e_1, e_2)$ *for all* n.

Soundness of the logical-relations model is immediate from Precongruence and Adequacy.

Theorem 6.5 (Soundness). *If* $\Gamma \vDash e_1 \preccurlyeq_{log} e_2 : \tau$, *then* $\Gamma \vDash e_1 \preccurlyeq_{ctx} e_2 : \tau$.

This soundness theorem is our license to use logical relatedness to prove contextual equivalence. In practice, logical-relations proofs often employ a third, even more usable approximation relation, called *ciu approximation*:

Definition 6.6 ($\preccurlyeq_{ciu}$*).* $\Gamma \vDash e_1 \preccurlyeq_{ciu} e_2 : \tau \stackrel{\text{def}}{=} \Gamma \vdash e_{1,2} : \tau \ \wedge \ \forall \gamma \ \vdash \gamma : \Gamma \Rightarrow \forall K \ \ \forall n \ \ O^n(K[\gamma e_1], K[\gamma e_2])$

It considers two terms equivalent when closed instantiations (i.e., substitution for free variables) of them yield related observable behaviors when used (i.e., placed inside evaluation contexts) [Mason and Talcott 1991]. We can show that ciu approximation entails logical approximation:

Lemma 6.7 ($\preccurlyeq_{ciu}$ entails $\preccurlyeq_{log}$). *If* $\Gamma \vDash e_1 \preccurlyeq_{ciu} e_2 : \tau$, *then* $\Gamma \vDash e_1 \preccurlyeq_{log} e_2 : \tau$.

It follows from Theorem 6.5 and Lemma 6.7 that contextual approximation can be established by showing ciu approximation. Ciu approximation is easier to use than contextual approximation because the universal quantification is over evaluation contexts K, a subclass of program contexts C that can be instantiated only by closed terms. It is also easier to use than the logical relation $\mathcal{E}$ because evaluation contexts on both sides are the same. We work with ciu approximation in §7.

6.3 Implementation in Coq

Building on the Coq development in §4.4, we further implemented the step-indexed logical-relations model. All results in §6.2 are machine-checked, in about 2,000 lines of code.

7 REASONING ABOUT NESTED QUERIES: EXAMPLES

We now demonstrate how to use the reasoning principles established in the preceding sections to prove novel equivalences. We use four examples: the first one equates programs that otherwise would not be equivalent without using **query** (§7.2); the second one proves nested queries semantically irrelevant in a usage previously considered prototypical (§7.3); the last two equate decision-making programs where subtle syntactic differences lead to exponential differences in inference efficiency (§7.4 and §7.5). To increase readability, the first example is simpler and explained in greater detail. The other examples are more condensed, but their proofs exhibit similar patterns.

7.1 A Catalog of Standard Equivalences

Before addressing more challenging equivalences, we first catalog a few standard ones validated using the logical relations. They help sanity-check that our semantics is reasonably conservative.

E1. If $\Gamma \vdash (\lambda x.\, e)\ v : \tau$, then $\Gamma \vDash (\lambda x.\, e)\ v \approx_{ctx} e\,\{v/x\} : \tau$.

E2. Let e_1, e_2, and e_3 be well-typed: (i) $\Gamma \vdash e_1 : \tau_1$, (ii) $\Gamma,\ x : \tau_1 \vdash e_2 : \tau_2$, and (iii) $\Gamma,\ y : \tau_2 \vdash e_3 : \tau_3$. Then $\Gamma \vDash \textbf{let}\ y = (\textbf{let}\ x = e_1\ \textbf{in}\ e_2)\ \textbf{in}\ e_3 \approx_{ctx} \textbf{let}\ x = e_1\ \textbf{in let}\ y = e_2\ \textbf{in}\ e_3 : \tau_3$.

E3. Let $e_1 \stackrel{\text{def}}{=} sample_{Normal}\ e_{m_1}\ \left|e_{s_1}\right| + sample_{Normal}\ e_{m_2}\ \left|e_{s_2}\right|$ and $e_2 \stackrel{\text{def}}{=} sample_{Normal}\ \left(e_{m_1}+e_{m_2}\right)\ \sqrt{e_{s_1}^2+e_{s_2}^2}$, where $\Gamma \vdash e_{1,2} : \mathbf{R}$. Then $\Gamma \vDash e_1 \approx_{ctx} e_2 : \mathbf{R}$.

E4. Let $\Gamma \vdash e : \tau$ and $r_{1,2} \in (0, 1]$. Then $\Gamma \vDash \textbf{score}\ \mathsf{c}_{r_1}; \textbf{score}\ \mathsf{c}_{r_2}; e \approx_{ctx} \textbf{score}\ \left(\mathsf{c}_{r_1} \times \mathsf{c}_{r_2}\right); e : \tau$.

E5. Let $r \in (0, 1]$. Then $\cdot \vDash \textbf{score}\ \mathsf{c}_r \approx_{ctx} \textbf{if}\ sample_{Bern}\ \mathsf{c}_r\ \textbf{then}\ (\textbf{score}\ \mathsf{c}_1)\ \textbf{else}\ (\textbf{score}\ \mathsf{c}_0) : \mathbf{1}$.

E6. Let $\Gamma \vdash e : \mathbf{R}$ and $r \in (0, 1]$. Then $\Gamma \vDash \textbf{query}\ e \approx_{ctx} \textbf{query}\ (\textbf{score}\ \mathsf{c}_r; e) : \textbf{dist}\ \mathbf{R}$.

E7. Let $x : \mathbf{R} \vdash e : \mathbf{R}$. Suppose that for all $r \in \mathbb{R}$, $e\,\{\mathsf{c}_r/x\}$ always terminates, with model evidence $f(r) \stackrel{\text{def}}{=} \mu_{\mathsf{NS}}(e\,\{\mathsf{c}_r/x\}) > 0$. Suppose op is a unary operator such that $interp\,(\mathrm{op}, \mathsf{c}_r) = \mathsf{c}_{f(r)}$. Then $x : \mathbf{R} \vDash e \approx_{ctx} \textbf{score}\ \mathrm{op}(x); \textbf{sample}\ (\textbf{query}\ e) : \mathbf{R}$.

These equivalences are adapted from those reported in Staton [2017], Culpepper and Cobb [2017], and Wand et al. [2018]. E1 is equivalence under β-reduction. E2 is associativity of **let**-binding. E3 shows that the sum of two Gaussians is a Gaussian. E4 is distributivity of scoring. E5 suggests that (1-bounded) soft constraints can be implemented in terms of hard constraints. E6 shows that introducing independent scoring into a computation does not change its normalized measure. E7 justifies the resampling technique: a computation e—especially when it performs conditioning—can equivalently be implemented by obtaining a normalized representation of e, resampling from it, and scoring the model evidence, to prevent a proliferation of low-weight samples.

7.2 Fixing False Equivalences Using Nested Queries

We revisit the first equivalence question posed in §1, wherein a false equivalence between e_1 and e_2 is fixed by a true equivalence between **sample** (**query** e_1) and e_2. Specifically, we show $y : \mathbf{2} \vDash \textbf{sample}\ (\textbf{query}\ e_1) \approx_{ctx} e_2 : \mathbf{2}$ where e_1 and e_2 are defined as follows (assuming an equality operator over Boolean values):

$$e_1 \stackrel{\text{def}}{=} \begin{array}{l}\textbf{let}\ x = sample_{Bern}\ \mathsf{c}_{0.2}\ \textbf{in}\\ \textbf{let}\ \_ = \textbf{score}\ (\textbf{if}\ x = y\ \textbf{then}\ \mathsf{c}_1\ \textbf{else}\ \mathsf{c}_0)\ \textbf{in}\\ x\end{array} \qquad e_2 \stackrel{\text{def}}{=} \begin{array}{l}\textbf{let}\ f = \textbf{fix}\ this.\ \lambda y.\ \textbf{let}\ x = sample_{Bern}\ \mathsf{c}_{0.2}\ \textbf{in}\\ \qquad\qquad \textbf{if}\ x = y\ \textbf{then}\ x\ \textbf{else}\ this\ ()\ y\ \textbf{in}\\ f\ ()\ y\end{array}$$

Proof. By Lemma 5.3, the goal amounts to showing contextual approximations in both directions. By Soundness (Theorem 6.5) and Lemma 6.7, we can show ciu approximations instead:

$$y : \mathbf{2} \vDash \textbf{sample}\ (\textbf{query}\ e_1) \preccurlyeq_{ciu} e_2 : \mathbf{2} \qquad y : \mathbf{2} \vDash e_2 \preccurlyeq_{ciu} \textbf{sample}\ (\textbf{query}\ e_1) : \mathbf{2}$$

By Definition 6.6, we need to show that for all $v \in \{\text{True}, \text{False}\}$, K, and n, the following observational approximations hold:

$$O^n(K[\textbf{sample}\ (\textbf{query}\ e_1\ \{v/y\})],\ K[e_2\ \{v/y\}]) \tag{7.1}$$

$$O^n(K[e_2\ \{v/y\}],\ K[\textbf{sample}\ (\textbf{query}\ e_1\ \{v/y\})]) \tag{7.2}$$

Here we verify them for the case $v = \text{True}$; the case $v = \text{False}$ is analogous. Because O is monotone (Lemma 6.1), it suffices to verify them for sufficiently large n's—below we assume $n \geq 11$.

By the definition of the O relation, proving (7.1) amounts to showing the following inequalities:

$$\mu^n_{\mathsf{NS}}(K[\textbf{sample}\ (\textbf{query}\ e_1\ \{\textsc{True}/y\})]) \geq \mu_{\mathsf{NS}}(K[e_2\ \{\textsc{True}/y\}]) \tag{7.3}$$

$$\forall V\ \mu^n_{\mathsf{TV}}(K[\textbf{sample}\ (\textbf{query}\ e_1\ \{\textsc{True}/y\})], V) \leq \mu_{\mathsf{TV}}(K[e_2\ \{\textsc{True}/y\}], V) \tag{7.4}$$

We simplify the step-indexed LHS of (7.4) first. For convenience, define J to be an evaluation context emerging during evaluation: $e_1\ \{\textsc{True}/y\} = J\left[sample_{Bern}\ \mathsf{c}_{0.2}\right]$. The calculation below works by repeatedly using the small-step semantics to step the term in question while applying Lemmas 4.9–4.12 to evolve measures:

$$\begin{aligned}
&\mu^n_{\mathsf{TV}}(K[\textbf{sample}\ (\textbf{query}\ e_1\ \{\textsc{True}/y\})], V) = \frac{\mu^{n-1}_{\mathsf{TV}}(K[e_1\ \{\textsc{True}/y\}], V)}{\mu^{n-1}_{\mathsf{NS}}(e_1\ \{\textsc{True}/y\})} \\
&= \frac{\mu^{n-1}_{\mathsf{TV}}(K\left[J\left[sample_{Bern}\ \mathsf{c}_{0.2}\right]\right], V)}{\mu^{n-1}_{\mathsf{NS}}(J\left[sample_{Bern}\ \mathsf{c}_{0.2}\right])} = \frac{\int \mu^{n-4}_{\mathsf{TV}}(K[J[\textbf{if}\ \mathsf{c}_r \leq \mathsf{c}_{0.2}\ \textbf{then}\ \textsc{True}\ \textbf{else}\ \textsc{False}]], V)\,\mathrm{d}r}{\int \mu^{n-4}_{\mathsf{NS}}(J[\textbf{if}\ \mathsf{c}_r \leq \mathsf{c}_{0.2}\ \textbf{then}\ \textsc{True}\ \textbf{else}\ \textsc{False}])\,\mathrm{d}r} \\
&= \frac{0.2 \cdot \mu^{n-6}_{\mathsf{TV}}(K[J[\textsc{True}]], V) + 0.8 \cdot \mu^{n-6}_{\mathsf{TV}}(K[J[\textsc{False}]], V)}{0.2 \cdot \mu^{n-6}_{\mathsf{NS}}(J[\textsc{True}]) + 0.8 \cdot \mu^{n-6}_{\mathsf{NS}}(J[\textsc{False}])} \\
&= \frac{0.2 \cdot \mu^{n-9}_{\mathsf{TV}}(K[\textbf{let}\ _ = \textbf{score}\ \mathsf{c}_1\ \textbf{in}\ \textsc{True}], V) + 0.8 \cdot \mu^{n-9}_{\mathsf{TV}}(K[\textbf{let}\ _ = \textbf{score}\ \mathsf{c}_0\ \textbf{in}\ \textsc{False}], V)}{0.2 \cdot \mu^{n-9}_{\mathsf{NS}}(\textbf{let}\ _ = \textbf{score}\ \mathsf{c}_1\ \textbf{in}\ \textsc{True}) + 0.8 \cdot \mu^{n-9}_{\mathsf{NS}}(\textbf{let}\ _ = \textbf{score}\ \mathsf{c}_0\ \textbf{in}\ \textsc{False})} \\
&= \frac{0.2 \cdot \mu^{n-11}_{\mathsf{TV}}(K[\textsc{True}], V) + 0.8 \cdot 0}{0.2 \cdot \mu^{n-11}_{\mathsf{NS}}(\textsc{True}) + 0.8 \cdot 0} = \mu^{n-11}_{\mathsf{TV}}(K[\textsc{True}], V) \qquad (\text{By Lemma 4.6, } \mu^{n-11}_{\mathsf{NS}}(\textsc{True}) = 1)
\end{aligned}$$

Likewise, we have for the LHS of (7.3) that $\mu^n_{\mathsf{NS}}(K[\textbf{sample}\ (\textbf{query}\ e_1\ \{\textsc{True}/y\})]) = \mu^{n-11}_{\mathsf{NS}}(K[\textsc{True}])$.

Next we simplify the RHS of (7.4). Define e_2' such that $\langle \sigma \mid e_2\ \{\textsc{True}/y\} \rangle \rightsquigarrow^m \left\langle \sigma \,\middle|\, e_2' \right\rangle \bullet 1$ for all m and σ. That is, $e_2' \stackrel{\text{def}}{=} (\textbf{fix}\ \mathit{this}.\ \ldots)\ ()\ \textsc{True}$. It takes 10 steps to evaluate the recursive call e_2' to a return value or until e_2' appears as the redex again. Thus, by repeatedly applying Lemmas 4.10 and 4.12 and taking evaluation steps, we can derive the following recursive equation:

$$\mu^m_{\mathsf{TV}}(K\left[e_2'\right], V) = \begin{cases} 0 & \text{if } 0 \leq m \leq 9 \\ 0.2 \cdot \mu^{m-10}_{\mathsf{TV}}(K[\textsc{True}], V) + 0.8 \cdot \mu^{m-10}_{\mathsf{TV}}\left(K\left[e_2'\right], V\right) & \text{if } m \geq 10 \end{cases} \tag{7.5}$$

Taking the limit of each side of (7.5), we have $\mu_{\mathsf{TV}}\left(K\left[e_2'\right], V\right) = \mu_{\mathsf{TV}}(K[\textsc{True}], V)$. So by Lemma 4.18, $\mu_{\mathsf{TV}}(K[e_2\ \{\textsc{True}/y\}], V) = \mu_{\mathsf{TV}}\left(K\left[e_2'\right], V\right) = \mu_{\mathsf{TV}}(K[\textsc{True}], V)$. Likewise, we can derive for the RHS of (7.3) that $\mu_{\mathsf{NS}}(K[e_2\ \{\textsc{True}/y\}]) = \mu_{\mathsf{NS}}(K[\textsc{True}])$. Thus (7.3) and (7.4) are true:

$$\mu^n_{\mathsf{NS}}(K[\textbf{sample}\ (\textbf{query}\ e_1\ \{\textsc{True}/y\})]) = \mu^{n-11}_{\mathsf{NS}}(K[\textsc{True}]) \geq \mu_{\mathsf{NS}}(K[\textsc{True}]) = \mu_{\mathsf{NS}}(K[e_2\ \{\textsc{True}/y\}])$$

$$\mu^n_{\mathsf{TV}}(K[\textbf{sample}\ (\textbf{query}\ e_1\ \{\textsc{True}/y\})], V) = \mu^{n-11}_{\mathsf{TV}}(K[\textsc{True}], V) \leq \mu_{\mathsf{TV}}(K[\textsc{True}], V) = \mu_{\mathsf{TV}}(K[e_2\ \{\textsc{True}/y\}], V)$$

Above we proved observational approximation (7.1). We can show that the other direction (7.2) holds too. By (7.5) and by induction on n,

$$\begin{aligned}
&\mu^n_{\mathsf{TV}}(K[e_2\ \{\textsc{True}/y\}], V) = \mu^{n-1}_{\mathsf{TV}}\left(K\left[e_2'\right], V\right) = 0.2 \cdot \mu^{n-11}_{\mathsf{TV}}(K[\textsc{True}], V) + 0.8 \cdot \mu^{n-11}_{\mathsf{TV}}\left(K\left[e_2'\right], V\right) \\
&\leq 0.2 \cdot \mu^{n-11}_{\mathsf{TV}}(K[\textsc{True}], V) + 0.8 \cdot \mu_{\mathsf{TV}}(K[\textsc{True}], V) \qquad (\text{By the induction hypothesis}) \\
&\leq \mu_{\mathsf{TV}}(K[\textsc{True}], V) = \mu_{\mathsf{TV}}(K[\textbf{sample}\ (\textbf{query}\ e_1\ \{\textsc{True}/y\})], V)
\end{aligned}$$

Similarly, we have $\mu^n_{\mathsf{NS}}(K[e_2\ \{\textsc{True}/y\}]) \geq \mu_{\mathsf{NS}}(K[\textbf{sample}\ (\textbf{query}\ e_1\ \{\textsc{True}/y\})])$. □

The example above shows that our semantics and logical relations can be applied to reason about programs like e_2 that admit nonterminating executions. Nonetheless, it is nonessential to equivalences of this flavor that one of the programs in the equivalence exhibits nontermination:

$Agents_1 \stackrel{\text{def}}{=} \textbf{fix}\ this.\ (Alice_1,\ Bob_1)$

$Alice_1 \stackrel{\text{def}}{=} \lambda d.\ \textbf{let}\ aLoc = sample_{Bern}\ aPrior\ \textbf{in}$
$\quad \textbf{let}\ bLoc = \textbf{snd}\ (this\ ())\ (d - \mathsf{c}_1)\ \textbf{in}$
$\quad \textbf{let}\ \_ = ScoreLoc\ aLoc\ bLoc\ \textbf{in}\ aLoc$

$Bob_1 \stackrel{\text{def}}{=} \lambda d.\ \textbf{let}\ bLoc = sample_{Bern}\ bPrior\ \textbf{in}$
$\quad \textbf{if}\ d > \mathsf{c}_0\ \textbf{then}$
$\quad\quad \textbf{let}\ aLoc = \textbf{fst}\ (this\ ())\ d\ \textbf{in}$
$\quad\quad \textbf{let}\ \_ = ScoreLoc\ aLoc\ bLoc\ \textbf{in}\ bLoc$
$\quad \textbf{else}\ bLoc$

$CG_1 \stackrel{\text{def}}{=} \lambda\_.\ \textbf{sample}\ (\textbf{query}\ (\textbf{fst}\ (Agents_1\ ())\ \mathsf{c}_8))$

$Agents_2 \stackrel{\text{def}}{=} \textbf{fix}\ this.\ (Alice_2,\ Bob_2)$

$Alice_2 \stackrel{\text{def}}{=} \lambda d.\ \textbf{let}\ aLoc = sample_{Bern}\ aPrior\ \textbf{in}$
$\quad \textbf{let}\ bLoc = \textbf{sample}\ (\textbf{query}\ (\textbf{snd}\ (this\ ())\ (d - \mathsf{c}_1)))\ \textbf{in}$
$\quad \textbf{let}\ \_ = ScoreLoc\ aLoc\ bLoc\ \textbf{in}\ aLoc$

$Bob_2 \stackrel{\text{def}}{=} \lambda d.\ \textbf{let}\ bLoc = sample_{Bern}\ bPrior\ \textbf{in}$
$\quad \textbf{if}\ d > \mathsf{c}_0\ \textbf{then}$
$\quad\quad \textbf{let}\ aLoc = \textbf{sample}\ (\textbf{query}\ (\textbf{fst}\ (this\ ())\ d))\ \textbf{in}$
$\quad\quad \textbf{let}\ \_ = ScoreLoc\ aLoc\ bLoc\ \textbf{in}\ bLoc$
$\quad \textbf{else}\ bLoc$

$CG_2 \stackrel{\text{def}}{=} \lambda\_.\ \textbf{sample}\ (\textbf{query}\ (\textbf{fst}\ (Agents_2\ ())\ \mathsf{c}_8))$

Figure 8. CG_2 encodes Figure 1 using the core calculus. Unlike CG_2, CG_1 does *not* use nested queries.

Holtzen et al. [2019, §A] pose (and correctly reject) a similar equivalence, which otherwise holds when terms are placed inside queries.

7.3 Coordination Game

Encodings. The right column of Figure 8 uses the core PPL to encode the coordination game in Figure 1. Mutual recursion between the agents is encoded by $Agents_2$, a fixpoint function whose body is a pair; recursive reference is via pair projection. Agents' prior preferences are represented by free variables $aPrior$ and $bPrior$. Now alter the encoding to obtain $Agents_1$, $Alice_1$, Bob_1, and CG_1 (shown in the left column of Figure 8) such that $Alice_1$ and Bob_1 are free of nested queries.

Equivalence. We prove the equivalence $aPrior : \mathbf{R}, bPrior : \mathbf{R} \vDash CG_1\ () \approx_{ctx} CG_2\ () : \mathbf{2}$.

Proof Sketch. As in the previous example, we show contextual equivalence by showing ciu approximation in both directions. Below we focus on the direction $aPrior : \mathbf{R}, bPrior : \mathbf{R} \vDash CG_1\ () \preccurlyeq_{ciu} CG_2\ () : \mathbf{2}$; the other direction is analagous.

We need to show for all γ substituting for $aPrior$ and $bPrior$, for all evaluation contexts K, and for all n, $O^n(K[\gamma CG_1\ ()],\ K[\gamma CG_2\ ()])$ holds. This goal consists of two inequalities, one for μ_{NS} and the other for μ_{TV}. The proof proceeds in a simlar pattern to the previous example, stepping terms on both sides and evolving measures per Lemmas 4.9–4.12 and Lemmas 4.17–4.22.

CG_2 uses nested queries, so calculations on the RHS need to handle normalization factors. The key enabler of this proof is the cancellation of normalization factors in the numerator and the denominator of the RHS. Below we illustrate this for $\mu_{\mathsf{TV}}(K[\gamma CG_2\ ()], V)$. For convenience, let $J_{\mathsf{T}} \stackrel{\text{def}}{=} \textbf{let}\ bLoc = [\cdot]\ \textbf{in let}\ \_ = ScoreLoc\ \textsc{True}\ bLoc\ \textbf{in}\ \textsc{True}$ denote an evaluation context that can emerge during evaluation; J_{F} is similarly defined, substituting False for True. Let c_a be the value that γ substitutes for $aPrior$. The RHS measure of interest is transformed as follows:

$$\mu_{\mathsf{TV}}(K[\gamma CG_2\ ()], V) = \mu_{\mathsf{TV}}(K[\textbf{sample}\ (\textbf{query}\ (\textbf{fst}\ (\gamma Agents_2\ ())\ \mathsf{c}_8))], V)$$

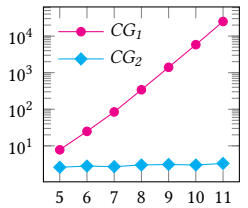
$$= \frac{\mu_{\mathsf{TV}}(K[\textbf{fst}\ (\gamma Agents_2\ ())\ \mathsf{c}_8], V)}{\mu_{\mathsf{NS}}(\textbf{fst}\ (\gamma Agents_2\ ())\ \mathsf{c}_8)} = \frac{\begin{array}{l} a \cdot \mu_{\mathsf{TV}}(K[J_{\mathsf{T}}[\textbf{sample}\ (\textbf{query}\ (\textbf{snd}\ (\gamma Agents_2\ ())\ \mathsf{c}_7))]], V) + \\ (1-a) \cdot \mu_{\mathsf{TV}}(K[J_{\mathsf{F}}[\textbf{sample}\ (\textbf{query}\ (\textbf{snd}\ (\gamma Agents_2\ ())\ \mathsf{c}_7))]], V) \end{array}}{\begin{array}{l} a \cdot \mu_{\mathsf{NS}}(J_{\mathsf{T}}[\textbf{sample}\ (\textbf{query}\ (\textbf{snd}\ (\gamma Agents_2\ ())\ \mathsf{c}_7))]) + \\ (1-a) \cdot \mu_{\mathsf{NS}}(J_{\mathsf{F}}[\textbf{sample}\ (\textbf{query}\ (\textbf{snd}\ (\gamma Agents_2\ ())\ \mathsf{c}_7))]) \end{array}}$$

$$= \frac{a \cdot \frac{\mu_{\mathsf{TV}}(K[J_{\mathsf{T}}[\mathbf{snd}\ (\gamma Agents_2\ ())\ \mathsf{c}_7]], V)}{\mu_{\mathsf{NS}}(\mathbf{snd}\ (\gamma Agents_2\ ())\ \mathsf{c}_7)} + (1-a) \cdot \frac{\mu_{\mathsf{TV}}(K[J_{\mathsf{F}}[\mathbf{snd}\ (\gamma Agents_2\ ())\ \mathsf{c}_7]], V)}{\mu_{\mathsf{NS}}(\mathbf{snd}\ (\gamma Agents_2\ ())\ \mathsf{c}_7)}}{a \cdot \frac{\mu_{\mathsf{NS}}(J_{\mathsf{T}}[\mathbf{snd}\ (\gamma Agents_2\ ())\ \mathsf{c}_7])}{\mu_{\mathsf{NS}}(\mathbf{snd}\ (\gamma Agents_2\ ())\ \mathsf{c}_7)} + (1-a) \cdot \frac{\mu_{\mathsf{NS}}(J_{\mathsf{F}}[\mathbf{snd}\ (\gamma Agents_2\ ())\ \mathsf{c}_7])}{\mu_{\mathsf{NS}}(\mathbf{snd}\ (\gamma Agents_2\ ())\ \mathsf{c}_7)}}$$

The cancellation of normalization factors allows the RHS to match the form of the LHS. The rest of the proof is unsurprising and is not shown here to save tediousness. □

As the proof makes explicit, the equivalence hinges on the cancellation of normalization factors, which is a consequence of Alice's reasoning about Bob's reasoning being independent of Alice's choice—by the problem definition, the agents cannot communicate. Given that the coordination game is the first example Stuhlmüller and Goodman [2014] use to illustrate the modeling power of nested queries, its equivalence to a nested-query-free variant comes as a surprise.

Performance. Nested queries prove semantically irrelevant in this example, but they have implications for inference efficiency. The figure below plots running times (in milliseconds) for CG_1 and CG_2 in logarithmic scale. The inference algorithm at work is WebPPL's default choice for discrete random variables, which enumerates random choices for exact inference.



Running time of CG_1 grows exponentially as the problem size (namely depth of metareasoning) increases, while that of CG_2 is inconspicuous. The contrast results from a simple optimization applied to nested queries: query results are deterministic—they are distributions—so it makes sense to memoize them. In CG_2, because different choices of location at a higher depth can lead to the same query at a lower depth, the same inference problem effectively gets queried multiple times, with subsequent queries fetching the memoized distribution and thus amortizing the cost of the first query. Hence, running time of CG_2 grows linearly, although the trend is indiscernible in the plot because problem sizes are too small.

More nesting is not necessarily better for arbitrary inference algorithms, though. It is possible that Monte Carlo algorithms may have better asymptotic behavior in the absence of nesting, measured by variance or convergence rate.

7.4 Markov Decision Processes

Encodings. Markov decision processes (MDPs) [Howard 1960] are a widely used framework for expressing sequential decision-making problems. Figure 9 shows two generic MDP encodings.

Each encoding consists of two mutually recursive functions, *Action* and *Cost*. *Action* samples an action and uses a **score** $\exp(-cost)$ term to bias the sample towards actions that will yield low cost over time: the score of an action is in proportion to the softmin of its cost.

Cost computes the cost of taking a given action in a given state. The return value of *Cost* sums two parts: the immediate cost obtained in the next state after applying a *Transition* function, and costs to be accumulated in future states (discounted by a factor r). An agent chooses its *nextAction* in *nextState* by using a nested query to recursively reason about how different actions play out in the future. The encoding is reminiscent of Bellman's principle of optimality [Bellman 1957].

Functions *Prior*, *Transition*, *IsTerminalState*, and *ImmediateCost* are domain-specific and possibly stochastic (but free of conditioning). The two robots mentioned in Figure 2, for example, instantiate the generic MDP encodings on the problem domain of 2D-space path planning.

Equivalence. The encodings differ in what the **query** terms encompass. We prove the equivalence $\cdot \vDash MDP_1 \approx_{ctx} MDP_2 : \mathbf{1} \to (\mathbf{R} \to \mathbf{2},\ \mathbf{2} \to \mathbf{R} \to \mathbf{R})$, where states and costs have type $\mathbf{R}$ and actions $\mathbf{2}$.

The proof proceeds similarly to §7.2 and §7.3, transforming measures under small-step reduction. Intuitively, the equivalence holds because the term **snd** (*this* ()) *nextAction nextState*, which the

```
MDP_1 ≝ fix this. (Action_1, Cost_1)

Action_1 ≝ λstate. let action = Prior () in
  let cost = snd (this ()) action state in
  let _ = score exp(−cost) in action

Cost_1 ≝ λaction. λstate.
  let nextState = Transition action state in
  if IsTerminalState nextState then
    ImmediateCost nextState
  else
    let nextAction =
      sample (query (fst (this ()) nextState)) in
    let futureCosts =
      snd (this ()) nextAction nextState in
    ImmediateCost nextState + c_r × futureCosts
```

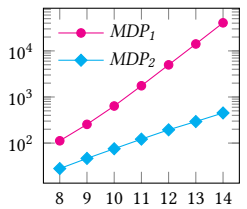
```
MDP_2 ≝ fix this. (Action_2, Cost_2)

Action_2 ≝ λstate. let action = Prior () in
  let cost = snd (this ()) action state in
  let _ = score exp(−cost) in action

Cost_2 ≝ λaction. λstate.
  let nextState = Transition action state in
  if IsTerminalState nextState then
    ImmediateCost nextState
  else
    let futureCosts = sample (query (
      let nextAction = fst (this ()) nextState in
      snd (this ()) nextAction nextState
    )) in
    ImmediateCost nextState + c_r × futureCosts
```

Figure 9. Two generic MDP encodings. They differ in what the **query** terms encompass.

query term in $Cost_2$ additionally encompasses, is free of conditioning effects: although it may further call $Action_2$ and score actions, the scoring is always screened off by an enclosing query and thus unable to influence any random choice made in outer queries. So the proof amounts to showing that the two nested queries have equal model evidence.

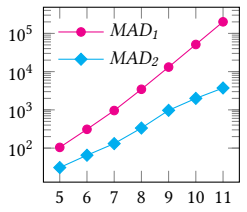
Unlike in §7.3, neither encoding is equivalent to a nested-query-free variant: an agent's choice of action enters into determining the model evidence of the nested query about the agent's future.

Performance. The subtle difference between MDP_1 and MDP_2 has marked performance implications. We instantiated the encodings in WebPPL on a finite-horizon MDP problem; the figure below plots running times of exact inference, increasing the problem size (namely MDP horizon). All nested queries are already memoized. Because nested queries are identified by *nextState* and because different sequences of actions can lead to the same state, the memoized result of a nested query can be reused for subsequent, identical queries. Running time increases exponentially with both encodings, because the number of distinct queries grows exponentially.



Importantly, MDP_1 scales exponentially worse than MDP_2: with a horizon of 14, inference of MDP_1 takes about 100 times as much time. The contrast is because each query in MDP_1 has to solve a larger inference problem than its counterpart in MDP_2 does. In MDP_1, within a query of horizon h, recursive call **snd** (*this ()*) *nextAction nextState* happens h times and thus sampling of **query** (**fst** (*this ()*) *nextState*) happens $h-1$ times, meaning that the enclosing query of the recursive calls has to exhaust all α^h executions of a search tree of depth h (where α is the branching factor). By contrast, in MDP_2, every recursive call happens within a query nested inside its caller, so each query works on a search tree of depth $O(1)$. Hence, MDP_2 optimizes MDP_1 while preserving semantics.

7.5 Multi-Agent Sequential Decision Making



The encodings and equivalence in §7.4 can be generalized to sequential decision making involving multiple agents who recursively reason about one another. Stuhlmüller and Goodman [2014, §3.3] give a generic encoding. This encoding, which we call MAD_1, is analogous to MDP_1. We construct a provably equivalent encoding, MAD_2, analogous to MDP_2. We instantiated

the encodings on a two-agent game: the agents roam in a grid world to collect prizes, but they are penalized if they get too close to each other. Notice that unlike the coordination game, neither MAD_1 nor MAD_2 is equivalent to a nested-query-free variant: agents reason recursively about each other, fully aware of the moves the other agent has made. The figure above plots running times: as the number of total rounds of play increases, MAD_2 scales exponentially better than MAD_1.

Our theory allows for formally relating probabilistic programs concerning usage of nested queries. The proven equivalences justify program optimizations that accelerate some inference algorithms by factorizing programs into recursively nested queries (§7.3) or by rearranging recursively nested queries (§7.4 and §7.5). Future research can explore ways to automate such optimizations.

8 RELATED WORK AND DISCUSSION

Operational semantics. Defining measure semantics by integrating a *sampling-based* operational semantics is most thoroughly investigated by Borgström et al. [2016], Culpepper and Cobb [2017], Wand et al. [2018], and Szymczak and Katoen [2019]. Culpepper and Cobb study a typed lambda calculus without recursion, Borgström et al. and Wand et al. study untyped lambda calculi (thus with recursion), and Szymczak and Katoen study an imperative while-language. Thus, except for Culpepper and Cobb, all need to step-index a density function and take the limit of its integration. None is concerned with nested queries.

These semantics represent the source of randomness differently. Culpepper and Cobb, Wand et al., and Szymczak and Katoen all use the entropy space $\mathbb{S}$ (Definition 4.1), where an entropy value can be infinitely split to generate samples. Borgström et al. use a measure space $\mathbb{T}$ of finite *traces*, i.e., sequences of sample values. Infinite traces are also possible [Park et al. 2008]. The stock measure over $\mathbb{S}$ is a probability measure, whereas that over $\mathbb{T}$ is σ-finite. Wand et al. show that the choice of $\mathbb{S}$ or $\mathbb{T}$ does not constitute fundamental differences in assigning meaning to probabilistic programs: the measure spaces result in identical value measures.

Borgström et al. [2016] and Staton et al. [2016] give *distribution-based* operational semantics, directly associating transitions with distributions rather than integrating over an entropy space. Staton et al. study a typed lambda calculus without recursion but with nested queries. The operational semantics assumes a function that produces normalization factors for nested queries, so in this sense the fixpoint knot remains to be tied.

Logical relations and contextual equivalence. Bizjak and Birkedal [2015], Culpepper and Cobb [2017], and Wand et al. [2018] use logical relations to establish contextual equivalence for probabilistic programs. Bizjak and Birkedal consider only discrete probabilistic programs *without conditioning*, while Culpepper and Cobb and Wand et al. support continuous distributions and scoring. Bizjak and Birkedal and Wand et al. employ biorthogonality [Pitts and Stark 1998] and step indexing [Ahmed 2006] to define logical relations. Only the formal development of Culpepper and Cobb is mechanized, and we reuse their axiomatization of $\mathbb{R}^+$ and Lebesgue integration.

Indistinguishability. Bizjak and Birkedal take it to mean identical probabilities of termination in a PPL without conditioning. Wand et al. take it to mean identical unnormalized *value* measures, so their contextual equivalence may not distinguish terms having different normalized measures. For example, it would consider e_1 and e_2 below equivalent because $\mu_{\mathsf{TV}}(K[e_1], V) = \mu_{\mathsf{TV}}(K[e_2], V)$ for all K and V, despite that e_2 diverges half the time while e_1 terminates certainly:

$$e_1 \stackrel{\text{def}}{=} \textbf{score}\ \mathsf{c}_{0.5}; \mathsf{c}_1 \qquad e_2 \stackrel{\text{def}}{=} \textbf{let}\ \mathit{diverge} = \textbf{fix}\ \mathit{this}.\ \mathit{this}\ ()\ \textbf{in if}\ \mathit{sample}_{\mathit{Bern}}\ \mathsf{c}_{0.5}\ \textbf{then}\ \mathit{diverge}\ ()\ \textbf{else}\ \mathsf{c}_1$$

Our notion of indistinguishability is also more discriminative than a variant comparing *normalized* measures. This variant is useful to Olmedo et al. [2018], who establish (via a weakest preexpectation

semantics) the correctness of a program transformation that compiles conditioning away, thus effectively declaring conditioning as syntactic sugar. We consider normalization to be triggered explicitly by the **query** construct and consider programs to otherwise denote unnormalized distributions.

Nontermination. Giving semantics to PPLs in the presence of conditioning and nontermination is the subject of recent studies. Olmedo et al. [2018], Bichsel et al. [2018], and Szymczak and Katoen [2019] all advocate semantics that distinguish programs—like e_1 and e_2 above—that have different nontermination measures. Bichsel et al. additionally distinguish programmer mistakes (namely stuck executions) from failing a hard constraint (namely **score** c_0); like Szymczak and Katoen, we lump them together for simplicity. Probabilistic programs with positive nontermination measure offer useful modeling power; Icard [2017] and Szymczak and Katoen give examples.

A subtle equivalence our semantics allows is $\cdot \vDash \textbf{score}\ \mathsf{c}_0 \approx_{ctx} (\textbf{fix}\ \mathit{this}.\ \textbf{score}\ \mathsf{c}_{0.5}; \mathit{this}\ ())\ () : \mathbf{1}$. The RHS computation loops, with its step-indexed model evidence approaching zero (but never reaching zero). This equivalence is however sensitive to the treatment of zero-model-evidence queries: treating **sample** (**query** (**score** c_0)) as stuck, rather than denoting a point mass distribution at **exn**, would invalidate the result, as the context **sample** (**query** $[\cdot]$) would distinguish the terms.

Scoring and normalization. Staton et al. [2016] show that unbounded **score** can result in infinite model evidence, causing normalization factors to be ill-defined. Borgström et al. [2016] show that even statically bounded **score** can cause infinite model evidence in the presence of recursion. Bichsel et al. [2018] show that statically bounded **score** can even cause the limit of step-indexed model evidence to not exist when recursion is present. We compile a list of examples in §A.

Borgström et al. and Szymczak and Katoen reconcile the tension between **score** and recursion by imposing the restriction that the argument of **score** should not exceed 1. Culpepper and Cobb remark that there is no syntactic restriction that can exclude programs having ill-defined model evidence without also limiting expressive power. They and Wand et al. need not impose restrictions because their semantics concern only unnormalized measures. We inherit the restriction of 1-bounded **score** to ensure that measure of nonterminating executions is well-defined, though showing this in the presence of nested queries is more difficult. Bounded scoring limits expressive power in general; they are only as expressive as hard constraints (cf. E5 in §7.1). We observe that in metareasoning applications, scoring in recursively nested queries tends to express goals and preferences of agents, rather than observed data in statistical modeling, which in general needs unbounded scoring. This observation seems to explain why examples in this paper are compatible with 1-bounded **score**.

Denotational semantics. Denotational semantics of probabilistic programs has received significant development (e.g., Kozen [1979], Staton et al. [2016], Heunen et al. [2017]). Not many have considered nested queries, though. Notably, Staton et al. give a denotational semantics to a calculus with scoring and nested queries, but without recursion. They show an example of denotational equality that resembles sequential Monte Carlo [Liu and Chen 1998], which can be viewed as using nested queries to resample from normalized distributions. The theoretical development is not readily applicable to equivalences motivating our study, which require the expressive power of nesting queries recursively inside queries. Nonetheless, since the subprobability monad of Staton et al. has a CPO structure, it should be possible to adapt the approach to address recursion. Future research can study whether this denotational equality coincides with contextual equivalence.

Approximate Inference. Operational semantics based on entropy or traces idealize the Monte Carlo inference method of likelihood weighting, by integrating over sample spaces and by infinitely tracing executions (i.e., taking limits). Relaxing integration to sums and bounding execution traces, our semantics corresponds to a simplistic implementation of nested Monte Carlo (NMC). NMC is known to incur computational costs exponential in levels of nesting [Rainforth 2018], as each

sample of an outer random variable in general defines a different inference problem for the inner query—a consequence of integration happening inside operational reduction. Efficient, general approximate inference methods for recursively nested queries are still outstanding. Unless a program has strong independence structures (e.g., coordination games), asymptotically exact inference is likely intractable.

Potpourri. Nested queries bear a resemblance to evaluation within evaluation, which is one form of computational reflection [Smith 1982]. In a historically damning paper on fexprs (a mechanism for total reification), Wand [1998] proves that too much reification can hinder all optimizations, admitting only "trivial" equalities. By contrast, nested queries have a richer equational theory.

Amortized exact inference (of the sort mentioned in §7.3) exists in varied forms [Stuhlmüller and Goodman 2012; Gershman and Goodman 2014; Holtzen et al. 2020]. They consist in exploiting conditional independence to factorize an overall inference problem into subproblems whose inference cost can be shared. As our example equivalences suggest, nested queries can be viewed as a language construct for compositionally delimiting Bayesian-inference subproblems within a larger program.

The Church PPL [Goodman et al. 2008] was first to advertise nested queries. Many PPLs support them, including the logic PPL ProbLog [Mantadelis and Janssens 2011]. Ackerman et al. [2011], Freer et al. [2014], and Huang et al. [2020] study the computability theory of **query**. Tavares et al. [2019] and Tolpin et al. [2021] introduce *random conditional distributions* and *stochastic conditioning*, respectively, in similar ways generalizing nested queries.

9 CONCLUSION

Nested queries, coupled with recursion, are a powerful language feature found in many PPLs. We have developed an operational-semantics-based foundation for them. The development consists of a recursive operational and measure semantics, capturing the semantic essence of nested queries, and a logical-relations model and its metatheories, granting powerful equational reasoning principles. Key, machine-checked results include well-definedness of limit measures and soundness of the logical-relations model. We demonstrate the results' usefulness by applying them in reasoning about Bayesian models of intelligent agents, formally proving practically relevant equivalences that were not suggested and could not be proved before.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable feedback. We thank Elena Glassman, Jianlin Li, Andrew Myers, François-René Rideau, Armando Solar-Lezama, and Zekun Wang for discussions and help. This work was partially supported by NSERC. The views and opinions expressed are those of the authors and do not necessarily reflect the position of any government agency.

A EXAMPLES OF SCORE CAUSING BAD MODEL EVIDENCE

In this appendix, we compile a list of examples from prior work to show how **score** can give rise to bad model evidence.

- Staton et al. [2016] show that unbounded **score** can result in infinite model evidence:

$$\textbf{let } x = \textbf{sample } \mathit{Exponential}(1.0) \textbf{ in score } \mathit{exp}(x)$$

  The value taken by random variable x is in general unbounded, since the support of the exponential distribution is $[0, \infty)$. The model evidence of the program is ∞.

- Borgström et al. [2016] show that even statically bounded **score** can cause infinite model evidence in the presence of recursion:

$$\textbf{let } f = (\textbf{fix } \mathit{this}.\, \textbf{if sample } \mathit{Bernoulli}(0.5) \textbf{ then let } \_ = \textbf{score } 2 \textbf{ in } \mathit{this}\ () \textbf{ else } ())\ \textbf{ in } f\ ()$$

  The scoring in the program is statically bounded, but recursion causes the model evidence of the program to be ∞. Also notice that the program terminates a.s.: sans scoring, the measure of nonterminating executions is zero.

- In the above example, the limit of step-indexed model evidence still exists, albeit ∞. Bichsel et al. [2018] further show that statically bounded **score** can even cause the limit of step-indexed model evidence to not exist when recursion is present.

$$\textbf{let } f = (\textbf{fix } \mathit{this}.\, \lambda x.\, \textbf{let } \_ = \textbf{score } (\textbf{if } x \textbf{ then } 2 \textbf{ else } 0.5) \textbf{ in } \mathit{this}\ ()\ (\neg\, x))\ \textbf{ in } f\ ()\ \textsc{False}$$

  The scoring in the program is statically bounded, but recursion causes the model evidence to not exist. The score of the only execution trace keeps switching between 2 and 1.

B AN EXAMPLE OF ρ^n_{NS} NOT CONVERGING

§4.2 mentions the possibility that ρ^n_{NS} may not converge as n goes to infinity. This appendix gives an example. Notice that unlike in §A, we are already imposing the restriction of 1-bounded **score**.

Consider the program e below:

$$\begin{aligned} e \stackrel{\text{def}}{=}\ & \textbf{let } f = \textbf{fix } \mathit{this}.\, (\textbf{let } \_ = \textbf{sample } (\textbf{query} \\ & \quad \textbf{let } x = \textbf{sample Unif in} \\ & \quad \textbf{score } (\textbf{if } x < \mathsf{c}_{0.5} \textbf{ then } \mathsf{c}_1 \textbf{ else } \mathsf{c}_{0.8})) \textbf{ in } \mathit{this}\ ()) \\ & \textbf{in } f\ () \end{aligned}$$

Program e invokes a recursive function, repeatedly querying a computation that makes a random choice and scores it.

Let σ^* be an entropy value leading to a sequence of random choices $\{x^{(i)}\}_{i \in \mathbb{N}^*}$. The table below shows the value of $\rho^n_{\mathsf{NS}}(\sigma^*, e)$ as n increases. Entropy σ^* is so picked that $x^{(i)} < 0.5$ for $i \in \{1, 3, 4, 5\}$ and $x^{(i)} \geq 0.5$ for $i \in \{2, 6, 7\}$.

n	8	16	24	32	40	48	56	...
$\rho^n_{\mathsf{NS}}(\sigma^*, e)$	1.111	.988	1.097	1.219	1.355	1.204	1.070	...

Every two recursive invocations are 8 steps apart, and the table captures the value of $\rho^n_{\mathsf{NS}}(\sigma, e^*)$ under step budgets that are multiples of 8. Notice that the model evidence of each nested query is $0.5 \times 1 + 0.5 \times 0.8 = 0.9$, so $\rho^{8i}_{\mathsf{NS}}(\sigma^*, e) = (1/0.9)^k (0.8/0.9)^{i-k}$, where k, determined by σ^*, is the number of recursive calls in which the random draw $x^{(i)}$ is less than 0.5. The sequence is clearly not monotone, and does not necessarily converge for the infinite sequence of random choices $\{x^{(i)}\}_{i \in \mathbb{N}^*}$ generated by σ^*.

REFERENCES

Nathanael L. Ackerman, Cameron E. Freer, and Daniel M. Roy. 2011. Noncomputable Conditional Distributions. In *Symp. on Logic In Computer Science (LICS)*. https://doi.org/10.1109/LICS.2011.49

Amal Ahmed. 2006. Step-Indexed Syntactic Logical Relations for Recursive and Quantified Types. In *European Symp. on Programming (ESOP)*. https://doi.org/10.1007/11693024_6

Andrew W. Appel and David McAllester. 2001. An Indexed Model of Recursive Types for Foundational Proof-Carrying Code. *ACM Tran. on Programming Languages and Systems (TOPLAS)* 23, 5 (Sept. 2001). https://doi.org/10.1145/504709.504712

Richard E. Bellman. 1957. *Dynamic Programming*. Princeton University Press.

Benjamin Bichsel, Timon Gehr, and Martin Vechev. 2018. Fine-Grained Semantics for Probabilistic Programs. In *European Symp. on Programming (ESOP)*. https://doi.org/10.1007/978-3-319-89884-1_6

Aleš Bizjak and Lars Birkedal. 2015. Step-Indexed Logical Relations for Probability. In *Int'l Conf. on Foundations of Software Science and Computation Structures (FoSSaCS)*. https://doi.org/10.1007/978-3-662-46678-0_18

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

Coq [n.d.]. The Coq proof assistant. https://coq.inria.fr.

Ryan Culpepper and Andrew Cobb. 2017. Contextual Equivalence for Probabilistic Programs with Continuous Random Variables and Scoring. In *European Symp. on Programming (ESOP)*. https://doi.org/10.1007/978-3-662-54434-1_14

Owain Evans, Andreas Stuhlmüller, John Salvatier, and Daniel Filan. 2017. Modeling Agents with Probabilistic Programs. https://agentmodels.org.

Matthias Felleisen and Robert Hieb. 1992. The Revised Report on the Syntactic Theories of Sequential Control and State. *Theoretical Computer Science* 103, 2 (1992). https://doi.org/10.1016/0304-3975(92)90014-7

Cameron E. Freer, Daniel M. Roy, and Joshua B. Tenenbaum. 2014. Towards Common Sense Reasoning via Conditional Simulation: Legacies of Turing in Artificial Intelligence. *Turing's Legacy* 42 (2014). arXiv:1212.4799

Samuel J. Gershman and Noah D. Goodman. 2014. Amortized Inference in Probabilistic Reasoning. In *Annual Meeting of the Cognitive Science Society (CogSci)*. https://cogsci.mindmodeling.org/2014/papers/098/paper098.pdf

Noah D. Goodman, Vikash K. Mansinghka, Daniel Roy, Keith Bonawitz, and Joshua B. Tenenbaum. 2008. Church: A Language for Generative Models. In *Uncertainty in Artificial Intelligence (UAI)*. arXiv:1206.3255

Noah D Goodman and Andreas Stuhlmüller. 2014. The Design and Implementation of Probabilistic Programming Languages. http://dippl.org

Chris Heunen, Ohad Kammar, Sam Staton, and Hongseok Yang. 2017. A Convenient Category for Higher-Order Probability Theory. In *Symp. on Logic In Computer Science (LICS)*. https://doi.org/10.1109/LICS.2017.8005137

Steven Holtzen, Todd Millstein, and Guy Van den Broeck. 2019. Symbolic Exact Inference for Discrete Probabilistic Programs. (2019). arXiv:1904.02079

Steven Holtzen, Guy Van den Broeck, and Todd Millstein. 2020. Scaling Exact Inference for Discrete Probabilistic Programs. *Proc. of the ACM on Programming Languages (PACMPL)* 4, OOPSLA (2020). https://doi.org/10.1145/3428208

Ronald A. Howard. 1960. *Dynamic Programming and Markov Processes*. The MIT Press.

Daniel Huang, Greg Morrisett, and Bas Spitters. 2020. *An Application of Computable Distributions to the Semantics of Probabilistic Programs*. Cambridge University Press. https://doi.org/10.1017/9781108770750.004

Thomas Icard. 2017. Beyond Almost-Sure Termination. In *Annual Meeting of the Cognitive Science Society (CogSci)*. https://cogsci.mindmodeling.org/2017/papers/0430/paper0430.pdf

Dexter Kozen. 1979. Semantics of Probabilistic Programs. In *Annual Symposium on Foundations of Computer Science (SFCS)*. https://doi.org/10.1109/SFCS.1979.38

Jun S. Liu and Rong Chen. 1998. Sequential Monte Carlo Methods for Dynamic Systems. *Journal of the American Statistical Association* 93, 443 (1998). https://doi.org/10.1080/01621459.1998.10473765

Theofrastos Mantadelis and Gerda Janssens. 2011. Nesting Probabilistic Inference. (2011). arXiv:1112.3785

Ian Mason and Carolyn Talcott. 1991. Equivalence in Functional Languages with Effects. *Journal of Functional Programming (JFP)* 1, 3 (1991). https://doi.org/10.1017/S0956796800000125

Micaela Mayero. 2001. *Formalisation et automatisation de preuves en analyses réelle et numérique.* Ph.D. Dissertation. Université Paris VI.

James H. Morris, Jr. 1968. *Lambda-Calculus Models of Programming Languages.* Ph.D. Dissertation. Massachusetts Institute of Technology. http://hdl.handle.net/1721.1/64850

Federico Olmedo, Friedrich Gretz, Nils Jansen, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Annabelle McIver. 2018. Conditioning in Probabilistic Programming. *ACM Tran. on Programming Languages and Systems (TOPLAS)* 40, 1 (2018). https://doi.org/10.1145/3156018

Sungwoo Park, Frank Pfenning, and Sebastian Thrun. 2008. A Probabilistic Language Based on Sampling Functions. *ACM Tran. on Programming Languages and Systems (TOPLAS)* 31, 1 (Dec. 2008). https://doi.org/10.1145/1452044.1452048

Andrew Pitts and Ian Stark. 1998. Operational Reasoning for Functions with Local State. In *Higher Order Operational Techniques in Semantics.* https://homepages.inf.ed.ac.uk/stark/operfl.pdf

Tom Rainforth. 2018. Nesting Probabilistic Programs. In *Uncertainty in Artificial Intelligence (UAI).* arXiv:1803.06328

Thomas C. Schelling. 1980. *The Strategy of Conflict.* Harvard University Press.

Iris Rubi Seaman, Jan-Willem van de Meent, and David Wingate. 2020. Nested Reasoning About Autonomous Agents Using Probabilistic Programs. (2020). arXiv:1812.01569

Brian Cantwell Smith. 1982. *Procedural Reflection in Programming Languages.* Ph.D. Dissertation. Massachusetts Institute of Technology.

Rick Statman. 1985. Logical Relations and the Typed λ-calculus. *Information and Control* 65, 2 (1985). https://doi.org/10.1016/S0019-9958(85)80001-2

Sam Staton. 2017. Commutative Semantics for Probabilistic Programming. In *European Symp. on Programming (ESOP).* https://doi.org/10.1007/978-3-662-54434-1_32

Sam Staton, Frank Wood, Hongseok Yang, Chris Heunen, and Ohad Kammar. 2016. Semantics for Probabilistic Programming: Higher-Order Functions, Continuous Distributions, and Soft Constraints. In *Symp. on Logic In Computer Science (LICS).* https://doi.org/10.1145/2933575.2935313

Elias M. Stein and Rami Shakarchi. 2005. *Real Analysis: Measure Theory, Integration, and Hilbert Spaces.* Princeton University Press. https://doi.org/10.2307/j.ctvd58v18

Andreas Stuhlmüller and Noah D. Goodman. 2012. A Dynamic Programming Algorithm for Inference in Recursive Probabilistic Programs. (2012). arXiv:1206.3555

Andreas Stuhlmüller and Noah D. Goodman. 2014. Reasoning about Reasoning by Nested Conditioning: Modeling Theory of Mind with Probabilistic Programs. *Cognitive Systems Research* 28 (2014). https://doi.org/10.1016/j.cogsys.2013.07.003

Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction.* MIT press.

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

Terence Tao. 2011. *An Introduction to Measure Theory.* American Mathematical Society. https://doi.org/10.1090/gsm/126

Zenna Tavares, Xin Zhang, Edgar Minaysan, Javier Burroni, Rajesh Ranganath, and Armando Solar-Lezama. 2019. The Random Conditional Distribution for Higher-Order Probabilistic Inference. (2019). arXiv:1903.10556

David Tolpin, Yuan Zhou, and Hongseok Yang. 2021. Probabilistic Programs with Stochastic Conditioning. (2021). arXiv:2010.00282

Mitchell Wand. 1998. The Theory of Fexprs is Trivial. *Lisp and Symbolic Computation* 10, 3 (1998). https://doi.org/10.1023/A:1007720632734

Mitchell Wand, Ryan Culpepper, Theophilos Giannakopoulos, and Andrew Cobb. 2018. Contextual Equivalence for a Probabilistic Language with Continuous Random Variables and Recursion. *Proc. of the ACM on Programming Languages (PACMPL)* 2, ICFP (2018). https://doi.org/10.1145/3236782