Step-Indexed Logical Relations for Nondeterministic and Probabilistic Choice

Alejandro Aguirre
Lars Birkedal

Abstract
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 combination of the algebraic theories for probabilistic and nondeterministic choice and the distributive property between them.

ACM Reference Format:

1 Introduction
Probabilistic programming languages are languages that provide commands to sample from probability distributions to generate random values. These can be used, for instance, to write randomized algorithms, to generate secret keys for encryption and decryption, or to write statistical models.

Nondeterminism arises when studying behaviors of a program 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 a fair scheduler, in which every thread will be eventually allowed to run after a finite, but arbitrarily large number of cycles.

The combination of probabilistic and nondeterministic choice has been the object of study of many articles over more than three decades [3, 9, 17, 24, 26, 36]. 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 [36]. From an algebraic point of view, the distributive law would correspond to the equation

\[ e_1 \oplus (e_2 \mathbin{\mathbf{or}} e_3) = (e_1 \oplus e_2) \mathbin{\mathbf{or}} (e_1 \oplus e_3), \]

where \( \oplus \) and \( \mathbf{or} \) represent binary probabilistic and nondeterministic choice, respectively. Varacca and Winskel [36] 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 [1, 6, 29, 34]. 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 [7, 37, 39] the behavior used to define equivalence is the probability of termination. In the nondeterministic setting [5], 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.\footnote{It is in fact sufficient to step-index up to the first nonrecursive ordinal.}

\textbf{Contributions.}

- We define probabilistic versions of may- and must-termination for programs written in a typed higher-order language that combines parametric 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 countables, 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 [5, 7].
- We define an alternative notion of probabilistic termination parametrized by a scheduler which 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 distributivity law between them.

\textbf{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 alternative 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).

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, which we extend with two choice operators. The first choice operator \texttt{rand} \(n\) represents a uniform probabilistic choice over the set \([1, \ldots, n]\). The second choice operator \texttt{?} represents a nondeterministic choice over all the natural numbers. 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.

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

\[
\begin{align*}
\tau, \sigma &::= 1 | \alpha | \tau \rightarrow \sigma | \tau \times \sigma | \tau + \sigma | \mu \alpha.\tau \\
v &::= () | x | n \in \mathbb{N} | \lambda x.e | \langle \langle v, v' \rangle \rangle \\
e &::= () | x | n \in \mathbb{N} | \lambda x.e | e' | \langle \langle e, e' \rangle \rangle \\
&\quad\mid \pi_1e | \text{inl}(e) | \text{inr}(e) | \text{case}(e, x_1.e_1, x_2.e_2) \\
&\quad\mid \text{rand } e | \text{fold } e | \text{unfold } e | \alpha.e | e_-
\end{align*}
\]

\[E ::= \mathbb{A} | E | \text{case}(E, x_1.e_1, x_2.e_2) | \text{rand } E | \text{fold } E | \text{inl}(E) | \text{inr}(E) | \text{unfold } E \]

A type formation context \(\Delta\) is a finite set of type variables. A typing context \(\Gamma\) is a finite partial map from term variables to types. A typing formation judgment is a tuple \(\Delta \vdash \tau\), where \(\Delta\) is a type formation context and \(\tau\) is a type (by extension, we write \(\Delta \vdash \Gamma\) when every type in the codomain of \(\Gamma\) is well-formed under \(\Delta\)). A typing judgment is a tuple \(\Delta \vdash \Gamma : \tau\), where \(\Delta\) is a type formation context, \(\Gamma\) is a typing context, \(\tau\) is an expression and \(\alpha\) 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 \(\text{Type}(\Delta)\) denote the set of types \(\tau\) such that \(\Delta \vdash \tau\), and \(\text{Type}\) denotes the set of closed types. For a type \(\tau\), \(\text{Val}(\tau)\) is the set of values \(v\) such that \(\emptyset \vdash \emptyset \vdash v : \tau\); \(\text{Expr}(\tau)\) is the set of expressions \(e\) such that \(\emptyset \vdash \emptyset \vdash e : \tau\); and \(\text{Eval}(\tau)\) is the set of evaluation contexts \(E \in \text{Eval}\) such that for any \(e \in \text{Expr}(\tau)\), there exists \(\sigma \in \text{Type}\) such that \(\emptyset \vdash \emptyset \vdash E[e] : \sigma\) (in which case we also write \(\vdash E : \tau \Rightarrow \sigma\)). We use \(\text{Val}\) and \(\text{Expr}\) to denote the sets of all closed values and all closed expressions respectively, and we embed \(\text{Val}\) into \(\text{Expr}\) as usual.

Given \(\delta : \Delta \rightarrow \text{Type}\) and \(\tau \in \text{Type}(\Delta)\) we write \(\tau \delta\) for the usual capture-avoiding substitution of every free type variable \(\alpha\) in \(\tau\) by \(\delta(\alpha)\). Given \(\delta : \Delta \rightarrow \text{Type}\), we will also use

\[
\begin{align*}
(x : \tau) \in \Gamma &\quad \Delta \vdash \Gamma \\
\Delta \vdash \Gamma &\quad \Delta \vdash \Gamma, x : \sigma + e : \tau \\
\Delta \vdash \Gamma, e : \sigma \rightarrow \tau &\quad \Delta \vdash \Gamma, \lambda x.e : \sigma \rightarrow \tau \\
\Delta \vdash \Gamma, e_1 : \sigma 
\end{align*}
\]
We define a call-by-value single step reduction relation and denote it by $e \rightarrow \in \{ \text{the probability of termination} \}$.

In the probabilistic setting, making the program terminate or must-termination (every choice makes the program diverge). In languages with nondeterministic choice, there are usually two notions of observation: 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.

### 2.2 Operational semantics

We define a call-by-value single step reduction relation $\rightarrow \subseteq \text{Expr} \times \text{Expr} \times \{0, 1\} \times \{D, N, P\}$. 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:

- $(\lambda x.e) v \rightarrow_D e[v/x]$
- $\pi_i(v_1,v_2) \rightarrow_D v_i \quad i = \{1, 2\}$
- $\text{case}(\text{inr}(v), x_1,x_2,x_2) \rightarrow_D e_1[v/x_1]$
- $\text{case}(\text{inr}(v), x_1,x_2,x_2) \rightarrow_D e_2[v/x_2]$

We define $e \rightarrow e'$ to denote that there exist $r \in \{0, 1\}$ and $X \in \{D, N, P\}$ such that $e \rightarrow^r X e'$. If there exists a chain of deterministic reductions $e \rightarrow_D e_1 \rightarrow_D \ldots \rightarrow_D e'$, we will denote it by $e \rightarrow_D^* e'$. In general, we will not define a multistep reduction relation. As we will see later, we can define the probability of termination as a fixed point of the single steps, and then reason directly about it.

### 3 Notions of observations

Broadly speaking, two programs are defined to be contextually equivalent if, under any context, they exhibit the same observable behavior. In this section we make precise what observations we can make about a program’s behavior. This definition will also be crucial in the construction of the logical relations by biorthogonality.

In an deterministic setting \([28]\) 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 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 \([5, 23]\): 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 \([7]\).

#### 3.1 Probabilistic may- and must-termination

As a generalization, we will consider two quantitative observations, which will induce two different 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 points of the following two operators of type $(\text{Expr} \rightarrow \{0, 1\}) \rightarrow \text{Expr} \rightarrow \{0, 1\}$:

$\Phi(f)(e) = \begin{cases} 1 & \text{if } e \in \text{Val} \\ \sup_{n \in \mathbb{N}} f(E[n]) & e = E[\text{rand } k] \\ f(e') & e \rightarrow_D e' \\ 0 & \text{otherwise} \end{cases}$

$\Psi(f)(e) = \begin{cases} 1 & \text{if } e \in \text{Val} \\ \inf_{n \in \mathbb{N}} f(E[n]) & e = E[\text{rand } k] \\ f(e') & e \rightarrow_D e' \\ 0 & \text{otherwise} \end{cases}$

Note that existence of the sup is guaranteed because $f$ is bounded. Analogously, we define the probability of must-termination as the least fixed point of the operator $\Psi: (\text{Expr} \rightarrow \{0, 1\}) \rightarrow \text{Expr} \rightarrow \{0, 1\}$ defined below:

$\Psi(f)(e) = \begin{cases} 1 & \text{if } e \in \text{Val} \\ \inf_{n \in \mathbb{N}} f(E[n]) & e = E[\text{rand } k] \\ f(e') & e \rightarrow_D e' \\ 0 & \text{otherwise} \end{cases}$

The existence of the least fixed-points is guaranteed by Tarski’s fixed point theorem, and by the fact that the operators are monotonic:

**Lemma 3.1.** The operators $\Phi, \Psi$ are monotonic, that is, for $f, g: \text{Expr} \rightarrow \{0, 1\}$ such that $f \leq g$ then $\Phi(f) \leq \Phi(g)$ and $\Psi(f) \leq \Psi(g)$.

Therefore, we can define

**Definition 3.2.** Let $e \in \text{Expr}$. The probability of may-termination of $e$, denoted $\mathbb{P}_\downarrow(e)$ is defined as $\mathbb{P}_\downarrow(e) = (\text{lfp } \Phi)(e)$. The probability of must-termination of $e$, denoted $\mathbb{P}_\uparrow(e)$ is defined as $\mathbb{P}_\uparrow(e) = (\text{lfp } \Psi)(e)$.

The following proposition presents some properties that are useful to reason about the probability of (may-,must-)termination:

**Proposition 3.3.** Let $e \in \text{Expr}$. Then:

1. If $e \in \text{Val}$, then $\mathbb{P}_\downarrow(e) = \mathbb{P}_\uparrow(e) = 1$.
2. If $e = E[?]$ then $\mathbb{P}_\downarrow(e) = \sup_{n \in \mathbb{N}} \mathbb{P}_\downarrow(E[n])$ and $\mathbb{P}_\uparrow(e) = \inf_{n \in \mathbb{N}} \mathbb{P}_\uparrow(E[n])$.
3. If $e = E[\text{rand } k]$ then $\mathbb{P}_\downarrow(e) = \sum_{1 \leq n \leq k} \frac{1}{k} \cdot \mathbb{P}_\downarrow(E[n])$ and $\mathbb{P}_\uparrow(e) = \sum_{1 \leq n \leq k} \frac{1}{k} \cdot \mathbb{P}_\uparrow(E[n])$.
4. If $e \rightarrow_D e'$, then $\mathbb{P}_\downarrow(e) = \mathbb{P}_\downarrow(e')$ and $\mathbb{P}_\uparrow(e) = \mathbb{P}_\uparrow(e')$. 

It is easy to show that $\Phi$ is $\omega$-continuous, so its fixed point is precisely $\sup_{\alpha<\omega_1} \Phi^\alpha(\bot)$. However, this is not the case for $\Psi$, for which we need to iterate up to a larger ordinal. Note that iteration up to an ordinal $\alpha$ can be defined naturally: for any $\beta$, we let $\Psi^{\beta+1} = \Psi \circ \Psi^\beta$; and for a limit ordinal $\xi$, we let $\Psi^\xi = \sup_{\beta<\xi} \Psi^\beta$.

Here we want to show that $\Psi$ is $\omega_1$-continuous, where $\omega_1$ is the smallest uncountable ordinal. Concretely, we will prove that there exists a countable ordinal $\beta$ such that for any $e \in \text{Expr}$, $\Psi^\beta(e) = \Psi^\beta(\bot)$. In other words, we want to show that the fixpoint iteration $\Psi(\bot)(e), \Psi^2(\bot)(e), \ldots, \Psi^{\alpha}(\bot)(e), \ldots$ eventually becomes constant. This follows by a monotonicity argument. First note that, since $\Psi$ is monotonic, its fixpoint iterations are a non-decreasing sequence:

**Lemma 3.4.** Let $F_e(\alpha) = \Psi^\alpha(\bot)(e)$. Then, $F_e$ is monotonically non-decreasing, i.e., for $\alpha < \beta$, $F_e(\alpha) \leq F_e(\beta)$.

The following lemma is the key to our argument. It states that any non-decreasing $\omega_1$-indexed sequence in a closed interval must eventually become constant:

**Lemma 3.5.** Let $f : \omega_1 \to [0, 1]$ be a monotonic function. Then, there exist $r \in [0, 1]$ and $\beta < \omega_1$ such that for every $\alpha$, 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 $f^* - 2^{-i} < f(\beta_i)$. Then we can take $\beta = \sup_{i \in \mathbb{N}} \beta_i$, which is a countable ordinal. Using monotonicity, we get that for every $\alpha > \beta$,

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

Finally, we will prove:

**Proposition 3.6.** There exists a countable ordinal $\beta$ such that $\Psi^\beta(\bot) = \text{lfp} \Psi$, and therefore, $\text{lfp} \Psi = \sup_{\beta<\omega_1} \Psi^\beta(\bot)$.

**Proof.** For every $e$, $F_e(\alpha) = \Psi^\alpha(\bot)(e)$ determines a monotonic map from $\omega_1$ to $[0, 1]$, so it must be constant from some countable $\beta$. Since the set of expressions is countable, $\beta = \sup_{e \in \text{Expr}} \beta_e$ is a countable ordinal and for every $e$, $\Psi^{\beta+1}(\bot)(e) = \Psi^\beta(\bot)(e)$.

### 3.2 Comparison to may and must termination

It is easy to see that if we remove nondeterministic choice from our language, the two observations collapse to the same one, which also coincides with the notion of observation used in previous work for contextual equivalence for probabilistic programs [7]. 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 presented in previous work focusing on only nondeterminism [5].

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

**Definition 3.7** (May convergence). Let $\hat{\Phi} : \mathcal{P}(\text{Expr}_{\text{ND}}) \to \mathcal{P}(\text{Expr}_{\text{ND}})$ defined by

$$\hat{\Phi}(X) = \{ e \in \text{Expr}_{\text{ND}} | e \in \text{Val} \lor \exists e' \in \text{Expr}_{\text{ND}}. e \rightarrow e' \land e' \in X \}.$$

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

**Definition 3.8** (Must convergence). Let $\hat{\Psi} : \mathcal{P}(\text{Expr}_{\text{ND}}) \to \mathcal{P}(\text{Expr}_{\text{ND}})$ defined by

$$\hat{\Psi}(X) = \{ e \in \text{Expr}_{\text{ND}} | e \in \text{Val} \lor (\exists e'. e \rightarrow e' \land \forall e' \in \text{Expr}_{\text{ND}}. e \rightarrow e' \Rightarrow e' \in X) \}.$$

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

Technically, the definition of must-convergence is slightly different from [5] since, unlike the previous work, we do not consider that stuck terms must-converge. This choice will also allow us to present both notions of termination as particular cases of a scheduler-based notion of termination later, see Section 5.

Note that by considering the order isomorphism between $\mathcal{P}(\text{Expr}_{\text{ND}})$, ordered by subset inclusion, and $\text{Expr}_{\text{ND}} \to \{0, 1\}$, ordered pointwise, $\hat{\Phi}$ and $\hat{\Psi}$ can be seen as operators of type $(\text{Expr}_{\text{ND}} \to \{0, 1\}) \to (\text{Expr}_{\text{ND}} \to \{0, 1\})$. 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 3.9.** Let $e \in \text{Expr}_{\text{ND}}$ and $f \in \text{Expr} \to \{0, 1\}$. Then $\Phi(f)(e) = \hat{\Phi}(f)(e)$. In particular, $\hat{\Psi}^\downarrow(e) = 1$ if and only if $e \Downarrow$.

**Proof.** We do a case distinction on $e$. If $e \in \text{Val}$, then $\Phi(f)(e) = 1 = \hat{\Phi}(f)(e)$. If $e$ is stuck, then $\Phi(f)(e) = 0 = \hat{\Phi}(f)(e)$. If $e \rightarrow D e'$, then there is no other reduction from $e$, and $\hat{\Phi}(f)(e) = 1$ if $f(e) = 1$, so $\Phi(f)(e) = \hat{\Phi}(f)(e)$.

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

**Proposition 3.10.** Let $e \in \text{Expr}_{\text{ND}}$ and $f \in \text{Expr} \to \{0, 1\}$. Then $\Psi(f)(e) = \hat{\Psi}(f)(e)$. In particular, $\hat{\Psi}^\downarrow(e) = 1$ if and only if $e \Downarrow$.

**Proof.** We do a case distinction on $e$. If $e \in \text{Val}$, 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 Ψ(e) = \inf_{n \in \mathbb{N}} f(E[n]), which can be 0 or 1. If e is 0, then there exists a particular m ∈ \mathbb{N} such that e → E[m] and f(E[m]) = 0, so Ψ(e) = 0. Otherwise it is 1 so for every e′ such that e → e′, f(e′) = 1, and \Psi(e) = 1.

Then, \Psi(e) = 1 if and only if e ∈ \mathbb{B} because both are least fixed points of isomorphic operators.

\[\square\]

4 Type-indexed relations

Contextual equivalence is often defined by quantifying over arbitrary contexts but it may also be defined as the largest equivalence relation closed under term constructors of the programming language [28]. 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

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

Definition 4.1 (May- and must-adequacy). Let \(\mathcal{R}\) be a type-indexed relation. We say that \(\mathcal{R}\) is may-adequate if, for any \(\emptyset \vdash e : \tau\), then \(\Psi(e) \leq \Psi(e')\). Analagously, we say that \(\mathcal{R}\) is must-adequate if, for any \(\emptyset \vdash e : \tau\), then \(\Psi(e) \leq \Psi(e')\).

The following definitions are standard [28]:

Definition 4.2 (Precongruence). Let \(\mathcal{R}\) be a type-indexed relation. We say that it is

- reflexive if \(\Delta \vdash \Gamma \vdash e : \tau\) implies \(\Delta \vdash \Gamma \vdash e \mathcal{R} e : \tau\),
- symmetric if \(\Delta \vdash \Gamma \vdash e_1 \mathcal{R} e_2 : \tau\) implies \(\Delta \vdash \Gamma \vdash e_2 \mathcal{R} e_1 : \tau\),
- transitive if \(\Delta \vdash \Gamma \vdash e_1 \mathcal{R} e_2 : \tau\) and \(\Delta \vdash \Gamma \vdash e_2 \mathcal{R} e_3 : \tau\) imply \(\Delta \vdash \Gamma \vdash e_1 \mathcal{R} e_3 : \tau\),
- compatible if it is closed under the typing rules. For instance in the case of abstraction and application the following must hold:

\[
\begin{align*}
\Delta \vdash \Gamma, x : \sigma \vdash e_1 : \mathcal{R} e_2 : \tau & \quad \Delta \vdash \Gamma \vdash e_1 \mathcal{R} e_2 : \sigma \rightarrow \tau \\
\Delta \vdash \Gamma, \lambda x. e_1 : \mathcal{R} \lambda x. e_2 : \sigma \rightarrow \tau & \quad \Delta \vdash \Gamma \vdash e_1 \mathcal{R} e_2 : \sigma \rightarrow \tau
\end{align*}
\]

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

Definition 4.3 (Contextual approximation and equivalence). Contextual may-approximation (denoted \(\leq_{ctx}^\Delta\)) is the largest may-adequate precongruence. Contextual must-approximation (denoted \(\leq_{ctx}^\Delta\)) is the largest must-adequate precongruence. Contextual may-equivalence (\(\equiv_{ctx}^\Delta\)) and must-equivalence (\(\equiv_{ctx}^\Delta\)) are defined respectively as the largest symmetric subrelations of contextual may- and must-approximation.

Definition 4.4 (CIU approximation and equivalence). The CIU may-approximation (resp. CIU must-approximation) relation \(\leq_{CIU}^\Delta\) (\(\leq_{CIU}^\Delta\)) is defined as follows: for every \(e, e'\) such that \(\Delta | \Gamma \vdash e : \tau\) and \(\Delta | \Gamma \vdash e' : \tau\) we write \(\Delta | \Gamma \vdash e \leq_{CIU}^\Delta e' : \tau\) if for every \(\delta : \Delta \rightarrow \text{Type}, \gamma \in \text{Subst}(\Gamma \delta), E \in \text{Ectx}(\tau \delta),\) we have that \(\Psi(E[e]) \leq \Psi(E[e'])\) (\(\Psi(E[e]) \leq \Psi(E[e'])\)). CIU may-equivalence (\(\equiv_{CIU}^\Delta\)) and must-equivalence (\(\equiv_{CIU}^\Delta\)) are defined respectively as the largest symmetric subrelations of CIU may- and must-approximation.

By definition, CIU approximation is reflexive, transitive and may-adequate. However, showing directly that it is compatible is non-trivial. Instead, we define a logical relation.

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

Let \(\tau, \sigma \in \text{Type}(\emptyset)\). A value relation between \(\tau\) and \(\sigma\) is an \(\omega_1\)-indexed relation over \(\text{Val}(\tau) \times \text{Val}(\sigma)\). An evaluation context relation between \(\tau\) and \(\sigma\) is an \(\omega_1\)-indexed relation over \(\text{Ectx}(\tau) \times \text{Ectx}(\sigma)\). An expression relation between \(\tau\) and \(\sigma\) is an \(\omega_1\)-indexed relation over \(\text{Exp}(\tau) \times \text{Exp}(\sigma)\). We let \(\text{RVal}(\tau, \sigma), \text{REctx}(\tau, \sigma), \text{REctx}(\tau, \sigma)\) denote the sets of value, evaluation context and expression relations between \(\tau, \sigma\), respectively.

Given a type context \(\Delta\), we let \(\text{RVal}(\Delta)\) denote the set of relations over \(\Delta\) defined by:

\[
\text{RVal}(\Delta) = \left\{(\delta_1, \delta_2, r) \mid \Delta \vdash \Gamma \vdash e_1 : \text{Val} \rightarrow \text{Val}, \forall \alpha \in \Delta, r(\alpha) \in \text{RVal}(\delta_1(\alpha), \delta_2(\alpha))\right\}
\]

4.2 Relation for may-termination

Given a type context \(\Delta\), a type \(\tau \in \text{Type}(\Delta)\), and a set of relations \(\varphi = (\delta_1, \delta_2, r) \in \text{RVal}(\Delta),\) a type \(\tau\) can be given an interpretation \([\Delta \vdash \Gamma] (\varphi) (n)\) can be seen as a pair \((\gamma, \gamma')\) of substitutions for the variables in \(\Gamma\) such that for every \(x : \tau \in \Gamma, (\gamma(x), \gamma'(x)) \in [\Delta \vdash \Gamma] (\varphi) (n)\).

We then extend the relation to expressions by using biorthogonality. Plainly speaking, we lift the relational interpretation.
Lemma 4.6. The relation \( \subseteq \) defined as follows:
\[
\subseteq \triangleq \{ (\cdot, \cdot) \mid \forall \varphi, \psi \in \mathbf{R} \}.
\]
From this, we can define a type-indexed relation, that we call logical approximation:

Definition 4.5. We say that \( \Delta \vdash \Gamma \vdash e \triangleq \) if for all \( \varphi \in \mathbf{R} \), \( \forall n < \omega \), \( \forall \gamma, \gamma' \in \Delta \vdash \Gamma \psi(e), e \rangle \triangleq \psi(e') \).

We have the following adequacy result:

Proposition 4.6. The relation \( \triangleq \) is may-adequate.

Proof. Assume \( e \triangleq e' \). Note that, for all \( n < \omega \), \( \psi_{\Delta}^{(e)} \subseteq \psi_{\Delta}^{(e')} \). Then, we have that, for all \( n < \omega \), \( \psi_{\Delta}^{(e)} \subseteq \psi_{\Delta}^{(e')} \), so in particular, \( \sup_{n<\omega} \psi_{\Delta}^{(e)} \leq \psi_{\Delta}^{(e')} \).

Before proving context extension, we state some useful lemmas:

Lemma 4.7. Let \( \Delta \vdash \Gamma \vdash e \), \( \Delta, \alpha \vdash \Gamma \psi(e) \). Then \( \Delta \vdash \alpha \psi(e) \).

Lemma 4.8. Let \( \tau, \tau' \in \textbf{Type}, \pi \in \mathbf{RVal}(\tau, \tau') \). Then \( \pi \subseteq \pi'' \).

Lemma 4.9. Let \( \tau, \tau' \in \textbf{Type}, \pi \in \mathbf{RVal}(\tau, \tau') \). Then \( \pi'' \subseteq \pi''' \).

We now turn to proving the context extension lemmas. Here we only show a few representative cases.

Lemma 4.10. Let \( n < \omega, \varphi \in \mathbf{RVal}(\Delta), (E, E') \in \Delta \vdash \varphi \psi(e), e \rangle \subseteq \psi(e') \).

Lemma 4.11. Let \( n < \omega, \varphi \in \mathbf{RVal}(\Delta), (E, E') \in \Delta \vdash \varphi \psi(e), e \rangle \subseteq \psi(e') \).

Lemma 4.12. Let \( n < \omega, \varphi \in \mathbf{RVal}(\Delta), (E, E') \in \Delta \vdash \varphi \psi(e), e \rangle \subseteq \psi(e') \).

Proposition 4.14 (Fundamental property). The relation \( \triangleq \) is compatible. In particular, it is reflexive: for any type context \( \Delta \), typing context \( \Gamma \), expression \( e \) and type \( \tau \), if \( \Delta \vdash \Gamma \vdash e \), then \( \Delta \vdash \Gamma \vdash e \).

Proof. We show here a couple of illustrative cases:

- If \( e \equiv \). Let \( n < \omega, (E, E') \in \Delta \vdash \varphi \psi(e), e \rangle \).

Note that \( \psi_{\Delta}^{(E)} \subseteq \psi_{\Delta}^{(E')} \) for all \( n \in \mathbb{N}, (\Delta \vdash \Gamma \psi(e), e \rangle \).

Assume \( \Gamma \vdash e \triangleq \psi(e) \). Then \( \Gamma \vdash \varphi \psi(e), e \rangle \).

Finally, we consider the case of \( \Delta \vdash \Gamma \varphi \psi(e), e \rangle \).

Note that \( \psi_{\Delta}^{(E)} \subseteq \psi_{\Delta}^{(E')} \) for all \( n \in \mathbb{N}, (\Delta \vdash \Gamma \psi(e), e \rangle \).

We will now turn to proving the context extension lemmas. Here, we only show a few representative cases.
Analogously to the may-termination case, we build a relation, however, is indexed over the countable ordinals.

\( \Psi \downarrow(E[\text{rand}(e)]) \leq \Psi\downarrow(E'[\text{rand}(e')]) \),

so we can conclude

\( (\text{rand} e)y, (\text{rand} e')y' \in \big[\Delta + \top\big](n) \).

- Assume \( \Delta | \Gamma + e \leq^{log}_{\downarrow} e': \tau. \) We will show \( \Delta | \Gamma + \text{fold} e \leq^{log}_{\downarrow} \text{fold} e': \mu.\tau \).

Let \( n < \omega, (y, y') \in \big[\Delta + \top\big](n), (E, E') \in \big[\Delta + \top\big](n) \) by proposition 4.14.

\( \Psi\downarrow(E[\text{fold} [\ ]]) \leq \Psi\downarrow(E'[\text{fold} [\ ]]) \).

\( \square \)

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

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

**Proof:**

1. \( \leq^{\mathit{CIU}}_{\downarrow} \leq^{\log}_{\downarrow} \).

Assume \( \Delta | \Gamma + e \leq^{\mathit{CIU}}_{\downarrow} e': \tau. \) It suffices to note that by proposition 4.14, \( \Delta | \Gamma + e \leq^{\downarrow}_{\downarrow} e': \tau. \) and that together with \( \Delta | \Gamma + e \leq^{\mathit{CIU}}_{\downarrow} e': \tau. \) this implies \( \Delta | \Gamma + e \leq^{\log}_{\downarrow} e': \tau. \) Indeed, let \( \varphi \in RVal(\Delta), n < \omega, (y, y') \in \big[\Delta + \top\big](n). \)

2. \( \leq^{\mathit{ctx}}_{\downarrow} \leq^{\mathit{CIU}}_{\downarrow} \).

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

3. \( \leq^{\mathit{ctx}}_{\downarrow} \leq^{\mathit{CIU}}_{\downarrow} \).

We show the case where both \( \Delta \) and \( \Gamma \) contain a single variable, which can then be generalized by induction on their length. Assume \( \alpha | x: \tau_1 + e \leq^{\mathit{ctx}}_{\downarrow} e': \tau_2. \) and let \( \sigma \in \text{Type}, v \in \text{Val}(\tau_1[\sigma/\alpha]), \) and \( E \in Ectx(\tau_1[\sigma/\alpha]) \) such that \( r : \tau_1[\sigma/\alpha] \Rightarrow \tau_2 \) for some \( \tau_2 \in \text{Type}. \) By reflexivity \( \emptyset | \emptyset + v \leq^{\mathit{ctx}}_{\downarrow} v: \tau_1[\sigma/\alpha]. \) By compatibility we can also show that \( \emptyset | \emptyset + E((\lambda \lambda x.e) - v \leq^{\mathit{ctx}}_{\downarrow} E((\lambda \lambda x.e') - v): \tau_2[\sigma/\alpha], \) and \( \emptyset | \emptyset + E((\lambda \lambda x.e) - v \leq^{\mathit{ctx}}_{\downarrow} E((\lambda \lambda x.e') - v): \tau_2 \).

By adequacy and the properties of deterministic reduction, \( \Psi\downarrow(E[e[v/x]]) \leq \Psi\downarrow(E'[e'[v/x]]) \).

\( \square \)

5 Connection to scheduler-based semantics

Previously we have defined observations that depended on either a maximal or a minimal probability of termination. In some sense, this is a global definition, it depends on every possible run of the program. A local notion of observation can instead be defined using a scheduler, that selects on each nondeterministic choice a natural number. This is analogous to the notion of schedulers used in concurrent settings, that chose on each step which thread to execute.

**Definition 5.1 (Scheduler).** A scheduler is a tuple \( \theta = (X, \theta_0, \theta_1) \) where \( X \) is a countable space state, \( \theta_0 : X \rightarrow \mathbb{N} \) is an output function and \( \theta_1 : X \times \text{Expr} \rightarrow X \) is a transition function. We let \( \mathcal{S} \) denote the set of all schedulers.
\[ \Delta \vdash \alpha \] \[ \phi (\xi) = r(\alpha)(\xi) \]
\[ \Delta \vdash \text{true} \]
\[ \phi (\xi) = (\xi, \epsilon) \]
\[ \Delta \vdash \text{true} \]
\[ \phi (\xi) = (k, k) | k \in \mathbb{N} \]
\[ \Delta \vdash \tau \rightarrow \sigma \]
\[ \phi (\xi) = \bigcup_{\beta \leq \xi} \{ (\lambda x. e, \lambda y. e') | \forall (v, v') \in \Delta \vdash \tau \phi (\beta).e[v/x], e'[v'/y] \} \in \Delta \vdash \sigma \phi (\tau) (\beta) \]
\[ \Delta \vdash \tau \rightarrow \sigma \]
\[ \phi (\xi) = \bigcup_{\beta \leq \xi} \{ (\text{in}(v), \text{in}(v')) | (v, v') \in \Delta \vdash \tau \phi (\beta) \} \cup \{ (\text{inr}(v), \text{inr}(v')) | (v, v') \in \Delta \vdash \sigma \phi (\xi) \}
\[ \Delta \vdash \forall \alpha. \tau \]
\[ \phi (\xi) = \{ (\lambda e. \Delta, e') \mid \forall \sigma, \sigma' \in \text{Type}, \forall r \in \text{RVal}(\sigma, \sigma').(e, e') \in \Delta, \alpha \vdash \tau \phi (\alpha \mapsto (\sigma, \sigma', r)) (\tau) (\xi) \}
\[ \Delta \vdash \mu \alpha. \tau \]
\[ \phi (0) = \text{Val}(\delta_1(\mu \alpha. \tau)) \times \text{Val}(\delta_2(\mu \alpha. \tau)) \]
\[ \Delta \vdash \mu \alpha. \tau \]
\[ \phi (\beta + 1) = \{ (\text{fold } v, \text{fold } v') | (v, v') \in \Delta, \alpha \vdash \tau \phi (\alpha \mapsto (\delta_1(\mu \alpha. \tau), \delta_2(\mu \alpha. \tau), \Delta \vdash \mu \alpha. \tau \phi (\beta))) (\beta) \]
\[ \Delta \vdash \mu \alpha. \tau \]
\[ \phi (\xi) = \{ (\text{fold } v, \text{fold } v') | (v, v') \in \Delta, \alpha \vdash \tau \phi (\alpha \mapsto (\delta_1(\mu \alpha. \tau), \delta_2(\mu \alpha. \tau), \Delta \vdash \mu \alpha. \tau \phi (\beta))) (\beta) \}

**Figure 3.** Definition of the logical relation on values for must-termination. We use \( \delta_1, \delta_2 \) and \( r \) to denote the three components of \( \phi \).

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 [9].

Other approaches [33] 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 traces.

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 takes a single step. For a scheduler \( \theta = (X, \theta_0, \theta_1) \) and a state \( x \in X \), we define:

\[ \Xi_\theta : (X \times \text{Expr} \rightarrow [0,1]) \rightarrow (X \times \text{Expr} \rightarrow [0,1]) \]
\[ \Xi_\theta (f)(x, e) = \begin{cases} 
1 & \text{if } e \in \text{Val} \\
\frac{1}{k} \sum_{m \leq k} f(\theta_t(x, E[m]), E[m]) & \text{if } e = E[?] \text{ and } \theta_2(x) = m \\
\frac{1}{k} \sum_{m \leq k} f(\theta_t(x, E[m]), E[m]) & \text{if } e = E[\text{rand } k] \\
f(\theta_t(x, e'), e') & \text{otherwise} \\
0 & \text{otherwise} 
\end{cases} \]

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 \( \omega \)-continuous. Therefore, we can define:

**Definition 5.2 (Probability of termination under a scheduler).** Let \( e \in \text{Expr}, \theta = (X, \theta_0, \theta_1) \) be a scheduler and \( x \in X \) be a state. The probability of termination of \( X \) under the scheduler \( \theta \) with initial state \( x \), is denoted by \( \Psi^\dagger_{\theta,x}(e) \), and defined as:

\[ \Psi^\dagger_{\theta,x}(e) = \sup_{e \in \text{Sch}} \Xi_\theta^\dagger(e) \]

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 prove the following lemmas:

**Lemma 5.3.** Let \( e \in \text{Expr}, \theta = (X, \theta_0, \theta_1), x \in X, n \in \mathbb{N} \). Then,

\[ \Xi_\theta^n(\bot)(x, e) \leq \Phi^\dagger(\bot)(e) \]

**Lemma 5.4.** Let \( e \in \text{Expr} \). Then for all \( n \in \mathbb{N}, e > 0 \) there exists \( \theta = (X, \theta_0, \theta_1) \) and \( x \in X \) such that

\[ \Xi_\theta^n(\bot)(x, e) \geq \Phi^\dagger(\bot)(e) - \epsilon \]

The proof is by induction on \( n \) and relies on the construction of an \( e \)-optimal scheduler. In particular, when resolving a nondeterministic choice \( E[?] \) there is always some \( m \in \mathbb{N} \) such that \( \sup_{e \in \text{Sch}} \Phi^\dagger(\bot)(E[m]) - \epsilon/2 \preceq \Phi^\dagger(\bot)(E[m]) \), so we can make the scheduler choose \( m \) in that step.

As a consequence, we get the following theorem:

**Theorem 5.5.** Let \( e \in \text{Expr} \). Then

\[ \Psi^\dagger(e) = \sup_{\theta \in \text{Sch}} \Xi_\theta^\dagger(e) \]

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

**Lemma 5.6.** Let \( e \in \text{Expr}, \theta = (X, \theta_0, \theta_1), x \in X, \alpha \in \omega_1 \). Then,

\[ \Xi_\theta^n(\bot)(x, e) \geq \Psi^\dagger(\bot)(e) \]
In particular,
\[ \Psi_\bot^\bot(e) \geq \Psi_\bot(e) \]

Lemma 5.7. Let \( e \in \text{Expr} \). Then for all \( n \in \mathbb{N} \), \( e > 0 \) there exists \( \theta = (X, \theta_0, \theta_1) \) and \( x \in X \) such that
\[ \Theta_\bot^\bot(x, e) \leq \Psi_\bot(e) + \epsilon \]

The proof constructs an \( \epsilon \)-pessimal scheduler. The idea is similar, except than when resolving a nondeterministic choice \( E[?] \) we pick \( m \in \mathbb{N} \) such that
\[ \Psi_\bot(E[m]) \leq \inf_{k \in \mathbb{N}} \Psi_\bot(E[k]) + 2^{-|\pi|} \cdot \epsilon \]
where \( |\pi| \) is the length of the current trace.

From the previous two lemmas, we get:

Theorem 5.8. For any expression \( e \in \text{Expr} \), we have
\[ \Psi_\bot(e) = \inf_{\theta, n \in \mathbb{N}} \sup_{x} \Xi_\theta^\bot(x, e) \]

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 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 will use some syntactic sugar in the examples. First, we assume we have a Boolean type, which is isomorphic to \( 1 + 1 \), and we write \( \text{if } b \text{ then } e \text{ else } e' \) for \( \text{case } (b, e, e') \). If we have two expressions \( e, e' \) we write \( \text{let } x = e \text{ in } e' \) for \( (\lambda x.e') \) \( e \) and if \( x \) is not free in \( e' \), we will occasionally just write \( e \) \( e' \). 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
\[ \text{if } (\text{rand } q) \leq p \text{ then } e \text{ else } e' \]
For the particular case of \( p = 1, q = 2 \) we simply write \( e \oplus e' \). Similarly, we write \( e \odot e' \) for
\[ \text{if } ? > 0 \text{ then } e \text{ else } e' \]
Note that both in \( e \oplus_{p/q} e' \) and \( e \odot e' \) the order of evaluation dictates that first the probabilistic or nondeterministic choice is made, and only then the corresponding subexpression is evaluated.

We can define a call-by-value fixpoint operator in our system as below:
\[
\begin{align*}
\text{fix} & : \forall \alpha. \forall \alpha'. (\alpha \rightarrow \alpha') \rightarrow (\alpha \rightarrow \alpha') \\
\text{fold} & : \Lambda. \Lambda. \Lambda. F. \Lambda. z. e_F (\text{fold } e_F) z \\
\text{where } e_F & = \lambda y. \text{let } y' = \text{unfold } y \text{ in } F (\lambda x.y' \ y \ x)
\end{align*}
\]
In particular, for every closed type \( \tau \), we can write a closed expression \( \Omega_\tau = \text{fix } (\lambda f. f) () \) such that \( \Omega_\tau : \tau \) and \( \Psi_\bot(\Omega_\tau) = \Psi_\bot(\Omega_\tau) = 0 \).

6.1 Extensionality

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

Proposition 6.1. Let \( \Delta \) be a type formation context, \( \Delta \vdash \Gamma, \sigma, \tau \in \text{Type}(\Delta) \) and suppose \( f, f' \) are values such that \( \Delta \vdash f : \sigma \rightarrow \tau \) and \( \Delta \vdash f' : \sigma \rightarrow \tau \). Then \( \Delta \vdash \Gamma \vdash f \leq_{\text{ctx}} f' \iff \text{for all values } v \text{ such that } \Delta \vdash \Gamma \vdash v : \sigma, \text{ we have } \Delta \vdash \Gamma \vdash v \leq_{\text{ctx}} f' \).

A similar extensionality result holds for types as well.

6.2 Probabilities of termination

In previous work [7], it was shown that, for a language without nondeterministic choice, the probability of termination of an expression \( e \) is a left-computable number. We recall here the definition of left- and right-computable numbers, and generalize their results to our setting.

Definition 6.2. A real number \( r \in \mathbb{R} \) is left-computable (resp. right-computable) if the set \( \mathbb{Q}_{<r} = \{ q \mid q \in \mathbb{Q} \land q < r \} \) (resp. \( \mathbb{Q}_{>r} = \{ q \mid q \in \mathbb{Q} \land q > r \} \) is recursively enumerable, or equivalently, if there exists a computable function \( \phi : \mathbb{N} \rightarrow \mathbb{Q} \) such that \( \phi(n) \) is non-decreasing (resp. nonincreasing) and \( r = \lim_{n \rightarrow \infty} \phi(n) \).

The following is a useful result:

Lemma 6.3. There exists an expression \( e^\bot \in \text{Expr}(\text{nat} \rightarrow \text{nat} \rightarrow 1 \rightarrow 1) \) such that for every \( m, n \in \mathbb{N} \) with \( n \leq m \) and \( m \neq 0 \), \( \Psi_\bot(e^\bot n m \langle \rangle) = \Psi_\bot(e^\bot m n \langle \rangle) = n/m \).

Proof. Let \( e^\bot = \lambda m. \lambda n. \text{let } x = \text{rand } m \text{ in } \text{if } x \leq n \text{ then } () \text{ else } \Omega_1 \).

First we study the computability of the probability of may-termination.

Proposition 6.4 ([7]). For every left-computable real \( r \) there exists an expression \( e^\bot_\tau \) such that \( \Psi_\bot(e^\bot_\tau) = \Psi_\bot(e^\bot_\tau) = r \).

Proof. We just use the fact that for the fragment without nondeterministic choice, \( \Psi_\bot() \) and \( \Psi_\bot() \) coincide.

The converse is also true. This follows from the fact that left-computable numbers are closed under suprema of computable sequences [40].

Proposition 6.5. For every \( e \in \text{Expr} \), \( \Psi_\bot(e) \) is a left-computable real.

For must-termination we have the following result:

Proposition 6.6. For every right-computable real \( r \) there exists an expression \( e^\bot_\tau \) such that \( \Psi_\bot(e^\bot_\tau) = r \).

Proof. Let \( \phi : \mathbb{N} \rightarrow \mathbb{N} \times \mathbb{N} \) be the computable function that produces the sequence whose limit is \( r \). Then we can define, \( e^\bot_\tau = \text{let } (m, n) = \phi \text{ in } e^\bot n m \).
and we have that
\[ \Psi(e_\omega) = \inf_{t \in T} \Psi(t) \iff \text{ let } (n, m) = \varphi \text{ in } e^L n m = r. \]

However, a characterization of the probability of must-termination is out of scope of this work. There are two challenges here: first, we are combining suprema and infima in the computation of the probability, so in general it will not be either left- or right-computable. Second, we take limits over the countable ordinals, which would require to generalize the notion of computable sequence to the ordinals.

### 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 6.7.** Let \( \tau \in \text{Type}, e_1, e_2, e_3 \in \text{Expr}(\tau) \). We have:

\[
e_1 \text{ or } e_1 \cong^{ctx} e_1
\]
\[
e_1 \text{ or } e_2 \cong^{ctx} e_2 \text{ or } e_1
\]
\[
e_1 \text{ or } (e_2 \text{ or } e_3) \cong^{ctx} (e_1 \text{ or } e_2) \text{ or } e_3
\]

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

**Proposition 6.8.** Let \( \tau \in \text{Type}, e_1, e_2, e_3 \in \text{Expr}(\tau), \) and \( p, q \in [0, 1] \cap \mathbb{Q} \). We have:

\[
e_1 \oplus_p e_1 \cong^{ctx} e_1
\]
\[
e_1 \oplus_p e_2 \cong^{ctx} e_2 \oplus_{1-p} e_1
\]
\[
(e_1 \oplus_p e_2) \oplus_q e_3 \cong^{ctx} e_1 \oplus_p e_2 \oplus_{1-p} e_3
\]
\[
(\; e_1 \oplus_p (e_2 \text{ or } e_3) \; \cong^{ctx} (e_1 \oplus_p e_2) \; \text{ or } (e_1 \oplus_p e_3) \;
\]

**6.4 Fixpoint combinator**

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

**Definition 6.10.** Let \( \sigma, \tau \) be types. We say that \( f \in \text{Val}(\sigma \rightarrow \tau) \) is deterministic if for every \( \nu \in \text{Val}(\sigma) \) there exists \( w \in \text{Val}(\nu) \) such that \( f \nu \cong^{ctx}\downarrow \; w. \)

Using extensionality, we can show:

**Proposition 6.11.** Let \( \sigma, \tau \) be types, and \( G \in \text{Val}(\sigma \rightarrow \tau) \rightarrow \tau \) deterministic. Then \( \text{fix}_\downarrow G \equiv^{ctx}_\downarrow G (\text{fix}_\downarrow G) \) and \( \text{fix}_\downarrow G \equiv^{ctx}_{\uparrow} G (\text{fix}_\downarrow G) \).

### 6.5 Guessing random tosses

Consider the two programs of type \( 1 \rightarrow \text{nat} \) below:

\[
toss \triangleq \lambda x.0 \oplus 1
\]
\[
guess \triangleq \lambda x.0 \text{ or } 1
\]

We then have:

\[
\text{if guess } () = \text{toss } () \text{ then } () \text{ else } \Omega \equiv^{ctx}_\downarrow () \oplus \Omega
\]
\[
\text{if guess } () = \text{toss } () \text{ then } () \text{ else } \Omega \equiv^{ctx}_\uparrow () \oplus \Omega
\]

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:

\[
\text{if toss } () = \text{guess } () \text{ then } () \text{ else } \Omega \equiv^{ctx}_\downarrow ()
\]
\[
\text{if toss } () = \text{guess } () \text{ then } () \text{ else } \Omega \equiv^{ctx}_\uparrow () \oplus \Omega
\]

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 [11]. We discuss this further in section 8.

### 6.6 Skip lists

Following the example of Tassarotti and Harper [33], 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 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.

We can model lists in our language using the type \([\tau] = \mu \alpha.1+\alpha \cdot \tau \alpha.\) As usual, we consider two constructors nil: \(\forall \alpha.\,\,1 \rightarrow [\alpha] \rightarrow [\alpha] (\text{sometimes written } :: \text{ and in}}
infix position) and an append function ++: \(\forall \alpha. [\alpha] \times [\alpha] \rightarrow [\alpha]\). 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 selects an element in a list, and splits the list in the elements that appear before and after it:

\[
\text{split } l \times \text{nil } \Downarrow (l, x, \text{nil})
\]

We also need the following lemmas:

**Lemma 6.12.** Let \(\tau \in \text{Type}, x \in \text{Val}(\tau), l, r \in \text{Val}([\tau])\). Then

\[
\begin{align*}
\text{let } (y, z) &= \text{split } l \times r \text{ in } \text{ys++cons}(z, y) \\
\text{skiplist’ } l \(ys ++ (z :: zs)) \end{align*}
\]

**Lemma 6.13.** Let \(\tau \in \text{Type}, x, y \in \text{Val}(\tau), l, y, zs \in \text{Val}([\tau])\). Then

\[
\text{skiplist’ } l \times (y :: x :: l t)
\]

**Lemma 6.14.** Let \(\tau \in \text{Type}, x, y \in \text{Val}(\tau), l, t l \in \text{Val}([\tau])\). Then

\[
\text{skiplist’ } x :: y :: l t l \Downarrow \text{skiplist’ } y :: x :: l t
\]

**Lemma 6.15.** Let \(\tau \in \text{Type}, x \in \text{Val}(\tau), l, r, t l \in \text{Val}([\tau])\). Then

\[
\text{skiplist’ } (l ++ \text{cons}(x, y)) \Downarrow \text{skiplist’ } \text{cons}(x, l ++ r)
\]

**Theorem 6.16.** Let \(\tau \in \text{Type}, l, t l \in \text{Val}([\tau])\). Then

\[
\text{skiplist } x \times l t l \Downarrow \text{skiplist’ } x s t l
\]

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

\[
\begin{align*}
\text{let } (l, y, r) &= \text{split } n i l \times x \times \text{in } \\
\text{skiplist } x \times l t l \Downarrow \text{split } \text{nil } \text{x } \text{nil} \text{ in } \\
\text{let } tl &= tl \text{ } \text{\textcopyright } \text{cons}(x, y) \text{tl} \text{ in } \\
\text{skiplist’ } x \times l t l'
\end{align*}
\]

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 sort the functions. For all \(x, y \in \text{Val}(\text{nat})\) and \(l_1, l_2, l_3 \in \text{Val}([\text{nat}])\):

\[
\begin{align*}
nil ++ l_1 &\Downarrow l_1 ++ \text{nil} \Downarrow l_1 \\
\text{x :: l_1} ++ l_2 &\Downarrow \text{sort(l_1 ++ l_2)} \\
l_1 ++ (l_2 ++ l_3) &\Downarrow \text{sort(l_1 ++ l_2)}
\end{align*}
\]
7 Related work

Semantic models for probabilistic programming languages were introduced in Kozen’s seminal paper [21]. Powerdomains, initially used to model nondeterminism [30] were later adapted to probabilistic languages [18, 31], but extending this construction to higher-order is challenging [19]. Semantics based on measurable spaces are also hard to generalize to higher-order since the category of measurable spaces is not Cartesian closed [4]. In recent years, Cartesian closed categories to denote higher-order probabilistic languages have been proposed [13, 16, 35]. All these models are fairly involved, and it is unclear how to incorporate further extensions, such as recursive types and impredicative polymorphism, into them. Operational models for an untyped lambda calculus with sampling and scoring have been developed by Borgström et al. [10].

Step-indexed logical relations were introduced by Appel and McAllester [2] to model recursion, and have been shown to scale well to higher-order languages that support recursive types and polymorphism [1], as well as a variety of effects, including concurrency [6, 34]. Bioorthogonality [29] 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 [7], and then studied for continuous choice [12, 37]. In a recent paper, Zhang and Amin [39] show how step-indexed logical relations can also be used to reason about higher-order probabilistic programs with nested queries.

Apt and Plotkin [3] observed that countable nondeterminism introduces non-continuous behavior, and that iterating up to $\omega$ is not enough to model it. In his thesis, Lassen [23] studies contextual equivalence of programs with countable nondeterministic choice using operational semantics. Later Birkedal et al. [5] 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 $D$ and the powerset monad $\mathcal{P}$, respectively, but, as Varacca and Winskel [36] point out, there does not exist a distributive law between them. They propose two solutions: modeling probabilities by a monad $I$ 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 [25] and nontermination [24]. Bonchi et al. [9] study the coalgebras of $C$ to reason about trace equivalence for transition systems combining probabilities and nondeterminism. Another approach [15] shows that $C$ can be recovered by lifting $\mathcal{P}$ to the category of $D$-algebras.

Other techniques to reason about the combination of probabilistic and nondeterministic choice include predicate transformers [20, 27] 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 [22, 32]. 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 [8] to define behavioral equivalence. In recent work, Bonchi et al. [9] 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 parametric 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.

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).

Future work includes adding continuous probabilities and conditioning, to be able to capture higher-order probabilistic languages such as Anglican [38] or WebPPL [14]. In future work, 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. Finally, it could be interesting to study other notions of equivalence, e.g., one where two expressions $e, e'$ are equivalent if they have the same termination probability for any scheduler in some class $S$ of schedulers, i.e. $\forall \theta \in S. \Psi^\theta_I(e) = \Psi^\theta_I(e')$. 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 in [11].
References


Proof of lemma 4.10. Suffices to show that for every \( \alpha \), \( F_\alpha(\alpha) \geq \sup_{\beta < \alpha} F_\beta(\beta) \). This can be proven by transfinite induction. The zero case is trivial, and the limit case follows directly from the definition. For the successor case, by monotonicity of \( \Psi \) we have
\[
F_\alpha(\alpha + 1) = \Psi(\Psi^\alpha(\bot))(e)
\[
\geq \Psi(\sup_{\beta < \alpha} \Psi^\beta(\bot))(e)
\[
\geq \sup_{\beta < \alpha} \Psi(\Psi^\beta(\bot))(e)
\[
= \sup_{\beta < \alpha} \Psi^\beta(\bot)(e) = \sup_{\beta < \alpha + 1} \Psi^\beta(\bot)(e) = \sup_{\beta < \alpha + 1} F_\beta(\beta)
\]
The first inequality uses the LH. In the reindexing of the last equality, we can ignore the case \( \beta = 0 \), since \( \Psi^0(\bot) = \bot \).

A.2 Proofs of Section 4

Proof of lemma 4.11. Let \( (v, v') \in \llbracket \Delta \vdash \sigma \rrbracket(\phi)(n) \). Then it must be the case that \( v = \lambda x.t, v' = \lambda x.t' \) for some \( t, t' \), and furthermore, for any \( m \leq n \), \( (t[w/x], t'[w'/x]) \in \llbracket \Delta \vdash \sigma \rrbracket(\phi)^T(m) \). Therefore, \( \Psi^\beta(E[v w]) \leq \Psi^\beta(E'[v' w']) \), so we can conclude \( (E'[v][], E'[v'][[]]) \in \llbracket \Delta \vdash \sigma \rrbracket(\phi)^T(n) \).

Proof of lemma 4.12. Let \( (v, v') \in \llbracket \Delta \vdash \text{nat} \rrbracket(\phi)(n) \). Then, there exists \( k \in \mathbb{N} \) such that \( v = v' = k \). Now, we have
\[
\Psi^\beta(E[\text{rand } k]) \leq \sum_{1 \leq l \leq k} \frac{1}{k} \cdot \Psi^\beta(E[l])
\]
\[
\leq \sum_{1 \leq l \leq k} \frac{1}{k} \cdot \Psi^\beta(E'[l])
\]
\[
= \Psi^\beta(E'[\text{rand } k])
\]
Therefore, \( (E[\text{rand } []], E'[\text{rand } []]) \in \llbracket \Delta \vdash \text{nat} \rrbracket(\phi)^T(n) \).

Lemma A.1. Let \( n < \omega, \phi \in \text{RVVal}(\Delta) \), and \( (E, E') \in \llbracket \Delta \vdash \tau \times \sigma \rrbracket(\phi)^T(n) \). Let also \( (v, v') \in \llbracket \Delta \vdash \sigma \rrbracket(\phi)^T(n) \). Then
\[
(E[(v, [])], E[(v', [])]) \in \llbracket \Delta \vdash \sigma \rrbracket(\phi)^T(n).
\]

Proof. Let \( (w, w') \in \llbracket \Delta \vdash \sigma \rrbracket(\phi)(n) \). Then, by definition, \( (v, v') \in \llbracket \Delta \vdash \sigma \rrbracket(\phi)(n) \), so for all \( m \leq n \), \( \Psi^\beta_m(E[(v, w)]) \leq \Psi^\beta(E'[v', w']) \). From this, it follows that \( (E[(v, []), E[(v', [])]) \in \llbracket \Delta \vdash \sigma \rrbracket(\phi)^T(n) \).

Lemma A.2. Let \( n < \omega, \phi \in \text{RVVal}(\Delta) \), and \( (E, E') \in \llbracket \Delta \vdash \tau \times \sigma \rrbracket(\phi)^T(n) \). Let also \( (e, e') \in \llbracket \Delta \vdash \sigma \rrbracket(\phi)^T(n) \). Then
\[
(E[([]), E'[([])]) \in \llbracket \Delta \vdash \tau \rrbracket(\phi)^T(n).
\]

Proof. Let \( (v, v') \in \llbracket \Delta \vdash \tau \rrbracket(\phi)(n) \). By Lemma A.1 follows that \( (E[(v, []), E[(v', [])]) \in \llbracket \Delta \vdash \sigma \rrbracket(\phi)^T(n) \), so for all \( m \leq n \), \( \Psi^\beta_m(E[(v, e)]) \leq \Psi^\beta(E'[v', e']) \). From this, it follows that \( (E[(v, e), E'[(v', e'])]) \in \llbracket \Delta \vdash \tau \rrbracket(\phi)^T(n) \).

Lemma A.3. Let \( m < \omega, \phi \in \text{RVVal}(\Delta) \), and \( (E, E') \in \llbracket \Delta \vdash \tau \rrbracket(\phi)(n) \). Assume further that for every \( (v_1, v'_1) \in \llbracket \Delta \vdash \tau \rrbracket(\phi)(n) \), it is the case that \( (e_1[v_1/x_1], v'_1[x_1/x_1]) \in \llbracket \Delta \vdash \tau \rrbracket(\phi)^T(n) \); and that for every \( (v_2, v'_2) \in \llbracket \Delta \vdash \sigma \rrbracket(\phi)(n) \) then \( (e_2[v_2/x_2], v'_2[x_2/x_2]) \in \llbracket \Delta \vdash \tau \rrbracket(\phi)^T(n) \). Then we have
\[
(E[\text{case}([]), x_1.e_1, x_2.e_2]), E'[\text{case}([], x_1.e'_1, x_2.e'_2)])
\]
\[
\in \llbracket \Delta \vdash \tau \rrbracket(\phi)^T(n).
\]
Proof. Let \((v, v') \in \Delta \vdash \tau + \sigma (\varphi) (n)\). Let us assume \(v = \text{inl}(w)\) and \(v' = \text{inl}(w')\) such that \((w, w') \in \Delta \vdash \tau (\varphi) (n)\) (otherwise, \(v = \text{inr}(w)\) and \(v' = \text{inr}(w')\) such that \((w, w') \in \Delta \vdash \sigma (\varphi) (n)\) and the proof proceeds analogously). Then, for every \(m \leq n\),
\[
\Psi^\downarrow_m (E \text{[case]}(v, x_1.e_1, x_2.e_2)) = \Psi^\downarrow_m (E \text{[case]}(\text{inl}(w), x_1.e_1, x_2.e_2)) \\
\quad \leq \Psi^\downarrow_m (E [e_1/x_1]) \\
\quad \leq \Psi^\downarrow (E' [e'_1/w'/x_1]) \\
\quad = \Psi^\downarrow (E' [\text{case} (\text{inl}(w'), x_1.e'_1, x_2.e'_2)]) \\
\quad = \Psi^\downarrow (E' [\text{case} (v', x_1.e'_1, x_2.e'_2)]).
\]
Therefore, \((E \text{[case]}([], x_1.e_1, x_2.e_2), E' \text{[case]}([], x_1.e'_1, x_2.e'_2)) \in \Delta \vdash \tau + \sigma (\varphi)^\top (n)\). □

Proof of lemma 4.13. Let \((v, v') \in \Delta \vdash \tau [\mu \alpha. \tau/\alpha] (\varphi) (n)\). By definition of the interpretation and applying lemma 4.7, \((\text{fold } v, \text{fold } v') \in \Delta \vdash \tau [\mu \alpha. \tau/\alpha] (\varphi) (n + 1)\). Therefore, for all \(m \leq n + 1\) (and in particular, \(m \leq n\)), \(\Psi^\downarrow_m (E [\text{fold } v]) \leq \Psi^\downarrow (E' [\text{fold } v']).\) Thus \((E [\text{fold } []], E' [\text{fold } []]) \in \Delta \vdash \tau [\mu \alpha. \tau/\alpha] (\varphi)^\top (n)\). □

Lemma A.4. Let \(n < \omega, \sigma \in RVal(\Delta), \text{ and } (E, E') \in \Delta \vdash \tau [\mu \alpha. \tau/\alpha] (\varphi)^\top (n). \) Then \((E [\text{unfold } w], E' [\text{unfold } w]) \in \Delta \vdash \mu \alpha. \tau (\varphi)^\top (n)\). Proof. Let \((v, v') \in \Delta \vdash \mu \alpha. \tau (\varphi)^\top (n)\). Then, \(v = \text{fold } w, v' = \text{fold } w'\) and for all \(m < n\), \((w, w') \in \Delta \vdash \mu \alpha. \tau (\varphi)^\top (m)\).
We will show that for every \(m \leq n\), \(\Psi^\downarrow_m (E [\text{unfold } w]) \leq \Psi^\downarrow (E' [\text{unfold } w']).\) If \(n = 0\), then \(\Psi^\downarrow_0 (E [\text{unfold } w]) = 0\) and we are done. Otherwise, \(n = n' + 1\). Let \(m \leq n = n' + 1\). Then:
\[
\Psi^\downarrow_m (E [\text{unfold } w]) \leq \Psi^\downarrow_{m+1} (E [\text{unfold } w]) \\
\quad = \Psi^\downarrow_m (E [w]) \\
\quad \leq \Psi^\downarrow (E' [w']) \\
\quad = \Psi^\downarrow (E' [\text{unfold } w']).
\]
Hence \((E [\text{unfold } []], E' [\text{unfold } []]) \in \Delta \vdash \mu \alpha. \tau (\varphi)^\top (n)\). □

Lemma A.5. Let \(n < \omega, \sigma \in RVal(\Delta), \text{ and } (E, E') \in \Delta \vdash \sqrt{\tau [\alpha/\alpha]} (\varphi)^\top (n). \) Then \((E [[], E' [[]]) \in \Delta \vdash \forall \alpha. \tau (\varphi)^\top (n)\). Proof. Let \((v, v') \in \Delta \vdash \forall \alpha. \tau (\varphi)^\top (n)\). Then, \(v = \text{Ax } e, v' = \text{Ax } e', \) and for every \((\sigma, \sigma') \in \text{Type} \text{ and every } r \in RVal (\sigma, \sigma')\), \((e, e') \in \Delta \vdash \tau (\varphi [\alpha \mapsto (\sigma, \sigma', r)])^\top (n)\). Therefore,
\[
\Psi^\downarrow_\tau (E ([\text{Ax } e][])) \leq \Psi^\downarrow (E [e]) \leq \Psi^\downarrow (E' [e']) = \Psi^\downarrow (E' ([\text{Ax } e'][[]]).
\]
Therefore, \((E [[], E' [[]]) \in \Delta \vdash \forall \alpha. \tau (\varphi)^\top (n)\). □

Proposition (proposition 4.14). The relation \(\leq^\downarrow_{\forall} \) is compatible. In particular, it is reflexive for any type context \(\Delta\), typing context \(\Gamma\), expression \(e\) and type \(\tau\), if \(\Delta \vdash \Gamma : e : \tau\), then \(\Delta \vdash \Gamma : e \leq^\downarrow_{\forall} e : \tau\).
Proof. The most interesting cases are:
- Assume \(\Delta \vdash \Gamma, x : \sigma_1 \vdash e \leq_{\forall}^\downarrow e' : \sigma_2\). We will show that \(\Delta \vdash \Gamma \vdash \lambda x. e \leq_{\forall}^\downarrow \lambda x. e' : \sigma_1 \rightarrow \sigma_2\). Let \(n < \omega\), \((u, u') \in \Delta \vdash \sigma \cap \sigma_1(n)\). Then, for any \((y, y') \in \Delta \vdash \Gamma (n)\) we have that \((y[x \mapsto u], y'[x \mapsto u']) \in \Delta \vdash \Gamma, x : \sigma_1(n), \) so \((e y[x \mapsto u], e'y'[x \mapsto u']) \in \Delta \vdash \Gamma (n)\). By definition, \(e y[x \mapsto u] = e'(u/x)y, e'y'[x \mapsto u'] = e'(u'/x)y\), so we can conclude.
- Assume \(\Gamma \vdash e_1 \leq_{\forall}^\downarrow e'_1 : \sigma \rightarrow \tau, \) and \(\Gamma \vdash e_2 \leq_{\forall}^\downarrow e'_2 : \sigma \rightarrow \sigma_2\). Let \(n < \omega\) and \((y, y') \in \Delta \vdash \Gamma (n)\), and \((E, E') \in \Delta \vdash \tau^\top (n)\). Then, by definition of the relational interpretation, \((e_1 y, e'_1 y') \in \Delta \vdash \sigma \rightarrow \tau^\top (n)\), and by lemma 4.10, \((E[e_1 y][]), E'[e'_1 y'][[]]) \in \Delta \vdash \sigma^\top (n)\). On the other hand, also by definition \((e_2 y, e'_2 y') \in \Delta \vdash \sigma^\top (n)\), so for every \(m \leq n\), \(\Psi^\downarrow_m (E [e'_1 y') (e'_2 y')) \leq \Psi^\downarrow (E' [(e'_1 y') (e'_2 y')])\), so \(((e_1 e_2) y, (e'_1 e'_2) y') \in \Delta \vdash \sigma^\top (n)\).
- If \(e \equiv \text{case} \ldots\). Let \(n < \omega, (E, E') \in \Delta \vdash \text{nat}^\top (n)\). We have to show that for every \(m \leq n\), \(\Psi^\downarrow_m (E [?]) \leq \Psi^\downarrow (E' [?])\). Note that \(\Psi^\downarrow_m (E [?]) \leq \sup_{k \in \mathbb{N}} \Psi^\downarrow_m (E [?])\) and by induction hypothesis, for all \(k \in \mathbb{N}\), \(\Psi^\downarrow_m (E [k]) \leq \Psi^\downarrow (E' [k])\), so \(\Psi^\downarrow_m (E [?]) \leq \sup_{k \in \mathbb{N}} \Psi^\downarrow (E' [k]) = \Psi^\downarrow (E' [?])\).
• Assume $\Gamma \vdash e \leq^\log_1 e' \colon \text{nat}$. We will show that $\Gamma \vdash \text{rand } e \leq^\log_1 \text{rand } e' \colon \text{nat}$. By assumption, for all $n < \omega$ and all $(y, y') \in \mathbb{A} + \Gamma(n)$, $(e, e') \in \mathbb{A} + \text{nat}^\top(n)$. Now let $(E, E') \in \mathbb{A} + \text{nat}^\top(n)$. By lemma 4.12,
\[ (E[[\text{rand } [ ]], E'[\text{rand } [ ]]) \in \mathbb{A} + \text{nat}^\top(n), \]
and therefore,
\[ \mathcal{P}_n(E[[\text{rand } ty]]) \leq \mathcal{P}(E'[\text{rand } (t' y')]), \]
so we can conclude
\[ \mathcal{P}_n(E[[\text{rand } e]]) \leq \mathcal{P}(E'[\text{rand } (e')]). \]

• Assume $\Delta \vdash e \leq^\log_1 e' \colon (\mu x. \tau / \alpha)$. We will show $\Delta \vdash \text{fold } e \leq^\log_1 \text{fold } e' \colon (\mu x. \tau / \alpha)$. Let $n < \omega$, $(y, y') \in \mathbb{A} + \Gamma(n)$, $(E, E') \in \mathbb{A} + (\mu x. \tau / \alpha)^\top(n)$. By assumption, $(e, e') \in \mathbb{A} + (\tau \rightarrow (\mu x. \tau / \alpha))^\top(n)$ and by lemma 4.13,
\[ (E[[\text{fold } [ ]], E'[\text{fold } [ ]]) \in \mathbb{A} + (\tau \rightarrow (\mu x. \tau / \alpha))^\top(n), \]
so we can conclude
\[ \mathcal{P}_n(E[[\text{fold } e]]) \leq \mathcal{P}(E'[\text{fold } (e')]). \]

• Assume $\Delta \vdash e \leq^\log_1 e' \colon \mu x. \tau$. We will show $\Delta \vdash \text{unfold } e \leq^\log_1 \text{unfold } e' \colon \mu x. \tau$. Let $n < \omega$, $(y, y') \in \mathbb{A} + \Gamma(n)$, $(E, E') \in \mathbb{A} + (\mu x. \tau)^\top(n)$. By assumption, $(e, e') \in \mathbb{A} + (\mu x. \tau)^\top(n)$ and by lemma A.4,
\[ (E[[\text{unfold } [ ]], E'[\text{unfold } [ ]]) \in \mathbb{A} + (\mu x. \tau)^\top(n), \]
so we can conclude
\[ \mathcal{P}_n(E[[\text{unfold } e]]) \leq \mathcal{P}(E'[\text{unfold } (e')]). \]

Some lemmas useful for proving context extension:

**Lemma A.6.** Let $\Delta \vdash \tau, \alpha + \sigma$ and $\varphi = (\delta_1, \delta_2, r) \in RVal(\Delta)$. Then
\[ [\Delta + \sigma[\tau/\alpha]](\varphi) = [\Delta, \alpha + \sigma](\varphi[\alpha \mapsto (\delta_1(\tau), \delta_2(\tau), \mathbb{A} + \tau]\varphi)). \]

**Lemma A.7.** Let $\tau, \tau' \in \text{Type}$, $r \in \mathbb{S} \subseteq RVal(\tau, \tau')$. Then $r \subseteq r^\top.$

**Lemma A.8.** Let $\tau, \tau' \in \text{Type}$, $r \in \mathbb{S} \subseteq RVal(\tau, \tau')$. Then $r^\top \subseteq r^\top.$

**Lemma A.9.** Let $\xi < \omega$, $\varphi \in RVal(\Delta)$, $(E, E') \in \mathbb{A} + \sigma[\varphi]^\top(\xi)$ and $(v, v') \in \mathbb{A} + \sigma \rightarrow \tau^\top(\varphi)(\xi)$. Then,
\[ (E[[v [ ]], E'[v' [ ]]) \in [\Delta + \sigma](\varphi)^\top(\xi). \]

The logical relation for must termination satisfies the following context extension lemmas:

**Proof:** Let $(w, w') \in [\Delta + \tau][\varphi](\xi)$. From the assumptions, it must be the case that $v = \lambda x.t$, $v' = \lambda x.t'$ for some $t, t'$, and furthermore, for any $\beta \leq \xi$, $t[w/x], t'[w'/x] \in [\Delta + \sigma]([\tau] (\varphi)^\top(\beta))$. Therefore, $\mathcal{P}(E[[v w]]) \leq \mathcal{P}(E'[v' w']) \leq \mathcal{P}(E'[v' w'])$, so we can conclude $(E[[v [ ]], E'[v' [ ]])] \in [\Delta + \sigma][\varphi]^\top(\alpha).$ \hfill $\square$

**Lemma A.10.** Let $\xi < \omega$, $\varphi \in RVal(\Delta)$, $(E, E') \in [\Delta + \tau][\varphi]^\top(\xi)$ and $(e, e') \in [\Delta + \sigma][\varphi]^\top(\xi)$. Then,
\[ (E[[e [ ]], E'[e' [ ]]) \in [\Delta + \sigma \rightarrow \tau][\varphi]^\top(\xi). \]

**Proof:** Let $(v, v') \in [\Delta + \sigma \rightarrow \tau][\varphi](\xi)$ and then it must be the case that $v = \lambda x.t$, $v' = \lambda x.t'$ for some $t, t'$. By the previous context extension lemma, then $(E[[v [ ]], E'[v' [ ]])] \in [\Delta + \sigma][\varphi]^\top(\xi)$, and by assumption on $(e, e')$ we have for all $\beta \leq \xi$, $\mathcal{P}(E[[e v]]) \leq \mathcal{P}(E'[v' e'])$, so $(E[[e [ ]], E'[e' [ ]])] \in [\Delta + \sigma \rightarrow \tau][\varphi]^\top(\xi).$ \hfill $\square$

**Lemma A.11.** Let $\xi < \omega$, $\varphi \in RVal(\Delta)$, and $(E, E') \in [\Delta + \mathbb{N}][\varphi]^\top(\xi)$. Then
\[ (E[[\text{rand } [ ]], E'[\text{rand } [ ]]) \in [\Delta + \mathbb{N}][\varphi]^\top(\xi). \]

**Proof:** Let $(v, v') \in [\Delta + \mathbb{N}][\varphi](\xi)$. Then, there exists $k \in \mathbb{N}$ such that $v = v' = k$. Now, we have
\[ \mathcal{P}(E[[\text{rand } k]]) \leq \sum_{1 \leq m \leq k} \frac{1}{k} \cdot \mathcal{P}(E[m]) \leq \sum_{1 \leq m \leq k} \frac{1}{k} \cdot \mathcal{P}(E'[m]) = \mathcal{P}(E'[\text{rand } k]). \]

Therefore, $(E[[\text{rand } [ ]], E'[\text{rand } [ ]]) \in [\Delta + \mathbb{N}][\varphi]^\top(\xi).$ \hfill $\square$
Lemma A.12. Let $\xi < \omega_1$, $\psi \in \text{RVal}(\Delta)$, and $(E, E') \in \Delta \downarrow \tau \times \sigma(\phi \uparrow(\xi))$. Let also $(v, v') \in \Delta \downarrow \tau(\phi(\xi))$. Then

\[
(E([v, []]), E([v', []])) \in \Delta \downarrow \sigma(\phi(\xi)).
\]

Proof. Let $(w, w') \in \Delta \downarrow \sigma(\phi(\xi))$. Then, by definition, $(v, w, \langle v', w' \rangle) \in \Delta \downarrow \tau \times \sigma(\phi(\xi))$, so for all $\beta \leq \xi$, $\Psi^\beta_E(E([v, w])) \leq \Psi^\beta_E(E'[\langle v', w' \rangle])$. From this follows that $(E([v, []]), E([v', []])) \in \Delta \downarrow \sigma(\phi(\xi))$. \hfill\(\square\)

Lemma A.13. Let $\xi < \omega_1$, $\phi \in \text{RVal}(\Delta)$, and $(E, E') \in \Delta \downarrow \tau \times \sigma(\phi(\xi))$. Let also $(e, e') \in \Delta \downarrow \sigma(\phi(\xi))$. Then $(E([]), E([, e'])) \in \Delta \downarrow \tau(\phi(\xi))$. Therefore, by Lemma A.12 follows that

\[
(E[\text{case}([], x_1, e_1, x_2, e_2)], E'[\text{case}([], x_1, e_1', x_2, e_2']) \in \Delta \downarrow \tau + \sigma(\phi(\xi)).
\]

Proof. Let $(v, v') \in \Delta \downarrow \tau + \sigma(\phi(\xi))$. Let also $v = \text{inl}(w)$ and $v' = \text{inl}(w')$ such that $(w, w') \in \Delta \downarrow \tau(\phi(\xi))$ and the proof proceeds analogously. Then, for every $\beta \leq \xi$,

\[
\Psi^\beta_E(E[\text{case}(v, x_1, e_1, x_2, e_2)]) = \Psi^\beta_E(E[\text{case}(\text{inl}(w), x_1, e_1, x_2, e_2)])
\]

\[
\leq \Psi^\beta_E(E[e_1,w/x_1]) \\
\leq \Psi^\beta_E(E'[e_1',w'/x_1]) \\
= \Psi^\beta_E(E'[\text{case}(\text{inl}(w'), x_1, e_1', x_2, e_2')]) \\
= \Psi^\beta_E(E'[\text{case}(v', x_1, e_1', x_2, e_2')])
\]

Therefore, $(E[\text{case}([], x_1, e_1, x_2, e_2)], E'[\text{case}([], x_1, e_1', x_2, e_2']) \in \Delta \downarrow \tau + \sigma(\phi(\xi))$. \hfill\(\square\)

Lemma A.14. Let $\xi < \omega_1$, $\phi \in \text{RVal}(\Delta)$, and $(E, E') \in \Delta \downarrow \tau(\phi(\xi))$. Assume further that for every $(v_1, v_1') \in \Delta \downarrow \tau(\phi(\xi))$ it is the case that $(e_1[v_1/x_1], e_1'[v_1'/x_1]) \in \Delta \downarrow \tau'(\phi(\xi'))$ and that for every $(v_2, v_2') \in \Delta \downarrow \tau(\phi(\xi))$ then $(e_2[v_2/x_2], e_2'[v_2'/x_2]) \in \Delta \downarrow \tau'(\phi(\xi'))$. Then we have

\[
(E[\text{case}([], x_1, e_1, x_2, e_2)], E'[\text{case}([], x_1, e_1', x_2, e_2']) \in \Delta \downarrow \tau + \sigma(\phi(\xi)).
\]

Proof. Let $(v, v') \in \Delta \downarrow \tau + \sigma(\phi(\xi))$. Let also $v = \text{inr}(\alpha, x_1)$ and $v' = \text{inr}(\alpha', x_2)$ such that $(\alpha, x_1) \in \Delta \downarrow \tau(\phi(\xi))$ and the proof proceeds analogously. Then, for every $\beta \leq \xi$,

\[
\Psi^\beta_E(E[\text{case}(v, x_1, e_1, x_2, e_2)]) = \Psi^\beta_E(E[\text{case}([\alpha, x_1], e_1, x_2, e_2)])
\]

\[
\leq \Psi^\beta_E(E[e_1,\alpha,x_1/x_1]) \\
= \Psi^\beta_E(E'[\text{case}(\alpha', x_2, e_2']) \\
= \Psi^\beta_E(E'[\text{case}(v', x_1, e_1', x_2, e_2')])
\]

Therefore, $(E[\text{case}([], x_1, e_1, x_2, e_2)], E'[\text{case}([], x_1, e_1', x_2, e_2']) \in \Delta \downarrow \tau + \sigma(\phi(\xi))$. \hfill\(\square\)
Therefore, \((E[\text{unfold }]\,\psi), E'[\text{unfold }]\,\psi)\) ∈ \(\Delta \vdash \mu \alpha.\tau/\alpha\)\((\phi)^\top(\xi)\).

\[\] **Lemma A.17.** Let \(\xi < \omega_1, \phi \in \text{RVal}(\Delta), \) and \((E, E') \in \{\tau/\alpha\}\((\phi)^\top(\xi)\). Then \((E[\{\_\}], E'[\{\_\}]) \in \Delta \vdash \forall \alpha.\tau\((\phi)^\top(\xi)\).\]

\[\]

**Proof.** Let \((a, a') \in \{\Delta \vdash \forall \alpha.\tau\}(\phi)^\top(\xi)\). Then, \(a = \Lambda.e, a' = \Lambda.e'\), and for every \((\sigma, \sigma') \in \text{Type}\) and every \(r \in \text{RVal}(\sigma, \sigma')\), \((e, e') \in \{\Delta \vdash \tau\((\alpha \mapsto (\sigma, \sigma', r))\)^\top(\xi)\).

\[\]

Therefore, \((E[\{\_\}], E'[\{\_\}]) \in \Delta \vdash \forall \alpha.\tau\((\phi)^\top(\xi)\).

\[\] **Proposition (proposition 4.18).** The relation \(\leq_{\text{log}}\) is compatible. In particular, it is reflexive: for any type context \(\Delta\), typing context \(\Gamma\), expression \(e\) and type \(\tau\), if \(\Delta \mid \Gamma \vdash e : \tau\), then \(\Delta \mid \Gamma \vdash e \leq_{\text{log}} e : \tau\).

\[\]

**Proof.** The most interesting cases are:

- Assume \(\Delta \mid \Gamma, x : \sigma_1 \vdash e \leq_{\text{log}} e' : \sigma_2\). We will show that \(\Delta \mid \Gamma \vdash \lambda x.e : \sigma_1 \rightarrow \sigma_2\). Let \(\xi < \omega_1, (u, u') \in \{\Delta \vdash \sigma_1\}(\xi)\). Then, for any \((y, y') \in \{\Delta \vdash \Gamma\}(\xi)\) we have that \((y[x \mapsto u], y'[x \mapsto u']) \in \{\Delta \vdash \Gamma, x : \sigma_1\}(\xi)\), so \((e[y[x \mapsto u]], e'[y'[x \mapsto u']]) \in \{\Delta \vdash \sigma_2\}(\xi)\). By definition, \(e[y[x \mapsto u]] = e[u/x]y\) and \(e'[y'[x \mapsto u']] = e'[u'/x]y\), so we can conclude.

- Assume \(\Gamma \vdash e_1 \leq_{\text{log}} e_1' : \sigma \rightarrow \tau, \) and \(\Gamma \vdash e_2 \leq_{\text{log}} e_2' : \sigma\). We will show that \(\Delta \mid \Gamma \vdash e_1 \cdot e_2 \leq_{\text{log}} e_1' \cdot e_2' : \sigma \rightarrow \tau\). Let \(\xi < \omega_1\) and \((y, y') \in \{\Delta \vdash \Gamma\}(\xi)\), and \((E, E') \in \{\Delta \vdash \tau\}(\xi)\). By definition of the relational interpretation, \((e_1y, e_1'y) \in \{\Delta \vdash \sigma \rightarrow \tau\}(\xi)\), and by lemma A.9, \((E[e_1y[\_]], E'[e_1'y[\_]]) \in \{\Delta \vdash \Gamma\}(\xi)\). On the other hand, also by definition \((e_2y, e_2'y) \in \{\Delta \vdash \sigma\}(\xi)\), so for every \(\beta \leq \xi\), \(\Psi^\beta_{\psi}(E[e_1y(e_2y)]) \leq \Psi^\beta_{\psi}(E'[e_1'y(e_2'y)])\), \((e_1, e_2)y, (e_1', e_2')y' \in \{\Delta \vdash \tau\}(\xi)\).

- If \(e \equiv \_\). Let \(\xi < \omega_1, (E, E') \in \{\Delta \vdash \text{nat}\}(\xi)\). We have to show that for every \(\beta \leq \xi, \Psi^\beta_{\psi}(E[\_]) \leq \Psi^\beta_{\psi}(E'[\_])\). Note that \(\Psi^\beta_{\psi}(E[\_]) \leq \inf_{k \in \mathbb{N}} \Psi^\beta_{\psi}(E[k])\) and that for all \(k \in \mathbb{N}, (k, k) \in \{0 \vdash \text{nat}\}(\beta)\), so \(\Psi^\beta_{\psi}(E[k]) \leq \Psi^\beta_{\psi}(E'[k])\), and therefore \(\Psi^\beta_{\psi}(E[\_]) \leq \inf_{k \in \mathbb{N}} \Psi^\beta_{\psi}(E'[k]) = \Psi^\beta_{\psi}(E'[\_])\).

- Assume \(\Gamma \vdash e \leq_{\text{log}} e' : \text{nat}\). We will show that \(\Delta \mid \Gamma \vdash \text{rand } e \leq_{\text{log}} \text{rand } e' : \text{nat}\). By assumption, for all \(\xi < \omega_1\) and all \((y, y') \in \{\Delta \vdash \Gamma\}(\xi)\), \((e, e') \in \{\Delta \vdash \text{nat}\}(\xi)\). Now let \((E, E') \in \{\Delta \vdash \text{nat}\}(\xi)\). By lemma A.11,

\(\{E[\text{rand }]\,\psi), E'[\text{rand }]\,\psi\} \in \{\Delta \vdash \text{nat}\}(\xi)\),

and therefore,

\(\Psi^\beta_{\psi}(E[\text{rand }]\,\psi) \leq \Psi^\beta_{\psi}(E'[\text{rand }]\,\psi)\).

so we can conclude

\((\text{rand } e)_y, (\text{rand } e')_y' \in \{\Delta \vdash \text{nat}\}(\xi)\).

- Assume \(\Delta \mid \Gamma \vdash t \leq_{\text{log}} t' : \tau/\mu \alpha.\tau/\alpha\). We will show \(\Delta \mid \Gamma \vdash \text{fold } e \leq_{\text{log}} \text{fold } e' : \mu \alpha.\tau\). Let \(\xi < \omega_1, (y, y') \in \{\Delta \vdash \Gamma\}(\xi)\), \((E, E') \in \{\Delta \vdash \mu \alpha.\tau\}(\phi)^\top(\xi)\). By assumption, \((e, e') \in \{\Delta \vdash \tau(\alpha \mapsto \mu \alpha.\tau/\alpha)\}(\phi)^\top(\xi)\), and by lemma A.15,

\(\{E[\text{fold }]\,\psi), E'[\text{fold }]\,\psi\} \in \{\Delta \vdash \tau(\alpha \mapsto \mu \alpha.\tau/\alpha)\}(\phi)^\top(\xi)\),

so we can conclude

\(\Psi^\beta_{\psi}(E[\text{fold }]\,\psi) \leq \Psi^\beta_{\psi}(E'[\text{fold }]\,\psi)\).

- Assume \(\Delta \mid \Gamma \vdash e \leq_{\text{log}} e' : \mu \alpha.\tau\). We will show \(\Delta \mid \Gamma \vdash \text{unfold } e \leq_{\text{log}} \text{unfold } e' : \tau/\mu \alpha.\tau\). Let \(\xi < \omega_1, (y, y') \in \{\Delta \vdash \Gamma\}, (E, E') \in \{\Delta \vdash \tau(\alpha \mapsto \mu \alpha.\tau)\}(\phi)^\top(\xi)\). By assumption, \((e, e') \in \{\Delta \vdash \mu \alpha.\tau\}(\phi)^\top(\xi)\), and by lemma A.16,

\(\{E[\text{unfold }]\,\psi), E'[\text{unfold }]\,\psi\} \in \{\Delta \vdash \mu \alpha.\tau\}(\phi)^\top(\xi)\),

so we can conclude

\(\Psi^\beta_{\psi}(E[\text{unfold }]\,\psi) \leq \Psi^\beta_{\psi}(E'[\text{unfold }]\,\psi)\).
A.3 Proofs of Section 5

Proof of lemma 5.3. By induction on $n$. If $n = 0$ it is simple to check. Otherwise, $n = l + 1$. We have the following cases:

- If $e \in Val$, it is easy to check.
- If $e \rightarrow_D e'$. Then,
  $$\Xi^0_\theta(\bot, e) = \Xi^0_\theta(\bot, (\theta_t(x, e'), e')) \leq \Phi^l(\bot)(e') = \Phi^0(\bot)(e)$$
- If $e = E[?]$. Then,
  $$\Xi^0_\theta(\bot, E[?]) = \Xi^0_\theta(\bot)(\theta_t(x, E[\theta_o(x)], E[\theta_o(x)]))$$
  $$\leq \Phi^l(\bot)(E[\theta_o(x)])$$
  $$\leq \sup_{m \in \mathbb{N}} \Phi^l(\bot)(E[m])$$
  $$= \Phi^0(\bot)(E[?])$$
- If $e = E[\text{rand } k]$. Then,
  $$\Xi^0_\theta(\bot, E[\text{rand } k]) = \sum_{1 \leq m \leq k} \frac{1}{k} \cdot \Xi^0_\theta(\bot, \theta_t(x, E[m]), E[m])$$
  $$\leq \sum_{1 \leq m \leq k} \frac{1}{k} \cdot \Phi^l(\bot)(E[m])$$
  $$= \Phi^0(E[\text{rand } k])$$

Proof of lemma 5.4. By induction on $n$. If $n = 0$, we can consider as scheduler the trivial scheduler with a single state and output 0, that is $\theta_0 = (\{\star\}, \lambda x.0, \lambda x.\lambda t.\star)$. We check that indeed $\Xi^0_\theta(\bot, e) = 0 \geq 0 - \epsilon = \Phi^0(\bot)(e) - \epsilon$.

If $n = l + 1$, we have the following cases:

- If $e \in Val$, we can take the trivial scheduler as before, and $\Xi^0_e(\bot, e) = 1 \geq 1 - \epsilon = \Phi^0(\bot)(e) - \epsilon$.
- If $e \rightarrow_D e'$. By I.H. there exists a scheduler $\theta = (X, \theta_o, \theta_t)$ and $x \in X$ such that $\Xi^l_\theta(\bot, e') \geq \Phi^l(\bot)(e') - \epsilon$. We construct a new scheduler $\theta'$ with state space $X \cup \{\star\}$ (assuming wlog $\star \notin X$); output function $\theta'_o$ such that $\theta'_o(\star) = 0$, and coincides with $\theta_0$ everywhere else; and transition function $\theta'_t$ such that $\theta'_t(\star) = \lambda_x.0$, and $\theta'_t(y) = \theta(y)$ for all $y \in X$. Then we have
  $$\Xi^{l+1}_\theta(\bot, e) = \Xi^l_\theta(\bot, (\theta_t(x, e'), e'))$$
  $$= \Xi^l_\theta(\bot, (x, e'))$$
  $$\geq \Phi^l(\bot)(e') - \epsilon$$
  $$= \Phi^{l+1}(\bot)(e)$$
- If $e = E[?]$. By the definition of least upper bound, there exists $m$ such that
  $$\Phi^{l+1}(\bot)(E[?]) - \epsilon / 2 = \sup_{k \in \mathbb{N}} \Phi^l(\bot)(E[k]) - \epsilon / 2 \leq \Phi^l(\bot)(E[m])$$
  By I.H. there exists a scheduler $\theta = (X, \theta_o, \theta_t)$ and $x \in X$ such that $\Xi^l_\theta(\bot, E[m]) \geq \Phi^l(\bot)(E[m]) - \epsilon / 2$. We construct a new scheduler $\theta'$ with state space $X \cup \{\star\}$ (assuming wlog $\star \notin X$); output function $\theta'_o$ such that $\theta'_o(\star) = m$, and coincides with $\theta_0$ everywhere else; and transition function $\theta'_t$ such that $\theta'_t(\star) = \lambda_x.0$, and $\theta'_t(y) = \theta(y)$ for all $y \in X$. Then we have
  $$\Xi^{l+1}_\theta(\bot, E[?]) = \Xi^l_\theta(\bot, (x, E[m]))$$
  $$= \Xi^l_\theta(\bot, (x, E[m]))$$
  $$\geq \Phi^l(\bot)(E[m]) - \epsilon / 2$$
  $$\geq (\Phi^{l+1}(\bot)(E[?]) - \epsilon / 2) - \epsilon / 2$$
  $$= \Phi^{l+1}(\bot)(E[?]) - \epsilon$$
Therefore, recall that proof of theorem 5.5.

Proof of lemma 5.6. By induction on $\alpha$.

- If $e \in \text{Val}$, it is easy to check.
- If $e \rightarrow_D e'$, then,
  \[
  \Xi_0^\alpha(\bot, e) = \Xi_0^\alpha(\bot, \theta_t(x, e'), e') \geq \Psi^\beta(\bot)(e') = \Psi^\alpha(\bot)(e)
  \]
- If $e = [?]$. Then,
  \[
  \Xi_0^\alpha(\bot)(x, [?]) = \Xi_0^\alpha(\bot, \theta_t(x, E[\theta_0(x)])), E[\theta_0(x)])
  \]
  \[
  \geq \Phi^\beta(\bot)(E[\theta_0(x)])
  \]
  \[
  \geq \inf_{m \in \mathbb{N}} \Phi^\beta(\bot)(E[m])
  \]
  \[
  = \Phi^\alpha(\bot)([?])
  \]
- If $e = E[\text{rand}]$. Then,
  \[
  \Xi_0^\alpha(\bot)(x, E[\text{rand}]) = \sum_{1 \leq m \leq k} \frac{1}{k} \cdot \Xi_0^\alpha(\bot, \theta_t(x, E[m]), E[m])
  \]
  \[
  \leq \sum_{1 \leq m \leq k} \frac{1}{k} \cdot \Phi^\beta(\bot)(E[m])
  \]
  \[
  = \Phi^\alpha(E[\text{rand}])
  \]

Finally, if $\alpha$ is a limit ordinal, and for every $\beta < \alpha$ we have $\Xi_0^\beta(\bot)(x, e) \geq \Psi^\beta(\bot)(e)$, then

\[
\Xi_0^\alpha(\bot)(x, e) = \sup_{\beta < \alpha} \Xi_0^\beta(\bot)(x, e) \geq \sup_{\beta < \alpha} \Psi^\beta(\bot)(e) \geq \Psi^\alpha(\bot)(e)
\]
Proof of lemma 5.7. We have to show that
\[
\forall e. \forall n. \forall \epsilon > 0. \exists \theta. \exists x \in X_\theta. \sup_{n \in \omega} \Xi^n_\theta(\bot)(x, e) \leq \sup_{\alpha \in \omega_1} \Psi^\alpha(\bot)(e) + \epsilon
\]

We will prove something stronger, namely that there exists a concrete scheduler \( \theta \) such that
\[
\forall e. \forall n. \forall \epsilon > 0. \exists x \in X_\theta. \sup_{n \in \omega} \Xi^n_\theta(\bot)(x, e) \leq \sup_{\alpha \in \omega_1} \Psi^\alpha(\bot)(e) + \epsilon
\] (1)

This scheduler is described below:
- Its (countable) space state \( X_\theta \) corresponds to finite non-empty traces of \( Expr \) (for convenience, we denote them backwards, and we use \( e :: \pi \) to denote appending \( e \) in front of \( \pi \))
- The transition function \( \theta_t(\pi, e) = e :: \pi \) corresponds to appending \( e \) to the trace.
- For any trace \( \pi \) starting in an expression of the form \( e \) where \( e \rightarrow_\Pi e' \), or an expression of the form \( E[\text{rand } m] \), the output function \( \theta_o \) returns 0.
- For any trace \( \pi \) of length \( |\pi| \) starting in an expression of the form \( E[?] \), the output function \( \theta_o \) returns \( m \), where \( m \) is chosen such that
  \[
  \Psi^\bot(E[m]) \leq \inf_{k \in \mathbb{N}} \Psi^\bot(E[k]) + 2^{-|\pi|} \cdot \epsilon
  \]
  Note that such \( m \) must exist by the definition of greatest lower bound.

We now show by induction that, for every \( n \in \mathbb{N} \), every \( e \in Expr \) and every trace \( \pi \in Expr \times Expr^* \) starting in \( e \)
\[
\Xi^n_\theta(\bot)(\pi, e) \leq \Psi^\bot(e) + \left( \frac{2^n - 1}{2^{n+|\pi| - 1}} \right) \cdot \epsilon
\]
By then choosing \( \pi = (e :: []) \) and taking limits when \( n \rightarrow \infty \), the factor of the \( \epsilon \) converges to 1 and we obtain (1).

If \( n = 0 \), either both sides of the equation are 1 (when \( e \in Val \)), or the left-hand side is 0. Now we assume the result holds for some arbitrary \( n \) and prove it for \( n + 1 \). Then we have the following cases:
- If \( e \in Val \), then the left-hand side of the equation is 1, and the second is strictly larger than 1.
- If \( e \rightarrow e' \), then
  \[
  \Xi^{n+1}_\theta(\bot)(\pi, e) = \Xi^n_\theta(\bot)((e' :: \pi), e') \leq \Psi^\bot(e') + \left( \frac{2^n - 1}{2^{n+|\pi| - 1}} \right) \cdot \epsilon = \Psi^\bot(e) + \left( \frac{2^n - 1}{2^{n+|\pi| - 1}} \right) \cdot \epsilon \leq \Psi^\bot(e) + \left( \frac{2^{n+1} - 1}{2^{n+|\pi| - 1}} \right) \cdot \epsilon
  \]
- If \( e = E[?] \), then, for \( m = \theta_o(\pi) \)
  \[
  \Xi^{n+1}_\theta(\bot)(\pi, e) = \Xi^n_\theta(\bot)((E[m] :: \pi), E[m]) \leq \Psi^\bot(E[m]) + \left( \frac{2^n - 1}{2^{n+|\pi| - 1}} \right) \cdot \epsilon \leq \inf_{k \in \mathbb{N}} \Psi^\bot(E[k]) + \left( \frac{2^n - 1}{2^{n+|\pi| - 1}} \right) \cdot \epsilon + 2^{-|\pi|} \cdot \epsilon = \Psi^\bot(E[?]) + \left( \frac{2^n - 1}{2^{n+|\pi| - 1}} \right) \cdot \epsilon + 2^{-|\pi|} \cdot \epsilon = \Psi^\bot(E[?]) + \left( \frac{2^{n+1} - 1}{2^{n+|\pi| - 1}} \right) \cdot \epsilon
  \]
• If \( e = E[\text{rand }] \), then,

\[
\Xi_{\pi}^{n+1}(\bot)(\pi, e) = \sum_{1 \leq k \leq m} \frac{1}{m} \cdot \Xi_{\pi}^{n}(\bot)((E[k] :: \pi), E[k]) \\
\leq \sum_{1 \leq k \leq m} \frac{1}{m} \cdot \Xi_{\pi}^{1}(E[k]) + \left( \frac{2^n - 1}{2^{n+1+|\pi|-1}} \right) \cdot \varepsilon \\
= \Xi_{\pi}^{1}(E[\text{rand } m]) + \left( \frac{2^n - 1}{2^{n+1+|\pi|-1}} \right) \cdot \varepsilon \\
\leq \Xi_{\pi}^{1}(E[\text{rand } m]) + \left( \frac{2^{n+1} - 1}{2^{n+1+|\pi|-1}} \right) \cdot \varepsilon
\]

\[\square\]

### A.4 Proofs of Section 6

**Proof of proposition 6.5.** We first prove by induction that for every \( n \), \( \Xi_{\pi}^{n}(e) \) is left-computable. The case of 0 is trivial. For the inductive case we have:

- If \( e \) is a value or stuck, the result is trivial.
- If \( e \rightarrow_{D} \bar{e} \), then \( \Xi_{\pi}^{n+1}(e) = \Xi_{\pi}^{n}(\bar{e}) \), so we apply I.H. directly.
- If \( e = E[\text{rand } k] \). Then \( \Xi_{\pi}^{n+1}(E[\text{rand } k]) = \sum_{1 \leq k \leq m} (1/k) \cdot \Xi_{\pi}^{1}(E[j]) \). By I.H. every one of the \( \Xi_{\pi}^{1}(E[j]) \) is left-computable, and since left-computable reals are closed under addition and multiplication by a nonnegative rational, \( \Xi_{\pi}^{n+1}(E[\text{rand } k]) \) is also left-computable.
- If \( e = E[?] \). Then \( \Xi_{\pi}^{n+1}(E[?]) = \sup_{i \in \mathbb{N}} \Xi_{\pi}^{1}(E[j]) \). By I.H., there exists for every \( i \) a computable nondecreasing sequence \( \varphi_i : \mathbb{N} \rightarrow \mathbb{Q} \) such that \( \sup_{j \in \mathbb{N}} \varphi_i(j) = \Xi_{\pi}^{1}(E[j]) \). Then we can define \( \varphi(m) = \sup_{i \leq m} \varphi_i(j) \) and observe that \( \varphi(m) \) is computable, nondecreasing, and that

\[
\sup_{m \in \mathbb{N}} \varphi(m) = \sup_{i,j \in \mathbb{N}} \varphi_i(j) = \sup_{i \in \mathbb{N}} \Xi_{\pi}^{1}(E[j]),
\]

so \( \Xi_{\pi}^{n+1}(E[?]) \) is left-computable

To conclude we argue as in the \( e = E[?] \) case, and obtain that \( \Xi_{\pi}^{1}(e) = \sup_{n \in \mathbb{N}} \Xi_{\pi}^{n}(e) \) is left-computable. \[\square\]

**Proof of proposition 6.7.** Let \( E \in \text{Ectx} \). We show the proof for may-equivalence, the one for must-equivalence being analogous. We have to show the following cases:

- \( e_1 \) or \( e_1 \leq_{\text{ctx}}^{\leq} e_1 \). Then

\[
\Xi_{\pi}^{1}(E[e_1 \text{ or } e_1]) = \Xi_{\pi}^{1}(E[\text{if } ? > 0 \text{ then } e_1 \text{ else } e_1]) \\
= \sup_{m \in \mathbb{N}} \Xi_{\pi}^{1}(E[\text{if } m > 0 \text{ then } e_1 \text{ else } e_1]) \\
= \max(\Xi_{\pi}^{1}(E[e_1])) \\
= \Xi_{\pi}^{1}(E[e_1])
\]

- \( e_1 \) or \( e_2 \leq_{\text{ctx}}^{\leq} e_2 \) or \( e_1 \). Then

\[
\Xi_{\pi}^{1}(E[e_1 \text{ or } e_2]) = \Xi_{\pi}^{1}(E[\text{if } ? > 0 \text{ then } e_1 \text{ else } e_2]) \\
= \sup_{m \in \mathbb{N}} \Xi_{\pi}^{1}(E[\text{if } m > 0 \text{ then } e_1 \text{ else } e_2]) \\
= \max(\Xi_{\pi}^{1}(E[e_1]), \Xi_{\pi}^{1}(E[e_2]))
\]
and
\[ \mathcal{P}^\dagger(E[e_2 \text{ or } e_1]) = \mathcal{P}^\dagger(E[\text{if } ? > 0 \text{ then } e_2 \text{ else } e_1]) \]
\[ = \sup_{m \in \mathbb{N}} \mathcal{P}^\dagger(E[\text{if } m > 0 \text{ then } e_2 \text{ else } e_1]) \]
\[ = \max(\mathcal{P}^\dagger(E[e_2]), \mathcal{P}^\dagger(E[e_1])) \]

• \( e_1 \) or \((e_2 \text{ or } e_3) \leq^c (e_1 \text{ or } e_2) \) or \( e_1 \). Then
\[ \mathcal{P}^\dagger(E[e_1 \text{ or } (e_2 \text{ or } e_3)]) = \mathcal{P}^\dagger(E[\text{if } ? > 0 \text{ then } e_1 \text{ else } (if ? > 0 \text{ then } e_2 \text{ else } e_3)]) \]
\[ = \sup_{m \in \mathbb{N}} \mathcal{P}^\dagger(E[\text{if } m > 0 \text{ then } e_1 \text{ else } (if ? > 0 \text{ then } e_2 \text{ else } e_3)]) \]
\[ = \max(\mathcal{P}^\dagger(E[e_1]), \mathcal{P}^\dagger(E[if ? > 0 \text{ then } e_2 \text{ else } e_3])) \]
\[ = \max(\mathcal{P}^\dagger(E[e_1]), \mathcal{P}^\dagger(E[e_2]), \mathcal{P}^\dagger(E[e_3])) \]

Analogously, we can show
\[ \mathcal{P}^\dagger(E[(e_1 \text{ or } e_2) \text{ or } e_3]) = \mathcal{P}^\dagger(E[\text{if } ? > 0 \text{ then } (if ? > 0 \text{ then } e_1 \text{ else } e_2) \text{ else } e_3]) \]
\[ = \max(\mathcal{P}^\dagger(E[e_1]), \mathcal{P}^\dagger(E[e_2]), \mathcal{P}^\dagger(E[e_3])) \]

\[ \square \]

**Proof of proposition 6.8.** Let \( E \in \text{Ectx} \). Assume also \( p = p_1/p_2 \) and \( q = q_1/q_2 \) where \( p_1 \leq p_2, q_1 \leq q_2 \) are naturals, \( p_2 \neq 0 \) and \( q_2 \neq 0 \). We have to show the following cases:

• \( e_1 \oplus_p e_1 \leq^c e_1 \). Then
\[ \mathcal{P}^\dagger(E[e_1 \oplus_p e_1]) = \mathcal{P}^\dagger(E[\text{if rand } p_2 \leq p_1 \text{ then } e_1 \text{ else } e_1]) \]
\[ = \sum_{1 \leq m \leq p_2} \mathcal{P}^\dagger(E[\text{if } m \leq p_1 \text{ then } e_1 \text{ else } e_1]) \]
\[ = \frac{p_1}{p_2} \cdot \mathcal{P}^\dagger(E[e_1]) + \frac{p_2 - p_1}{p_2} \cdot \mathcal{P}^\dagger(E[e_1]) \]
\[ = \mathcal{P}^\dagger(E[e_1]) \]

• \( e_1 \oplus_p e_2 \leq^c e_2 \oplus_{1-(p_1/p_2)} e_1 \). Then
\[ \mathcal{P}^\dagger(E[e_1 \oplus_p e_2]) = \mathcal{P}^\dagger(E[\text{if rand } p_2 \leq p_1 \text{ then } e_1 \text{ else } e_2]) \]
\[ = \sum_{1 \leq m \leq p_2} \mathcal{P}^\dagger(E[\text{if } m \leq p_1 \text{ then } e_1 \text{ else } e_2]) \]
\[ = \frac{p_1}{p_2} \cdot \mathcal{P}^\dagger(E[e_1]) + \frac{p_2 - p_1}{p_2} \cdot \mathcal{P}^\dagger(E[e_2]) \]

and
\[ \mathcal{P}^\dagger(E[e_2 \oplus_{1-(p_1/p_2)} e_1]) = \mathcal{P}^\dagger(E[e_2 \oplus (p_2-p_1)/p_2 \text{ e}_1]) \]
\[ = \mathcal{P}^\dagger(E[\text{if rand } p_2 \leq (p_2 - p_1) \text{ then } e_2 \text{ else } e_1]) \]
\[ = \sum_{1 \leq m \leq p_2} \mathcal{P}^\dagger(E[\text{if } m \leq (p_2 - p_1) \text{ then } e_2 \text{ else } e_1]) \]
\[ = \frac{p_2 - p_1}{p_2} \cdot \mathcal{P}^\dagger(E[e_2]) + \frac{p_1}{p_2} \cdot \mathcal{P}^\dagger(E[e_1]) \]
\begin{itemize}
  \item \((e_1 \oplus_{p_1/p_2} e_2) \oplus_{q_1/q_2} e_3 \subseteq^{\forall \downarrow} e_1 \oplus_{(p_1/q_1)/(p_2/q_2)} (e_2 \oplus_{(q-pq)/(1-pq)} e_3)\). Then
  \[
  \mathbb{P}^\downarrow(E[(e_1 \oplus_{p_1/p_2} e_2) \oplus_{q_1/q_2} e_3]) = \mathbb{P}^\downarrow(E[\text{if rand } q_2 \leq q_1 \text{ then (if rand } p_2 \leq p_1 \text{ then } e_1 \text{ else } e_2 \text{) else } e_3])
  \]
  \[
  = \sum_{1 \leq m \leq q_2} \mathbb{P}^\downarrow(E[\text{if } q_2 \leq q_1 \text{ then (if rand } p_2 \leq p_1 \text{ then } e_1 \text{ else } e_2 \text{) else } e_3])
  \]
  \[
  = \frac{q_1}{q_2} \cdot \mathbb{P}^\downarrow(E[\text{if rand } p_2 \leq p_1 \text{ then } e_1 \text{ else } e_2]) + \frac{q_2 - q_1}{q_2} \cdot \mathbb{P}^\downarrow(E[e_3])
  \]
  \[
  = \frac{q_1}{q_2} \left( \sum_{k \in \mathbb{N}} \mathbb{P}^\downarrow(E[\text{if } k \leq p_1 \text{ then } e_1 \text{ else } e_2]) \right) + \frac{q_2 - q_1}{q_2} \cdot \mathbb{P}^\downarrow(E[e_3])
  \]
  \[
  = \frac{p_1 q_1}{p_2 q_2} \mathbb{P}^\downarrow(E[e_1]) + \frac{p_1 (p_2 - p_1)}{p_1 p_2} \mathbb{P}^\downarrow(E[e_2]) + \frac{q_2 - q_1}{q_2} \cdot \mathbb{P}^\downarrow(E[e_3])
  \]
\end{itemize}

Analogously, we can show
\[
\mathbb{P}^\downarrow(E[e_1 \oplus_{(p_1/q_1)/(p_2/q_2)} (e_2 \oplus_{(q-pq)/(1-pq)} e_3)])
\]
\[
= \frac{p_1 q_1}{p_2 q_2} \mathbb{P}^\downarrow(E[e_1]) + \frac{p_1 (p_2 - p_1)}{p_1 p_2} \mathbb{P}^\downarrow(E[e_2]) + \frac{q_2 - q_1}{q_2} \cdot \mathbb{P}^\downarrow(E[e_3])
\]

\[\square\]

\textbf{Proof of proposition 6.9.} To simplify the notation, we assume \(p = 1/2\), but the proof is analogous for any other rational in \([0, 1]\). Consider contexts \((E, E') \in \llbracket0 \rightarrow \rrbracket^\mathbb{N}\). By expanding the definitions we get:

\[
e_1 \oplus (e_2 \text{ or } e_3) \triangleq \text{if (rand } 2 \leq 1 \text{ then } e_1 \text{ else (if } ? > 0 \text{ then } e_2 \text{ else } e_3) =
\]
\[
(e_1 \text{ or } e_2) \oplus (e_1 \text{ or } e_3) \triangleq \text{if } ? > 0 \text{ then (if (rand } 2 \leq 1 \text{ then } e_1 \text{ else } e_2 \text{) else (if (rand } 2 \leq 1 \text{ then } e_1 \text{ else } e_3),
\]

which amount to a commuting conversion. That is, consider the contexts
\[
C_P \triangleq \text{if (rand } 2 \leq 1 \text{ then } e_1 \text{ else [ ]}
\]

and
\[
C_N \triangleq \text{if } ? > 0 \text{ then } e_1 \text{ else [ ]}
\]

Then have to show:

\[
C_P[\text{if } ? > 0 \text{ then } e_2 \text{ else } e_3] \equiv^{\forall \downarrow} \text{if } ? > 0 \text{ then } C_P[e_2] \text{ else } C_P[e_3]
\]

We have:

\[
\mathbb{P}^\downarrow(E[C_P[\text{if } ? > 0 \text{ then } e_2 \text{ else } e_3])
\]
\[
= (1/2) \cdot \mathbb{P}^\downarrow(E[e_1]) + (1/2) \cdot \mathbb{P}^\downarrow(E[\text{if } ? > 0 \text{ then } e_2 \text{ else } e_3])
\]
\[
= (1/2) \cdot \mathbb{P}^\downarrow(E[e_1]) + (1/2) \cdot \sup_{k \in \mathbb{N}} \mathbb{P}^\downarrow(E[\text{if } k > 0 \text{ then } e_2 \text{ else } e_3])
\]
\[
= (1/2) \cdot \mathbb{P}^\downarrow(E[e_1]) + (1/2) \cdot \max(\mathbb{P}^\downarrow(E[e_2]), \mathbb{P}^\downarrow(E[e_3]))
\]

and

\[
\mathbb{P}^\downarrow(E'[\text{if } ? > 0 \text{ then } C_P[e_2] \text{ else } C_P[e_3]])
\]
\[
= \sup_{k \in \mathbb{N}} \mathbb{P}^\downarrow(E'[\text{if } k > 0 \text{ then } C_P[e_2] \text{ else } C_P[e_3]])
\]
\[
= \max(\mathbb{P}^\downarrow(E'[C_P[e_2]]), \mathbb{P}^\downarrow(E'[C_P[e_3]]))
\]
\[
= \max(\{1/2) \cdot \mathbb{P}^\downarrow(E'[e_1]), (1/2) \cdot \mathbb{P}^\downarrow(E'[e_2]), (1/2) \cdot \mathbb{P}^\downarrow(E'[e_3])\}
\]
\[
= (1/2) \cdot \mathbb{P}^\downarrow(E'[e_1]) + (1/2) \cdot \max(\mathbb{P}^\downarrow(E'[e_2]), \mathbb{P}^\downarrow(E'[e_3]))
\]

By the fundamental lemma, \(e_1 \equiv^{\forall \downarrow} e_1 \text{, } e_2 \equiv^{\forall \downarrow} e_2 \text{ and } e_3 \equiv^{\forall \downarrow} e_3\), therefore, \(\mathbb{P}^\downarrow(E[e_1]) = \mathbb{P}^\downarrow(E'[e_1]), \mathbb{P}^\downarrow(E[e_2]) = \mathbb{P}^\downarrow(E'[e_2])\) and \(\mathbb{P}^\downarrow(E[e_3]) = \mathbb{P}^\downarrow(E'[e_3])\). This concludes the proof. \(\square\)
Proof of proposition 6.11. By deterministic reduction, we can show
\[(\text{fix }_\_ G) \trianglerighteqctx 1 G (\lambda z. e_G (\text{fold } e_G) z) \trianglerighteqctx 1 v \]  
Since $G$ is deterministic, there exists $w \in \text{Val}(\sigma \rightarrow \tau)$ such that $G (\lambda x. e_G (\text{fold } e_G) x) \trianglerighteqctx 1 w$, and therefore,
\[(\text{fix }_\_ G) \trianglerighteqctx 1 w \trianglerighteqctx 1 v \]  
On the other hand by deterministic reduction we have
\[(\text{fix }_\_ G) \trianglerighteqctx 1 G (\lambda z. e_G (\text{fold } e_G) z) \]  
so by transitivity and compatibility
\[(\lambda z. e_G (\text{fold } e_G) z) \trianglerighteqctx 1 w \trianglerighteqctx 1 v \]  
Now we can apply extensionality (proposition 6.1)
\[(\lambda z. e_G (\text{fold } e_G) z) \trianglerighteqctx 1 G (\lambda z. e_G (\text{fold } e_G) z) \]  
and again by compatibility and transitivity
\[(\text{fix }_\_ G) \trianglerighteqctx 1 G (\text{fix }_\_ G) \]  
\[\square\]

Proof of lemma 6.12. By induction on the length of $r$. If $r = \text{nil}$, then
\[\text{split } l \times \text{nil} \rightarrow (l, x, \text{nil}) \]  
so
\[\text{let } (y_s, z, z_s) = \text{split } l \times \text{nil} \text{ in } y_s + z :: z_s \trianglerighteqctx 0 l + x :: \text{nil} \]
Otherwise, $r = w :: w_s$, and
\[\text{split } l \times w :: w_s \rightarrow (l, x, w :: w_s) \text{ or } (\text{split } (l + x :: \text{nil}) w w_s) \]  
We have that
\[\text{let } (y_s, z, z_s) = (l, x, w :: w_s \text{ in } y_s + z :: z_s \trianglerighteqctx 0 l + x :: w :: w_s) \]  
and by inductive hypothesis,
\[\text{let } (y_s, z, z_s) = (\text{split } (l + x :: \text{nil}) w w_s) \text{ in } y_s + z :: z_s \trianglerighteqctx 0 (l + x :: \text{nil}) + w :: w_s \]
\[\trianglerighteqctx 0 (x :: \text{nil} + w :: w_s) \]
\[\trianglerighteqctx 0 (x :: (\text{nil} + w :: w_s)) \]
\[\trianglerighteqctx 0 (x :: w :: w_s) \]
So in summary,
\[\text{split } l \times w :: w_s \trianglerighteqctx 0 l + x :: w :: w_s \text{ or } l + x :: w :: w_s \trianglerighteqctx 0 l + x :: w :: w_s \]
\[\square\]

If $r = \text{nil}$, we just use the fact that
\[\text{sort}(y_s + (z :: z_s)) \trianglerighteqctx 0 \text{sort}(z :: (y_s + z_s)) \]
Otherwise, $l = x :: xs$. Let $E \in Ectx([\text{nat}])$. Then
\[
\Psi^\parallel(E[\text{skiplist}' x :: xs (ys ++ (z :: zs))])
= \Psi^\parallel(E[\text{skiplist}' xs (ys ++ (z :: zs))]
\oplus \text{skiplist}' xs x :: (ys ++ (z :: zs)))
= (1/2) \cdot \Psi^\parallel(E[\text{skiplist}' xs (ys ++ (z :: zs))])
+ (1/2) \cdot \Psi^\parallel(E[\text{skiplist}' xs x :: (ys ++ (z :: zs))])
= (1/2) \cdot \Psi^\parallel(E[\text{skiplist}' xs z :: (ys ++ zs)])
+ (1/2) \cdot \Psi^\parallel(E[\text{skiplist}' xs x :: (z :: ys ++ zs)])
\]
Similarly, we can show
\[
\Psi^\parallel(E[\text{skiplist}' x :: xs (z :: (ys ++ zs))])
= (1/2) \cdot \Psi^\parallel(E[\text{skiplist}' xs z :: (ys ++ zs)])
+ (1/2) \cdot \Psi^\parallel(E[\text{skiplist}' xs x :: (z :: ys ++ zs)])
\]
\[
\square
\]

\textit{Proof of lemma 6.14.} This is a direct consequence of lemma 6.13 and the definition of \text{skiplist}'. Let $E \in Ectx([\text{nat}])$. Then
\[
\Psi^\parallel(E[\text{skiplist}' x :: y :: l tl])
= \Psi^\parallel(E[\text{skiplist}' y :: l x :: tl \oplus \text{skiplist}' y :: l tl])
= \Psi^\parallel(E[(\text{skiplist}' l x :: tl \oplus \text{skiplist}' l y :: x :: tl)
\oplus (\text{skiplist}' l tl \oplus \text{skiplist}' l y :: tl)])
= \Psi^\parallel(E[(\text{skiplist}' l x :: tl \oplus \text{skiplist}' l x :: y :: tl)
\oplus (\text{skiplist}' l tl \oplus \text{skiplist}' l y :: tl)])
\]
\[
\Psi^\parallel(E[\text{skiplist}' y :: x :: l tl])
\]
\[
\square
\]

\textit{Proof of lemma 6.15.} By induction on the length of $l$. If it is empty, then both sides reduce to
\[
\text{skiplist}' (\text{cons}(x, r)) tl
\]
Otherwise,
\[
\text{skiplist}' (\text{cons}(z, zs) ++ \text{cons}(x, r)) tl
\]
\[
\equiv^\text{stx} \text{skiplist}' (\text{cons}(z, zs ++ \text{cons}(x, r)) tl)
\]
\[
\text{(Def.)} \equiv^\text{stx} \text{let } tl' = tl \oplus \text{cons}(z, tl) \text{ in}
\text{skiplist}' (z :: cons(x, r)) tl'
\]
\[
\text{(I.H.)} \equiv^\text{stx} \text{let } tl' = tl \oplus \text{cons}(z, tl) \text{ in}
\text{skiplist}' (\text{cons}(x, zs :: r)) tl'
\]
\[
\text{(I.H.)} \equiv^\text{stx} \text{skiplist}' (\text{cons}(z, \text{cons}(x, zs :: r))) tl
\]
\[
(\text{lemma 6.14}) \equiv^\text{stx} \text{skiplist}' (\text{cons}(x, \text{cons}(z, zs :: r))) tl
\]
\[
\equiv^\text{stx} \text{skiplist}' (\text{cons}(x, \text{cons}(z, zs :: r))) tl
\]
\[
\square