Step-Indexed Logical Relations for Countable Nondeterminism and Probabilistic Choice

ALEJANDRO AGUIRRE and LARS BIRKEDAL, Aarhus University, Denmark

Developing denotational models for higher-order languages that combine probabilistic and nondeterministic choice is known to be very challenging. In this paper, we propose an alternative approach based on operational techniques. We study a higher-order language combining parametric polymorphism, recursive types, discrete probabilistic choice and countable nondeterminism. We define probabilistic generalizations of may- and must-termination as the optimal and pessimal probabilities of termination. Then we define step-indexed logical relations and show that they are sound and complete with respect to the induced contextual preorders. For may-equivalence we use step-indexing over the natural numbers whereas for must-equivalence we index over the countable ordinals. We then show than the probabilities of may- and must-termination coincide with the maximal and minimal probabilities of termination under all schedulers. Finally we derive the equational theory induced by contextual equivalence and show that it validates the distributive combination of the algebraic theories for probabilistic and nondeterministic choice.

CCS Concepts: • **Theory of computation → Probabilistic computation**; **Operational semantics**; **Program verification**.

Additional Key Words and Phrases: Functional Languages, Probabilistic programming, Logical Relations

ACM Reference Format:
Alejandro Aguirre and Lars Birkedal. 2023. Step-Indexed Logical Relations for Countable Nondeterminism and Probabilistic Choice. *Proc. ACM Program. Lang.* 7, POPL, Article 2 (January 2023), 28 pages. https://doi.org/10.1145/3571195

1 INTRODUCTION

Probabilistic programming languages are languages extended with commands that generate random values by sampling them from a particular probability distribution. These can be used, for instance, to write randomized algorithms, to implement encryption and decryption protocols, or to write statistical models.

Nondeterminism is a behavior that arises when studying programs that depend on an unknown external source, such as the user input or a scheduler that selects which thread of a concurrent program to run. This is often modeled by a binary operator that chooses nondeterministically between two programs. Countable nondeterminism, more concretely, arises in the setting of concurrent execution under a fair scheduler, in which every thread will be eventually allowed to run after a finite, but arbitrarily large number of cycles.

Consider for instance the following pseudocode:

$$x = 0; \mathsf{fork}\{\mathsf{while}(\mathsf{true})\{\mathsf{x} :=\!\mathsf{!x} + 1\}\}; f(!x)$$

Authors' address: Alejandro Aguirre, alejandro@cs.au.dk; Lars Birkedal, birkedal@cs.au.dk, Aarhus University, Åbogade 34, 8200, Aarhus, Denmark.

2475-1421/2023/1-ART2
https://doi.org/10.1145/3571195

This program first initializes a variable x to 0. Then it forks a thread that loops while incrementing x and at at some point calls f with the value of x at that point. Under a fair scheduler, this value of x is arbitrarily large, but finite: the parent thread must be scheduled at some point.

In the sequential setting we will consider, this behavior can be captured by adding to our language a nullary operator that evaluates in one step to any nondeterministically chosen natural number. Note that binary nondeterminism does not suffice to capture this behavior, since a loop that chooses between increasing a variable or stopping will also include the possibility of diverging.

The combination of probabilistic and nondeterministic choice has been the object of study of many articles over more than three decades [Apt and Plotkin 1986; Bonchi et al. 2019; Jacobs 2021; Mio et al. 2021; Mislove 2000; Varacca and Winskel 2006]. In denotational semantics, it is well-known that probabilistic and nondeterministic choice can be modeled by the probability and the powerset monad, respectively, but there does not exist a distributive law between them that allows us to form a combined monad [Varacca and Winskel 2006]. From an algebraic point of view, the distributive law would correspond to the equation

$$e_1 \oplus (e_2 \;\mathbf{or}\; e_3) = (e_1 \oplus e_2) \;\mathbf{or}\; (e_1 \oplus e_3),$$

where $\oplus$ and $\mathbf{or}$ represent binary probabilistic and binary nondeterministic choice, respectively. Varacca and Winskel [2006] propose two solutions to this problem. The first one consists of using the monad of indexed valuations to model probabilistic choice. This monad has a distributive law over the powerset monad, but it does not satisfy the equation $e \oplus e = e$. The second option consists of identifying a monad that simultaneously models probabilistic and nondeterministic choice and distributivity between them; this is the monad of nonempty convex sets of probability distributions.

In this work, rather than using denotational semantics, we instead study the combination of probabilistic and nondeterministic choice from an operational point of view. Operational techniques based on step-indexed logical relations have proven to be successful to study contextual equivalence of higher-order programs in a variety of settings [Ahmed 2006; Birkedal et al. 2012; Pitts and Stark 1998; Turon et al. 2013]. Generally, two expressions are considered contextually equivalent if they exhibit the same termination behavior under any context. Step-indexed relations have been extended to languages with probabilistic and countable nondeterminism separately. In the probabilistic setting [Bizjak and Birkedal 2015; Wand et al. 2018; Zhang and Amin 2022] the behavior used to define equivalence is the probability of termination. In the nondeterministic setting [Birkedal et al. 2013], there are actually two notions of equivalence that are of interest: one based on may-termination (i.e., there is a set of choices that makes the program evaluate to a value) and another based on must-termination (there does not exist a set of choices that makes the program diverge or get stuck); and whereas indexing over the natural numbers is adequate for may-termination, for must-termination one must step-index over the countable ordinals.[1]

Contributions.

- We define probabilistic versions of may- and must-termination for programs written in a typed higher-order language that combines impredicative polymorphism, recursive types, and probabilistic and countable nondeterministic choice.
- We define two logical relations to reason about probabilistic may- and must- contextual equivalence. The former is step-indexed over the naturals, and the latter is step-indexed over the countable ordinals. We prove that they are sound and complete, and we further show that they extend the notions of contextual equivalence defined in previous work [Birkedal et al. 2013; Bizjak and Birkedal 2015].

[1] It is in fact sufficient to step-index up to the least nonrecursive ordinal

- We define an alternative notion of probabilistic termination parametrized by a scheduler that selects which nondeterministic choice to make. We show that probabilistic may- and must-termination coincide, respectively, with the optimal and pessimal probability of termination under all schedulers.
- We apply our contextual equivalence relation to case studies. In particular, we show that it satisfies both the algebraic theories for probabilistic and nondeterministic choice, and the distributive law between them.

Structure of the paper. This paper is structured as follows: we first introduce the syntax and operational semantics of the language we study (§2). Then, we present the notions of observations that we will use to define contextual equivalence (§3). After this, we define two notions of contextual equivalence, and define two sound and complete step-indexed logical relations (§4). Then, we define an alternate notion of observation based on schedulers, and show that it coincides with the one defined earlier (§5). Afterwards, we study the algebraic theory induced by contextual equivalence, and present other examples (§6). Finally, we discuss related work (§7), and conclude (§8). The full version of the paper includes an Appendix with omitted proofs.[2]

2 LANGUAGE

2.1 Syntax

We consider a version of System F with product, sum, universal, and recursive types, as well as a native type of natural numbers (for simplicity we assume $\mathbb{N}$ starts at 1), which we extend with two choice operators. The first choice operator "$\mathsf{rand}\ n$" represents a uniform probabilistic choice over the set $\{1, \ldots, n\}$. The second choice operator, denoted "?", represents a nondeterministic choice over all the natural numbers. Although our language does not feature concurrency, this operator captures the behavior of the concurrent increment loop presented in the introduction. The concrete operational semantics of these operators is presented in the next subsection. While the main focus of the present work is on the combination of probabilistic and nondeterministic choice, we include universal and recursive types to make sure that the techniques we employ scale to those important language features. This also makes our target language more expressive as a setting in which to model other languages including features like concurrency.

The syntax of types, values, expressions and evaluation contexts are defined as follows:

$$\begin{aligned}
\tau, \sigma &::= 1 \mid \mathsf{nat} \mid \alpha \mid \tau \to \sigma \mid \tau \times \sigma \mid \tau + \sigma \mid \forall \alpha.\tau \mid \mu\alpha.\tau \\
v &::= \langle\rangle \mid x \mid \underline{n}\ (n \in \mathbb{N}) \mid \lambda x.e \mid \langle v, v' \rangle \mid \mathsf{inl}(v) \mid \mathsf{inr}(v) \mid \mathsf{fold}\ (v) \mid \Lambda.e \\
e &::= \langle\rangle \mid x \mid \underline{n}\ (n \in \mathbb{N}) \mid \lambda x.e \mid e\ e' \mid \langle e, e' \rangle \mid \pi_i e \mid \mathsf{inl}(e) \mid \mathsf{inr}(e) \mid \mathsf{case}(e, x_1.e_1, x_2.e_2) \\
&\qquad \mid \mathsf{rand}\ e \mid ? \mid \mathsf{fold}\ e \mid \mathsf{unfold}\ e \mid \Lambda.e \mid e\_ \mid \mathrm{S}\ e \mid \mathrm{P}\ e \\
E &::= [\,] \mid E\ e \mid v\ E \mid \langle E, e \rangle \mid \langle v, E \rangle \mid \mathsf{inl}(E) \mid \mathsf{inr}(E) \mid \mathsf{fold}\ E \mid \mathsf{unfold}\ E \mid \mathsf{case}(E, x_1.e_1, x_2.e_2) \\
&\qquad \mid \mathsf{rand}\ E \mid E\_
\end{aligned}$$

A type formation context Δ is a finite set of type variables. A typing context Γ is a finite partial map from term variables to types. A typing formation judgment is a tuple $\Delta \vdash \tau$, where Δ is a type formation context and τ is a type (by extension, we write $\Delta \vdash \Gamma$ when every type in the codomain of Γ is well-formed under Δ). A typing judgment is a tuple $\Delta \mid \Gamma \vdash e : \tau$, where Δ is a type formation context, Γ is a typing context, e is an expression and τ is a type. The set of valid typing formation and typing judgments are defined inductively as usual; we show a few selected rules in fig. 1. We let *Type*(Δ) denote the set of types τ such that $\Delta \vdash \tau$, and *Type* denotes the set of closed types. For

[2] https://cs.au.dk/ birke/papers/step-nondet-prob.pdf

$$\frac{(x\colon\tau)\in\Gamma \qquad \Delta\vdash\Gamma}{\Delta\mid\Gamma\vdash x\colon\tau} \qquad \frac{\Delta\mid\Gamma,x\colon\sigma\vdash e\colon\tau}{\Delta\mid\Gamma\vdash\lambda x.e\colon\sigma\to\tau} \qquad \frac{\Delta\mid\Gamma\vdash e_1\colon\sigma\to\tau \qquad \Delta\mid\Gamma\vdash e_2\colon\sigma}{\Delta\mid\Gamma\vdash e_1\;e_2\colon\tau}$$

$$\frac{\Delta\mid\Gamma\vdash e\colon\mathsf{nat}}{\Delta\mid\Gamma\vdash \mathsf{rand}\;e_1\colon\mathsf{nat}} \qquad \frac{}{\Delta\mid\Gamma\vdash\,?\colon\mathsf{nat}} \qquad \frac{\Delta\mid\Gamma\vdash e\colon\tau[\mu\alpha.\tau/\alpha]}{\Delta\mid\Gamma\vdash\mathsf{fold}\,e\colon\mu\alpha.\tau} \qquad \frac{\Delta\mid\Gamma\vdash e\colon\mu\alpha.\tau}{\Delta\mid\Gamma\vdash\mathsf{unfold}\,e\colon\tau[\mu\alpha.\tau/\alpha]}$$

$$\frac{\Delta,\alpha\mid\Gamma\vdash e\colon\tau}{\Delta\mid\Gamma\vdash\Lambda.e\colon\forall\alpha.\tau} \qquad \frac{\Delta\mid\Gamma\vdash e\colon\forall\alpha.\tau \qquad \Delta\vdash\sigma}{\Delta\mid\Gamma\vdash e\_\colon\tau[\sigma/\alpha]} \qquad \frac{\Delta\mid\Gamma\vdash e\colon\mathbb{N}}{\Delta\mid\Gamma\vdash\mathsf{S}\,e\colon\mathbb{N}} \qquad \frac{\Delta\mid\Gamma\vdash e\colon\mathbb{N}}{\Delta\mid\Gamma\vdash\mathsf{P}\,e\colon\mathbb{N}}$$

Fig. 1. Selected typing rules

a type τ, $\mathit{Val}(\tau)$ is the set of values v such that $\emptyset\mid\emptyset\vdash v\colon\tau$; $\mathit{Expr}(\tau)$ is the set of expressions e such that $\emptyset\mid\emptyset\vdash e\colon\tau$; and $\mathit{Ectx}(\tau)$ is the set of evaluation contexts $E\in\mathit{Ectx}$ such that for any $e\in\mathit{Expr}(\tau)$, there exists $\sigma\in\mathit{Type}$ such that $\emptyset\mid\emptyset\vdash E[e]\colon\sigma$ (in which case we also write $\vdash E\colon\tau\Rightarrow\sigma$). We use Val and Expr to denote the sets of all closed values and all closed expressions respectively, and we embed Val into Expr as usual.

Given $\delta\colon\Delta\to\mathit{Type}$ and $\tau\in\mathit{Type}(\Delta)$ we write $\tau\delta$ for the usual capture-avoiding substitution of every free type variable α in τ by $\delta(\alpha)$. Given $\delta\colon\Delta\to\mathit{Type}$, we will also use it to denote its canonical extension to $\mathit{Type}(\Delta)\to\mathit{Type}$. A substitution for a context Γ of closed types is a map $\gamma\colon\mathrm{dom}(\Gamma)\to\mathit{Val}$ such that, for every $x\colon\tau$ in Γ, $\gamma(x)\in\mathit{Val}(\tau)$. We use $\mathit{Subst}(\Gamma)$ to denote the set of substitutions for Γ, and $e\gamma$ to denote the substitution of every free variable x in e by $\gamma(x)$.

We also define some syntactic sugar to make writing programs easier. First, we use the type $1+1$ to model Booleans, and we write if b then e else e' for case$(b,_.e,_.e')$ (here underscore just denotes an unimportant variable). If we have two expressions e,e' we write let $x=e$ in e' for $(\lambda x.e')\;e$, and if x is not free in e', we will occasionally just write $e;e'$.

2.2 Operational Semantics

We define a call-by-value single step reduction relation $\longrightarrow\;\subseteq\mathit{Expr}\times\mathit{Expr}\times[0,1]\times\{D,N,P\}$. The index $[0,1]$ indicates the probability of the reduction taking place, while the letter indicates the type of reduction (Deterministic, Nondeterministic or Probabilistic). If the probability of the reduction is 1, we will often omit the index. The reduction rules are:

$$\begin{array}{rll@{\qquad}rll}
(\lambda x.e)v & \longrightarrow_D e[v/x] & & ? & \longrightarrow_N \underline{k} & k\in\mathbb{N}\\
\pi_i\langle v_1,v_2\rangle & \longrightarrow_D v_i & i\in\{1,2\} & (\Lambda.e)\_ & \longrightarrow_D e & \\
\mathsf{case}(\mathsf{inl}(v),x_1.e_1,x_2.e_2) & \longrightarrow_D e_1[v/x_1] & & \mathsf{rand}\;\underline{n} & \longrightarrow_P^{1/n} \underline{k} & 1\le k\le n\\
\mathsf{case}(\mathsf{inr}(v),x_1.e_1,x_2.e_2) & \longrightarrow_D e_2[v/x_2] & & \mathsf{S}\;\underline{k} & \longrightarrow \underline{k+1} & k\in\mathbb{N}\\
\mathsf{unfold}\;\mathsf{fold}\;v & \longrightarrow_D v & & \mathsf{P}\;\underline{k} & \longrightarrow \max\{\underline{1},\underline{k-1}\} & k\in\mathbb{N}
\end{array}$$

$$\frac{e\longrightarrow_X^q e'}{E[e]\longrightarrow_X^q E[e']} \qquad X\in\{D,N,P\}$$

We write $e\longrightarrow e'$ to denote that there exist $r\in[0,1]$ and $X\in\{D,N,P\}$ such that $e\longrightarrow_X^r e'$. If there exists a chain of deterministic reductions $e\longrightarrow_D e_1\longrightarrow_D\ldots\longrightarrow_D e'$, we will denote it by $e\longrightarrow_D^* e'$.

These rules induce a Markov Decision Process where the set of states is Expr. However, we will not define a multistep reduction relation. The property that we are interested in is the probability of termination, which we will define in the next section as a fixed point of the single step reduction.

3 NOTIONS OF OBSERVATIONS

In this section we make precise the observations we can make about the behavior of a program containing probabilistic and nondeterministic choices. We will use these observations to define the contextual equivalence relation. These notions will also be crucial in the construction of the logical relations by biorthogonality.

In a deterministic setting [Pitts 2004] the observation used to define contextual equivalence is termination. Note that this suffices to define interesting equivalence relations: if we have two distinct values of a ground type, we can always define a context that makes only one of them diverge. In languages with nondeterministic choice, there are usually two notions of observation to consider [Birkedal et al. 2013; Lassen 1998]: may-termination (i.e., there are choices that make the program terminate) or must-termination (every choice makes the program terminate). In the probabilistic setting, quantitative notions of observations are used, namely the probability of termination [Bizjak and Birkedal 2015].

3.1 Preliminaries: Order Theory

We review here some concepts that will be useful in defining the observations using fixed points. We will not present here a formal definition of the ordinal numbers (see e.g. [Kunen 2011]). Suffices to know that the set of countable ordinals (denoted by ω_1) is well-ordered (we use $<$ and $\in$ indistinctively) and (1) $0 \in \omega_1$, (2) for every $\beta \in \omega_1$, $\beta + 1 \in \omega_1$, and (3) for every countable family $\{\beta_i\}_{i\in\omega} \subseteq \omega_1$, $\sup_{i\in\omega} \beta_i \in \omega_1$. Here ω denotes the ordinal corresponding to the set of natural numbers (0 included). A non-zero ordinal is a *limit ordinal* if it is the supremum of all the ordinals below it. This can be used to define an induction principle known as *ordinal induction.*

Definition 3.1. *A set L equipped with a preorder relation $(\leq) \subseteq L \times L$ is a complete lattice if for every subset $S \subseteq L$, $\sup S \in L$ and $\inf S \in L$.*

Definition 3.2. *Let L be a complete lattice. An operator $F\colon L \longrightarrow L$ is ω-continuous if for any monotonically non-decreasing sequence $\{x_i\}_{i\in\omega}$, we have $\sup_{i\in\omega} F(x_i) = F(\sup_{i\in\omega} x_i)$.*

A fixed point of an operator $F\colon L \longrightarrow L$ is an element $x \in L$ such that $F(x) = x$. A classical result by Tarski gives sufficient conditions for the existence of a least fixed point:

Theorem 3.3 (Tarski's fixed point theorem [Tarski 1955])**.** *Let L be a complete lattice and let $F\colon L \longrightarrow L$ be monotone. Then F has a least fixed point $\mathsf{lfp}(F)$.*

Given a monotonically non-decreasing operator $F\colon L \longrightarrow L$ we can define its iterations up to an arbitrary ordinal by ordinal induction: $F^0(x) = x$, $F^{\beta+1}(x) = F(F^\beta(x))$ and for a limit ordinal α, $F^\alpha(x) = \sup_{\beta<\alpha} F^\beta(x)$. The following result from [Cousot and Cousot 1979] states that this iteration reaches the least fixed point at some ordinal.

Theorem 3.4 ([Cousot and Cousot 1979])**.** *Let L be a complete lattice with bottom element $\bot$ and let $F\colon L \longrightarrow L$ be monotonically non-decreasing. Then there exists a (possibly uncountable) ordinal ξ such that $\mathsf{lfp}\ F = F^\xi(\bot)$.*

The following result, known as Kleene's fixed-point theorem, gives us the value of this ordinal for ω-continuous operators.

Theorem 3.5 (Kleene's fixed point theorem [Cousot and Cousot 1979])**.** *Let L be a complete lattice with bottom element $\bot$ and let $F\colon L \longrightarrow L$ be $\omega-$continuous. Then $\mathsf{lfp}\ F = F^\omega(\bot) = \sup_{i\in\omega} F^i(\bot)$.*

3.2 Probabilistic May- and Must-Termination

In the probabilistic setting, the observable behavior of a program can be defined by a map of type $Expr \longrightarrow [0, 1]$ mapping an expression e to the probability that it evaluates to a value. As a generalization of probabilistic termination, we will consider two quantitative observations, which will induce two different (and in general incomparable) contextual equivalence relations.

First we consider the probability of may-termination, which is the maximal probability of termination among all possible nondeterministic choices. We will define it as the least fixed point of the following operator of type $(Expr \longrightarrow [0,1]) \longrightarrow Expr \longrightarrow [0,1]$:

$$\Phi(f)(e) = \begin{cases} 1 & \text{if } e \in Val \\ \sup_{n \in \mathbb{N}} f(E[\underline{n}]) & e = E[?] \\ \sum_{1 \leq n \leq k} \frac{1}{k} \cdot f(E[\underline{n}]) & e = E[\mathsf{rand}\ \underline{k}] \\ f(e') & e \longrightarrow_D e' \\ 0 & \text{otherwise} \end{cases}$$

Note that existence of the sup in the case for $E[?]$ is guaranteed because f is bounded. Analogously, we define the probability of must-termination as the least fixed point of the operator $\Psi \colon (Expr \longrightarrow [0,1]) \longrightarrow Expr \longrightarrow [0,1]$ defined below:

$$\Psi(f)(e) = \begin{cases} 1 & \text{if } e \in Val \\ \inf_{n \in \mathbb{N}} f(E[\underline{n}]) & e = E[?] \\ \sum_{1 \leq n \leq k} \frac{1}{k} \cdot f(E[\underline{n}]) & e = E[\mathsf{rand}\ \underline{k}] \\ f(e') & e \longrightarrow_D e' \\ 0 & \text{otherwise} \end{cases}$$

The set of functions $(Expr \longrightarrow [0,1])$ ordered pointwise forms a complete lattice. The existence of the least fixed points is therefore guaranteed by Tarski's fixed point theorem, and by the fact that the operators are monotonic:

Lemma 3.6. *The operators Φ, Ψ are monotonically nondecreasing, that is, for $f, g \colon Expr \longrightarrow [0,1]$ such that $f \leq g$ then $\Phi(f) \leq \Phi(g)$ and $\Psi(f) \leq \Psi(g)$.*

The proof is by case analysis. Thus, both operators have least fixed points, so the concepts below are well-defined:

Definition 3.7. *Let $e \in Expr$. The* probability of may-termination *of e, denoted $\mathfrak{P}^{\downarrow}(e)$ is defined as $\mathfrak{P}^{\downarrow}(e) = (\mathsf{lfp}\ \Phi)(e)$. The* probability of must-termination *of e, denoted $\mathfrak{P}^{\Downarrow}(e)$ is defined as $\mathfrak{P}^{\Downarrow}(e) = (\mathsf{lfp}\ \Psi)(e)$.*

In order to connect the fixed point definition of the probabilities of termination to the operational semantics of our language, we use the properties below. The proof follows by first using the facts that $\mathfrak{P}^{\downarrow} = \Phi(\mathfrak{P}^{\downarrow})$, $\mathfrak{P}^{\Downarrow} = \Psi(\mathfrak{P}^{\Downarrow})$, and then unfolding the definitions of Φ and Ψ.

Proposition 3.8. *Let $e \in Expr$. Then:*

(1) If $e \in Val$, then $\mathfrak{P}^{\downarrow}(e) = \mathfrak{P}^{\Downarrow}(e) = 1$.
(2) If $e = E[?]$ then $\mathfrak{P}^{\downarrow}(e) = \sup_{n \in \mathbb{N}} \mathfrak{P}^{\downarrow}(E[\underline{n}])$ and $\mathfrak{P}^{\Downarrow}(e) = \inf_{n \in \mathbb{N}} \mathfrak{P}^{\Downarrow}(E[\underline{n}])$.
(3) If $e = E[\mathsf{rand}\ \underline{k}]$ then $\mathfrak{P}^{\downarrow}(e) = \sum_{1 \leq n \leq k} \frac{1}{k} \cdot \mathfrak{P}^{\downarrow}(E[\underline{n}])$ and $\mathfrak{P}^{\Downarrow}(e) = \sum_{1 \leq n \leq k} \frac{1}{k} \cdot \mathfrak{P}^{\Downarrow}(E[\underline{n}])$.
(4) If $e \longrightarrow_D^* e'$, then $\mathfrak{P}^{\downarrow}(e) = \mathfrak{P}^{\downarrow}(e')$ and $\mathfrak{P}^{\Downarrow}(e) = \mathfrak{P}^{\Downarrow}(e')$.

In section 4 we will define step-indexed logical relations to reason about these notions of termination. Therefore, we develop a step-indexed version of the probability of termination. Let $e \in Expr$. For any $n \in \omega$ we use $\mathfrak{P}_n^{\downarrow}(e)$ as shorthand for $\Phi^n(\bot)(e)$. For any $\beta \in \omega_1$ we use $\mathfrak{P}_\beta^{\Downarrow}(e)$

to denote $\Psi^{\beta}(\bot)(e)$ (the reason for using different indices for may- and must-termination will become clear in short). Note that this makes the step-indexing count every operational steps. Other work [Birkedal et al. 2013; Bizjak and Birkedal 2015] considers step-indexed notions of termination that only increment the step-index on choices and on unfold-fold reductions. This makes the resulting logical relation easier to use for some applications, for instance β-reduction is an equivalence at the same step index, not just at the limit. However, for our intended purposes it suffices to count every step.

We will use the following two results about the step-indexed probability of termination. First, it is monotonically non-decreasing in the number of steps:

Proposition 3.9. *Let $e \in Expr$. Then, for any $n < m < \omega$, $\mathfrak{P}^{\downarrow}_n(e) \leq \mathfrak{P}^{\downarrow}_m(e)$, and for any $\beta < \xi < \omega_1$, $\mathfrak{P}^{\Downarrow}_\beta(e) \leq \mathfrak{P}^{\Downarrow}_\xi(e)$.*

This result is proven by (ordinal) induction. Second, we have a step-indexed version of proposition 3.8:

Proposition 3.10. *Let $e \in Expr$, $n \leq m < \omega$ and $\beta \leq \xi < \omega_1$. Then:*

(1) If $e \in Val$, then $\mathfrak{P}^{\downarrow}_n(e) = \mathfrak{P}^{\Downarrow}_\beta(e) = 1$.

(2) If $e = E[?]$ then $\mathfrak{P}^{\downarrow}_n(e) \leq \sup_{k \in \mathbb{N}} \mathfrak{P}^{\downarrow}_m(E[\underline{k}])$ and $\mathfrak{P}^{\Downarrow}_\beta(e) \leq \inf_{k \in \mathbb{N}} \mathfrak{P}^{\Downarrow}_\xi(E[\underline{k}])$.

(3) If $e = E[\mathit{rand}\,\underline{k}]$ then $\mathfrak{P}^{\downarrow}_n(e) \leq \sum_{1 \leq n \leq k} \dfrac{1}{k} \cdot \mathfrak{P}^{\downarrow}_m(E[\underline{n}])$ and $\mathfrak{P}^{\Downarrow}_\beta(e) \leq \sum_{1 \leq n \leq k} \dfrac{1}{k} \cdot \mathfrak{P}^{\Downarrow}_\xi(E[\underline{n}])$.

(4) If $e \longrightarrow^*_D e'$, then $\mathfrak{P}^{\downarrow}_n(e) \leq \mathfrak{P}^{\downarrow}_m(e')$ and $\mathfrak{P}^{\Downarrow}_\beta(e) \leq \mathfrak{P}^{\Downarrow}_\xi(e')$.

The logical relation for may-termination is going to be defined in terms of $\mathfrak{P}^{\downarrow}_n$, but in the end we want to use it to reason about $\mathfrak{P}^{\downarrow}$, so we need to take our step indices from an ordinal ξ large enough so that $\sup_{\beta<\xi} \mathfrak{P}^{\downarrow}_\beta = \mathfrak{P}^{\downarrow}$. In the case of may-termination, it suffices to step-index up to ω due to the following result:

Proposition 3.11. *Φ is ω-continuous. In particular, for any $e \in Expr$, $\mathfrak{P}^{\downarrow}(e) = \sup_{i \in \omega} \mathfrak{P}^{\downarrow}_i(e)$.*

However, this is not the case for Ψ, due to the following counterexample:

Example 3.12. Consider the family $\{f_i \colon Expr \longrightarrow [0,1]\}$ where $f_i(e) = \begin{cases} 1 & e = \underline{n} \wedge n < i \\ 0 & \text{otherwise} \end{cases}$. Then $\sup_{i \in \omega}(\Psi^i(f_i)(?)) = \sup_{i\in\omega}(\inf_{n \in \mathbb{N}} f_i(\underline{n})) = 0$, but $(\sup_{i\in\omega} \Psi^i(f_i))(?) = \inf_{n\in\mathbb{N}} \sup_{i\in\omega} f_i(\underline{n}) = 1$. Therefore Ψ is not ω-continuous.

We will show that the iterations of Ψ reach a fixed point in at most ω_1 steps, and therefore it suffices to step-index up to ω_1. This generalizes to the probabilistic setting a similar result from [Birkedal et al. 2013] for must termination of programs with countable nondeterministic choice. Concretely, we will prove that there exists a countable ordinal β such that for any $e \in Expr$, $\mathfrak{P}^{\Downarrow}(e) = \Psi^{\beta}(\bot)(e)$. In other words, we will show that the fixpoint iteration $\Psi(\bot)(e), \Psi^2(\bot)(e), \ldots, \Psi^{\omega}(\bot)(e), \ldots$ eventually becomes constant at some ordinal below ω_1.

The following lemma is the key to our argument. It states that any non-decreasing ω_1-indexed sequence in a closed interval must eventually become constant:

Lemma 3.13. *Let $f : \omega_1 \longrightarrow [0,1]$ be a monotonic function. Then, there exist $r \in [0,1]$ and $\beta < \omega_1$ such that for every α, if $\beta < \alpha < \omega_1$ then $f(\alpha) = r$.*

Proof. Let $f^* = \sup_{\alpha<\omega_1} f(\alpha)$, which must exist since f is bounded. By the definition of least upper bound, for every $i \in \mathbb{N}$ there exists $\beta_i < \omega_1$ such that $f^* - 2^{-i} < f(\beta_i) \leq f^*$. Then we can take $\beta = \sup_{i\in\mathbb{N}} \beta_i$ which is a countable ordinal. Using monotonicity, we get that for every $\alpha \geq \beta$ with $\alpha < \omega_1$,

$$f^* \geq f(\alpha) \geq f(\beta) \geq \sup_{i\in\mathbb{N}} f(\beta_i) \geq \sup_{i\in\mathbb{N}}(f^* - 2^{-i}) = f^*.$$

□

Finally, we will prove:

Proposition 3.14. *There exists a countable ordinal β such that $\Psi^\beta(\bot) = \mathsf{lfp}\ \Psi$, and therefore, $\mathsf{lfp}\Psi = \sup_{\beta<\omega_1} \Psi^\beta(\bot)$. In particular, for any $e \in Expr$, $\mathfrak{P}^{\Downarrow}(e) = \sup_{\beta\in\omega_1} \mathfrak{P}^{\Downarrow}_\beta(e)$.*

Proof. For every e, $F_e(\alpha) \triangleq \Psi^\alpha(\bot)(e)$ determines a monotonic map from ω_1 to $[0, 1]$, so it must be constant from some countable β_e. Since the set of expressions is countable, $\beta = \sup_{e\in Expr} \beta_e$ is a countable ordinal and for every e, $\Psi^{\beta+1}(\bot)(e) = \Psi^\beta(\bot)(e)$. □

3.3 Comparison to May- and Must-Termination

This work builds on two previous works that study contextual equivalence for probabilistic choice [Bizjak and Birkedal 2015] and countable nondeterministic choice [Birkedal et al. 2013]. In this section we argue how the notions of observation we have defined can be seen as generalizations of the observations used in those works. From another point of view, the notions of termination we have defined can also be seen as extensions to the higher-order setting of notions of probabilistic test satisfaction from the literature of process algebra [Yi and Larsen 1992].

First, it is easy to see that if we remove nondeterministic choice from our language, the (may- and must-) probabilities of termination collapse to the same one, which also coincides with the notion of observation used in previous work for contextual equivalence for probabilistic programs [Bizjak and Birkedal 2015]. This defines the probability of termination as the least fixed-point of the Φ operator without the case for countable choice.

We will further justify our choice of observations by showing that, after removing probabilistic choice from our language, they coincide with the notions of may- and must-termination for languages with countable nondeterminism presented in [Birkedal et al. 2013].

Throughout this subsection, we will consider the fragment of the language without probabilistic choice (we denote such expressions by $Expr_{ND}$). We recall here the definitions of may and must convergence.

Definition 3.15 (May-convergence). *Let $\hat{\Phi} : \mathcal{P}(Expr_{ND}) \to \mathcal{P}(Expr_{ND})$ defined by*

$$\hat{\Phi}(X) = \{e \in Expr_{ND} \mid e \in Val \vee \exists e' \in Expr_{ND}.e \longrightarrow e' \wedge e' \in X\}.$$

We say that e may-converges (denoted $e \downarrow$) if $e \in \mathsf{lfp}\ \hat{\Phi}$, which exists by Tarski's fixed point theorem.

Definition 3.16 (Must-convergence). *Let $\hat{\Psi} : \mathcal{P}(Expr_{ND}) \to \mathcal{P}(Expr_{ND})$ defined by*

$$\hat{\Psi}(X) = \{e \in Expr_{ND} \mid e \in Val \vee (\exists e'.e \longrightarrow e' \wedge \forall e' \in Expr_{ND}.e \longrightarrow e' \Rightarrow e' \in X)\}.$$

We say that e must-converges (denoted $e \Downarrow$) if $e \in \mathsf{lfp}\ \hat{\Psi}$, which exists by Tarski's fixed point theorem.

Technically, the definition of must-convergence is slightly different from [Birkedal et al. 2013] which, unlike us, considers stuck terms to be must-convergent. This difference is minor, since it only arises with ill-typed terms. Our choice of presentation also allows us to present both notions of termination as particular cases of a scheduler-based notion of termination, see section 5.

Note that by considering the order isomorphism between $\mathcal{P}(Expr_{ND})$, ordered by subset inclusion, and $Expr_{ND} \to \{0, 1\}$, ordered pointwise, $\hat{\Phi}$ and $\hat{\Psi}$ can be seen as operators of type $(Expr_{ND} \to \{0, 1\}) \to (Expr_{ND} \to \{0, 1\})$. In particular, may- and must-convergence can be regarded as

$\{0,1\}$-valued functions. This provides the connection to the notions of termination studied in this paper.

Proposition 3.17. *Let $e \in Expr_{ND}$, and $f \in Expr \to \{0,1\}$. Then $\Phi(f)(e) = \hat{\Phi}(f)(e)$. In particular, $\mathfrak{P}^{\downarrow}(e) = 1$ if and only if $e \downarrow$.*

Proof. We do a case distinction on e. If $e \in Val$, then $\Phi(f)(e) = 1 = \hat{\Phi}(f)(e)$. If e is stuck, then $\Phi(f)(e) = 0 = \hat{\Phi}(f)(e)$. If $e = E[?]$ then $\Phi(f)(e) = \sup_{n\in\mathbb{N}} f(E[\underline{n}])$. Since for every n, $f(E[\underline{n}])$ is either 0 or 1 the sup can also only be 0 or 1. It is 1 iff there exists a particular m for which $f(E[\underline{m}]) = 1$, so $\Phi(f)(e) = \hat{\Phi}(f)(e)$. If $e \longrightarrow_D e'$, then there is no other reduction from e, and $\hat{\Phi}(f)(e) = 1$ iff $f(e) = 1$, so $\Phi(f)(e) = \hat{\Phi}(f)(e)$.

Then, $\mathfrak{P}^{\downarrow}(e) = 1$ if and only if $e \downarrow$ because both are least fixed points of isomorphic operators. □

Proposition 3.18. *Let $e \in Expr_{ND}$, and $f \in Expr \to \{0,1\}$. Then $\Psi(f)(e) = \hat{\Psi}(f)(e)$. In particular, $\mathfrak{P}^{\Downarrow}(e) = 1$ if and only if $e \Downarrow$.*

Proof. We do a case distinction on e. If $e \in Val$, then $\Psi(f)(e) = 1 = \hat{\Psi}(f)(e)$. If e is stuck, then $\Psi(f)(e) = 0 = \hat{\Psi}(f)(e)$. If $e = E[?]$, then $\Psi(f)(e) = \inf_{n\in\mathbb{N}} f(E[\underline{n}])$, which can be 0 or 1. If it is 0, then there exists a particular $m \in \mathbb{N}$ such that $e \longrightarrow E[\underline{m}]$ and $f(E[\underline{m}]) = 0$, so $\hat{\Psi}(f)(e) = 0$. Otherwise it is 1 so for every e' such that $e \longrightarrow e'$, $f(e') = 1$, and $\hat{\Psi}(f)(e) = 1$. If $e \longrightarrow_D e'$, then there is no other reduction from e, and $\Psi(f)(e) = f(e') = \hat{\Psi}(f)(e)$.

Then, $\mathfrak{P}^{\Downarrow}(e) = 1$ if and only if $e \Downarrow$ because both are least fixed points of isomorphic operators. □

4 TYPE-INDEXED RELATIONS

Contextual equivalence is a standard notion of equivalence between higher-order programs. Informally speaking, it states that two programs are equivalent if they have the same observable behavior under any context. Alternatively, contextual equivalence may also be defined as the largest equivalence relation that is adequate under some notion of observation and closed under the term constructors of the programming language [Pitts 2004]. It is well-known that it is hard to reason directly about contextual equivalence. To prove contextual equivalences, we will instead define a step-indexed logical relation, and then show that it is sound for contextual equivalence (i.e., it is an equivalence relation and it is closed under term constructors).

To show completeness of the logical relation, we will show that it coincides with CIU (Closed Instantiation of Uses) equivalence, which is defined in terms of evaluation contexts, as opposed to arbitrary contexts, and which also coincides with contextual equivalence.

4.1 Background

Definition 4.1. *A* type-indexed relation *is a set $\mathcal{R}$ of tuples $(\Delta, \Gamma, e_1, e_2, \tau)$ such that $\Delta \vdash \Gamma$, $\Delta \vdash \tau$, $\Delta \mid \Gamma \vdash e_1 : \tau$ and $\Delta \mid \Gamma \vdash e_2 : \tau$. If $(\Delta, \Gamma, e_1, e_2, \tau) \in \mathcal{R}$, we also denote it by $\Delta \mid \Gamma \vdash e_1 \mathrel{\mathcal{R}} e_2 : \tau$.*

Definition 4.2 (May- and must-adequacy). *Let $\mathcal{R}$ be a type-indexed relation. We say that $\mathcal{R}$ is may-adequate if, for any $\emptyset \mid \emptyset \vdash e \mathrel{\mathcal{R}} e'$, then $\mathfrak{P}^{\downarrow}(e) \leq \mathfrak{P}^{\downarrow}(e')$. Analogously, we say that $\mathcal{R}$ is must-adequate if, for any $\emptyset \mid \emptyset \vdash e \mathrel{\mathcal{R}} e'$ then $\mathfrak{P}^{\Downarrow}(e) \leq \mathfrak{P}^{\Downarrow}(e')$.*

The following definitions are standard [Pitts 2004]:

Definition 4.3 (Precongruence). *Let $\mathcal{R}$ be a type-indexed relation. We say that it is*

- reflexive *if $\Delta \mid \Gamma \vdash e : \tau$ implies $\Delta \mid \Gamma \vdash e \mathrel{\mathcal{R}} e : \tau$,*
- symmetric *if $\Delta \mid \Gamma \vdash e_1 \mathrel{\mathcal{R}} e_2 : \tau$ implies $\Delta \mid \Gamma \vdash e_2 \mathrel{\mathcal{R}} e_1 : \tau$*
- transitive *if $\Delta \mid \Gamma \vdash e_1 \mathrel{\mathcal{R}} e_2 : \tau$ and $\Delta \mid \Gamma \vdash e_2 \mathrel{\mathcal{R}} e_3 : \tau$ imply $\Delta \mid \Gamma \vdash e_1 \mathrel{\mathcal{R}} e_3 : \tau$*

- compatible *if it is closed under the typing rules. For instance in the case of abstraction and application the following must hold:*

$$\frac{\Delta \mid \Gamma, x\colon \sigma \vdash e_1 \mathcal{R}\, e_2 \colon \tau}{\Delta \mid \Gamma \vdash \lambda x.e_1 \mathcal{R}\, \lambda x.e_2 \colon \sigma \to \tau} \qquad \frac{\begin{array}{c}\Delta \mid \Gamma \vdash e_1 \mathcal{R}\, e_2 \colon \sigma \to \tau \\ \Delta \mid \Gamma \vdash e_1' \mathcal{R}\, e_2' \colon \sigma\end{array}}{\Delta \mid \Gamma \vdash e_1\, e_1' \mathcal{R}\, e_2\, e_2' \colon \tau}$$

We say that $\mathcal{R}$ is a precongruence if it is reflexive, transitive and compatible.

Definition 4.4 (Contextual approximation and equivalence)**.** *Contextual may-approximation (denoted $\lesssim^{ctx}_{\downarrow}$) is the largest may-adequate precongruence. Contextual must-approximation (denoted $\lesssim^{ctx}_{\Downarrow}$) is the largest must-adequate precongruence. Contextual may-equivalence ($\cong^{ctx}_{\downarrow}$) and must-equivalence ($\cong^{ctx}_{\Downarrow}$) are defined respectively as the largest symmetric subrelations of contextual may- and must-approximation.*

The CIU relation is an alternative way to define equivalence between higher-order programs. It is defined with respect with evaluation contexts instead of arbitrary contexts, which makes it easier to use for some applications.

Definition 4.5 (CIU approximation and equivalence)**.** *The CIU may-approximation (resp. CIU must-approximation) relation $\lesssim^{CIU}_{\downarrow}$ ($\lesssim^{CIU}_{\Downarrow}$) is defined as follows: for every e, e' such that $\Delta \mid \Gamma \vdash e \colon \tau$ and $\Delta \mid \Gamma \vdash e' \colon \tau$ we write $\Delta \mid \Gamma \vdash e \ \lesssim^{CIU}_{\downarrow}\ e' \colon \tau$ ($\Delta \mid \Gamma \vdash e \ \lesssim^{CIU}_{\Downarrow}\ e' \colon \tau$) if for every $\delta \colon \Delta \to Type$, $\gamma \in Subst(\Gamma\delta)$, and $E \in Ectx(\tau\delta)$, we have that $\mathfrak{P}^{\downarrow}(E[e\gamma]) \leq \mathfrak{P}^{\downarrow}(E[e'\gamma])$ ($\mathfrak{P}^{\Downarrow}(E[e\gamma]) \leq \mathfrak{P}^{\Downarrow}(E[e'\gamma])$). CIU may-equivalence ($\cong^{CIU}_{\downarrow}$) and must-equivalence ($\cong^{CIU}_{\Downarrow}$) are defined respectively as the largest symmetric subrelations of CIU may- and must-approximation.*

By definition, CIU approximation is reflexive, transitive and may-adequate. In fact we will show that it is also compatible and it coincides with contextual approximation. However, a direct proof of this fact is challenging. Instead we define a step-indexed logical relation, and we will show that it coincides with both the CIU relation and contextual approximation.

4.2 Step-Indexed Logical Relations

Logical relations were introduced as a technique to prove properties of higher order programs, such as strong normalization [Tait 1967] or contextual equivalence [Pitts and Stark 1998]. The idea is to define an interpretation of a type by induction on the construction of the type ensuring that all inhabitants satisfy a desired property. In the case of contextual approximation, every type is given a (relational) interpretation as a set of pairs of related terms. Step-indexed logical relations are a generalization of logical relations to avoid circularity issues when treating recursive types. They index the interpretation of a type by a numerical value (the step-index) that morally corresponds to the number of evaluation steps for which the terms are related. This allows the interpretation to be well-defined by induction on both the structure of the type and the step-index.

In this work, we will use the countable ordinals as step indices, as indicated by the following definition:

Definition 4.6 (ω_1-indexed relation)**.** *Let S, S' be two sets. An ω_1-*indexed relation *r is a map $\omega_1 \to \mathcal{P}(S \times S')$ such that for every countable ordinal ξ, $r(\xi) \subseteq \cap_{\beta<\xi}\, r(\xi)$ (note that this implies $r(0) \supseteq r(1) \supseteq \cdots \supseteq r(\omega) \supseteq \dots$). An ω-indexed relation r is an ω_1-indexed relation such that for every $\xi > \omega$, $r(\xi) = r(\omega)$.*

Let $\tau, \sigma \in Type(\emptyset)$. A *value relation* between τ and σ is an ω_1-indexed relation over $Val(\tau) \times Val(\sigma)$. An *evaluation context relation* between τ and σ is an ω_1-indexed relation over $Ectx(\tau) \times Ectx(\sigma)$. An *expression relation* between τ, σ is an ω_1-indexed relation over $\mathit{Expr}(\tau) \times \mathit{Expr}(\sigma)$. We let $\mathit{RVal}(\tau, \sigma)$, $\mathit{REctx}(\tau, \sigma)$, and $\mathit{RExpr}(\tau, \sigma)$ denote the sets of value, evaluation context and expression relations between τ and σ, respectively.

To define the interpretation of a type we also need to interpret each of the free type variables that appear in it. Given a type context Δ, we let $\mathit{RVal}(\Delta)$ denote the set of interpretations of Δ defined by:

$$\mathit{RVal}(\Delta) = \left\{ (\delta_1, \delta_2, r) \;\middle|\; \begin{array}{l} \delta_1, \delta_2 \colon \Delta \to \mathit{Type}, r \colon \Delta \to \omega_1 \to \mathcal{P}(\mathit{Val} \times \mathit{Val}), \\ \forall \alpha \in \Delta. r(\alpha) \in \mathit{RVal}(\delta_1(\alpha), \delta_2(\alpha)) \end{array} \right\}$$

In other words, a relation over Δ is a triple consisting of two substitutions δ_1 and δ_2 for Δ and a map r assigning to every $\alpha \in \Delta$ a value relation over $\delta_1(\alpha), \delta_2(\alpha)$.

In the rest of the section, we will define two logical relations to reason about may- and must-contextual approximation. The construction follows a similar pattern in both cases. Given a type context Δ, a type $\tau \in \mathit{Type}(\Delta)$, and a set of relations $\varphi = (\delta_1, \delta_2, r) \in \mathit{RVal}(\Delta)$ giving an interpretation to every type variable, a type τ can be given a relational interpretation $[\![\Delta \vdash \tau]\!](\varphi)$, which is a relation in $\mathit{RVal}(\delta_1(\tau), \delta_2(\tau))$. This can then be lifted to a relational interpretation over closed expressions. Finally, we define a logical relation over open expressions by substituting every free variable by a pair of related values.

The logical relations we define here can be seen as extensions of the ones in [Birkedal et al. 2013] with probabilistic choice (see also the logical relation defined in [Bizjak and Birkedal 2015]). The combination is smooth due to the use of biorthogonality, which allows us to prove adequacy and compatibility of the logical relations from the convergence properties of $\mathfrak{P}^{\downarrow}_{n}(\cdot)$ and $\mathfrak{P}^{\Downarrow}_{\beta}(\cdot)$, as well as propositions 3.8 to 3.10.

4.3 Relation for May-Termination

The relational interpretation for values is defined in fig. 2. This definition is mostly standard, see e.g. [Birkedal et al. 2013; Bizjak and Birkedal 2015]. To give some intuition, the inhabitants of this relation should be the pairs of values of type τ such that the value on the left contextually approximates the value on the right. Hence, for base types the relation is just equality. For a type variable α, its relational interpretation is given in φ. Two values of an arrow type $\tau \to \sigma$ are related if they map related values of type τ to related expressions of type σ. For product and sum types, the relation is the respectively the product and the sum of the relations for the individual types. In the case of a universal type $\forall \alpha. \tau$, we take the pairs $(\Lambda. e, \Lambda, e')$ such that (e, e') are related at every possible interpretation of the variable α.

The case for recursive types is more interesting, and it justifies the use of step-indices. Two values $(\mathsf{fold}\ v, \mathsf{fold}\ v')$ are in $[\![\Delta \vdash \mu\alpha.\tau]\!]$ at a step index n if (v, v') are in $[\![\Delta \vdash \tau[(\mu\alpha.\tau)/\alpha]]\!]$ at a smaller step-index (see also lemma 4.9). Circularity in the definition is avoided by requiring the step index to strictly decrease.

The definition can be generalized to typing contexts $[\![\Delta \vdash \Gamma]\!]$ by taking the pointwise product. For every $\varphi \in \mathit{RVal}(\Delta)$ and every natural number n, $[\![\Delta \vdash \Gamma]\!](\varphi)(n)$ can be seen as a pair (γ, γ') of substitutions for the variables in Γ such that for every $(x \colon \tau) \in \Gamma$, $(\gamma(x), \gamma'(x)) \in [\![\Delta \vdash \tau]\!](\varphi)(n)$.

So far we have only defined the relational interpretation for values. We can then extend it to a relational interpretation for expressions by using biorthogonality. This consists of lifting the relational interpretation first from values to evaluation contexts and then to expressions using the probability of may-termination as a dualizing observation. The two steps of the lifting are given by

$$\llbracket \Delta \vdash \alpha \rrbracket(\varphi)(n) = r(\alpha)(n) \qquad \llbracket \Delta \vdash 1 \rrbracket(\varphi)(n) = \{(\langle\rangle, \langle\rangle)\} \qquad \llbracket \Delta \vdash \mathsf{nat} \rrbracket(\varphi)(n) = \{(\underline{k}, \underline{k}) \mid k \in \mathbb{N}\}$$

$$\begin{aligned}
\llbracket \Delta \vdash \tau \to \sigma \rrbracket(\varphi)(n) &= \bigcap_{m \le n} \{(\lambda x.e, \lambda y.e') \mid \forall (v, v') \in \llbracket \Delta \vdash \tau \rrbracket(\varphi)(m).(e[v/x], e'[v'/y]) \in \llbracket \Delta \vdash \sigma \rrbracket(\varphi)^{\top\top}(m)\} \\
\llbracket \Delta \vdash \tau \times \sigma \rrbracket(\varphi)(n) &= \{(\langle v_1, v_2 \rangle, \langle v_1', v_2' \rangle) \mid (v_1, v_1') \in \llbracket \Delta \vdash \tau \rrbracket(\varphi)(n) \wedge (v_2, v_2') \in \llbracket \Delta \vdash \sigma \rrbracket(\varphi)(n)\} \\
\llbracket \Delta \vdash \tau + \sigma \rrbracket(\varphi)(n) &= \{(\mathsf{inl}(v), \mathsf{inl}(v')) \mid (v, v') \in \llbracket \Delta \vdash \tau \rrbracket(\varphi)(n)\} \cup \\
&\qquad \{(\mathsf{inr}(v), \mathsf{inr}(v')) \mid (v, v') \in \llbracket \Delta \vdash \sigma \rrbracket(\varphi)(n)\} \\
\llbracket \Delta \vdash \forall \alpha.\tau \rrbracket(\varphi)(n) &= \{(\Lambda.e, \Lambda, e') \mid \forall \sigma, \sigma' \in \mathit{Type}, \forall r \in \mathit{RVal}(\sigma, \sigma'). \\
&\qquad\qquad (e, e') \in (\llbracket \Delta, \alpha \vdash \tau \rrbracket(\varphi[\alpha \mapsto (\sigma, \sigma', r)]))^{\top\top}(n)\}
\end{aligned}$$

$$\begin{aligned}
\llbracket \Delta \vdash \mu\alpha.\tau \rrbracket(\varphi)(0) &= \mathit{Val}(\delta_1(\mu\alpha.\tau)) \times \mathit{Val}(\delta_2(\mu\alpha.\tau)) \\
\llbracket \Delta \vdash \mu\alpha.\tau \rrbracket(\varphi)(n{+}1) &= \{(\mathsf{fold}\, v, \mathsf{fold}\, v') \mid (v, v') \in \llbracket \Delta, \alpha \vdash \tau \rrbracket(\varphi')(n)\} \\
&\text{where } \varphi' \triangleq \varphi[\alpha \mapsto (\delta_1(\mu\alpha.\tau), \delta_2(\mu\alpha.\tau), \llbracket \Delta \vdash \mu\alpha.\tau \rrbracket(\varphi))]
\end{aligned}$$

Fig. 2. Definition of the logical relation on values for may-termination. We use δ_1, δ_2 and r to denote the three components of φ.

two operators $(\cdot)^{\top} : \mathit{RVal}(\tau, \tau') \to \mathit{REctx}(\tau, \tau')$ and $(\cdot)^{\perp} : \mathit{REctx}(\tau, \tau') \to \mathit{RExpr}(\tau, \tau')$ defined as:

$$r^{\top}(n) = \{(E, E') \mid \forall m \le n, \forall (v, v') \in r(m).\mathfrak{P}^{\downarrow}_m(E[v]) \le \mathfrak{P}^{\downarrow}(E'[v'])\}$$

$$r^{\perp}(n) = \{(e, e') \mid \forall m \le n, \forall (E, E') \in r(m).\mathfrak{P}^{\downarrow}_m(E[e]) \le \mathfrak{P}^{\downarrow}(E'[e'])\}$$

We write $r^{\top\top}$ for $(r^{\top})^{\perp}$. Biorthogonality is convenient because it ensures that the relational interpretation of expressions is closed under evaluation contexts and may-adequate. The fact that step-indices appear in the probability of termination only on the left is due to the fact that we want the relation to coincide with approximation, and to show $\mathfrak{P}^{\downarrow}(E[e]) \le \mathfrak{P}^{\downarrow}(E'[e'])$ it suffices to show $\mathfrak{P}^{\downarrow}_m(E[e]) \le \mathfrak{P}^{\downarrow}(E'[e'])$ at every index.

By closing the relation under substitutions and taking the limit over all step-indices, we can define a type-indexed relation, that we call *logical approximation*:

Definition 4.7. *We say that $\Delta \mid \Gamma \vdash e \lesssim^{log}_{\downarrow} e' : \tau$ if for all $\varphi \in \mathit{RVal}(\Delta)$, for all $n < \omega$, and all $(\gamma, \gamma') \in \llbracket \Delta \vdash \Gamma \rrbracket(\varphi)(n)$, we have $(e\gamma, e'\gamma') \in \llbracket \Delta \vdash \tau \rrbracket(\varphi)^{\top\top}(n)$.*

We want to show that logical approximation coincides with contextual approximation. We begin by showing that it is adequate.

Proposition 4.8. *The relation $\lesssim^{log}_{\downarrow}$ is may-adequate.*

Proof. Assume $\vdash e \lesssim^{log}_{\Downarrow} e' : \tau$. Note that, for all $n < \omega$, $([\,], [\,]) \in \llbracket \vdash \tau \rrbracket^{\top}(n)$. Then, we have that, for all $n < \omega$, $\mathfrak{P}^{\downarrow}_n(e) \le \mathfrak{P}^{\downarrow}(e')$, so in particular, $\sup_{n<\omega} \mathfrak{P}^{\downarrow}_n(e) \le \mathfrak{P}^{\downarrow}(e')$. By definition, $\mathfrak{P}^{\downarrow}(e) = \sup_{n<\omega} \Psi^n(\perp)(e) = \sup_{n<\omega} \mathfrak{P}^{\downarrow}_n(e)$, so $\mathfrak{P}^{\downarrow}(e) \le \mathfrak{P}^{\downarrow}(e')$. □

To show that it is compatible, we first need to show the so-called extension lemmas. Before showing them, we state some useful results:

Lemma 4.9. *Let $\Delta \vdash \tau$, and $\Delta, \alpha \vdash \sigma$ and $\varphi = (\delta_1, \delta_2, r) \in \mathit{RVal}(\Delta)$, and define $\varphi' = \varphi[\alpha \mapsto (\tau\delta_1, \tau\delta_2, \llbracket \Delta \vdash \tau \rrbracket(\varphi))]$. Then $\llbracket \Delta \vdash \sigma[\tau/\alpha] \rrbracket(\varphi) = \llbracket \Delta, \alpha \vdash \sigma \rrbracket(\varphi')$.*

Lemma 4.10. *Let $\tau, \tau' \in \mathit{Type}$, $r \in \mathit{RVal}(\tau, \tau')$. Then $r \subseteq r^{\top\top}$.*

Lemma 4.11. *Let $\tau, \tau' \in Type$, $r \subseteq s \in RVal(\tau, \tau')$. Then $r^{\top\top} \subseteq s^{\top\top}$.*

The context extension lemmas, as the name suggests, show that the interpretation of evaluation contexts $[\![\Delta \vdash \sigma]\!](\varphi)^{\top}$ is in a precise sense closed under evaluation context composition. To give a flavor of how they look like, we state and prove here some representative cases:

Lemma 4.12. *Let $n < \omega$, $\varphi \in RVal(\Delta)$, and $(E, E') \in [\![\Delta \vdash \mathsf{nat}]\!](\varphi)^{\top}(n)$. Then*

$$(E[\mathit{rand}\,[\,]\,], E'[\mathit{rand}\,[\,]\,]) \in [\![\Delta \vdash \mathsf{nat}]\!](\varphi)^{\top}(n).$$

Proof. Let $(v, v') \in [\![\Delta \vdash \mathsf{nat}]\!](\varphi)(n)$. Then, there exists $k \in \mathbb{N}$ such that $v = v' = \underline{k}$. We have

$$\mathfrak{P}^{\downarrow}_{\alpha}(E[\mathsf{rand}\ \underline{k}]) \leq \sum_{1 \leq l \leq k} \frac{1}{k} \cdot \mathfrak{P}^{\downarrow}_{\alpha}(E[\underline{l}]) \leq \sum_{1 \leq l \leq k} \frac{1}{k} \cdot \mathfrak{P}^{\downarrow}(E'[\underline{l}]) = \mathfrak{P}^{\downarrow}(E'[\mathsf{rand}\ \underline{k}])$$

Therefore, $(E[\mathsf{rand}\ [\,]\,], E'[\mathsf{rand}\ [\,]\,]) \in [\![\Delta \vdash \mathsf{nat}]\!](\varphi)^{\top}(n)$. □

Lemma 4.13. *Let $n < \omega$, $\varphi \in RVal(\Delta)$, and $(E, E') \in [\![\Delta \vdash \mu\alpha.\tau]\!](\varphi)^{\top}(n)$. Then we have that $(E[\mathit{fold}\,[\,]], E'[\mathit{fold}\,[\,]]) \in [\![\Delta \vdash \tau[\mu\alpha.\tau/\alpha]]\!](\varphi)^{\top}(n)$.*

Proof. Let $(v, v') \in [\![\Delta \vdash \tau[\mu\alpha.\tau/\alpha]]\!](\varphi)(n)$. By definition of the interpretation and applying lemma 4.9, $(\mathsf{fold}\,v, \mathsf{fold}\,v') \in [\![\Delta \vdash \tau[\mu\alpha.\tau/\alpha]]\!](\varphi)(n+1)$. Therefore, for all $m \leq n+1$ (and in particular, $m \leq n$), $\mathfrak{P}^{\downarrow}_{m}(E[\mathsf{fold}\,v]) \leq \mathfrak{P}^{\downarrow}(E'[\mathsf{fold}\,v'])$. Thus $(E[\mathsf{fold}\,[\,]], E'[\mathsf{fold}\,[\,]]) \in [\![\Delta \vdash \tau[\mu\alpha.\tau/\alpha]]\!](\varphi)^{\top}(n)$. □

Now we are in shape to show that the logical relation is compatible. This is often known as the *fundamental property*:

Proposition 4.14 (Fundamental property). *The relation $\lesssim^{log}_{\downarrow}$ is compatible. As a consequence, it is reflexive: for any type context Δ, typing context Γ, expression e and type τ, if $\Delta \mid \Gamma \vdash e\colon \tau$, then $\Delta \mid \Gamma \vdash e \lesssim^{log}_{\downarrow} e\colon \tau$.*

Proof. We show here a couple of illustrative cases:

- If $e = {?}$. Let $n < \omega$, $(E, E') \in [\![\Delta \vdash \mathsf{nat}]\!](n)^{\top}$. We have to show that for every $m \leq n$, $\mathfrak{P}^{\downarrow}_{m}(E[?]) \leq \mathfrak{P}^{\downarrow}(E'[?])$. Note that $\mathfrak{P}^{\downarrow}_{m}(E[?]) \leq \sup_{k \in \mathbb{N}} \mathfrak{P}^{\downarrow}_{m}(E[\underline{k}])$ and $\forall k \in \mathbb{N}$, $(\underline{k}, \underline{k}) \in [\![\emptyset \vdash \mathsf{nat}]\!](m)$, so $\mathfrak{P}^{\downarrow}_{m}(E[\underline{k}]) \leq \mathfrak{P}^{\downarrow}(E'[\underline{k}])$, and $\mathfrak{P}^{\downarrow}_{m}(E[?]) \leq \sup_{k \in \mathbb{N}} \mathfrak{P}^{\downarrow}(E'[\underline{k}]) = \mathfrak{P}^{\downarrow}(E'[?])$.
- Assume $\Gamma \vdash e \lesssim^{log}_{\downarrow} e'\colon \mathsf{nat}$. We will show that $\Gamma \vdash \mathsf{rand}\ e \lesssim^{log}_{\downarrow} \mathsf{rand}\ e'\colon \mathsf{nat}$. By assumption, for all $n < \omega$ and all $(\gamma, \gamma') \in [\![\Delta \vdash \Gamma]\!](n)$, $(e\gamma, e'\gamma') \in [\![\Delta \vdash \mathsf{nat}]\!]^{\top\top}(n)$. Now let $(E, E') \in [\![\Delta \vdash \mathsf{nat}]\!]^{\top}(n)$. By lemma 4.12,
$$(E[\mathsf{rand}\ [\,]\,], E'[\mathsf{rand}\ [\,]\,]) \in [\![\Delta \vdash \mathsf{nat}]\!]^{\top}(n),$$
and therefore,
$$\mathfrak{P}^{\downarrow}_{n}(E[\mathsf{rand}(e\gamma)]) \leq \mathfrak{P}^{\downarrow}(E'[\mathsf{rand}(e'\gamma')]),$$
so we can conclude
$$((\mathsf{rand}\ e)\gamma, (\mathsf{rand}\ e')\gamma') \in [\![\Delta \vdash \mathsf{nat}]\!]^{\top\top}(n).$$
- Assume $\Delta \mid \Gamma \vdash e \lesssim^{log}_{\downarrow} e'\colon \tau[\mu\alpha.\tau/\alpha]$. We will show $\Delta \mid \Gamma \vdash \mathsf{fold}\,e \lesssim^{log}_{\downarrow} \mathsf{fold}\,e'\colon \mu\alpha.\tau$. Let $n < \omega$, $(\gamma, \gamma') \in [\![\Delta \vdash \Gamma]\!]$, $(E, E') \in [\![\Delta \vdash \mu\alpha.\tau]\!](\varphi)^{\top}$. By assumption, $(e\gamma, e'\gamma') \in [\![\Delta \vdash \tau[\mu\alpha.\tau/\alpha]]\!](\varphi)^{\top\top}(n)$ and by lemma 4.13, $(E[\mathsf{fold}\,[\,]], E'[\mathsf{fold}\,[\,]]) \in [\![\Delta \vdash \tau[\mu\alpha.\tau/\alpha]]\!](\varphi)^{\top}(n)$, so we can conclude
$$\mathfrak{P}^{\downarrow}_{n}(E[\mathsf{fold}\,[e]]) \leq \mathfrak{P}^{\downarrow}(E'[\mathsf{fold}\,[e']]).$$
□

As a consequence, we can show that the logical relation is a sound and complete method for reasoning about contextual and CIU equivalence:

Theorem 4.15 (CIU-theorem)**.** *The relations $\lesssim^{log}_{\downarrow}$, $\lesssim^{ctx}_{\downarrow}$ and $\lesssim^{CIU}_{\downarrow}$ coincide.*

Proof. (1) ($\lesssim^{CIU}_{\downarrow} \subseteq \lesssim^{log}_{\downarrow}$). Assume $\Delta \mid \Gamma \vdash e \lesssim^{CIU}_{\downarrow} e' : \tau$. It suffices to note that by proposition 4.14, $\Delta \mid \Gamma \vdash e \lesssim^{log}_{\downarrow} e : \tau$, and that together with $\Delta \mid \Gamma \vdash e \lesssim^{CIU}_{\downarrow} e' : \tau$ this implies $\Delta \mid \Gamma \vdash e \lesssim^{log}_{\downarrow} e' : \tau$. Indeed, let $\varphi \in RVal(\Delta)$, $n < \omega$, $(\gamma, \gamma') \in [\![\Delta \vdash \Gamma]\!](\varphi)(n)$. From the definition of $[\![\Delta \vdash \tau]\!]^{\top\top}$ it follows that if $(e\gamma, e\gamma') \in [\![\Delta \vdash \tau]\!](\varphi)^{\top\top}(n)$ and for every $E \in Ectx(\tau)$, $\mathfrak{P}^{\downarrow}(E[e\gamma']) \leq \mathfrak{P}^{\downarrow}(E[e'\gamma'])$, then also $(e\gamma, e'\gamma')[\![\Delta \vdash \tau]\!](\varphi)^{\top\top}(n)$.

(2) ($\lesssim^{log}_{\downarrow} \subseteq \lesssim^{ctx}_{\downarrow}$). The transitive closure of $\lesssim^{log}_{\downarrow}$ is reflexive, transitive, compatible and may-adequate, so it is contained in $\lesssim^{ctx}_{\downarrow}$.

(3) ($\lesssim^{ctx}_{\downarrow} \subseteq \lesssim^{CIU}_{\downarrow}$). We show the case where both Δ and Γ contain a single variable, which can then be generalized by induction on their length. Assume $\alpha \mid x : \tau_1 \vdash e \lesssim^{ctx}_{\downarrow} e' : \tau_2$, and let $\sigma \in Type$, $v \in Val(\tau_1[\sigma/\alpha])$, and $E \in Ectx(\tau_2[\sigma/\alpha])$ such that $\vdash E : \tau_2[\sigma/\alpha] \Rightarrow \tau_3$ for some $\tau_3 \in Type$. By reflexivity $\emptyset \mid \emptyset \vdash v \lesssim^{ctx}_{\downarrow} v : \tau_1[\sigma/\alpha]$. By compatibility we can also show that $\emptyset \mid \emptyset \vdash (\Lambda.\lambda x.e)\ \_\ v \lesssim^{ctx}_{\downarrow} (\Lambda.\lambda x.e')\ \_\ v : \tau_2[\sigma/\alpha]$, and $\emptyset \mid \emptyset \vdash E[(\Lambda.\lambda x.e)\ \_\ v] \lesssim^{ctx}_{\downarrow} E[(\Lambda.\lambda x.e')\ \_\ v] : \tau_3$ By adequacy and the properties of deterministic reduction, we can conclude that $\mathfrak{P}^{\downarrow}(E[e[v/x]]) = \mathfrak{P}^{\downarrow}(E[(\Lambda.\lambda x.e)\ \_\ v]) \leq \mathfrak{P}^{\downarrow}(E[(\Lambda.\lambda x.e')\ \_\ v]) = \mathfrak{P}^{\downarrow}(E[e'[v/x]])$. □

4.4 Relation for Must-Termination

Analogously to the may-termination case, we build a relational interpretation of types that is must-adequate. This relation, however, is indexed over the countable ordinals. The relational interpretation of values can be found in fig. 3, which is essentially analogous to the relational interpretation of values in the may-termination case, except for the fact that step indices are taken from ω_1. In particular the interpretation of recursive types is defined by ordinal induction (and thus it is well-founded), and at a limit ordinal it is defined as the intersection of the interpretation of the type at all smaller ordinals. The other main difference lies in the way we use biorthogonality to lift the relation to contexts and expressions, which is based on the probability of must-termination:

$$r^{\top}(\xi) = \{(E, E') \mid \forall \beta \leq \xi, \forall (v, v') \in r(\beta).\quad \mathfrak{P}^{\Downarrow}_{\beta}(E[v]) \leq \mathfrak{P}^{\Downarrow}(E'[v'])\}$$
$$r^{\perp}(\xi) = \{(e, e') \mid \forall \beta \leq \xi, \forall (E, E') \in r(\beta).\quad \mathfrak{P}^{\Downarrow}_{\beta}(E[e]) \leq \mathfrak{P}^{\Downarrow}(E'[e'])\}$$

The logical relation for expressions in context is defined below:

Definition 4.16. *We say that $\Delta \mid \Gamma \vdash e \lesssim^{log}_{\Downarrow} e' : \tau$ if for all $\varphi \in RVal(\Delta)$, for all $\xi < \omega_1$, and all $(\gamma, \gamma') \in [\![\Delta \vdash \Gamma]\!](\varphi)(\xi)$, we have $(e\gamma, e'\gamma') \in [\![\Delta \vdash \tau]\!](\varphi)^{\top\top}(\xi)$.*

Proposition 4.17. *The relation $\lesssim^{log}_{\Downarrow}$ is must-adequate.*

Proof. Assume $\vdash e \lesssim^{log}_{\Downarrow} e' : \tau$. Note that, for all ξ, $([\,], [\,]) \in [\![\tau]\!]^{\top}(\xi)$. Then, we have that, for all $\xi < \omega_1$, $\mathfrak{P}^{\Downarrow}_{\xi}(e) \leq \mathfrak{P}^{\Downarrow}(e')$, so in particular, $\sup_{\beta<\omega_1} \mathfrak{P}^{\Downarrow}_{\beta}(e) \leq \mathfrak{P}^{\Downarrow}(e')$. On the other hand, $\mathfrak{P}^{\Downarrow}(e) = \sup_{\beta<\omega_1} \Psi^{\beta}(\perp)(e) = \sup_{\beta<\omega_1} \mathfrak{P}^{\Downarrow}_{\beta}(e)$, so $\mathfrak{P}^{\Downarrow}(e) \leq \mathfrak{P}^{\Downarrow}(e')$. □

This proof shows the need for step-indexing up to ω_1: to ensure that $\mathfrak{P}^{\Downarrow}(e) \leq \mathfrak{P}^{\Downarrow}(e')$, we need to require that (e, e') is in the relational interpretation until a β that makes $\mathfrak{P}^{\Downarrow}_{\beta}$ reach the fixpoint.

$$\llbracket \Delta \vdash \alpha \rrbracket(\varphi)(\xi) = r(\alpha)(\xi) \qquad \llbracket \Delta \vdash 1 \rrbracket(\varphi)(\xi) = \{(\langle\rangle, \langle\rangle)\} \qquad \llbracket \Delta \vdash \mathsf{nat} \rrbracket(\varphi)(\xi) = \{(\underline{k}, \underline{k}) \mid k \in \mathbb{N}\}$$

$$\llbracket \Delta \vdash \tau \to \sigma \rrbracket(\varphi)(\xi) = \bigcap_{\beta \le \xi} \{(\lambda x.e, \lambda y.e') \mid \forall (v, v') \in \llbracket \Delta \vdash \tau \rrbracket(\varphi)(\beta).(e[v/x], e'[v'/y]) \in \llbracket \Delta \vdash \sigma \rrbracket(\varphi)^{\top\top}(\beta)\}$$

$$\llbracket \Delta \vdash \tau \times \sigma \rrbracket(\varphi)(\xi) = \{(\langle v_1, v_2\rangle, \langle v_1', v_2'\rangle) \mid (v_1, v_1') \in \llbracket \Delta \vdash \tau \rrbracket(\varphi)(\xi) \wedge (v_2, v_2') \in \llbracket \Delta \vdash \sigma \rrbracket(\varphi)(\xi)\}$$

$$\begin{aligned}\llbracket \Delta \vdash \tau + \sigma \rrbracket(\varphi)(\xi) = {} & \{(\mathsf{inl}(v), \mathsf{inl}(v')) \mid (v, v') \in \llbracket \Delta \vdash \tau \rrbracket(\varphi)(\xi)\} \cup \\ & \{(\mathsf{inr}(v), \mathsf{inr}(v')) \mid (v, v') \in \llbracket \Delta \vdash \sigma \rrbracket(\varphi)(\xi)\}\end{aligned}$$

$$\begin{aligned}\llbracket \Delta \vdash \forall \alpha.\tau \rrbracket(\varphi)(\xi) = \{(\Lambda.e, \Lambda, e') \mid {} & \forall \sigma, \sigma' \in \mathit{Type}, \forall r \in \mathit{RVal}(\sigma, \sigma'). \\ & (e, e') \in \llbracket \Delta, \alpha \vdash \tau \rrbracket(\varphi[\alpha \mapsto (\sigma, \sigma', r)])^{\top\top}(\xi)\}\end{aligned}$$

$$\begin{aligned}\llbracket \Delta \vdash \mu\alpha.\tau \rrbracket(\varphi)(0) &= \mathit{Val}(\delta_1(\mu\alpha.\tau)) \times \mathit{Val}(\delta_2(\mu\alpha.\tau)) \\ \llbracket \Delta \vdash \mu\alpha.\tau \rrbracket(\varphi)(S\beta) &= \{(\mathsf{fold}\, v, \mathsf{fold}\, v') \mid (v, v') \in \llbracket \Delta, \alpha \vdash \tau \rrbracket(\varphi')(\beta)\} \\ \llbracket \Delta \vdash \mu\alpha.\tau \rrbracket(\varphi)(\chi) &= \bigcap_{\beta < \chi} \{(\mathsf{fold}\, v, \mathsf{fold}\, v') \mid (v, v') \in \llbracket \Delta, \alpha \vdash \tau \rrbracket(\varphi')(\beta)\} \\ \text{where } \varphi' &\triangleq \varphi[\alpha \mapsto (\delta_1(\mu\alpha.\tau), \delta_2(\mu\alpha.\tau), \llbracket \Delta \vdash \mu\alpha.\tau \rrbracket(\varphi))]\end{aligned}$$

Fig. 3. Definition of the logical relation on values for must-termination. We use δ_1, δ_2 and r to denote the three components of φ, $S\beta$ to denote a successor ordinal, and χ to denote a limit ordinal $\sup_{\beta<\chi} \beta$.

We can prove context extension lemmas analogous to the ones in the previous section. We show here the case for unfolding, which illustrates how ordinals are used in the proof:

Lemma 4.18. *Let $\xi < \omega_1$, $\varphi \in \mathit{RVal}(\Delta)$, and $(E, E') \in \llbracket \Delta \vdash \tau[\mu\alpha.\tau/\alpha] \rrbracket(\varphi)^{\top}(\xi)$. Then we have that $(E[\mathsf{unfold}\,[]], E'[\mathsf{unfold}\,[]]) \in \llbracket \Delta \vdash \mu\alpha.\tau/\alpha \rrbracket(\varphi)^{\top}(\xi)$.*

Proof. Let $(v, v') \in \llbracket \Delta \vdash \mu\alpha.\tau/\alpha \rrbracket(\varphi)^{\top}(\xi)$. Then, $v = \mathsf{fold}\, w$, $v' = \mathsf{fold}\, w'$ and for all $\beta < \xi$, $(w, w') \in \llbracket \Delta \vdash \tau[\mu\alpha.\tau/\alpha] \rrbracket(\varphi)^{\top}(m)$. We will show that for every $\beta \le \xi$, $\mathfrak{P}^{\Downarrow}_{\beta}(E[\mathsf{unfold}\ \mathsf{fold}\, w]) \le \mathfrak{P}^{\Downarrow}(E'[\mathsf{unfold}\ \mathsf{fold}\, w'])$. If $\xi = 0$, then $\mathfrak{P}^{\Downarrow}_{0}(E[\mathsf{unfold}\ \mathsf{fold}\, w]) = 0$ and we are done. If $\xi = \xi' + 1$, then let $\beta \le \xi = \xi' + 1$. Then:

$$\begin{aligned}\mathfrak{P}^{\Downarrow}_{\beta}(E[\mathsf{unfold}\ \mathsf{fold}\, w]) &\le \mathfrak{P}^{\Downarrow}_{\xi'+1}(E[\mathsf{unfold}\ \mathsf{fold}\, w]) \\ &= \mathfrak{P}^{\Downarrow}_{\xi'}(E[w]) \\ &\le \mathfrak{P}^{\Downarrow}(E'[w']) = \mathfrak{P}^{\Downarrow}(E'[\mathsf{unfold}\ \mathsf{fold}\, w']).\end{aligned}$$

Otherwise, $\xi = \sup_{\beta<\xi}$ is a limit ordinal. Then, let $\beta \le \xi$. We have:

$$\begin{aligned}\mathfrak{P}^{\Downarrow}_{\beta}(E[\mathsf{unfold}\ \mathsf{fold}\, w]) &\le \mathfrak{P}^{\Downarrow}_{\xi}(E[\mathsf{unfold}\ \mathsf{fold}\, w]) \\ &= \sup_{\zeta<\xi} \mathfrak{P}^{\Downarrow}_{\zeta}(E[\mathsf{unfold}\ \mathsf{fold}\, w]) \\ (\text{proposition } 3.10) &\le \sup_{\zeta<\xi} \mathfrak{P}^{\Downarrow}_{\zeta}(E[w]) \\ &\le \mathfrak{P}^{\Downarrow}(E'[w']) = \mathfrak{P}^{\Downarrow}(E'[\mathsf{unfold}\ \mathsf{fold}\, w']).\end{aligned}$$

Hence $(E[\mathsf{unfold}\,[]], E'[\mathsf{unfold}\,[]]) \in \llbracket \Delta \vdash \mu\alpha.\tau/\alpha \rrbracket(\varphi)^{\top}(\xi)$. □

As a consequence of the context extension lemmas, we have:

Proposition 4.19 (Fundamental property)**.** *The relation $\lesssim_{\Downarrow}^{log}$ is compatible. As a consequence, it is reflexive: for any type context Δ, typing context Γ, expression e and type τ, if $\Delta \mid \Gamma \vdash e : \tau$, then $\Delta \mid \Gamma \vdash e \lesssim_{\Downarrow}^{log} e : \tau$.*

Proof. The proof is analogous to the case of may-equivalence, after generalizing to the countable ordinals. We show the case of countable nondeterministic choice. Let $e \equiv ?$, and let $\xi < \omega_1$, $(E, E') \in [\![\Delta \vdash \mathsf{nat}]\!]^{\top}(\xi)$. We have to show that for every $\beta \leq \xi$, $\mathfrak{P}^{\Downarrow}_{\beta}(E[?]) \leq \mathfrak{P}^{\Downarrow}(E'[?])$. Note that $\mathfrak{P}^{\Downarrow}_{\beta}(E[?]) \leq \inf_{k \in \mathbb{N}} \mathfrak{P}^{\Downarrow}_{\beta}(E[\underline{k}])$ and that for all $k \in \mathbb{N}$, $(\underline{k}, \underline{k}) \in [\![\emptyset \vdash \mathsf{nat}]\!](\beta)$, so $\mathfrak{P}^{\Downarrow}_{\beta}(E[\underline{k}]) \leq \mathfrak{P}^{\Downarrow}(E'[\underline{k}])$, and therefore $\mathfrak{P}^{\Downarrow}_{\beta}(E[?]) \leq \inf_{k \in \mathbb{N}} \mathfrak{P}^{\Downarrow}(E'[\underline{k}]) = \mathfrak{P}^{\Downarrow}(E'[?])$. □

Theorem 4.20 (CIU-theorem)**.** *The relations $\lesssim_{\Downarrow}^{log}$, $\lesssim_{\Downarrow}^{ctx}$ and $\lesssim_{\Downarrow}^{CIU}$ coincide.*

5 CONNECTION TO SCHEDULER-BASED SEMANTICS

In the previous sections we have defined a way to resolve nondeterminism that consisted in taking at every point either the supremum or the infimum over all possible choices of the probability of termination of the continuation. In some sense, this is a global definition, since it needs to consider all possible runs of the program.

An alternative way of resolving nondeterminism is by the use of a scheduler, that selects locally on each nondeterministic choice a natural number. By parametrizing the reduction relation over schedulers, it reduces to a Markov chain, and then we can reason about the probability that it reaches a terminating state. This is analogous to the notion of schedulers used in concurrent settings, that choose on each step which thread to execute.

In this section we will present a notion of observation for programs that depends on a scheduler, and then show how it relates to may- and must-probabilistic termination.

Definition 5.1 (Scheduler)**.** *A scheduler is a tuple $\theta = (X, \theta_o, \theta_t)$ where X is a countable space state, $\theta_o : X \longrightarrow \mathbb{N}$ is an output function and $\theta_t : X \times Expr \longrightarrow X$ is a transition function. We let* Sch *denote the set of all schedulers.*

Morally, the scheduler is an automaton that can receive two kinds of queries. The output queries select a natural number that can be used to resolve nondeterminism. The transition function can be used to update the scheduler with information about the expression that is currently being evaluated. This is similar to the notion of resolution employed e.g. by [Bonchi et al. 2019].

Other approaches [Tassarotti and Harper 2019] allow the scheduler to know the full trace before resolving a nondeterministic choice. This is covered by our setting, since we can have a scheduler whose set of states is the set of finite traces. However, as it will later become clear, a scheduler that only knows the current term but not the full trace of execution cannot in general minimize the probability of termination.

We will now define the probability of termination under a scheduler. As in section 3 we will define it as the fixed point of an operator that computes the probability of termination after a single step. For a scheduler $\theta = (X, \theta_o, \theta_t)$ and a state $x \in X$, we define:

$$\Xi_\theta \colon (X \times \mathit{Expr} \to [0,1]) \to (X \times \mathit{Expr} \to [0,1])$$

$$\Xi_\theta(f)(x,e) = \begin{cases} 1 & \text{if } e \in \mathit{Val} \\ f(\theta_t(x, E[\underline{m}]), E[\underline{m}]) & e = E[?] \land \theta_o(x) = m \\ \sum_{1 \le m \le k} \frac{1}{k} \cdot f(\theta_t(x, E[\underline{m}]), E[\underline{m}]) & e = E[\mathsf{rand}\ \underline{k}] \\ f(\theta_t(x, e'), e') & e \longrightarrow_D e' \\ 0 & \text{otherwise} \end{cases}$$

Note that in particular, when resolving nondeterminism we make an output query, and that the scheduler is updated with the expression we are reducing to.

We can show that this operator is ω-continuous. Therefore, we can define:

Definition 5.2 (Probability of termination under a scheduler). *Let $e \in \mathit{Expr}$, $\theta = (X, \theta_o, \theta_t)$ be a scheduler and $x \in X$ be a state. The probability of termination of e under the scheduler θ with initial state $x \in X$, is denoted by $\mathfrak{P}^{\downarrow}_{\theta,x}(e)$, and defined as:*

$$\mathfrak{P}^{\downarrow}_{\theta,x}(e) = \sup_{n \in \omega} \Xi^n_\theta(\bot)(x,e)$$

We will now show that the probabilities of termination under the optimal and the pessimal schedulers coincide with the probabilities of may- and must-termination respectively.

To show the coincidence in the case of may-termination, we first prove the following lemma:

Lemma 5.3. *Let $e \in \mathit{Expr}$, $\theta = (X, \theta_o, \theta_t)$, $x \in X$, $n \in \mathbb{N}$. Then, $\Xi^n_\theta(\bot)(x,e) \le \Phi^n(\bot)(e)$.*

An optimal scheduler does not exist in general. Consider for instance the following program, which terminates with probability $(n-1)/n$ where n is picked by the scheduler.:

$$e \triangleq \mathsf{let}\ n = ?\ \mathsf{in}\ \mathsf{if}\ (\mathsf{rand}\ n) = 1\ \mathsf{then}\ \mathrm{diverge}\ \mathsf{else}\ \langle\rangle$$

Here, $\mathfrak{P}^{\downarrow}(e) = 1$, but there does not exist a scheduler θ such that $\mathfrak{P}^{\downarrow}_{\theta,x}(e) = 1$. However, for any ϵ we can find a scheduler that is ϵ-optimal, as stated by the following lemma:

Lemma 5.4. *Let $e \in \mathit{Expr}$. Then for all $n \in \mathbb{N}$, $\epsilon > 0$ there exists $\theta = (X, \theta_o, \theta_t)$ and $x \in X$ such that*

$$\Xi^n_\theta(\bot)(x,e) \ge \Phi^n(\bot)(e) - \epsilon$$

The proof is by induction on n. In particular, when resolving a nondeterministic choice $E[?]$, for every $\epsilon' > 0$ there is always some $m \in \mathbb{N}$ such that $\Phi^n(\bot)(E[\underline{m}]) \ge \sup_{k \in \mathbb{N}} \Phi^n(\bot)(E[\underline{k}]) - \epsilon'$, so we can make the scheduler choose m in that step. For instance in the example program above, we would make the scheduler choose the first integer larger than $1/\epsilon'$.

As a consequence, we get the following theorem:

Theorem 5.5. *Let $e \in \mathit{Expr}$. Then*

$$\mathfrak{P}^{\downarrow}(e) = \sup_{\theta \in \mathsf{Sch}, \theta = (X, \theta_o, \theta_t), x \in X} \mathfrak{P}^{\downarrow}_{\theta,x}(e)$$

An analogous result can be proven for the probability of must-termination. We begin by showing:

Lemma 5.6. *Let $e \in \mathit{Expr}$, $\theta = (X, \theta_o, \theta_t)$, $x \in X$, $\alpha \in \omega_1$. Then, $\Xi^\alpha_\theta(\bot)(x,e) \ge \Psi^\alpha(\bot)(e)$. In particular, $\mathfrak{P}^{\downarrow}_{\theta}(e) \ge \mathfrak{P}^{\Downarrow}(e)$.*

Lemma 5.7. *Let $e \in Expr$. Then for all $n \in \mathbb{N}$, $\epsilon > 0$ there exists $\theta = (X, \theta_o, \theta_t)$ and $x \in X$ such that*

$$\mathfrak{P}^{\downarrow}_{\theta}(x, e) \leq \mathfrak{P}^{\Downarrow}(e) + \epsilon$$

The proof constructs an ϵ-pessimal scheduler θ such that

$$\forall e. \forall n. \forall \epsilon > 0. \exists x \in X_{\theta}. \sup_{n \in \omega} \Xi^{n}_{\theta}(\bot)(x, e) \leq \sup_{\alpha \in \omega_1} \Psi^{\alpha}(\bot)(e) + \epsilon \tag{1}$$

The idea is similar to the may-termination case, except than when resolving a nondeterministic choice $E[?]$ we pick $m \in \mathbb{N}$ such that

$$\mathfrak{P}^{\Downarrow}(E[\underline{m}]) \leq \inf_{k \in \mathbb{N}} \mathfrak{P}^{\Downarrow}(E[\underline{k}]) + 2^{-|\pi|} \cdot \epsilon$$

where $|\pi|$ is the length of the current trace. This implies that the scheduler needs to keep track not only of the current term, but also of the length of the trace in order to minimize the probability of termination after nondeterministic choices. Indeed, consider the following program (using syntactic sugar for recursion):

$$\mathsf{letrec}\, f\ \_ = (\mathsf{let}\, n = ?\, \mathsf{in}\, \mathsf{if}\, (\mathsf{rand}\ n) = 1\, \mathsf{then}\, \langle\rangle\, \mathsf{else}\, f\ \langle\rangle)\, \mathsf{in}\, f\ \langle\rangle$$

This program iterates a fair random choice between 1 and n, where n is chosen nondeterministically by the scheduler. A scheduler that depends only on the current term will necessarily always pick the same n in the expression above every time the recursion is unfolded. Therefore the program will terminate with probability 1. A scheduler that observes the trace can increase n (e.g., duplicating it) after every unfolding, and therefore the probability of termination can be made arbitrarily close to zero.

From the previous two lemmas, we get:

Theorem 5.8. *For any expression $e \in Expr$, we have*

$$\mathfrak{P}^{\Downarrow}(e) = \inf_{\theta \in \mathsf{Sch}, \theta = (X, \theta_o, \theta_t), x \in X} \sup_{n \in \mathbb{N}} \Xi^{n}_{\theta}(\bot)(x, e)$$

To summarize, Theorems 5.5 and 5.8 show that we can find schedulers that get as close as we want to the probability of may-termination or the probability of must termination. The probability of may-termination coincides with the supremum of the probabilities of termination over all schedulers, and the probability of must-termination coincides with the infimum of the probabilities of termination over all schedulers.

6 APPLICATIONS

We first define syntactic sugar for binary probabilistic and nondeterministic choice, which we will use in the examples. Given two expressions e, e' and a two naturals p, q such that $q \neq 0$ and $p \leq q$, we write $e \oplus_{p/q} e'$ for $\mathsf{if}\, (\mathsf{rand}\ q) \leq p\, \mathsf{then}\, e\, \mathsf{else}\, e'$. For the particular case of $p = 1, q = 2$ we simply write $e \oplus e'$. Similarly, we write e **or** e' for $\mathsf{if}\, ? > 1\, \mathsf{then}\, e\, \mathsf{else}\, e'$. Note that both in $e \oplus_{p/q} e'$ and e **or** e' the order of evaluation dictates that first the probabilistic or nondeterministic choice is made, and only then the corresponding subexpression is evaluated.

We can also define a call-by-value fixpoint operator in our system as below:

$$\mathsf{fix} \colon \forall \alpha. \forall \alpha'. ((\alpha \to \alpha') \to (\alpha \to \alpha')) \to \alpha \to \alpha'$$
$$\mathsf{fix} = \Lambda. \Lambda. \lambda F. \lambda z. e_F\ (\mathsf{fold}\ \ e_F)\ z \qquad (\text{where } e_F = \lambda y. \mathsf{let}\, y' = \mathsf{unfold}\ \ y\, \mathsf{in}\, F\ (\lambda x. y'\ y\ x))$$

In particular, for every closed type τ, we can write a closed expression $\Omega_{\tau} \triangleq \mathsf{fix}\ \_\ \_\ (\lambda f. f)\ \langle\rangle$ such that $\Omega_{\tau} \colon \tau$ and $\mathfrak{P}^{\downarrow}(\Omega_{\tau}) = \mathfrak{P}^{\Downarrow}(\Omega_{\tau}) = 0$.

6.1 Extensionality

As a first consequence of the CIU theorems, we obtain extensionality for values:

Proposition 6.1. *Let Δ be a type formation context, $\Delta \vdash \Gamma$, $\sigma, \tau \in \mathit{Type}(\Delta)$ and suppose f, f' are values such that $\Delta \mid \Gamma \vdash f : \sigma \to \tau$ and $\Delta \mid \Gamma \vdash f' : \sigma \to \tau$. Then $\Delta \mid \Gamma \vdash f \lesssim^{ctx}_{\downarrow} f' : \sigma \to \tau$ iff for all values v such that $\Delta \mid \Gamma \vdash v : \sigma$, we have $\Delta \mid \Gamma \vdash f\ v \lesssim^{ctx}_{\downarrow} f'\ v : \tau$. The analogous holds for $\lesssim^{ctx}_{\Downarrow}$.*

A similar extensionality result holds for types as well.

6.2 Fixpoint Combinator

We show helpful properties of the fixpoint combinator to use in further examples.

Definition 6.2. *Let σ, τ be types. We say that $f \in \mathit{Val}(\sigma \to \tau)$ is deterministic if for every $v \in \mathit{Val}(\sigma)$ there exists $w \in \mathit{Val}(\tau)$ such that $f\ v \cong^{ctx}_{\downarrow} w$ and $f\ v \cong^{ctx}_{\Downarrow} w$.*

As a consequence of extensionality (proposition 6.1), we can show:

Proposition 6.3. *Let σ, τ be types, and $G \in \mathit{Val}((\sigma \to \tau) \to \sigma \to \tau)$ deterministic. Then $\mathsf{fix}\,\_\,\_\,G \cong^{ctx}_{\downarrow} G\ (\mathsf{fix}\,\_\,\_\,G)$ and $\mathsf{fix}\,\_\,\_\,G \cong^{ctx}_{\Downarrow} G\ (\mathsf{fix}\,\_\,\_\,G)$.*

Proof. By deterministic reduction, we can show

$$(\mathsf{fix}\,\_\,\_\,G)\ v \cong^{ctx}_{\downarrow} G\ (\lambda z.e_G\ (\mathsf{fold}\,e_G)\ z)\ v$$

Since G is deterministic, there exists $w \in \mathit{Val}(\sigma \to \tau)$ such that $G\ (\lambda x.e_G\ (\mathsf{fold}\,e_G)\ x) \cong^{ctx}_{\downarrow} w$, and therefore, $(\mathsf{fix}\,\_\,\_\,G)\ v \cong^{ctx}_{\downarrow} w\ v$. On the other hand by deterministic reduction we have

$$(\mathsf{fix}\,\_\,\_\,G) \cong^{ctx}_{\downarrow} (\lambda z.e_G\ (\mathsf{fold}\,e_G)\ z)$$

so by transitivity and compatibility $(\lambda z.e_G\ (\mathsf{fold}\,e_G)\ z)\ v \cong^{ctx}_{\downarrow} w\ v$. Now we can apply proposition 6.1

$$(\lambda z.e_G\ (\mathsf{fold}\,e_G)\ z) \cong^{ctx}_{\downarrow} G\ (\lambda z.e_G\ (\mathsf{fold}\,e_G)\ z)$$

and again by compatibility and transitivity $(\mathsf{fix}\,\_\,\_\,G) \cong^{ctx}_{\downarrow} G\ (\mathsf{fix}\,\_\,\_\,G)$. Analogously, we show $(\mathsf{fix}\,\_\,\_\,G) \cong^{ctx}_{\Downarrow} G\ (\mathsf{fix}\,\_\,\_\,G)$. □

6.3 Algebraic Theory

First we study the algebraic theories induced by the contextual equivalence notions; see the summary in fig. 4. We present only the results for may-equivalence, but analogous results hold for must-equivalence. For the binary, non-deterministic choice, the language satisfies the equational theory of a join semilattice (meet semilattice in the case of must-equivalence):

Proposition 6.4. *Let $\tau \in \mathit{Type}$, $e_1, e_2, e_2 \in \mathit{Expr}(\tau)$. We have:*

$$e_1\ \mathbf{or}\ e_1 \cong^{ctx}_{\downarrow} e_1, \qquad e_1\ \mathbf{or}\ e_2 \cong^{ctx}_{\downarrow} e_2\ \mathbf{or}\ e_1, \qquad e_1\ \mathbf{or}\ (e_2\ \mathbf{or}\ e_3) \cong^{ctx}_{\downarrow} (e_1\ \mathbf{or}\ e_2)\ \mathbf{or}\ e_3$$

For the binary probabilistic choice we get the equational theory of a convex algebra:

Proposition 6.5. *Let $\tau \in \mathit{Type}$, $e_1, e_2, e_2 \in \mathit{Expr}(\tau)$, and $p, q \in [0, 1] \cap \mathbb{Q}$. We have:*

$$e_1 \oplus_p e_1 \cong^{ctx}_{\downarrow} e_1, \qquad e_1 \oplus_p e_2 \cong^{ctx}_{\downarrow} e_2 \oplus_{1-p} e_1, \qquad (e_1 \oplus_p e_2) \oplus_q e_3 \cong^{ctx}_{\downarrow} e_1 \oplus_{pq} (e_2 \oplus_{\frac{q-pq}{1-pq}} e_3)$$

The combination of a join semilattice and a convex algebra together with a distributive law is known as a convex semilattice [Mio et al. 2021]. The distributive law also holds here:

$$\begin{aligned}
e_1 \oplus_p e_1 &\cong^{ctx}_{\downarrow} e_1 & e_1 \oplus_p e_1 &\cong^{ctx}_{\Downarrow} e_1 \\
e_1 \oplus_p e_2 &\cong^{ctx}_{\downarrow} e_2 \oplus_{1-p} e_1 & e_1 \oplus_p e_2 &\cong^{ctx}_{\Downarrow} e_2 \oplus_{1-p} e_1 \\
(e_1 \oplus_p e_2) \oplus_q e_3 &\cong^{ctx}_{\downarrow} e_1 \oplus_{pq} (e_2 \oplus_{\frac{q-pq}{1-pq}} e_3) & (e_1 \oplus_p e_2) \oplus_q e_3 &\cong^{ctx}_{\Downarrow} e_1 \oplus_{pq} (e_2 \oplus_{\frac{q-pq}{1-pq}} e_3) \\
e_1 \textbf{ or } e_1 &\cong^{ctx}_{\downarrow} e_1 & e_1 \textbf{ or } e_1 &\cong^{ctx}_{\Downarrow} e_1 \\
e_1 \textbf{ or } e_2 &\cong^{ctx}_{\downarrow} e_2 \textbf{ or } e_1 & e_1 \textbf{ or } e_2 &\cong^{ctx}_{\Downarrow} e_2 \textbf{ or } e_1 \\
e_1 \textbf{ or } (e_2 \textbf{ or } e_3) &\cong^{ctx}_{\downarrow} (e_1 \textbf{ or } e_2) \textbf{ or } e_3 & e_1 \textbf{ or } (e_2 \textbf{ or } e_3) &\cong^{ctx}_{\Downarrow} (e_1 \textbf{ or } e_2) \textbf{ or } e_3 \\
e_1 \textbf{ or } \Omega_\tau &\cong^{ctx}_{\downarrow} e_1 & e_1 \textbf{ or } \Omega_\tau &\cong^{ctx}_{\Downarrow} \Omega_\tau \\
e_1 \oplus_p (e_2 \textbf{ or } e_3) &\cong^{ctx}_{\downarrow} (e_1 \oplus_p e_2) \textbf{ or } (e_1 \oplus_p e_3) & e_1 \oplus_p (e_2 \textbf{ or } e_3) &\cong^{ctx}_{\Downarrow} (e_1 \oplus_p e_2) \textbf{ or } (e_1 \oplus_p e_3)
\end{aligned}$$

Fig. 4. Equational theory for may- and must-equivalence. Here, $\tau \in \mathit{Type}$ and $e_1, e_2, e_3 \in \mathit{Expr}(\tau)$.

Proposition 6.6. *Let $\tau \in \mathit{Type}$, $e_1, e_2, e_3 \in \mathit{Expr}(\tau)$, and $p \in [0,1] \cap \mathbb{Q}$. Then*

$$e_1 \oplus_p (e_2 \textbf{ or } e_3) \cong^{ctx}_{\downarrow} (e_1 \oplus_p e_2) \textbf{ or } (e_1 \oplus_p e_3)$$

The three propositions above are proven using the CIU relation. We show one of the cases below:

Proof of Proposition 6.6. To simplify the notation, we assume $p = 1/2$, but the proof is analogous for any other rational in $[0,1]$. Consider a context $E \in \mathit{Ectx}(\tau)$. By expanding the definitions we get:

$$\begin{aligned}
e_1 \oplus (e_2 \textbf{ or } e_3) &\triangleq \text{if}\,(\text{rand } 2) \le 1 \text{ then } e_1 \text{ else } (\text{if } ? > 1 \text{ then } e_2 \text{ else } e_3) \\
(e_1 \oplus e_2) \textbf{ or } (e_1 \oplus e_3) &\triangleq \\
&\text{if } ? > 1 \text{ then } (\text{if}\,(\text{rand } 2) \le 1 \text{ then } e_1 \text{ else } e_2) \text{ else } (\text{if}\,(\text{rand } 2) \le 1 \text{ then } e_1 \text{ else } e_3)
\end{aligned}$$

Showing that the two expressions are contextually equivalent amounts to showing a commuting conversion. That is, consider the context $C_P \triangleq \text{if}\,(\text{rand } 2) \le 1 \text{ then } e_1 \text{ else } [\,]$. Then have to show:

$$C_P[\text{if } ? > 1 \text{ then } e_2 \text{ else } e_3] \cong^{log}_{\downarrow} \text{if } ? > 1 \text{ then } C_P[e_2] \text{ else } C_P[e_3]$$

By applying Proposition 3.8, we have:

$$\begin{aligned}
\mathfrak{P}^{\downarrow}(E[C_P[\text{if } ? > 1 \text{ then } e_2 \text{ else } e_3]) &= (1/2) \cdot \mathfrak{P}^{\downarrow}(E[e_1]) + (1/2) \cdot \mathfrak{P}^{\downarrow}(E[\text{if } ? > 1 \text{ then } e_2 \text{ else } e_3]) \\
&= (1/2) \cdot \mathfrak{P}^{\downarrow}(E[e_1]) + (1/2) \cdot \sup_{k \in \mathbb{N}} \mathfrak{P}^{\downarrow}(E[\text{if } \underline{k} > 1 \text{ then } e_2 \text{ else } e_3]) \\
&= (1/2) \cdot \mathfrak{P}^{\downarrow}(E[e_1]) + (1/2) \cdot \max\{\mathfrak{P}^{\downarrow}(E[e_2]), \mathfrak{P}^{\downarrow}(E[e_3])\}
\end{aligned}$$

and

$$\begin{aligned}
&\mathfrak{P}^{\downarrow}(E[\text{if } ? > 1 \text{ then } C_P[e_2] \text{ else } C_P[e_3]]) \\
&= \sup_{k \in \mathbb{N}} \mathfrak{P}^{\downarrow}(E[\text{if } \underline{k} > 1 \text{ then } C_P[e_2] \text{ else } C_P[e_3]]) \\
&= \max\{\mathfrak{P}^{\downarrow}(E[C_P[e_2]]), \mathfrak{P}^{\downarrow}(E[C_P[e_3]])\} \\
&= \max\{(1/2) \cdot \mathfrak{P}^{\downarrow}(E[e_1]) + (1/2) \cdot \mathfrak{P}^{\downarrow}(E[e_2]), (1/2) \cdot \mathfrak{P}^{\downarrow}(E[e_1]) + \mathfrak{P}^{\downarrow}(E[e_3])\} \\
&= (1/2) \cdot \mathfrak{P}^{\downarrow}(E[e_1]) + (1/2) \cdot \max\{\mathfrak{P}^{\downarrow}(E[e_2]), \mathfrak{P}^{\downarrow}(E[e_3])\}
\end{aligned}$$

Therefore, we can conclude that

$$\mathfrak{P}^{\downarrow}(E[C_P[\mathsf{if}\,?>1\,\mathsf{then}\,e_2\,\mathsf{else}\,e_3]) = \mathfrak{P}^{\downarrow}(E[\mathsf{if}\,?>1\,\mathsf{then}\,C_P[e_2]\,\mathsf{else}\,C_P[e_3]])$$

and thus $C_P[\mathsf{if}\,?>1\,\mathsf{then}\,e_2\,\mathsf{else}\,e_3] \cong_{\downarrow}^{CIU} \mathsf{if}\,?>1\,\mathsf{then}\,C_P[e_2]\,\mathsf{else}\,C_P[e_3]$. By the CIU theorem, we can also show that the two expressions are contextually equivalent. This concludes the proof. □

Furthermore, the results can be extended to countable choice. Indeed,

Proposition 6.7. *Let $\tau \in Type$, $e \in Expr(\tau)$, $f \in Val(\mathsf{nat} \to \tau)$ and $p \in [0,1] \cap \mathbb{Q}$. Then*

$$e \oplus_p (f\ ?) \cong_{\downarrow}^{ctx} (\lambda n.e \oplus_p f\ n)\ ?$$

Proof. We assume w.l.o.g. $p = 1/2$. Consider an evaluation context $E \in Ectx(\tau)$. Then:

$$\begin{aligned}\mathfrak{P}^{\downarrow}(E[e_1 \oplus_p (f\ ?)]) &= (1/2)\cdot\mathfrak{P}^{\downarrow}(E[e_1]) + (1/2)\cdot\mathfrak{P}^{\downarrow}(E[(f\ ?)]) \\ &= (1/2)\cdot\mathfrak{P}^{\downarrow}(E[e_1]) + (1/2)\cdot\sup_{k\in\mathbb{N}}\mathfrak{P}^{\downarrow}(E[f\ \underline{k}])\end{aligned}$$

and

$$\begin{aligned}\mathfrak{P}^{\downarrow}(E[(\lambda n.e \oplus_p f\ n)\ ?]) &= \sup_{k\in\mathbb{N}}\mathfrak{P}^{\downarrow}(E[(\lambda n.e \oplus_p f\ n)\ \underline{k}]) \\ &= \sup_{k\in\mathbb{N}}\mathfrak{P}^{\downarrow}(E[e \oplus_p f\ \underline{k}]) \\ &= \sup_{k\in\mathbb{N}}\Big((1/2)\cdot\mathfrak{P}^{\downarrow}(E'[e]) + (1/2)\cdot\mathfrak{P}^{\downarrow}(E[f\ \underline{k}])\Big) \\ &= (1/2)\cdot\mathfrak{P}^{\downarrow}(E[e]) + \sup_{k\in\mathbb{N}}(1/2)\cdot\mathfrak{P}^{\downarrow}(E[f\ \underline{k}])\end{aligned}$$

Therefore, $\mathfrak{P}^{\downarrow}(E[e_1 \oplus_p (f\ ?)]) = \mathfrak{P}^{\downarrow}(E[(\lambda n.e \oplus_p f\ n)\ ?])$, so $e_1 \oplus_p (f\ ?) \cong_{\downarrow}^{CIU} (\lambda n.e \oplus_p f\ n)\ ?$. By applying the CIU theorem, we obtain that the two expressions are also contextually equivalent. This concludes the proof. □

Example 6.8. Recall the example program from the introduction. We can write a similar program that initializes x to 0, and then chooses with equal probability between running e (that does not have x as a free variable), or forking a thread that increments x and then calling f with x:

$$x = 0; e \oplus \{\mathsf{fork}\{\mathsf{while(true)}\{\mathsf{x := !x + 1}\}\}; f(!x)\}$$

Using countable nondeterministic choice, we can write this program in out language as $e \oplus (f\ ?)$, where we choose at random between running e or running f with a nondeterministically chosen argument. The proposition above implies that this program should be contextually equivalent to $(\lambda n.e \oplus f\ n)\ ?$, where an argument is chosen nondeterministically, and then we choose at random between running e or running f with said argument. Morally, this corresponds to the concurrent program below:

$$x = 0; \mathsf{fork}\{\mathsf{while(true)}\{\mathsf{x := !x + 1}\}\}; e \oplus f(!x)$$

In other words, the programs are equivalent whether the thread that increments x in a loop is forked before or after the probabilistic choice is resolved.

Example 6.9. As observed by Varacca and Winskel [2006], if the distributive law holds, so does the convexity law below: $e_1\ \mathbf{or}\ e_2 \cong_{\downarrow}^{ctx} e_1\ \mathbf{or}\ e_2\ \mathbf{or}\ (e_1 \oplus e_2)$.

6.4 Guessing Coin Tosses

Consider the two programs of type $1 \longrightarrow \mathsf{nat}$ below

$$\mathsf{toss} \triangleq \lambda x.\underline{1} \oplus \underline{2} \qquad\qquad \mathsf{guess} \triangleq \lambda x.\underline{1}\ \mathbf{or}\ \underline{2}$$

We then have:

$$\mathsf{if}\,\mathsf{guess}\ \langle\rangle = \mathsf{toss}\ \langle\rangle\ \mathsf{then}\ \langle\rangle\ \mathsf{else}\ \Omega_1 \cong^{ctx}_{\downarrow} \langle\rangle \oplus \Omega_1$$
$$\mathsf{if}\,\mathsf{guess}\ \langle\rangle = \mathsf{toss}\ \langle\rangle\ \mathsf{then}\ \langle\rangle\ \mathsf{else}\ \Omega_1 \cong^{ctx}_{\Downarrow} \langle\rangle \oplus \Omega_1$$

Here guess is evaluated before toss and thus guess provides the scheduler with a choice between two programs that diverge with probability 1/2. Therefore, the best and the worst scheduler have the same probability of termination. But if we reverse the order, the result of the toss has already been fixed when guess is called and can be seen by the scheduler. Therefore, the scheduler can choose between terminating with probability 1 or diverging with probability 1:

$$\mathsf{if}\,\mathsf{toss}\ \langle\rangle = \mathsf{guess}\ \langle\rangle\ \mathsf{then}\ \langle\rangle\ \mathsf{else}\ \Omega_1 \cong^{ctx}_{\downarrow} \langle\rangle$$
$$\mathsf{if}\,\mathsf{toss}\ \langle\rangle = \mathsf{guess}\ \langle\rangle\ \mathsf{then}\ \langle\rangle\ \mathsf{else}\ \Omega_1 \cong^{ctx}_{\Downarrow} \Omega_1$$

This result suggests that, in general, the order of probabilistic and nondeterministic choices cannot be reversed, even though in this case it is tempting to. It could be interesting to study equivalences under a more restricted class of schedulers that is not allowed to observe the entire program as suggested in [Chadha et al. 2010]. We discuss this further in section 8.

6.5 Skip Lists

Following the example of Tassarotti and Harper [2019], we use nondeterminism to study the concurrent behavior of skip lists. Skip lists are a probabilistic data structure to represent sets of integers in such a way that checking membership has low expected cost. In their simplest formulation they consist of two levels of lists: an ordered bottom list, and an ordered top list that is a subset of the bottom list. When inserting an element, we always insert it (in a sorted manner) in the bottom list, and then choose at random whether to insert it in the top list. When checking membership, we will then search first in the top list, and if we fail, we will proceed to search in the bottom list.

We will model the behavior of a skip list in a concurrent setting by assuming we have a list of insertion operations that are nondeterministically ordered by a scheduler. We will then show that the resulting top list is contextually equivalent to the top list resulting from running the operations in a predetermined order. Morally this implies that for any client that uses the sequential implementation, the concurrent implementation can be used in its place.

We can model lists in our language using the type $[\tau] = \mu\alpha.1 + \tau \times \alpha$. As usual, we consider two constructors $\mathsf{nil}\colon \forall\alpha.[\alpha]$ and $\mathsf{cons}\colon \forall\alpha.\alpha \longrightarrow [\alpha] \longrightarrow [\alpha]$ (sometimes written :: and in infix position) and an append function $\mathsf{++}\colon \forall\alpha.[\alpha] \times [\alpha] \longrightarrow [\alpha]$. To simplify notation, we will use syntactic sugar to define recursive functions over lists by pattern matching.

The following is the model of the concurrent implementation of the skip list. We first define the following function that nondeterministically splits the list in three parts: an initial segment, an intermediate element, and the final segment:

$$\mathsf{split}\ l\ x\ \mathsf{nil} \triangleq \langle l, x, \mathsf{nil}\rangle$$
$$\mathsf{split}\ l\ x\ \mathsf{cons}(y, ys) \triangleq \langle l, x, \mathsf{cons}(y, ys)\rangle\ \mathbf{or}\ (\mathsf{split}\ (l\mathsf{++}\mathsf{cons}(x, \mathsf{nil}))\ y\ ys)$$

Using this operation we implement the skiplist method; for simplicity we (1) return only the top list, and (2) sort the list at the end, rather than maintaining a sorted list.

$$\begin{aligned}\mathsf{skiplist}\ \mathsf{nil}\ tl &\triangleq \mathsf{sort}\ tl\\ \mathsf{skiplist}\ \mathsf{cons}(x, xs)\ tl &\triangleq \mathsf{let}\ (l, y, r) = \mathsf{split}\ \mathsf{nil}\ x\ xs\ \mathsf{in}\ \mathsf{let}\ tl' = tl \oplus \mathsf{cons}(y, tl)\ \mathsf{in}\ \mathsf{skiplist}\ (l{+\!+}r)\ tl'\end{aligned}$$

We also define another implementation where elements are inserted in the predetermined order:

$$\begin{aligned}\mathsf{skiplist}'\ \mathsf{nil}\ tl &\triangleq \mathsf{sort}\ tl\\ \mathsf{skiplist}'\ \mathsf{cons}(x, xs)\ tl &\triangleq \mathsf{let}\ tl' = tl \oplus \mathsf{cons}(x, tl)\ \mathsf{in}\ \mathsf{skiplist}'\ xs\ tl'\end{aligned}$$

We now prove that both implementations are contextually equivalent, which intuitively expresses that the (nondeterministic model of the) concurrent implementation is a refinement of a sequential implementation. In the proof we assume the following facts about the ++ and the sort functions. For all $x, y \in \mathit{Val}(\mathsf{nat})$ and $l_1, l_2, l_3 \in \mathit{Val}([\mathsf{nat}])$:

$$\mathsf{nil} ++ l_1 \cong^{ctx}_{\Downarrow} l_1 ++ \mathsf{nil} \cong^{ctx}_{\Downarrow} l_1 \qquad (x :: l_1) ++ l_2 \cong^{ctx}_{\Downarrow} x :: (l_1 ++ l_2)$$

$$l_1 ++ (l_2 ++ l_3) \cong^{ctx}_{\Downarrow} (l_1 ++ l_2) ++ l_3 \qquad \mathsf{sort}(l_1 ++ x :: l_2) \cong^{ctx}_{\Downarrow} \mathsf{sort}(x :: l_1 ++ l_2)$$

We also need the following lemmas:

Lemma 6.10. *Let $\tau \in \mathit{Type}$, $x \in \mathit{Val}(\tau)$, $l, r \in \mathit{Val}([\tau])$. Then*

$$\mathit{let}\ (ys, z, zs) = \mathsf{split}\ l\ x\ r\ \mathit{in}\ ys{+\!+}\mathsf{cons}(z, zs) \quad \cong^{ctx}_{\Downarrow} l{+\!+}\mathsf{cons}(x, r)$$

Lemma 6.11. *Let $\tau \in \mathit{Type}$, $x, y \in \mathit{Val}(\tau)$, $l, ys, zs \in \mathit{Val}([\tau])$. Then*

$$\mathsf{skiplist}'\ l\ (ys ++ (z :: zs)) \cong^{ctx}_{\Downarrow} \mathsf{skiplist}'\ l\ (z :: (ys ++ zs))$$

As a consequence, we get:

Lemma 6.12. *Let $\tau \in \mathit{Type}$, $x, y \in \mathit{Val}(\tau)$, $l, tl \in \mathit{Val}([\tau])$. Then*

$$\mathsf{skiplist}'\ x :: y :: l\ tl \cong^{ctx}_{\Downarrow} \mathsf{skiplist}'\ y :: x :: l\ tl$$

Lemma 6.13. *Let $\tau \in \mathit{Type}$, $x \in \mathit{Val}(\tau)$, $l, r, tl \in \mathit{Val}([\tau])$. Then*

$$\mathsf{skiplist}'\ (l ++ \mathsf{cons}(x, r))\ tl \cong^{ctx}_{\Downarrow} \mathsf{skiplist}'\ (\mathsf{cons}(x, l ++ r))\ tl$$

Finally, we can prove:

Theorem 6.14. *Let $\tau \in \mathit{Type}$, $l, tl \in \mathit{Val}([\tau])$. Then* $\mathsf{skiplist}\ xs\ tl \cong^{ctx}_{\Downarrow} \mathsf{skiplist}'\ xs\ tl$

Proof. By induction on the length of xs. If it is 0, then both sides reduce to sort tl. Otherwise,

$$\begin{aligned}
\mathsf{skiplist}\ \mathsf{cons}(x, xs)\ tl \cong^{ctx}_{\Downarrow} &\ \mathsf{let}\ (l, y, r) = \mathsf{split}\ \mathsf{nil}\ x\ xs\ \mathsf{in}\\
&\ \mathsf{let}\ tl' = tl \oplus \mathsf{cons}(y, tl)\ \mathsf{in}\\
&\ \mathsf{skiplist}\ (l{+\!+}r)\ tl'\\
\text{(I.H.)} \cong^{ctx}_{\Downarrow} &\ \mathsf{let}\ (l, y, r) = \mathsf{split}\ \mathsf{nil}\ x\ xs\ \mathsf{in}\\
&\ \mathsf{let}\ tl' = tl \oplus \mathsf{cons}(y, tl)\ \mathsf{in}\\
&\ \mathsf{skiplist}'\ (l{+\!+}r)\ tl'\\
\text{(Def.)} \cong^{ctx}_{\Downarrow} &\ \mathsf{let}\ (l, y, r) = \mathsf{split}\ \mathsf{nil}\ x\ xs\ \mathsf{in}\\
&\ \mathsf{skiplist}'\ \mathsf{cons}(y, l{+\!+}r)\ tl\\
\text{(lemma 6.13)} \cong^{ctx}_{\Downarrow} &\ \mathsf{let}\ (l, y, r) = \mathsf{split}\ \mathsf{nil}\ x\ xs\ \mathsf{in}\\
&\ \mathsf{skiplist}'\ l{+\!+}\mathsf{cons}(y, r)\ tl\\
\text{(lemma 6.10)} \cong^{ctx}_{\Downarrow} &\ \mathsf{skiplist}'\ (\mathsf{nil}{+\!+}\mathsf{cons}(x, xs))\ tl\\
\cong^{ctx}_{\Downarrow} &\ \mathsf{skiplist}'\ \mathsf{cons}(x, xs)\ tl
\end{aligned}$$

□

7 RELATED WORK

Denotational semantic models for probabilistic programming languages were introduced in Kozen's seminal paper [Kozen 1981]. Powerdomains, initially used to model nondeterminism [Plotkin 1976] were later adapted to probabilistic languages [Jones and Plotkin 1989; Saheb-Djahromi 1980], but extending this construction to higher-order is challenging [Jung and Tix 1998]. Semantics based on measurable spaces are also hard to generalize to higher-order since the category of measurable spaces is not Cartesian closed [Aumann 1961]. In recent years, Cartesian closed categories to denote higher-order probabilistic languages have been proposed in [Ehrhard et al. 2018, 2014; Heunen et al. 2017]. Later, these models were extended to support recursion [Ehrhard and Tasson 2019; Vákár et al. 2019]. However, these models are fairly involved, and in general it is unclear how to incorporate further extensions, such as countable nondeterministic choice, or impredicative polymorphism into them.

Step-indexed logical relations were introduced by Appel and McAllester [2001] to model recursion, and have been shown to scale well to higher-order languages that support recursive types and polymorphism [Ahmed 2006], as well as a variety of effects, including concurrency [Birkedal et al. 2012; Turon et al. 2013]. Biorthogonality [Pitts and Stark 1998] is a method of constructing logical relations from a notion of observation that simplifies the proof of completeness.

Step-indexed logical relations have been used to reason about higher-order probabilistic programs with various features. They were first proposed for discrete probabilistic choice [Bizjak and Birkedal 2015], and then studied for continuous choice [Culpepper and Cobb 2017; Wand et al. 2018], following the operational model from Borgström et al. [2016]. In a recent paper, Zhang and Amin [2022] show how step-indexed logical relations can also be used to reason about higher-order probabilistic programs with nested queries. Other techniques to reason about the relation between probabilistic programs include contextual distance [Crubillé and Dal Lago 2017; Crubillé and Dal Lago 2015], which is a metric generalization of contextual equivalence and measures the probability that a context can distinguish two terms.

Apt and Plotkin [1986] observed that countable nondeterminism introduces non-continuous behavior, and that iterating up to ω is not enough to model it. In his thesis, Lassen [1998] studies contextual equivalence of programs with countable nondeterministic choice using operational semantics. Later Birkedal et al. [2013] showed that step-indexed logical relations can be used to model a language with countable nondeterminism and recursive types by using step-indexing over the countable ordinals for must-termination.

The combination of probabilistic and nondeterministic choice has been studied in a variety of settings. From a denotational point of view, it is known that probabilistic and nondeterministic choice can be modeled by the distribution monad $\mathcal{D}$ and the powerset monad $\mathcal{P}$, respectively, but, as Varacca and Winskel [2006] point out, there does not exist a distributive law between them. They propose two solutions: modeling probabilities by a monad $\mathcal{I}$ of indexed valuations, which distributes over the powerset monad; or using the distributive combination of the algebraic theories of probabilistic and nondeterministic choice to present a new monad C of convex sets of distributions. Recently the latter monad has been extended to the category of metric spaces to support metric reasoning [Mio and Vignudelli 2020] and nontermination [Mio et al. 2021]. Bonchi et al. [2019] study the coalgebras of C to reason about trace equivalence for transition systems combining probabilities and nondeterminism. Another approach [Goy and Petrişan 2020] shows that C can be recovered by lifting $\mathcal{P}$ to the category of $\mathcal{D}$-algebras.

Other techniques to reason about the combination of probabilistic and nondeterministic choice include predicate transformers [Kaminski et al. 2016; Morgan and McIver 1999] in the setting of first-order imperative languages. This combination also appears in the literature of transition systems to study different notions of probabilistic automata [Kwiatkowska et al. 2011; Segala 2006]. Nondeterminism is often resolved by using a scheduler (also called resolution or policy) that selects which nondeterministic choices to make. Other approaches use coalgebraic techniques [Bonchi et al. 2017] to define behavioral equivalence. In recent work, Bonchi et al. [2019] unify these views. Analogously to our theorems 5.5 and 5.8 they prove that the maximal (resp. minimal) probability of termination among all traces coincides with the maximal (resp. maximal) probability of termination under all schedulers.

8 CONCLUSION AND FUTURE WORK

We have presented step-indexed logical relations for reasoning about contextual equivalence of programs written in a typed higher-order language that combines impredicative polymorphism, recursive types, and probabilistic and countable nondeterministic choice, and shown how to apply them to reason about challenging examples, including a distributive law showing that probabilistic choice distributes over nondeterministic choice. To the best of our knowledge, our work is the first to study a language with such a combination of features, either using operational or denotational methods.

We think our operational approach offers an interesting alternative to denotational models, because (1) it is arguably quite simple; (2) it is adequate for the natural operational semantics, which includes the distributive law, which is non-trivial to model denotationally; and (3) scales well to languages with recursive types and parametric polymorphism (a combination which is challenging to model denotationally, where one typically has to resort to some kind of model based on partial equivalence relations over a universal domain).

There are multiple directions for future work. First, we could study other notions of contextual equivalence for programs combining probabilities and nondeterminism. May- and must-contextual equivalence are convenient to work with, but one could argue they are not fine-grained enough. Instead, we could consider two expressions e, e' to be equivalent if, under any context, they have the same termination probability for any scheduler in some class $\mathcal{S}$ of schedulers, i.e. $\forall \theta \in \mathcal{S}. \mathfrak{P}^{\downarrow}_{\theta}(C[e]) = \mathfrak{P}^{\downarrow}_{\theta}(C[e'])$. The class of schedulers we have considered here are, however, too powerful for this purpose, and would induce the trivial equivalence relation, so we would have to identify a restricted class of schedulers that cannot observe the whole program execution, similarly to what is suggested by Chadha et al. [2010]. Second, we could extend the language to make it more expressive. For instance, adding continuous probabilities and conditioning would enable to capture higher-order probabilistic languages such as Anglican [Wood et al. 2014] or Hakaru [Narayanan et al. 2016]. The line of work presented in [Culpepper and Cobb 2017; Wand et al. 2018] shows how to prove contextual equivalence of programs with continuous probabilities, but extending their framework with countable nondeterminism is challenging, particularly in the case of must-termination, due to the interaction between integration and ω_1-indexed sequences. Finally, we also wish to extend our model to reason about probabilistic concurrent programs. Concurrency already has an inherent nondeterministic nature coming from the scheduler that selects which thread to run, and we believe that the techniques introduced in this work constitute a first step towards reasoning about contextual equivalence of concurrent probabilistic programs.

ACKNOWLEDGMENTS

This research was supported in part by a Villum Investigator grant (no. 25804), Center for Basic Research in Program Verification (CPV), from the VILLUM Foundation.

REFERENCES

Amal Ahmed. 2006. Step-Indexed Syntactic Logical Relations for Recursive and Quantified Types. In *Proceedings of the 15th European Conference on Programming Languages and Systems* (Vienna, Austria) *(ESOP'06)*. Springer-Verlag, Berlin, Heidelberg, 69–83. 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 Trans. Program. Lang. Syst.* 23, 5 (Sept. 2001), 657–683. https://doi.org/10.1145/504709.504712

K. R. Apt and G. D. Plotkin. 1986. Countable nondeterminism and random assignment. *J. ACM* 33, 4 (Aug. 1986), 724–767. https://doi.org/10.1145/6490.6494

Robert J. Aumann. 1961. Borel structures for function spaces. *Illinois J. Math.* 5, 4 (Dec. 1961), 614–630. https://doi.org/10.1215/ijm/1255631584

Lars Birkedal, Aleš Bizjak, and Jan Schwinghammer. 2013. Step-Indexed Relational Reasoning for Countable Nondeterminism. *Log. Meth. Comput. Sci.* 9, 4 (Oct. 2013), 4. https://doi.org/10.2168/lmcs-9(4:4)2013

Lars Birkedal, Filip Sieczkowski, and Jacob Thamsborg. 2012. A Concurrent Logical Relation. In *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France (LIPIcs, Vol. 16)*, Patrick Cégielski and Arnaud Durand (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 107–121. https://doi.org/10.4230/LIPIcs.CSL.2012.107

Aleš Bizjak and Lars Birkedal. 2015. Step-Indexed Logical Relations for Probability. In *Foundations of Software Science and Computation Structures*, Andrew Pitts (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 279–294.

Filippo Bonchi, Alexandra Silva, and Ana Sokolova. 2017. The Power of Convex Algebras. In *28th International Conference on Concurrency Theory (CONCUR 2017) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 85)*, Roland Meyer and Uwe Nestmann (Eds.). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 23:1–23:18. https://doi.org/10.4230/LIPIcs.CONCUR.2017.23

Filippo Bonchi, Ana Sokolova, and Valeria Vignudelli. 2019. The Theory of Traces for Systems with Nondeterminism and Probability. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)* (Vancouver, Canada) *(LICS '19)*. IEEE, Article 19, 14 pages. https://doi.org/10.1109/lics.2019.8785673

Johannes Borgström, Ugo Dal Lago, Andrew D. Gordon, and Marcin Szymczak. 2016. A lambda-calculus foundation for universal probabilistic programming. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*. ACM, Nara, Japan, 33–46. https://doi.org/10.1145/2951913.2951942

Rohit Chadha, A. Prasad Sistla, and Mahesh Viswanathan. 2010. Model Checking Concurrent Programs with Nondeterminism and Randomization. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 8)*, Kamal Lodaya and Meena Mahajan (Eds.). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 364–375. https://doi.org/10.4230/LIPIcs.FSTTCS.2010.364

Patrick Cousot and Radhia Cousot. 1979. Constructive versions of Tarski's fixed point theorems. *Pac. J. Math.* 82, 1 (May 1979), 43–57. https://doi.org/10.2140/pjm.1979.82.43

Raphaëlle Crubillé and Ugo Dal Lago. 2017. Metric Reasoning About λ-Terms: The General Case. In *Programming Languages and Systems: 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22–29, 2017, Proceedings* (Uppsala, Sweden). Springer-Verlag, Berlin, Heidelberg, 341–367. https://doi.org/10.1007/978-3-662-54434-1_13

Raphaëlle Crubillé and Ugo Dal Lago. 2015. Metric reasoning about λ-terms: The affine case. In *2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science*. 633–644. https://doi.org/10.1109/LICS.2015.64

Ryan Culpepper and Andrew Cobb. 2017. Contextual Equivalence for Probabilistic Programs with Continuous Random Variables and Scoring. In *Programming Languages and Systems*, Hongseok Yang (Ed.). Vol. 10201. Springer Berlin Heidelberg, Berlin, Heidelberg, 368–392. https://doi.org/10.1007/978-3-662-54434-1_14

Thomas Ehrhard, Michele Pagani, and Christine Tasson. 2018. Measurable cones and stable, measurable functions: A model for probabilistic higher-order programming. *Proceedings of the ACM on Programming Languages* 2, POPL, Article 59 (Jan. 2018), 28 pages. https://doi.org/10.1145/3158147

Thomas Ehrhard and Christine Tasson. 2019. Probabilistic call by push value. *Log. Meth. Comput. Sci.* Volume 15, Issue 1 (Jan. 2019). https://doi.org/10.23638/LMCS-15(1:3)2019

Thomas Ehrhard, Christine Tasson, and Michele Pagani. 2014. Probabilistic coherence spaces are fully abstract for probabilistic PCF. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Diego, California, USA) *(POPL '14)*. ACM, New York, NY, USA, 309–320. https://doi.org/10.1145/2535838.2535865

Alexandre Goy and Daniela Petrişan. 2020. Combining probabilistic and non-deterministic choice via weak distributive laws. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*. ACM, Saarbrücken Germany, 454–464. https://doi.org/10.1145/3373718.3394795

Chris Heunen, Ohad Kammar, Sam Staton, and Hongseok Yang. 2017. A convenient category for higher-order probability theory. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, 1–12. https://doi.org/10.

1109/lics.2017.8005137

Bart Jacobs. 2021. From Multisets over Distributions to Distributions over Multisets. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, New York, NY, USA, Article 39, 13 pages. https://doi.org/10.1109/lics52264.2021.9470678

C. Jones and G.D. Plotkin. 1989. A probabilistic powerdomain of evaluations. In *[1989] Proceedings. Fourth Annual Symposium on Logic in Computer Science*. IEEE Comput. Soc. Press, 186–195. https://doi.org/10.1109/lics.1989.39173

Achim Jung and Regina Tix. 1998. The Troublesome Probabilistic Powerdomain. *Electron. Notes Theor. Comput. Sci.* 13 (1998), 70–91. https://doi.org/10.1016/s1571-0661(05)80216-6

Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. 2016. Weakest Precondition Reasoning for Expected Run–Times of Probabilistic Programs. In *Programming Languages and Systems*, Peter Thiemann (Ed.). Vol. 9632. Springer Berlin Heidelberg, Berlin, Heidelberg, 364–389. https://doi.org/10.1007/978-3-662-49498-1_15

Dexter Kozen. 1981. Semantics of probabilistic programs. *J. Comput. Syst. Sci.* 22, 3 (June 1981), 328–350. https://doi.org/10.1016/0022-0000(81)90036-2

Kenneth Kunen. 2011. *Set Theory*. College Publications.

Marta Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: Verification of probabilistic real-time systems. In *International conference on computer aided verification*. Springer, 585–591.

Søren Bøgh Lassen. 1998. *Relational Reasoning about Functions and Nondeterminism*. Ph. D. Dissertation. BRICS. https://www.brics.dk/DS/98/2/BRICS-DS-98-2.pdf

Matteo Mio, Ralph Sarkis, and Valeria Vignudelli. 2021. Combining Nondeterminism, Probability, and Termination: Equational and Metric Reasoning. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, New York, NY, USA, Article 18, 14 pages. https://doi.org/10.1109/lics52264.2021.9470717

Matteo Mio and Valeria Vignudelli. 2020. Monads and Quantitative Equational Theories for Nondeterminism and Probability. In *31st International Conference on Concurrency Theory (CONCUR 2020) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 171)*, Igor Konnov and Laura Kovács (Eds.). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 28:1–28:18. https://doi.org/10.4230/LIPIcs.CONCUR.2020.28

Michael Mislove. 2000. Nondeterminism and Probabilistic Choice: Obeying the Laws. In *CONCUR 2000 — Concurrency Theory*, Catuscia Palamidessi (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 350–365.

Carroll Morgan and Annabelle McIver. 1999. pGCL: Formal reasoning for random algorithms. (1999).

Praveen Narayanan, Jacques Carette, Wren Romano, Chung-chieh Shan, and Robert Zinkov. 2016. Probabilistic inference by program transformation in Hakaru (system description). In *International Symposium on Functional and Logic Programming - 13th International Symposium, FLOPS 2016, Kochi, Japan, March 4-6, 2016, Proceedings*. Springer, 62–79. https://doi.org/10.1007/978-3-319-29604-3_5

Andrew Pitts. 2004. Typed Operational Reasoning. In *Advanced Topics in Types and Programming Languages*, Benjamin C. Pierce (Ed.). The MIT Press, Cambridge, Mass, Chapter 7. https://doi.org/10.7551/mitpress/1104.003.0011

Andrew Pitts and Ian Stark. 1998. Operational Reasoning for Functions with Local State. In *Higher Order Operational Techniques in Semantics*, Andrew Gordon and Andrew Pitts (Eds.). Publications of the Newton Institute, Cambridge University Press, 227–273. http://www.inf.ed.ac.uk/~stark/operfl.html

G. D. Plotkin. 1976. A Powerdomain Construction. *SIAM J. Comput.* 5, 3 (Sept. 1976), 452–487. https://doi.org/10.1137/0205035

N. Saheb-Djahromi. 1980. Cpo's of measures for nondeterminism. *Theor. Comput. Sci.* 12, 1 (Sept. 1980), 19–37. https://doi.org/10.1016/0304-3975(80)90003-1

Roberto Segala. 2006. Probability and Nondeterminism in Operational Models of Concurrency. In *CONCUR 2006 – Concurrency Theory*, Christel Baier and Holger Hermanns (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 64–78.

W. W. Tait. 1967. Intensional interpretations of functionals of finite type I. *J. Symbolic Logic* 32, 2 (Aug. 1967), 198–212. https://doi.org/10.2307/2271658

Alfred Tarski. 1955. A lattice-theoretical fixpoint theorem and its applications. *Pac. J. Math.* 5, 2 (June 1955), 285–309. https://doi.org/10.2140/pjm.1955.5.285

Joseph Tassarotti and Robert Harper. 2019. A separation logic for concurrent randomized programs. *Proceedings of the ACM on Programming Languages* 3, POPL, Article 64 (Jan. 2019), 30 pages. https://doi.org/10.1145/3290377

Aaron J. Turon, Jacob Thamsborg, Amal Ahmed, Lars Birkedal, and Derek Dreyer. 2013. Logical relations for fine-grained concurrency. *Acm Sigplan Notices* 48, 1 (Jan. 2013), 343–356. https://doi.org/10.1145/2480359.2429111

Matthijs Vákár, Ohad Kammar, and Sam Staton. 2019. A domain theory for statistical probabilistic programming. *Proceedings of the ACM on Programming Languages* 3, POPL (Jan. 2019), 1–29. https://doi.org/10.1145/3290349

Daniele Varacca and Glynn Winskel. 2006. Distributing probability over non-determinism. *Math. Struct. Comp. Sci.* 16, 01 (Feb. 2006), 87. https://doi.org/10.1017/s0960129505005074

Mitchell Wand, Ryan Culpepper, Theophilos Giannakopoulos, and Andrew Cobb. 2018. Contextual equivalence for a probabilistic language with continuous random variables and recursion. *Proceedings of the ACM on Programming Languages* 2, ICFP, Article 87 (July 2018), 30 pages. https://doi.org/10.1145/3236782

Frank Wood, Jan Willem van de Meent, and Vikash Mansinghka. 2014. A New Approach to Probabilistic Programming Inference. In *Proceedings of the 17th International conference on Artificial Intelligence and Statistics*. 1024–1032.

Wang Yi and Kim Guldstrand Larsen. 1992. Testing Probabilistic and Nondeterministic Processes. In *Proceedings of the IFIP TC6/WG6.1 Twelth International Symposium on Protocol Specification, Testing and Verification XII*. North–Holland Publishing Co., 15 pages.

Yizhou Zhang and Nada Amin. 2022. Reasoning about "Reasoning about Reasoning": Semantics and Contextual Equivalence for Probabilistic Programs with Nested Queries and Recursion. *Proc. ACM Program. Lang.* 6, POPL, Article 16 (jan 2022), 28 pages. https://doi.org/10.1145/3498677

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