Modelling Recursion and Probabilistic Choice in Guarded Type Theory

PHILIPP STASSEN, Aarhus University, Denmark
RASMUS EJLERS MØGELBERG, IT University of Copenhagen, Denmark
MAAIKE ANNEBET ZWART, IT University of Copenhagen, Denmark
ALEJANDRO AGUIRRE, Aarhus University, Denmark
LARS BIRKEDAL, Aarhus University, Denmark

Constructive type theory combines logic and programming in one language. This is useful both for reasoning about programs written in type theory, as well as for reasoning about other programming languages inside type theory. It is well-known that it is challenging to extend these applications to languages with recursion and computational effects such as probabilistic choice, because these features are not easily represented in constructive type theory.

We show how to define and reason about $\mathsf{FPC}_{\oplus}$, a programming language with probabilistic choice and recursive types, in guarded type theory. We use higher inductive types to represent finite distributions and guarded recursion to model recursion. We define both operational and denotational semantics of $\mathsf{FPC}_{\oplus}$, as well as a relation between the two. The relation can be used to prove adequacy, but we also show how to use it to reason about programs up to contextual equivalence.

CCS Concepts: • **Theory of computation → Type theory**; **Denotational semantics**; **Operational semantics**; **Probabilistic computation**.

Additional Key Words and Phrases: Probabilistic Programming Languages, Type Theory, Guarded Recursion, Recursive Types

ACM Reference Format:
Philipp Stassen, Rasmus Ejlers Møgelberg, Maaike Annebet Zwart, Alejandro Aguirre, and Lars Birkedal. 2025. Modelling Recursion and Probabilistic Choice in Guarded Type Theory. *Proc. ACM Program. Lang.* 9, POPL, Article 48 (January 2025), 29 pages. https://doi.org/10.1145/3704884

1 Introduction

Traditionally, modelling and reasoning about programming languages is done using either operational or denotational techniques. Denotational semantics provides mathematical abstractions that are used to see beyond the details of the operational implementation, to describe principles common to many different languages, and to provide modular building blocks that can be extended to settings with more language features. On the other hand, denotational semantics can easily become very sophisticated mathematically, and modelling combinations of recursion and other features such as probabilistic sampling and higher-order store can be difficult. Operational techniques are often

Authors' Contact Information: Philipp Stassen, Aarhus University, Aarhus, Denmark, stassen@cs.au.dk; Rasmus Ejlers Møgelberg, IT University of Copenhagen, Copenhagen, Denmark, mogel@itu.dk; Maaike Annebet Zwart, IT University of Copenhagen, Copenhagen, Denmark, mazw@itu.dk; Alejandro Aguirre, Aarhus University, Aarhus, Denmark, alejandro@cs.au.dk; Lars Birkedal, Aarhus University, Aarhus, Denmark, birkedal@cs.au.dk.

This work is licensed under a Creative Commons Attribution 4.0 International License.
© 2025 Copyright held by the owner/author(s).
ACM 2475-1421/2025/1-ART48
https://doi.org/10.1145/3704884

more direct and flexible, and are usually easier to implement in proof assistants, but reasoning requires constructing complex operational models and logics.

Synthetic guarded domain theory (SGDT) provides an alternative approach to both denotational and operational semantics. The idea is to work in an expressive meta-language with guarded recursion in the sense of Nakano [32], and to use recursion on the meta-level to model recursion on the object level. The specific meta-language used in this paper is Clocked Cubical Type Theory (CCTT) [26]. It includes a modal type-operator $\triangleright^\kappa$, indexed by a so-called clock κ (see Section 2), to describe data that is available one time step from now. It is possible to define elements by guarded recursion by means of a fixed point combinator $\mathsf{fix}^\kappa : (\triangleright^\kappa A \to A) \to A$. By applying the fixed point combinator to an operator on a universe one can obtain solutions to guarded recursive type equations [8]. For example, one can define a *guarded delay monad* L^κ, which maps a type A to $L^\kappa A$ satisfying $L^\kappa A \simeq A + \triangleright^\kappa (L^\kappa A)$. This guarded delay monad has been used to model recursive computations, both in operational semantics, as well as in denotational semantics, which simply models function types of the object language using the Kleisli functions of the metalanguage: $[\![A \to B]\!] \triangleq [\![A]\!] \to L^\kappa [\![B]\!]$. Guarded recursion is then also used for reasoning about the model, in particular to establish a relation between syntax and semantics to prove adequacy. These results were originally proved for the simply typed lambda calculus extended with recursive terms [33] and recursive types [29], but have been recently extended to cover also languages with general store and polymorphism [36].

SGDT has several of the benefits listed for operational and denotational techniques above: the models can be described at a high level of abstraction, and once one is familiar with programming in the meta-language, the mathematical knowledge required for constructing the models is limited. The main reason for both of these is that recursion and other tools needed for constructing and reasoning about these models, are build into the meta-language, so the mathematical difficulties appear on the next meta-level. Moreover, since metalanguages such as CCTT are dependent type theories, these results can potentially be directly implemented in proof assistants. For example, CCTT has been implemented[1] as an experimental extension of Cubical Agda [41]. Finally, the denotational semantics of the language can be seen as a shallow embedding of the object language, so one can potentially also use the meta-language as a language for both programming and reasoning.

In this paper we show how to extend SGDT to model probabilistic programming languages, i.e., languages that include commands that generate random values by sampling from a probability distribution. It is well known that it is challenging to develop semantic models for reasoning about higher-order probabilistic programming languages that include recursion, even in a classical meta-theory. Nonetheless, a plethora of denotational approaches have been investigated in recent years, [15, 16, 21, 24, 40]. Other operational-based approaches to reason about probabilistic programs have also been shown to scale to rich languages with a variety of features, using techniques such as logical relations [9, 14, 23, 43, 45], or bisimulations [13, 27].

Precisely, we show how to develop operational and denotational semantics of $\mathsf{FPC}_\oplus$[2], a call-by-value higher-order probabilistic programming language with recursive types, in CCTT, and prove that the denotational semantics is adequate with respect to the operational semantics. We also show how to use these results for reasoning about $\mathsf{FPC}_\oplus$ programs in CCTT. One of the challenges for doing so is that most of the previously developed theory for probabilistic languages relies on classical reasoning, which is not available in CCTT. For example, many types used in our model do not have decidable equality, and as a consequence, even the finite distribution monad $\mathcal{D}$ cannot be defined in the standard classical way. Fortunately, CCTT not only models guarded recursion but

[1]https://github.com/agda/guarded

[2]The name derives from Plotkin's Fixed Point Calculus (FPC) [17, 35]

also higher-inductive types (HITs), which we use to define $\mathcal{D}$: on a set A, $\mathcal{D}A$ is the free convex algebra [22], that is, a set $\mathcal{D}A$ together with a binary operation $\oplus_p$, indexed by a rational number p, satisfying the natural equational theory (idempotency, associativity, and commutativity). Using a HIT to represent distributions allows us to use the equational properties of the meta-language level when reasoning about the semantics of $\mathsf{FPC}_\oplus$, and it provides us with a useful induction principle for proving propositions ranging over $\mathcal{D}A$.

To model the combination of recursion and probabilistic choice,we use the *guarded convex delay monad* which we denote by D^κ. On a type A, $\mathsf{D}^\kappa A \simeq \mathcal{D}(A + \triangleright^\kappa(\mathsf{D}^\kappa A))$. Intuitively, this means that a computation of type A will be a distribution over values of A (immediately available) and delayed computations of type A. Quantifying over clocks gives the *convex delay monad* $\mathsf{D}^\forall A \triangleq \forall\kappa.\mathsf{D}^\kappa A$ which is a coinductive solution to the equation $\mathsf{D}^\forall A \simeq \mathcal{D}(A + \mathsf{D}^\forall A)$. Both operational and denotational semantics are defined using D^κ because it allows for guarded recursive definitions. These definitions can then be clock quantified to give elements of $\mathsf{D}^\forall$, which has the benefit that steps can be eliminated, because $\triangleright^\kappa$ does not get in the way.

As our main result, we define a logical relation, relating the denotational and operational semantics, and prove that it is sound with respect to contextual refinement. Traditionally, defining a logical relation relating denotational and operational semantics for a language with recursive types is non-trivial, see, e.g., [34]. Here we use the guarded type theory to define the logical relation by guarded recursion. As usual, the logical relation is divided into a relation for values and a relation for computations. Since computations compute to distributions, the challenge here lies in defining the relation for computations in terms of the relation for values. Earlier work on operationally-based logical relations [2, 9] used bi-orthogonality to reduce the problem to relating termination probabilities for computations of ground type. Here, we follow the approach of the recent article [20], which uses couplings [6, 28, 38, 42] to lift relations on values to relations on distributions. Note that the development in [20] relies on classical logic (in particular, the composition of couplings, the so-called bind lemma, relies on the axiom of choice) and thus does not apply here. Instead, we define a novel constructive notion of lifting of relations $\mathcal{R} : A \to B \to \mathsf{Prop}$ to relations on convex delay algebras $\overline{\mathcal{R}}^\kappa : \mathsf{D}^\kappa A \to \mathsf{D}^\forall B \to \mathsf{Prop}$ by guarded recursion. We establish a series of basic results for this, including a version of the important bind lemma that allows us to compose these liftings in proofs.

Finally, we use the semantics and the logical relation to prove contextual refinement of examples that combine probabilistic choice and recursion. The examples illustrate yet another benefit of the denotational semantics: Since fewer steps are needed in the denotational semantics than in the operational semantics, using this for reasoning makes the arguments less cluttered by steps than a direct relation between syntax and syntax would.

Contributions.

(1) To the best of our knowledge, we present the first constructive type theoretic account of operational and denotational semantics of $\mathsf{FPC}_\oplus$.
(2) We develop the theory of the finite distribution monad in cubical type theory, and we introduce the convex delay monad in CCTT, and develop the basic theory for it.
(3) We develop the basic constructive theory of couplings for convex delay algebras and use it to define a logical relation, relating denotational and operational semantics.
(4) We demonstrate how to use the semantics to reason about examples that combine probabilistic choice and recursion.

$$\frac{\Gamma \vdash \qquad \kappa : \mathsf{clock} \in \Gamma}{\Gamma, \alpha : \kappa \vdash} \qquad \frac{\Gamma \vdash t : \triangleright(\alpha{:}\kappa).A \qquad \Gamma, \beta : \kappa, \Gamma' \vdash}{\Gamma, \beta : \kappa, \Gamma' \vdash t\,[\beta] : A\,[\beta/\alpha]} \qquad \frac{\Gamma, \alpha : \kappa \vdash t : A}{\Gamma \vdash \lambda(\alpha{:}\kappa).t : \triangleright(\alpha{:}\kappa).A}$$

$$\frac{\Gamma, \kappa : \mathsf{clock} \vdash t : A}{\Gamma \vdash \Lambda\kappa.t : \forall\kappa.A} \qquad \frac{\Gamma \vdash t : \forall\kappa.A \qquad \Gamma \vdash \kappa' : \mathsf{clock}}{\Gamma \vdash t[\kappa'] : A[\kappa'/\kappa]} \qquad \frac{\Gamma \vdash t : \triangleright^\kappa A \to A}{\Gamma \vdash \mathsf{dfix}^\kappa\, t : \triangleright^\kappa A}$$

$$\frac{\Gamma \vdash t : \triangleright^\kappa A \to A}{\Gamma \vdash \mathsf{pfix}^\kappa\, t : \triangleright(\alpha{:}\kappa).(\mathsf{dfix}^\kappa t)\,[\alpha] =_A t(\mathsf{dfix}^\kappa t)}$$

Fig. 1. Selected typing rules for Clocked Cubical Type Theory.

2 Clocked Cubical Type Theory

We will work in Clocked Cubical Type Theory (CCTT) [26], an extension of Cubical Type Theory [12] with guarded recursion. At present, CCTT is the only existing type theory containing all constructions needed for this paper. We will not describe CCTT in detail, but only describe the properties we need, with the hope of making the paper more accessible, and so that the results can be reused in other (future) type theories with the same properties.

2.1 Basic Properties and HITs

CCTT has an infinite hierarchy of (Tarski style) universes (U_i), as well as identity types satisfying the standard rules and function extensionality. Precisely, CCTT, being based on Cubical Type Theory, has a path type as primitive, rather than an identity type in the traditional sense. However, the differences between the two are inessential to this work. We will write $a =_A b$, or just $a = b$, for the identity type associated with terms $a, b : A$ of the same type. Following conventions from homotopy type theory [39], we say that a type A is a (homotopy) proposition if for any $x, y : A$, the type $x = y$ is contractible, and that it is a (homotopy) set if for any $x, y : A$, the type $x = y$ is a proposition. We write Prop_i, and Set_i for the subuniverses of U_i of propositions and sets respectively, often omitting the universe level i. We write $A \simeq B$ for the type of equivalences from A to B. We shall mostly use this in the case where A and B are sets, and in that case, an equivalence is simply given by the standard notion of isomorphism of sets as phrased inside type theory using propositional equality. All types used represent to syntax, as well as the denotation of all types, are sets. Likewise all relations are valued in Prop. These choices simplify reasoning, as no higher dimensional structure needs to be accounted for. The only types used that are not sets are the universes Set and U.

CCTT also has higher inductive types (HITs), and we will use these to construct propositional truncation, set truncation and the finite distributions monad (see section 3). In particular, this means that one can express ordinary propositional logic with operators $\wedge, \vee, \exists, \forall$ on Prop, using the encodings of $\exists$ and $\vee$ defined using propositional truncation [39]. Recall in particular the elimination principles for $\exists$: When proving a proposition ψ assuming $\exists(x : X).\phi(x)$ we may assume we have an x in hand satisfying $\phi(x)$, but we cannot do that when mapping from $\exists(x : X).\phi(x)$ to an arbitrary type. We will also need inductive types to represent the type $\mathbb{N}$ of natural numbers, as well as types and terms of the language $\mathsf{FPC}_\oplus$ described in section 5. These are all captured by the schema for higher inductive types in CCTT [26].

2.2 Guarded Recursion

Guarded recursion uses a modal operator $\triangleright$ (pronounced 'later') to describe data that is available one time step from now. In multiclocked guarded recursion, $\triangleright$ is indexed by a clock κ. Clocks can be variables of the pretype clock or they can be the clock constant κ_0. Clocks can be universally quantified in the type $\forall\kappa.A$, with rules similar to Π-types, including a functional extensionality principle. The modality $\triangleright$ is a Fitch-style modality [7, 11], in the sense that introduction and elimination for $\triangleright$ is by abstraction and application to *ticks*, i.e., assumptions of the form $\alpha : \kappa$, to be thought of as evidence that time has ticked on clock κ. Because ticks can occur in terms, they can also occur in types, and the type $\triangleright(\alpha : \kappa).A$ binds α in A. When α does not occur in A we simply write $\triangleright^\kappa A$ for $\triangleright(\alpha:\kappa).A$. The rules for tick abstraction and application that we shall use in this paper are presented in Figure 1. In the figure, the notation $\Gamma \vdash$ means that Γ is a well-formed context. Note that the rule for tick application assumes that β (or anything occurring after that in the context) does not already occur in t. This is to avoid terms of the type $\triangleright^\kappa \triangleright^\kappa A \to \triangleright^\kappa A$ merging two time steps into one. We will sometimes use the notation

$$\mathsf{next}^\kappa \triangleq \lambda x.\lambda(\alpha:\kappa).x : A \to \triangleright^\kappa A. \tag{1}$$

The rules presented in Figure 1 are special cases of those of CCTT. The general rules allow certain 'timeless' assumptions in Γ' to occur in t in the tick application rule. This allows typing of an extensionality principle for $\triangleright$ of type

$$(x =_{\triangleright^\kappa A} y) \simeq \triangleright(\alpha:\kappa).(x\,[\alpha] =_A y\,[\alpha]). \tag{2}$$

In this paper we shall simply take this as an axiom. Intuitively, (2) states that x and y are equal, if the elements they deliver in the next time step are equal. One consequence of this is that $\triangleright$ preserves the property of being a set or a proposition. Other omitted rules for ticks allow typing of a *tick-irrelevance* axiom

$$\mathsf{tirr}^\kappa : \Pi(x : \triangleright^\kappa A).\triangleright(\alpha:\kappa).\triangleright(\beta:\kappa).(x\,[\alpha]) =_A (x\,[\beta]). \tag{3}$$

stating that all ticks are propositionally equal (although they should not be considered judgementally equal [4]).

The fixed point combinator dfix allows for defining and programming with guarded recursive types. Define $\mathsf{fix}^\kappa : (\triangleright^\kappa A \to A) \to A$ as $\mathsf{fix}^\kappa f = f(\mathsf{dfix}^\kappa f)$. Then one can prove that

$$\mathsf{fix}^\kappa t = t(\lambda(\alpha:\kappa).\mathsf{fix}^\kappa t). \tag{4}$$

Applying the fix point operator to maps on a universe, as in [8], one can define *guarded recursive types* such as the *guarded delay monad* L^κ mapping a type A to $L^\kappa A$ satisfying

$$L^\kappa A \simeq A + \triangleright^\kappa(L^\kappa A). \tag{5}$$

In this paper, we will not spell out how such guarded recursive types are defined as fixed points, but just give the defining guarded recursive equation. Intuitively, the reason this is well defined is that $L^\kappa A$ only occurs under a $\triangleright$ on the right-hand side of (5), which allows the recursive equation to be phrased as a map $\triangleright^\kappa \mathsf{U} \to \mathsf{U}$. One can also use fix to program with L^κ defining, e.g., the diverging computation as $\bot = \mathsf{fix}(\lambda x.\mathsf{inr}(x))$ (leaving the type equivalence between $L^\kappa A$ and its unfolding implicit). In this case (4) specialises to $\bot = \mathsf{inr}(\lambda(\alpha:\kappa).\bot)$.

2.3 Coinductive Types

Coinductive types can be represented by quantifying over clocks in guarded recursive types [3]. For example, the type $LA \triangleq \forall\kappa.L^\kappa A$ defines a coinductive solution to $LA \simeq A + LA$ in CCTT, provided A is *clock-irrelevant*, meaning that the canonical map $A \to \forall\kappa.A$ is an equivalence. More generally, one can prove that any functor $F : \mathsf{U} \to \mathsf{U}$ (in the naive sense of having a functorial action

$(A \to B) \to FA \to FB)$ commuting with clock quantification in the sense that $F(\forall\kappa.X) \simeq \forall\kappa.F(X)$ via the canonical map, has a final coalgebra defined as $\nu(F) \triangleq \forall\kappa.\nu^\kappa(F)$, where $\nu^\kappa(F) \simeq F(\triangleright^\kappa(\nu^\kappa(F))$ is defined using fix. This encoding also works for indexed coinductive types. The correctness of the encoding of coinductive types can be proved in CCTT, and relies on the type equivalence

$$\forall\kappa.A \simeq \forall\kappa.\triangleright^\kappa A.$$

For the encoding to be useful, one needs a large collection of clock-irrelevant types and functors commuting with clock quantification. We will need that $\mathbb{N}$ is clock-irrelevant, and that clock quantification commutes with sums in the sense that $\forall\kappa.(A+B) \simeq (\forall\kappa.A) + (\forall\kappa.B)$. Both of these can be proved in CCTT using the notion of induction under clocks for higher inductive types [26]. Note also that all propositions P are clock-irrelevant, because the clock constant κ_0 can be used to define a map $(\forall\kappa.P) \to P$. The following lemma can be used to prove types clock irrelevant.

Lemma 2.1. *Suppose $i : A \to B$ is injective in the sense of the existence of a map $\Pi x, y : A.(i(x) = i(y)) \to x = y$, and suppose B is clock irrelevant. Then also A is clock-irrelevant.*

For the rest of the paper, we work informally in CCTT.

3 Finite Distributions

In the previous section we saw the definition of the guarded delay monad L^κ (Equation 5), which models non-terminating computations. In this section, we introduce the distribution monad $\mathcal{D}$, which models finite probability distributions. These two monads will be combined to form the guarded convex delay monad in the next section.

To construct the distribution monad $\mathcal{D}$, we first need some infrastructure to reason about probabilities. We will assume types representing the open rational interval $(0, 1)$ and closed rational interval $[0, 1]$, and we assume that we have the operations product, division of smaller numbers by larger numbers, and inversion $(1 - (-))$. We also need that $(0, 1)$ is a set with decidable equality, and that $(0, 1)$ is clock irrelevant. In practice, there are several ways of specifying $(0, 1)$, for example as a type of pairs (n, d), of mutually prime, positive natural numbers satisfying $n < d$, and $[0, 1]$ can be obtained, e.g., by just adding two points to $(0, 1)$. For this encoding, clock irrelevance of $(0, 1)$ follows from the embedding into $\mathbb{N} \times \mathbb{N}$ and Lemma 2.1. However, for this paper the specific implementation is irrelevant, so we leave this open.

Then, we need to find the right representation of probability distributions in type theory. In classical presentations of probability theory, a finite distribution on a set A is a finite map into $[0, 1]$, whose values sum to 1. In type theory, this would be represented by a subtype of $A \to [0, 1]$, which would need some notion of finite support, as well as a definition of the functorial action of $\mathcal{D}$. Especially the latter is difficult to do: Given a map $f : A \to B$, then $\mathcal{D}(f) : \mathcal{D}(A) \to \mathcal{D}(B)$ should map a probability distribution μ to the distribution which, for each $b : B$, sums the probabilities $\mu(a)$ for each a such that $f(a) = b$. To define this, we would need decidable equality on B to compute for which a the equality $f(a) = b$ holds, which is too restrictive. Another approach could be to use lists of key-value pairs, but this requires a quotient to obtain the correct notion of equality of distributions, which would make it very hard to work with in practice.

Here we instead choose to represent $\mathcal{D}$ as the free monad for the theory of convex algebras, which is known to generate the finite distribution monad [22]. The operation $\oplus$ in the definition below should be thought of as a convex sum: $x \oplus_p y = p \cdot x + (1 - p) \cdot y$.

Definition 3.1 (Convex Algebra). A *convex algebra* is a set A together with an operation $\oplus : (0,1) \to A \to A \to A$, such that

$$\mu \oplus_p \mu = \mu \tag{idem}$$

$$\mu \oplus_p \nu = \nu \oplus_{1-p} \mu \tag{comm}$$

$$\left(\mu_1 \oplus_p \mu_2\right) \oplus_q \mu_3 = \mu_1 \oplus_{pq} \left(\mu_2 \oplus_{\frac{q-pq}{1-pq}} \mu_3\right) \tag{assoc}$$

A map $f : A \to B$ between convex algebras A and B is a *homomorphism* if $f(\mu \oplus_p \nu) = f(\mu) \oplus_p f(\nu)$ holds for all μ, ν, p.

Definition 3.2. Let $\mathcal{D}(A)$ be the higher inductive type defined using two constructors

$$\delta : A \to \mathcal{D}(A) \qquad\qquad \oplus : (0,1) \to \mathcal{D}(A) \to \mathcal{D}(A) \to \mathcal{D}(A),$$

and equations (paths) for idempotency, commutativity and associativity as in the definition of convex algebra, plus an equation for set truncation.

One advantage of HITs is that they come with an easy induction principle. For $\mathcal{D}(A)$, it is as follows: If $\phi(x)$ is a proposition for all $x : \mathcal{D}(A)$ and $\phi(\delta(a))$ holds for all a, and moreover, $\phi(\mu)$ and $\phi(\nu)$ implies $\phi(\mu \oplus_p \nu)$ for all μ, ν, p, then $\phi(x)$ holds for all x. We will use this proof technique in numerous places throughout this paper. Similarly, the recursion principle for $\mathcal{D}(A)$ states that to define a map f from $\mathcal{D}(A)$ into a set B, it suffices to define the cases of $f(\delta(x))$ and $f(x \oplus_p y)$, the latter using $f(x)$ and $f(y)$, in such a way that the convex algebra equations are respected. The recursion principle implies the following.

Proposition 3.3. *$\mathcal{D}(A)$ is the free convex algebra on A, in the sense that for any convex algebra B, and function $f : A \to B$, there exists a unique homomorphism of convex algebras $\overline{f} : \mathcal{D}(A) \to B$ satisfying $f = \overline{f} \circ \delta$. As a consequence, $\mathcal{D}(-)$ forms a monad on the category of sets.*

Now that we have a type $\mathcal{D}(A)$ of probability distributions over a set A, we would like to reason about these distributions. In particular, we would like to know when two distributions are equal. For that, we use the following trick: we first relate probability distributions on finite sets to the classical probability distributions on these sets (that is, to functions into $[0,1]$ with finite support). Then, we use functoriality of $\mathcal{D}$ to generalise to arbitrary sets A.

Define the Bishop finite sets in the standard way by induction on n as $\mathsf{Fin}(0) = 0$ and $\mathsf{Fin}(n+1) = \mathsf{Fin}(n) + 1$, where the + on the right hand side refers to sum of types. For these sets, we can relate $\mathcal{D}$ to its classical definition. First, we use that if any set A has decidable equality, we can associate a probability function $f_\mu : A \to [0,1]$ to a distribution μ by induction, using $f_{\delta(x)}(y) = 1$ if $x = y$ and $f_{\delta(x)}(y) = 0$ else. This gives us an equivalence of types:

Lemma 3.4. $\mathcal{D}(\mathsf{Fin}(n)) \simeq \Sigma(f : \mathsf{Fin}(n) \to [0,1]).\mathsf{sum}(f) = 1$

Here, sum is the sum of the values of f, defined by induction on n. Note that the right hand side of this equivalence indeed captures the classical definition of probability distributions over $\mathsf{Fin}(n)$. We use Lemma 3.4 to reason about equality for probability distributions as follows.

Example 3.5. The isomorphism of Lemma 3.4 allows us to prove equations of distributions by first mapping to the right hand side of the isomorphism and then using functional extensionality. Consider, for example, the equation

$$(a \oplus_p b) \oplus_p (c \oplus_p a) = (b \oplus_{\frac{1}{2}} c) \oplus_{2p(1-p)} a,$$

where $a, b, c : \mathcal{D}(A)$ for some A. In the case where $A = \mathsf{Fin}(3)$ and $a = \delta(0), b = \delta(1), c = \delta(2)$, we can prove this by noting that both the left and right-hand sides of the equation correspond to the

map $f(0) = p^2 + (1-p)^2 = 1 + 2p^2 - 2p = 1 - 2p(1-p)$, $f(1) = f(2) = p(1-p)$. Finally, the case of general A, a, b, c follows from applying functoriality to the canonical map $\mathsf{Fin}(3) \to \mathcal{D}A$.

Lastly, the guarded convex delay monad in the next section will be defined as a distribution on a sum type. To reason about such distributions, we will frequently use the fact that any distribution $\mu : \mathcal{D}(A+B)$ on a sum type can be written uniquely as a convex sum of two subdistributions: One on A and one on B. In the classical setting where a distribution is a map with finite support this is trivial: The two subdistributions are simply the normalised restrictions of the distribution map to A and B respectively. In the constructive type theoretic setting the proof requires a bit more work, so we mention this as a separate theorem.

Theorem 3.6. *For all sets A and B, the map*

$$\mathcal{D}(A) + \mathcal{D}(B) + \mathcal{D}(A) \times (0,1) \times \mathcal{D}(B) \to \mathcal{D}(A+B),$$

defined by

$$\begin{aligned} f(\mathsf{in}_1(\mu)) &\triangleq \mathcal{D}(\mathsf{inl})(\mu) \\ f(\mathsf{in}_2(\mu)) &\triangleq \mathcal{D}(\mathsf{inr})(\mu) \\ f(\mathsf{in}_3(\mu, p, \nu)) &\triangleq (\mathcal{D}(\mathsf{inl})(\mu)) \oplus_p (\mathcal{D}(\mathsf{inr})(\nu)), \end{aligned}$$

is an equivalence of types.

4 Convex Delay Algebras

In this section, we define the guarded convex delay monad D^κ and the convex delay monad $\mathsf{D}^\forall$ modelling the combination of probabilistic choice and recursion. We first recall the notion of delay algebra (sometimes called a lifting or a $\triangleright^\kappa$-algebra [33]) and define a notion of convex delay algebra.

Definition 4.1 (Convex Delay Algebra). A (κ-)*delay algebra* is a set A together with a map $\mathsf{step}^\kappa : \triangleright^\kappa A \to A$. A delay algebra homomorphism is a map $f : A \to B$ such that $f(\mathsf{step}^\kappa(a)) = \mathsf{step}^\kappa(\lambda(\alpha{:}\kappa).f(a\,[\alpha]))$ for all $a : \triangleright^\kappa A$. A *convex delay algebra* is a set A with both a delay algebra structure and a convex algebra structure, and a homomorphism of these is a map respecting both structures.

Defining the functorial action of $\triangleright^\kappa$ as $\triangleright^\kappa(f)(a) \triangleq \lambda(\alpha{:}\kappa).f(a\,[\alpha])$, the requirement for f being a delay algebra homomorphism can be expressed as the following commutative diagram

$$\begin{array}{ccc} \triangleright^\kappa A & \xrightarrow{\triangleright^\kappa(f)} & \triangleright^\kappa B \\ \downarrow{\scriptstyle \mathsf{step}^\kappa} & & \downarrow{\scriptstyle \mathsf{step}^\kappa} \\ A & \xrightarrow{f} & B \end{array}$$

Recall the guarded delay monad L^κ satisfying $L^\kappa A \simeq A + \triangleright^\kappa(L^\kappa A)$. The type $L^\kappa A$ is easily seen to be the free guarded delay algebra on a set A. Similarly, define the *guarded convex delay monad* as the guarded recursive type

$$\mathsf{D}^\kappa A \simeq \mathcal{D}(A + \triangleright^\kappa(\mathsf{D}^\kappa A)).$$

Again, this can be constructed formally as a fixed point on a universe (assuming A lives in the same universe), but we shall not spell this out here. We show that it is a free convex delay algebra. First define the convex delay algebra structure $(\mathsf{step}^\kappa, \oplus^\kappa)$ on $\mathsf{D}^\kappa A$ and the inclusion $\delta^\kappa : A \to \mathsf{D}^\kappa A$ by

$$\mathsf{step}^\kappa(a) = \delta(\mathsf{inr}(a)) \qquad \mu \oplus^\kappa_p \nu = \mu \oplus_p \nu \qquad \delta^\kappa a = \delta(\mathsf{inl}(a))$$

where the convex algebra structure on the right-hand sides of the equations above refers to those of $\mathcal{D}$.

Proposition 4.2. *$D^\kappa A$ is the free convex delay algebra structure on A, for any set A. As a consequence D^κ defines a monad on the category of sets.*

Proof. Suppose $f : A \to B$ and that B is a convex delay algebra. We define the extension $\overline{f} : D^\kappa A \to B$ by guarded recursion, so suppose we are given $g : \triangleright^\kappa(D^\kappa A \to B)$ and define

$$\overline{f}(\mathsf{step}^\kappa(a)) = \mathsf{step}^\kappa(\lambda(\alpha\!:\!\kappa).g\,[\alpha](a\,[\alpha]))$$
$$\overline{f}(\delta^\kappa(a)) = f(a)$$
$$\overline{f}(\mu \oplus_p^\kappa \nu) = \overline{f}(\mu) \oplus_p \overline{f}(\nu).$$

Note that these cases define $\overline{f}$ by induction on $\mathcal{D}$ and +. Unfolding the guarded recursive definition using (4) gives

$$\overline{f}(\mathsf{step}^\kappa(a)) = \mathsf{step}^\kappa(\lambda(\alpha\!:\!\kappa).\overline{f}(a\,[\alpha])).$$

so that $\overline{f}$ is a homomorphism of convex delay algebras. For uniqueness, suppose g is another homomorphism extending f. We show that $g = \overline{f}$ by guarded recursion and function extensionality. So suppose we are given $p : \triangleright^\kappa(\Pi(x : D^\kappa A)(g(a) = \overline{f}(a)))$. Then, for any $a : \triangleright^\kappa(D^\kappa A)$, the term $\lambda(\alpha\!:\!\kappa).p\,[\alpha](a\,[\alpha])$ proves

$$\triangleright(\alpha\!:\!\kappa).(g(a\,[\alpha]) = \overline{f}(a\,[\alpha])),$$

which by (2) is equivalent to $\lambda(\alpha\!:\!\kappa).g(a\,[\alpha]) = \lambda(\alpha\!:\!\kappa).\overline{f}(a\,[\alpha])$. Then,

$$\begin{aligned} g(\mathsf{step}^\kappa(a)) &= \mathsf{step}^\kappa(\lambda(\alpha\!:\!\kappa).g(a\,[\alpha])) \\ &= \mathsf{step}^\kappa(\lambda(\alpha\!:\!\kappa).\overline{f}(a\,[\alpha])) \\ &= \overline{f}(\mathsf{step}^\kappa(a)). \end{aligned}$$

The rest of the proof that $g(a) = \overline{f}(a)$, for all a, then follows by HIT induction on $\mathcal{D}$.

In terms of category theory, we have shown that D^κ is left adjoint to the forgetful functor from convex delay algebras to the category of sets. It therefore defines a monad on the category of sets. □

We will often write $t \gg\!=^\kappa f$ for $\overline{f}(t)$ where $\overline{f}$ is the unique extension of f to a convex delay algebra homomorphism.

Example 4.3. The type $D^\kappa A$ can be thought of as a type of probabilistic processes returning values in A. Define, for example, a geometric process with probability $p : (0,1)$ as $\mathsf{geo}_p^\kappa\,0$ where

$$\mathsf{geo}_p^\kappa : \mathbb{N} \to D^\kappa\mathbb{N}$$
$$\mathsf{geo}_p^\kappa\,n \triangleq (\delta^\kappa n) \oplus_p \mathsf{step}^\kappa(\lambda(\alpha\!:\!\kappa).\mathsf{geo}_p^\kappa\,(n+1))$$

Note that this gives

$$\begin{aligned} \mathsf{geo}_p^\kappa(0) &= (\delta^\kappa 0) \oplus_p \mathsf{step}^\kappa(\lambda(\alpha\!:\!\kappa).\mathsf{geo}_p^\kappa\,(1)) \\ &= (\delta^\kappa 0) \oplus_p \Big(\mathsf{step}^\kappa\lambda(\alpha\!:\!\kappa).\Big((\delta^\kappa 1) \oplus_p (\mathsf{step}^\kappa\lambda(\alpha\!:\!\kappa).\mathsf{geo}_p^\kappa\,(2))\Big)\Big) \\ &= \dots \end{aligned}$$

The modal delay $\triangleright^\kappa$ in the definition of $D^\kappa A$ prevents us from accessing values computed later, thereby also preventing us from, *e.g.*, computing probabilities of 'termination in n steps' as elements of $[0,1]$. Such operations should instead be defined on the convex delay monad $D^\forall$. This monad is defined in terms of D^κ by quantifying over clocks:

$$D^\forall A \triangleq \forall\kappa.D^\kappa A.$$

Proposition 4.4. *If A is clock-irrelevant then $\mathsf{D}^{\forall}A$ is the final coalgebra for the functor $F(X) = \mathcal{D}(A+X)$.*

Proof. We must show that F commutes with clock quantification, and this reduces easily to showing that $\mathcal{D}$ commutes with clock quantification. The latter can be either proved directly by using the same technique as for the similar result for the finite powerset functor [26], or by referring to [31, Proposition 14], which states that any free model monad for a theory with finite arities commutes with clock quantification. □

Remark 1. *From now on, whenever we look at a type $\mathsf{D}^{\forall}A$, we assume A to be clock-irrelevant.*

In particular, $\mathsf{D}^{\forall}A \simeq \mathcal{D}(A + \mathsf{D}^{\forall}A)$, and so $\mathsf{D}^{\forall}$ carries a convex algebra structure $(\delta^{\forall}, \oplus^{\forall})$ as well as a map $\mathsf{step}^{\forall} : \mathsf{D}^{\forall}A \to \mathsf{D}^{\forall}A$ defined by $\mathsf{step}^{\forall}(x) = \delta(\mathsf{inr}(x))$. Define also $\delta^{\forall} : A \to \mathsf{D}^{\forall}A$ as $\delta^{\forall}x \triangleq \delta(\mathsf{inl}\, x)$.

Example 4.5. The geometric process can be defined as an element of $\mathbb{N} \to \mathsf{D}^{\forall}\mathbb{N}$ as $\mathsf{geo}_p\, n \triangleq \Lambda\kappa.\mathsf{geo}^{\kappa}_p\, n$. This satisfies the equations

$$\begin{aligned}\mathsf{geo}_p\, 0 &= (\delta^{\forall}\, 0) \oplus^{\forall}_p \mathsf{step}^{\forall}(\mathsf{geo}_p\,(1)) \\ &= (\delta^{\forall}\, 0) \oplus_p \mathsf{step}^{\forall}((\delta^{\forall}\, 1) \oplus_p \mathsf{step}^{\forall}(\mathsf{geo}_p\,(2))) \\ &= \dots\end{aligned}$$

Lemma 4.6. *$\mathsf{D}^{\forall}$ carries a monad structure whose unit is $\delta^{\forall}$ and where the Kleisli extension $\overline{f} : \mathsf{D}^{\forall}A \to \mathsf{D}^{\forall}B$ of a map $f : A \to \mathsf{D}^{\forall}B$ satisfies*

$$\overline{f}(\mathsf{step}^{\forall}x) = \mathsf{step}^{\forall}(\overline{f}(x)) \qquad\qquad \overline{f}(\mu \oplus^{\forall}_p \nu) = \overline{f}(\mu) \oplus^{\forall}_p \overline{f}(\nu).$$

Proof. This is a consequence of [31, Lemma 16]. □

4.1 Probability of Termination

Unfolding the type equivalence $\mathsf{D}^{\forall}A \simeq \mathcal{D}(A + \mathsf{D}^{\forall}A)$ we can define maps out of $\mathsf{D}^{\forall}A$ by cases. For example, we can define the probability of immediate termination $\mathsf{PT}_0 : \mathsf{D}^{\forall}A \to [0,1]$, as:

$$\begin{aligned}\mathsf{PT}_0(\delta^{\forall}a) &= 1 \\ \mathsf{PT}_0(\mathsf{step}^{\forall}d) &= 0 \\ \mathsf{PT}_0(x \oplus^{\forall}_p y) &= p \cdot \mathsf{PT}_0(x) + (1-p) \cdot \mathsf{PT}_0(y).\end{aligned}$$

Likewise, we define a function $\mathsf{run} : \mathsf{D}^{\forall}A \to \mathsf{D}^{\forall}A$ that runs a computation for one step, eliminating a single level of $\mathsf{step}^{\forall}$ operations:

$$\mathsf{run}(\delta^{\forall}a) = \delta^{\forall}a \qquad \mathsf{run}(\mathsf{step}^{\forall}d) = d \qquad \mathsf{run}(x \oplus^{\forall}_p y) = (\mathsf{run}\, x) \oplus^{\forall}_p (\mathsf{run}\, y).$$

We can hence compute the probability of termination in n steps, by first running a computation for n steps, and then computing the probability of immediate termination of the result:

$$\mathsf{PT}_n(x) = \mathsf{PT}_0(\mathsf{run}^n x).$$

For example, running the geometric process for one step gives:

$$\mathsf{run}(\mathsf{geo}_p\, 0) = (\delta^{\forall}\, 0) \oplus_p ((\delta^{\forall}\, 1) \oplus_p \mathsf{step}^{\forall}(\mathsf{geo}_p\,(2))),$$

so $\mathsf{PT}_1(\mathsf{geo}_p\, 0) = p + (1-p)p = 2p - p^2$.

We will later prove (Lemma 4.14) that $\mathsf{PT}_{(.)}(\mu) : \mathbb{N} \to [0,1]$ is monotone for all μ. Intuitively, the probability of termination for μ is the limit of this sequence. The following definition gives a simple way of comparing such limits, without introducing real numbers to our type theory.

Definition 4.7. Let $f, g\colon \mathbb{N} \to [0,1]$ be monotone. Write $f \leq_{\lim} g$ for the following proposition: for every $n\colon \mathbb{N}$ and every rational $\varepsilon > 0$ there exists $m\colon \mathbb{N}$ such that $f(n) \leq g(m) + \epsilon$. We write $f =_{\lim} g$ for the conjunction $f \leq_{\lim} g$ and $g \leq_{\lim} f$.

It is easy to show that the relation $\leq_{\lim}$ is reflexive and transitive and thus $=_{\lim}$ is an equivalence relation. Furthermore, $\leq_{\lim}$ is closed under pointwise convex sums.

Lemma 4.8. *Let $f_1, g_1, f_2, g_2\colon \mathbb{N} \to [0,1]$ be monotone and let $p \in (0,1)$. If $f_1 \leq_{\lim} g_1$ and $f_2 \leq_{\lim} g_2$, then $p \cdot f_1 + (1-p) \cdot f_2 \leq_{\lim} p \cdot g_1 + (1-p) \cdot g_2$.*

4.2 Step Reductions

Rather than eliminating a full level of $\mathsf{step}^\forall$ operations with run, it is sometimes useful to allow different branches to run for a different number of steps. We capture this in the *step reduction relation* $\rightsquigarrow$. This is the reflexive, transitive, and convex closure of the one-step reduction relation:

Definition 4.9. Define the step reduction relation $\rightsquigarrow\colon \mathsf{D}^\forall A \to \mathsf{D}^\forall A \to \mathsf{Prop}$ inductively by the following rules

$$\frac{}{v \rightsquigarrow v} \qquad \frac{}{\mathsf{step}^\forall v \rightsquigarrow v} \qquad \frac{v \rightsquigarrow v' \qquad v' \rightsquigarrow v''}{v \rightsquigarrow v''} \qquad \frac{v_1 \rightsquigarrow v_1' \qquad v_2 \rightsquigarrow v_2'}{v_1 \oplus_p^\forall v_2 \rightsquigarrow v_1' \oplus_p^\forall v_2'}$$

Remark 2. *CCTT lacks higher inductive families as primitive, but the special case of the step reduction relation can be represented by defining a fuelled version $\rightsquigarrow^n$ by induction on n, and existentially quantifying n.*

The step reduction relation $\rightsquigarrow$ is closely related to run.

Lemma 4.10. *Let $v, v' : \mathsf{D}^\forall A$. Then:*

(1) For all $n : \mathbb{N}$, $v \rightsquigarrow \mathsf{run}^n v$.
(2) If $v \rightsquigarrow v'$ then, for all $n : \mathbb{N}$, $\mathsf{run}^n v \rightsquigarrow \mathsf{run}^n v'$.
(3) If $v \rightsquigarrow v'$ then there exists an $n : \mathbb{N}$ such that $v' \rightsquigarrow \mathsf{run}^n v$.

Proof. The first two facts are proven by induction on n, the last by induction on $\rightsquigarrow$. □

Unlike run, the relation $\rightsquigarrow$ can execute different branches of a probabilistic computation for a different number of steps, as illustrated in the following example.

Example 4.11. Consider: $v = (\mathsf{step}^\forall(\delta^\forall a)) \oplus_p^\forall (\mathsf{step}^\forall(\mathsf{step}^\forall(\delta^\forall b)))$. Then run removes $\mathsf{step}^\forall$ operations from both branches.

$$\begin{aligned} \mathsf{run}(v) &= (\delta^\forall a) \oplus_p^\forall (\mathsf{step}^\forall(\delta^\forall b)) \\ \mathsf{run}^2(v) &= (\delta^\forall a) \oplus_p^\forall (\delta^\forall b) \\ \mathsf{run}^n(v) &= (\delta^\forall a) \oplus_p^\forall (\delta^\forall b) \text{ for } n \geq 2. \end{aligned}$$

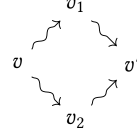
As per Lemma 4.10, we have $v \rightsquigarrow (\mathsf{run}^n v)$, for each $n : \mathbb{N}$, as well as:

$$\begin{aligned} v &\rightsquigarrow (\delta^\forall a) \oplus_p^\forall (\mathsf{step}^\forall(\mathsf{step}^\forall(\delta^\forall b))) \\ v &\rightsquigarrow (\mathsf{step}^\forall(\delta^\forall a)) \oplus_p^\forall (\mathsf{step}^\forall(\delta^\forall b)) \\ v &\rightsquigarrow (\mathsf{step}^\forall(\delta^\forall a)) \oplus_p^\forall (\delta^\forall b). \end{aligned}$$

We will use this flexibility in running different branches for a different number of steps in both our proofs and examples, such as in the proof of Lemma 7.9.

The step reduction relation is confluent, since we can always match two probabilistic processes that stem from the same parent process by running all branches long enough.

Corollary 4.12 (Confluence).
Let $v, v_1, v_2 : \mathsf{D}^\forall A$.
If both $v \rightsquigarrow v_1$ *and* $v \rightsquigarrow v_2$,
then there is a $v' : \mathsf{D}^\forall$ *such that* $v_1 \rightsquigarrow v'$ *and* $v_2 \rightsquigarrow v'$.



Proof. We apply Lemma 4.10.3, giving us an n_1 and n_2 such that $v_1 \rightsquigarrow \mathsf{run}^{n_1} v$ and $v_2 \rightsquigarrow \mathsf{run}^{n_2} v$. Taking $N = \max(n_1, n_2)$ then gives the confluence via Lemma 4.10.1. □

We mention two more useful properties of step reductions. Firstly, the bind operation preserves the $\rightsquigarrow$ relation, which follows by an easy induction on $\rightsquigarrow$.

Lemma 4.13. *If* $f : A \to \mathsf{D}^\forall(B)$, *and* $v, v' : \mathsf{D}^\forall A$ *such that* $v \rightsquigarrow v'$. *Then also* $\overline{f}(v) \rightsquigarrow \overline{f}(v')$.

Secondly, the probability of termination is monotone with respect to n, and along step reductions:

Lemma 4.14. *For* $v, v' : \mathsf{D}^\forall A$, *we have:*
(1) For all $n, m : \mathbb{N}$ *such that* $n \leq m$: $\mathsf{PT}_n(v) \leq \mathsf{PT}_m(v)$.
(2) If $v \rightsquigarrow v'$ *then for all* $n : \mathbb{N}$: $\mathsf{PT}_n(v) \leq \mathsf{PT}_n(v')$.
(3) If $v \rightsquigarrow v'$ *then* $\mathsf{PT}_{(.)}(v) =_{\lim} \mathsf{PT}_{(.)}(v')$.

Proof. The first statement is by induction on n and case analysis for v, the second statement by induction on n and $\rightsquigarrow$. The third statement follows by induction on $\rightsquigarrow$, the properties of $\leq_{\lim}$ and monotonicity of $\mathsf{PT}_{(.)}(-)$ (i.e. the first statement of this lemma). □

4.3 Approximate Step Reductions

The monads D^κ and $\mathsf{D}^\forall$ model the fact that at any point in time we might not have complete information about a probability distribution, with more information coming in at every time step. The relations run and $\rightsquigarrow$ process the incoming information and give new probability distributions that are future variants of the original (delayed) distribution, after a finite number of unfoldings. However, it is often useful to talk about what happens in the limit, after infinitely many unfoldings. For this purpose we introduce the relation $\rightsquigarrow^\approx$. Intuitively, $v \rightsquigarrow^\approx v'$ holds if we can get arbitrarily close to v' by applying sufficiently many unfoldings to v.

Definition 4.15. Define the *approximate step reduction* relation $\rightsquigarrow^\approx : \mathsf{D}^\forall A \to \mathsf{D}^\forall A \to \mathsf{Prop}$ as: $v \rightsquigarrow^\approx v'$ if for all $p \in (0, 1)$ there is a $v'' : \mathsf{D}^\forall A$ such that $v \rightsquigarrow v' \oplus^\forall_p v''$.

Example 4.16. The approximate step reduction relation $\rightsquigarrow^\approx$ allows us to talk about limits constructively. For example, say h_x is a hesitant point distribution h_x, if $h_x \rightsquigarrow \delta^\forall(x) \oplus^\forall_q h_x$. For instance, $h_x = \Lambda\kappa.(\mathsf{fix}^\kappa(\lambda y.\delta^\kappa(x) \oplus^\kappa_q \mathsf{step}^\kappa y))$ satisfies this, but we will see another example in section 9.1. In the limit, h_x should be the same distribution as the normal point distribution $\delta^\forall(x)$. Indeed,

$$h_x \rightsquigarrow^\approx \delta^\forall(x).$$

To prove this, we need to show that for all $p \in (0, 1)$ there is a v such that $h_x \rightsquigarrow \delta^\forall(x) \oplus^\forall_p v$. Let, for any natural number n, $q_n = \sum_{i=0}^{n} q(1-q)^i$, then by applying transitivity of $\rightsquigarrow$ n times:

$$h_x \rightsquigarrow \delta^\forall(x) \oplus^\forall_{q_n} h_x.$$

If we choose n such that $q_n \geq p$, then we can write

$$\delta^\forall(x) \oplus^\forall_{q_n} h_x = \delta^\forall(x) \oplus^\forall_p (\delta^\forall(x) \oplus^\forall_{\frac{q_n - p}{1-p}} h_x).$$

$$\frac{x : \sigma \in \Gamma \qquad \vdash \Gamma}{\Gamma \vdash x : \sigma} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \langle\rangle : 1} \qquad \frac{n : \mathbb{N} \qquad \vdash \Gamma}{\Gamma \vdash \underline{n} : \mathsf{Nat}} \qquad \frac{\Gamma \vdash L : \mathsf{Nat} \qquad \Gamma \vdash M : \sigma \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash \mathsf{ifz}(L, M, N) : \sigma}$$

$$\frac{\Gamma \vdash M : \mathsf{Nat}}{\Gamma \vdash \mathsf{suc}\, M : \mathsf{Nat}} \qquad \frac{\Gamma \vdash M : \sigma}{\Gamma \vdash \mathsf{inl}\, M : \sigma + \tau} \qquad \frac{\Gamma \vdash L : \sigma + \sigma' \qquad \Gamma, x : \sigma \vdash M : \tau \qquad \Gamma, y : \sigma' \vdash N : \tau}{\Gamma \vdash \mathsf{case}(L, x.M, y.N) : \tau}$$

$$\frac{\Gamma \vdash M : \sigma \qquad \Gamma \vdash N : \tau}{\Gamma \vdash \langle M, N\rangle : \sigma \times \tau} \qquad \frac{\Gamma \vdash M : \sigma \times \tau}{\Gamma \vdash \mathsf{fst}\, M : \sigma} \qquad \frac{\Gamma, (x : \sigma) \vdash M : \tau}{\Gamma \vdash \mathsf{lam}\, x.M : \sigma \to \tau} \qquad \frac{\Gamma \vdash N : \sigma \to \tau \qquad \Gamma \vdash M : \sigma}{\Gamma \vdash NM : \tau}$$

$$\frac{\Gamma \vdash M : \tau[\mu X.\tau/X]}{\Gamma \vdash \mathsf{fold}\, M : \mu X.\tau} \qquad \frac{\Gamma \vdash M : \mu X.\tau}{\Gamma \vdash \mathsf{unfold}\, M : \tau[\mu X.\tau/X]} \qquad \frac{\Gamma \vdash M : \sigma \qquad \Gamma \vdash N : \sigma \qquad p : (0,1)}{\Gamma \vdash \mathsf{choice}^p(M, N) : \sigma}$$

Fig. 2. Typing rules for $\mathsf{FPC}_\oplus$. Rules for pred, inr, snd omitted.

Then choosing $v = \delta^\forall(x) \oplus^\forall_{\frac{qn-p}{1-p}} h_x$ gives $h_x \rightsquigarrow \delta^\forall(x) \oplus^\forall_p v$, which is what we needed to show.

We again prove some useful properties of $\rightsquigarrow^\approx$ that we need in future proofs. We start with the defining properties of the step reductions. These also hold for approximate step reductions:

Lemma 4.17. *The relation $\rightsquigarrow^\approx$ is reflexive and transitive. Furthermore,* $\mathsf{step}^\forall v \rightsquigarrow^\approx v$, *and $\rightsquigarrow^\approx$ is preserved by sums: if $v_1 \rightsquigarrow^\approx v_1'$ and $v_2 \rightsquigarrow^\approx v_2'$, then for all $p \in (0,1)$, $v_1 \oplus^\forall_p v_2 \rightsquigarrow^\approx v_1' \oplus^\forall_p v_2'$.*

The bind operation also preserves approximate step reductions, which follows directly from the fact that the bind operation preserves $\rightsquigarrow$.

Lemma 4.18. *If $f : A \to \mathsf{D}^\forall(B)$, and $v, v' : \mathsf{D}^\forall A$ such that $v \rightsquigarrow^\approx v'$. Then also $\overline{f}(v) \rightsquigarrow^\approx \overline{f}(v')$.*

Step reductions are also approximate step reductions, and approximately reducing step reductions just results in further approximate reductions, as does step reducing an approximate step reduction.

Lemma 4.19. *Let $v, v_1, v_2 : \mathsf{D}^\forall A$. Then:*

(1) If $v \rightsquigarrow v_1$, then $v \rightsquigarrow^\approx v_1$.
(2) If $v \rightsquigarrow v_1$ and $v_1 \rightsquigarrow^\approx v_2$, then $v \rightsquigarrow^\approx v_2$.
(3) If $v \rightsquigarrow^\approx v_1$ and $v_1 \rightsquigarrow v_2$, then $v \rightsquigarrow^\approx v_2$.

Finally we show that approximate step reductions preserve limit probabilities of termination:

Lemma 4.20. *Let $v_1, v_2 : \mathsf{D}^\forall A$. If $v_1 \rightsquigarrow^\approx v_2$ then $\mathsf{PT}_{(.)}(v_1) =_{\lim} \mathsf{PT}_{(.)}(v_2)$.*

5 Probabilistic FPC

In this section we define the language $\mathsf{FPC}_\oplus$, its typing rules, and operational semantics in CCTT. $\mathsf{FPC}_\oplus$ is the extension of simply typed lambda calculus with recursive types and probabilistic choice. The typing rules of $\mathsf{FPC}_\oplus$ are presented in Figure 2. In typing judgements, Γ is assumed to be a variable context and all types (which include recursive types of the form $\mu X.\tau$) are closed.

We assume a given representation of terms and types of $\mathsf{FPC}_\oplus$ within CCTT. For example, as inductive types with the notion of closedness defined as decidable properties on these types. We will also assume that terms are annotated with enough types so that one can deduce the type of subterms from terms. This allows the evaluation function and the denotational semantics to be defined by induction on terms rather than typing judgement derivations. It also means the typing judgement $\Gamma \vdash M : \sigma$ is decidable. We write Ty for the type of closed types, and $\mathsf{Tm}^\Gamma_\sigma$ for the type of

$$
\begin{aligned}
\mathsf{eval}^\kappa(V) &\triangleq \delta^\kappa V \\
\mathsf{eval}^\kappa(\mathsf{suc}\,M) &\triangleq \mathsf{D}^\kappa(\mathbf{suc})(\mathsf{eval}^\kappa M) \\
\mathsf{eval}^\kappa(\mathsf{inl}\,M) &\triangleq \mathsf{D}^\kappa(\mathbf{inl})(\mathsf{eval}^\kappa M) \\
\mathsf{eval}^\kappa(\mathsf{ifz}(L, M, N)) &\triangleq \mathsf{eval}^\kappa L \mathbin{>\!\!>\!\!=^\kappa} \begin{cases} \underline{0} \mapsto \mathsf{eval}^\kappa(M) \\ \underline{n+1} \mapsto \mathsf{eval}^\kappa(N) \end{cases} \\
\mathsf{eval}^\kappa(\mathsf{fst}\,M) &\triangleq (\mathsf{eval}^\kappa M) \mathbin{>\!\!>\!\!=^\kappa} \lambda\langle V, W\rangle.\delta^\kappa V \\
\mathsf{eval}^\kappa\langle M, N\rangle &\triangleq \mathsf{eval}^\kappa M \mathbin{>\!\!>\!\!=^\kappa} \lambda V.\mathsf{eval}^\kappa N \mathbin{>\!\!>\!\!=^\kappa} \lambda W.\delta^\kappa\langle V, W\rangle \\
\mathsf{eval}^\kappa(\mathsf{case}(L, x.M, y.N)) &\triangleq \mathsf{eval}^\kappa(L) \mathbin{>\!\!>\!\!=^\kappa} \begin{cases} \mathsf{inl}\,V \mapsto \mathsf{step}^\kappa(\lambda(\alpha\!:\!\kappa).\mathsf{eval}^\kappa(M[V/x])) \\ \mathsf{inr}\,V \mapsto \mathsf{step}^\kappa(\lambda(\alpha\!:\!\kappa).\mathsf{eval}^\kappa(N[V/y])) \end{cases} \\
\mathsf{eval}^\kappa(M\,N) &\triangleq \mathsf{eval}^\kappa(M) \mathbin{>\!\!>\!\!=^\kappa} \lambda(\mathsf{lam}\,x.M').\ \mathsf{eval}^\kappa(N) \\
&\qquad \mathbin{>\!\!>\!\!=^\kappa} \lambda V.\mathsf{step}^\kappa(\lambda(\alpha\!:\!\kappa).\mathsf{eval}^\kappa(M'[V/x])) \\
\mathsf{eval}^\kappa(\mathsf{fold}\,M) &\triangleq \mathsf{D}^\kappa(\mathsf{fold}\,)(\mathsf{eval}^\kappa(M)) \\
\mathsf{eval}^\kappa(\mathsf{unfold}\,M) &\triangleq \mathsf{eval}^\kappa M \mathbin{>\!\!>\!\!=^\kappa} \lambda(\mathsf{fold}\,V).\mathsf{step}^\kappa(\lambda(\alpha\!:\!\kappa).\delta^\kappa V) \\
\mathsf{eval}^\kappa(\mathsf{choice}^p(M, N)) &\triangleq (\mathsf{eval}^\kappa M) \oplus^\kappa_p (\mathsf{eval}^\kappa N)
\end{aligned}
$$

Fig. 3. The evaluation function eval^κ. Rules for pred, inr, snd omitted.

terms M satisfying $\Gamma \vdash M : \sigma$, constructed as a subtype of the type of all terms. For empty Γ we write simply Tm_σ. We write Val_σ for the set of closed values of type σ, as captured by the grammar

$$V, W := \langle\rangle \mid \underline{n} \mid \mathsf{lam}\,x.M \mid \mathsf{fold}\,V \mid \langle V, W\rangle \mid \mathsf{inl}\,V \mid \mathsf{inr}\,V$$

The *operational semantics* is given by a function eval^κ of type

$$\mathsf{eval}^\kappa : \{\sigma : \mathsf{Ty}\} \to \mathsf{Tm}_\sigma \to \mathsf{D}^\kappa(\mathsf{Val}_\sigma), \tag{6}$$

associating to a term M with type σ an element of the free guarded convex delay algebra over Val_σ. The corresponding element of the coinductive convex delay algebra can be defined as

$$\mathsf{eval}(M) \triangleq \Lambda\kappa.\mathsf{eval}^\kappa(M) : \mathsf{D}^\forall(\mathsf{Val}_\sigma).$$

The function eval^κ is defined by an outer guarded recursion and an inner induction on terms. In other words, by first assuming given a term of type $\triangleright^\kappa(\{\sigma : \mathsf{Ty}\} \to \mathsf{Tm}_\sigma \to \mathsf{D}^\kappa(\mathsf{Val}_\sigma))$, then defining $\mathsf{eval}^\kappa(M)$ by induction on M. The cases are given in Figure 3. The figure overloads names of some term constructors to functions of values, e.g. $\mathbf{inl} : \mathsf{Val}_\sigma \to \mathsf{Val}_{\sigma+\tau}$ mapping V to $\mathsf{inl}\,V$, and $\mathbf{pred}, \mathbf{suc} : \mathsf{Val}_{\mathsf{Nat}} \to \mathsf{Val}_{\mathsf{Nat}}$ defined using $\mathsf{Val}_{\mathsf{Nat}} \simeq \mathbb{N}$. The figure also uses pattern matching within bindings, e.g. in the case of function application, where we use that all values of function type are of the form $\mathsf{lam}\,x.M'$ for some M'.

In the cases of $\mathsf{case}(L, x.M, y.N)$ and $M\,N$ the recursive calls to eval^κ are under step^κ and so can be justified by guarded recursion. This is necessary, because they are not instantiated at a structurally smaller term. The case for $\mathsf{unfold}\,M$ also introduces a computation step using step^κ. While this is not strictly necessary to define the operational semantics, we introduce it to simulate the steps of the denotational semantics defined in Section 6. This simplifies the proof of Theorem 6.1.

Example 5.1. Define the fixed point combinator $\cdot \vdash \mathsf{Y} : ((\sigma \to \tau) \to (\sigma \to \tau)) \to \sigma \to \tau$ by $\mathsf{Y} \triangleq \mathsf{lam}\, f.\mathsf{lam}\, z.e_f(\mathsf{fold}\, e_f)z$, where

$$e_f : (\mu X.(X \to \sigma \to \tau)) \to \sigma \to \tau$$
$$e_f \triangleq \mathsf{lam}\, y.\mathsf{let}\, y' = \mathsf{unfold}\, y \,\mathsf{in}\, f(\mathsf{lam}\, x.y'yx)$$

Here, $y : \mu X.(X \to \sigma \to \tau)$, $y' : (\mu X.(X \to \sigma \to \tau)) \to \sigma \to \tau$ and $\mathsf{lam}\, x.y'yx : \sigma \to \tau$.

Then, for any values $\cdot \vdash f : (\sigma \to \tau) \to \sigma \to \tau$ and $\cdot \vdash V : \sigma$,

$$\mathsf{eval}^\kappa((\mathsf{Y}f)(V)) = (\Delta^\kappa)^4(\mathsf{eval}^\kappa((f(\mathsf{Y}f))(V)))$$

where $\Delta^\kappa \triangleq (\mathsf{step}^\kappa \circ \mathsf{next}^\kappa)$.

Contextual Refinement. Two terms M, N are contextually equivalent if for any closing context $C[-]$ of ground type the terms $C[M]$ and $C[N]$ have the same observational behaviour. Here, the only observable behaviour we consider is the probability of termination for programs of type 1, which, for a closed term M should be the limit of the sequence $\mathsf{PT}_n(\mathsf{eval}\, M)$.

Definition 5.2 (Contextual Refinement). Let $\Gamma \vdash M, N : \tau$ be terms. We say that M *contextually refines* N if for any closing context of unit type $C : (\Gamma \vdash \sigma) \Rightarrow (\cdot \vdash 1)$, $\mathsf{PT}_{(.)}(\mathsf{eval}(C[M])) \leq_{\mathrm{lim}} \mathsf{PT}_{(.)}(\mathsf{eval}(C[N]))$. In this case, write $M \preceq_{\mathsf{Ctx}} N$. We say that M and N are contextually equivalent ($M \equiv_{\mathsf{Ctx}} N$) if $M \preceq_{\mathsf{Ctx}} N$ and $N \preceq_{\mathsf{Ctx}} M$.

Definition 5.2 should be read as defining a predicate on M and N in CCTT, *i.e.*, a function $\mathsf{Tm}^\Gamma_\sigma \to \mathsf{Tm}^\Gamma_\sigma \to \mathsf{Prop}$. The notion of closing context $C : (\Gamma \vdash \sigma) \Rightarrow (\cdot \vdash 1)$ is a special case of a typing judgement on contexts $C : (\Gamma \vdash \sigma) \Rightarrow (\Delta \vdash \tau)$ defined in a standard way. Note that $C[-]$ may capture free variables in M.

6 Denotational Semantics

We now define a *denotational semantics* for $\mathsf{FPC}_\oplus$. The semantics is based on two ideas: (i) we use guarded recursion to interpret recursive types, inspired by [29], and (ii) we use the guarded convex delay monad to interpret effectful computations, similarly to how monads are used to interpret effectful languages in standard denotational semantics. Specifically we define two functions

$$[\![-]\!]^\kappa : \mathsf{Ty} \to \mathsf{Set}$$
$$[\![-]\!]^\kappa_- : \{\Gamma : \mathsf{Ctx}\} \to \{\sigma : \mathsf{Ty}\} \to \mathsf{Tm}^\Gamma_\sigma \to [\![\Gamma]\!]^\kappa \to \mathsf{D}^\kappa[\![\sigma]\!]^\kappa$$

which interpret types and terms, respectively, see Figure 4. Note that by using guarded recursion to interpret types, it suffices to define the denotation of closed types.

The interpretation of terms is defined by induction on terms. Note that all syntactic values are interpreted as semantic values in the sense that we can define a map

$$[\![-]\!]^{\mathsf{Val},\kappa} : \{\sigma : \mathsf{Ty}\} \to \mathsf{Val}_\sigma \to [\![\sigma]\!]^\kappa$$

satisfying $[\![V]\!]^\kappa = \delta^\kappa([\![V]\!]^{\mathsf{Val},\kappa})$ for all V. Note also that the denotational semantics only steps when unfolding from a recursive type. As a consequence, the denotational semantics is much less cluttered by steps than the operational semantics, which makes it simpler to use for reasoning. Despite the fewer steps, the two semantics refine each other. In particular we have the theorem below, which we will prove using the techniques we develop in later sections:

THEOREM 6.1. *For any well-typed closed expression $\cdot \vdash M : 1$ we have that*

$$\mathsf{PT}_{(.)}(\mathsf{eval}\, M) =_{\mathrm{lim}} \mathsf{PT}_{(.)}([\![M]\!])$$

where $[\![M]\!] \triangleq \Lambda\kappa.[\![M]\!]^\kappa$

Interpretation of types

$$[\![\mathsf{Nat}]\!]^\kappa \triangleq \mathbb{N} \qquad [\![\sigma \times \tau]\!]^\kappa \triangleq [\![\sigma]\!]^\kappa \times [\![\tau]\!]^\kappa \qquad [\![\sigma \to \tau]\!]^\kappa \triangleq [\![\sigma]\!]^\kappa \to \mathsf{D}^\kappa[\![\tau]\!]^\kappa$$
$$[\![1]\!]^\kappa \triangleq 1 \qquad [\![\sigma + \tau]\!]^\kappa \triangleq [\![\sigma]\!]^\kappa + [\![\tau]\!]^\kappa \qquad [\![\mu X.\tau]\!]^\kappa \triangleq \triangleright^\kappa[\![\tau[\mu X.\tau/X]]\!]^\kappa$$

Interpretation of terms

$$[\![\langle\rangle]\!]^\kappa_\rho \triangleq \delta^\kappa(\star) \quad [\![\underline{n}]\!]^\kappa_\rho \triangleq \delta^\kappa(n) \quad [\![x]\!]^\kappa_\rho \triangleq \delta^\kappa(\rho(x))$$
$$[\![\mathsf{suc}\, M]\!]^\kappa_\rho \triangleq \mathsf{D}^\kappa(\mathbf{suc})\left([\![\Gamma \vdash M : \mathsf{Nat}]\!]^\kappa_\rho\right)$$
$$[\![\mathsf{ifz}(t, M, N)]\!]^\kappa_\rho \triangleq [\![L]\!]^\kappa_\rho \gg\!\!=^\kappa \begin{cases} 0 & \mapsto [\![M]\!]^\kappa_\rho \\ n+1 & \mapsto [\![N]\!]^\kappa_\rho \end{cases}$$
$$[\![\mathsf{lam}\, x.M]\!]^\kappa_\rho \triangleq \delta^\kappa\left(\lambda(v : [\![\sigma]\!]^{\mathsf{Val},\kappa}_\rho).[\![M]\!]^\kappa_{\rho.x\mapsto v}\right)$$
$$[\![M\, N]\!]^\kappa_\rho \triangleq [\![M]\!]^\kappa_\rho \gg\!\!=^\kappa \lambda f.[\![N]\!]^\kappa_\rho \gg\!\!=^\kappa \lambda v.fv$$
$$[\![\langle M, N\rangle]\!]^\kappa_\rho \triangleq [\![M]\!]^\kappa_\rho \gg\!\!=^\kappa \lambda v.\left([\![N]\!]^\kappa_\rho \gg\!\!=^\kappa \lambda w.\delta^\kappa(v, w)\right)$$
$$[\![\mathsf{fst}\, M]\!]^\kappa_\rho \triangleq \mathsf{D}^\kappa(\mathsf{pr}_1)\left([\![M]\!]^\kappa_\rho\right)$$
$$[\![\mathsf{inl}\, M]\!]^\kappa_\rho \triangleq \mathsf{D}^\kappa(\mathbf{inl})\left([\![M]\!]^\kappa_\rho\right)$$
$$[\![\mathsf{case}(L, x.M, y.N)]\!]^\kappa_\rho \triangleq [\![L]\!]^\kappa_\rho \gg\!\!=^\kappa \begin{cases} \mathbf{inl}\, v \mapsto [\![M]\!]^\kappa_{\rho.x\mapsto v} \\ \mathbf{inr}\, v \mapsto [\![N]\!]^\kappa_{\rho.y\mapsto v} \end{cases}$$
$$[\![\mathsf{fold}\, M]\!]^\kappa_\rho \triangleq \mathsf{D}^\kappa(\mathsf{next}^\kappa)([\![M]\!]^\kappa_\rho)$$
$$[\![\mathsf{unfold}\, M]\!]^\kappa_\rho \triangleq [\![M]\!]^\kappa_\rho \gg\!\!=^\kappa \lambda v.\mathsf{step}^\kappa(\lambda(\alpha : \kappa).\delta^\kappa(v\,[\alpha]))$$
$$[\![\mathsf{choice}^p(M, N)]\!]^\kappa_\rho \triangleq [\![M]\!]^\kappa_\rho \oplus^\kappa_p [\![N]\!]^\kappa_\rho$$

Fig. 4. Denotational semantics for $\mathsf{FPC}_\oplus$. Rules for pred, inr, snd omitted.

The denotational semantics satisfies the following substitution lemma:

Lemma 6.2 (Substitution Lemma). *For any well-typed term $\Gamma.(x : \sigma) \vdash M : \tau$ as well as every well typed value $\Gamma \vdash V : \sigma$ we have*

$$[\![M[V/x]]\!]^\kappa_\rho = [\![M]\!]^\kappa_{\rho.x\mapsto [\![V]\!]^{\mathsf{Val},\kappa}_\rho}$$

The following example presents a useful equality to work with denotational semantics of recursive functions defined via a fixed point combinator:

Example 6.3. Recall the fixed point operator Y from Example 5.1. The denotational semantics satisfies the following equations

$$[\![(\mathsf{lam}\, y.M)\, x]\!]^\kappa_{x\mapsto v} = [\![M]\!]^\kappa_{y\mapsto v} \tag{7}$$
$$[\![\mathsf{Y} f\, x]\!]^\kappa_{x\mapsto v} = (\mathsf{step}^\kappa \circ \mathsf{next}^\kappa)([\![f(\mathsf{Y} f)\, x]\!]^\kappa_{x\mapsto v}) \tag{8}$$

for any value f and semantic value v, and where x does not appear free in M. By applying the substitution lemma, we also get

$$[\![(\mathsf{lam}\ y.M)\ V]\!]^\kappa = [\![M[V/x]]\!]^\kappa$$
$$[\![\mathsf{Y} f\ V]\!]^\kappa = (\mathsf{step}^\kappa \circ \mathsf{next}^\kappa)([\![f(\mathsf{Y}f)\ V]\!]^\kappa)$$

7 Couplings and Lifting Relations

Our next goal is to define a logical relation between the syntax and semantics and use it to reason about contextual equivalence of $\mathsf{FPC}_\oplus$ terms. This is done by defining a value relation (of type $[\![\sigma]\!]^\kappa \to \mathsf{Val}_\sigma \to \mathsf{Prop}$) and an expression relation (of type $\mathsf{D}^\kappa[\![\sigma]\!]^\kappa \to \mathsf{D}^\forall(\mathsf{Val}_\sigma) \to \mathsf{Prop}$) by mutual recursion. We start by defining a relational lifting [25] operation mapping a relation $\mathcal{R} : A \to B \to \mathsf{Prop}$ on values to a relation $\overline{\mathcal{R}}^\kappa : \mathsf{D}^\kappa A \to \mathsf{D}^\forall B \to \mathsf{Prop}$ on computations. This will be used to define the expression relation as the relational lifting of the value relation. As opposed to standard lifting constructions, that consider the same monad on both sides of the relation, there is an asymmetry in the type of $\overline{\mathcal{R}}^\kappa$. The reason is that the logical relation will be defined by guarded recursion in the first argument, while using arbitrarily deep unfoldings of the term on the right-hand side. This section defines the relational lifting construction and proves its basic properties.

The first step is lifting a relation $\mathcal{R}$ over $A \times B$ to a relation over $\mathcal{D}A \times \mathcal{D}B$. Recall the notion of $\mathcal{R}$-coupling [6, 28], which achieves this precise end:

Definition 7.1. Let $\mathcal{R} : A \to B \to \mathsf{Prop}$, and let $\mu : \mathcal{D}A, \nu : \mathcal{D}B$ be finite distributions. An $\mathcal{R}$-coupling between μ and ν is a distribution on the total space of $\mathcal{R}$, *i.e.*, $\rho : \mathcal{D}(\Sigma(a{:}A), (b{:}B).a\,\mathcal{R}\,b)$, whose marginals are μ and ν:

$$\mathcal{D}(\mathsf{pr}_1)(\rho) = \mu \qquad\qquad \mathcal{D}(\mathsf{pr}_2)(\rho) = \nu,$$

where pr_1 and pr_2 are the projection maps $A \xleftarrow{\mathsf{pr}_1} A \times B \xrightarrow{\mathsf{pr}_2} B$. We write $\mathsf{Cpl}_\mathcal{R}(\mu, \nu)$ for the type of $\mathcal{R}$-couplings between μ and ν.

In other words, an $\mathcal{R}$-coupling is a joint distribution over $A \times B$ with the property that $\mathcal{R}$ always holds for any pair of values sampled from it. Our definition of $\mu\,\overline{\mathcal{R}}^\kappa\,\nu$ will generalise this idea for pairs of computations, whose final values might become available at different times. Recall that by Theorem 3.6 a distribution $\mu : \mathsf{D}^\kappa A \simeq \mathcal{D}(A + \triangleright^\kappa(\mathsf{D}^\kappa A))$ must be either a distribution of values, one of delayed computations or a convex combination of the two. The relation $\mu\,\overline{\mathcal{R}}^\kappa\,\nu$ should then hold if (1) the values available in μ now can be matched by ν after possibly some computation steps, and (2) the delayed computation part of μ can be matched later, in a guarded recursive step. The matching of values is done via an $\mathcal{R}$-coupling, and the computation steps are interpreted using $\rightsquigarrow^\approx$. We choose $\rightsquigarrow^\approx$ over the more straightforward $\rightsquigarrow$ to allow the matching values for μ to be delivered in the limit of an infinite sequence of computation steps. By choosing $\rightsquigarrow^\approx$, we just require that any fraction of it can be delivered in finite time. In the following definition, we leave the inclusions $\mathcal{D}(\mathsf{inl})$ and $\mathcal{D}(\mathsf{inr})$ implicit and simply write, e.g., $\mu : \mathcal{D}A$ for the first case mentioned above.

Definition 7.2 (Lifting of Relations). Given a relation $\mathcal{R} : A \to B \to \mathsf{Prop}$ we define its lift $\overline{\mathcal{R}}^\kappa : \mathsf{D}^\kappa A \to \mathsf{D}^\forall B \to \mathsf{Prop}$ as: $\mu\,\overline{\mathcal{R}}^\kappa\,\nu$ if one of the following three options is true:

(1) $\mu : \mathcal{D}A$, there is a $\nu' : \mathcal{D}B$ such that $\nu \rightsquigarrow^\approx \nu'$, and there is a $\rho : \mathsf{Cpl}_\mathcal{R}(\mu, \nu')$.
(2) $\mu : \mathcal{D}(\triangleright^\kappa \mathsf{D}^\kappa A)$ and $\triangleright^\kappa(\alpha : \kappa)\ (((\zeta^\kappa(\mu))[\alpha])\,\overline{\mathcal{R}}^\kappa\,\nu)$,
where $\zeta^\kappa : \mathcal{D}(\triangleright^\kappa \mathsf{D}^\kappa A) \to \triangleright^\kappa \mathsf{D}^\kappa A$ is defined as

$$\zeta^\kappa(\delta(x)) \triangleq x \qquad\qquad \zeta^\kappa(\mu \oplus_p \nu) \triangleq \lambda(\alpha{:}\kappa).\zeta^\kappa(\mu)\ [\alpha] \oplus^\kappa_p \zeta^\kappa(\nu)\ [\alpha]$$

(3) There exist $\mu_1 : \mathcal{D}A, \mu_2 : \mathcal{D}(\triangleright^\kappa \mathsf{D}^\kappa A)$, and $p \in (0,1)$ such that $\mu = \mu_1 \oplus_p \mu_2$, there exist $\nu_1, \nu_2 : \mathsf{D}^\forall B$ such that $\nu \rightsquigarrow^\approx \nu_1 \oplus^\forall_p \nu_2$, and $\mu_1 \overline{\mathcal{R}}^\kappa \nu_1$ and $\mu_2 \overline{\mathcal{R}}^\kappa \nu_2$.

We then define $\overline{\mathcal{R}} : \mathsf{D}^\forall A \to \mathsf{D}^\forall B \to \mathsf{Prop}$ as:

$$\mu \overline{\mathcal{R}} \nu \triangleq \forall \kappa.(\mu[\kappa] \overline{\mathcal{R}}^\kappa \nu).$$

The following is a consequence of the encoding of coinductive types using guarded recursion.

Lemma 7.3. *If A is clock irrelevant, then $\overline{\mathcal{R}}$ is the coinductive solution to the equation that $\mu \overline{\mathcal{R}} \nu$ if and only if one of the following:*

(1) $\mu : \mathcal{D}A$, there is a $\nu' : \mathcal{D}B$ such that $\nu \rightsquigarrow^\approx \nu'$, and there is a $\rho : \mathsf{Cpl}_{\mathcal{R}}(\mu, \nu')$.
(2) $\mu : \mathcal{D}(\mathsf{D}^\forall A)$ and $\mathsf{run}(\mu) \overline{\mathcal{R}} \nu$.
(3) There exist $\mu_1 : \mathcal{D}A, \mu_2 : \mathcal{D}(\mathsf{D}^\forall A)$, and $p : (0,1)$ such that $\mu = \mu_1 \oplus_p \mu_2$, there exist $\nu_1, \nu_2 : \mathsf{D}^\forall B$ such that $\nu \rightsquigarrow^\approx \nu_1 \oplus^\forall_p \nu_2$, and $\mu_1 \overline{\mathcal{R}} \nu_1$ and $\mu_2 \overline{\mathcal{R}} \nu_2$.

The following lemma allows us to use existence of a lifting to prove an inequality between probabilities of termination, which will be useful when reasoning about contextual refinement

Lemma 7.4. *Let $\mu, \nu : \mathsf{D}^\forall 1$, and let $\mathsf{eq}_1 : 1 \to 1 \to \mathsf{Prop}$ be the identity relation, relating the unique element to itself. Then,*

$$\mu \overline{\mathsf{eq}_1} \nu \Rightarrow \mathsf{PT}_{(.)}(\mu) \leq_{\lim} \mathsf{PT}_{(.)}(\nu).$$

In the remainder of this section, we will prove some useful reasoning principles for working with $\overline{\mathcal{R}}^\kappa$ at a more abstract level, without the need for unfolding the definition. These are summarized in Figure 5, where the double bar indicates that the rule is bidirectional.

$$\frac{\nu \rightsquigarrow \nu' \qquad \mu \overline{\mathcal{R}}^\kappa \nu}{\mu \overline{\mathcal{R}}^\kappa \nu'} \qquad \frac{\nu \rightsquigarrow \nu' \qquad \mu \overline{\mathcal{R}}^\kappa \nu'}{\mu \overline{\mathcal{R}}^\kappa \nu} \qquad \frac{\nu \rightsquigarrow^\approx \nu' \qquad \mu \overline{\mathcal{R}}^\kappa \nu'}{\mu \overline{\mathcal{R}}^\kappa \nu}$$

$$\frac{(\mathsf{step}^\kappa \mu_1) \oplus^\kappa_p (\mathsf{step}^\kappa \mu_2) \overline{\mathcal{R}}^\kappa \nu}{\mathsf{step}^\kappa(\lambda(\alpha{:}\kappa).(\mu_1[\alpha]) \oplus^\kappa_p (\mu_2[\alpha])) \overline{\mathcal{R}}^\kappa \nu} \qquad \frac{\mu \overline{\mathcal{R}}^\kappa \mathsf{step}^\forall(\nu_1 \oplus^\forall_p \nu_2)}{\mu \overline{\mathcal{R}}^\kappa (\mathsf{step}^\forall(\nu_1)) \oplus^\forall_p (\mathsf{step}^\forall(\nu_2))}$$

$$\frac{\mu_1 \overline{\mathcal{R}}^\kappa \nu_1 \qquad \mu_2 \overline{\mathcal{R}}^\kappa \nu_2}{(\mu_1 \oplus^\kappa_p \mu_2) \overline{\mathcal{R}}^\kappa (\nu_1 \oplus^\forall_p \nu_2)} \qquad \frac{\mu \overline{\mathcal{R}}^\kappa \overline{\nu} \qquad \forall a, b. a \mathcal{R} b \to f(a) \overline{\mathcal{S}}^\kappa g(b)}{\overline{f}(\mu) \overline{\mathcal{S}}^\kappa \overline{g}(\nu)}$$

Fig. 5. Reasoning principles for $\overline{\mathcal{R}}^\kappa$

First we show that $\overline{\mathcal{R}}^\kappa$ is invariant with respect to $\rightsquigarrow$ and $\rightsquigarrow^\approx$-reductions on its second argument.

Lemma 7.5. *If $\nu \rightsquigarrow \nu'$ then $\mu \overline{\mathcal{R}}^\kappa \nu$ iff $\mu \overline{\mathcal{R}}^\kappa \nu'$. In particular $\mu \overline{\mathcal{R}}^\kappa (\mathsf{step}^\forall \nu)$ iff $\mu \overline{\mathcal{R}}^\kappa \nu$.*

Proof. Left to right follows from Lemma 4.10, and right to left is by Lemma 4.19. □

Lemma 7.6. *If $\nu \rightsquigarrow^\approx \nu'$ and $\mu \overline{\mathcal{R}}^\kappa \nu'$, then also $\mu \overline{\mathcal{R}}^\kappa \nu$.*

Proof. This follows from transitivity of $\rightsquigarrow^\approx$. □

The lemma below allows us to commute computation steps with probabilistic choices on either side of a lifted relation:

Lemma 7.7. *Let* $\mathcal{R} : A \to B \to \mathsf{Prop}$.

(1) If $\mu_1, \mu_2 : \triangleright^\kappa \mathsf{D}^\kappa A$, $\nu : \mathsf{D}^\forall B$ *then*

$$(\mathsf{step}^\kappa \mu_1) \oplus_p^\kappa (\mathsf{step}^\kappa \mu_2)\ \overline{\mathcal{R}}^\kappa\ \nu \text{ iff } \mathsf{step}^\kappa(\lambda(\alpha{:}\kappa).(\mu_1\,[\alpha]) \oplus_p^\kappa (\mu_2\,[\alpha]))\ \overline{\mathcal{R}}^\kappa\ \nu.$$

(2) If $\mu : \mathsf{D}^\kappa A$ *and* $\nu_1, \nu_2 : \mathsf{D}^\forall B$ *then* $\mu\ \overline{\mathcal{R}}^\kappa\ \mathsf{step}^\forall(\nu_1 \oplus_p^\forall \nu_2)$ *iff* $\mu\ \overline{\mathcal{R}}^\kappa\ (\mathsf{step}^\forall(\nu_1)) \oplus_p^\forall (\mathsf{step}^\forall(\nu_2))$.

Proof. The first statement follows from applying Definition 7.2(2). The second statement follows from Lemma 7.5. □

Liftings are a useful technique to reason about computations because of the way they interact with choice, and with the monad structures of D^κ and $\mathsf{D}^\forall$. First, liftings are closed under choice operators, which allows us to construct them by parts:

Lemma 7.8. *Let* $\mathcal{R} : A \to B \to \mathsf{Prop}$, *and suppose that* $\mu_1\ \overline{\mathcal{R}}^\kappa\ \nu_1$ *and* $\mu_2\ \overline{\mathcal{R}}^\kappa\ \nu_2$. *Then also* $(\mu_1 \oplus_p^\kappa \mu_2)\ \overline{\mathcal{R}}^\kappa\ (\nu_1 \oplus_p^\forall \nu_2)$.

Proof (sketch). There are 9 cases to consider, one for each of the possible combinations of cases for $\mu_1\ \overline{\mathcal{R}}^\kappa\ \nu_1$ and $\mu_2\ \overline{\mathcal{R}}^\kappa\ \nu_2$. The proof for each of these cases follows quite directly from the assumptions that we get from $\mu_1\ \overline{\mathcal{R}}^\kappa\ \nu_1$ and $\mu_2\ \overline{\mathcal{R}}^\kappa\ \nu_2$, after some rewriting of terms using the axioms of convex algebras. For example, if $\mu_1 : \mathcal{D}A$ and for μ_2 there exist $\mu_{21} : \mathcal{D}A$, $\mu_{22} : \mathcal{D}(\triangleright^\kappa \mathsf{D}^\kappa A)$ such that $\mu_2 = \mu_{21} \oplus_{p'} \mu_{22}$, then:

$$\mu_1 \oplus_p^\kappa \mu_2 = \mu_1 \oplus_p^\kappa (\mu_{21} \oplus_{p'}^\kappa \mu_{22}) = (\mu_1 \oplus^\kappa_{\frac{p}{p+(1-p)p'}} \mu_{21}) \oplus^\kappa_{p+(1-p)p'} \mu_{22},$$

where $\mu_1 \oplus^\kappa_{\frac{p}{p+(1-p)p'}} \mu_{21} : \mathcal{D}A$ and $\mu_{22} : \mathcal{D}(\triangleright^\kappa \mathsf{D}^\kappa A)$. We can then combine information coming from the assumptions $\mu_1\ \overline{\mathcal{R}}^\kappa\ \nu_1$ and $\mu_2\ \overline{\mathcal{R}}^\kappa\ \nu_2$ using the same probabilities as above to reach the desired conclusion. □

Second, we can prove a bind lemma, that allows us to sequence computations related by liftings. This lemma will be crucial e.g. in the proof of the fundamental lemma in the following section.

Lemma 7.9 (Bind Lemma). *Let* $\mathcal{R} : A \to B \to \mathsf{Prop}$ *and* $\mathcal{S} : A' \to B' \to \mathsf{Prop}$.

(1) If $f : A \to \mathsf{D}^\kappa A'$ *and* $g : B \to \mathsf{D}^\forall B'$ *satisfy* $f(a)\ \overline{\mathcal{S}}^\kappa\ g(b)$ *whenever* $a\,\mathcal{R}\,b$, *then for all* $\mu : \mathcal{D}A$ *and* $\nu : \mathcal{D}B$, *if there is a* $\rho : \mathsf{Cpl}_{\mathcal{R}}(\mu, \nu)$, *then* $\overline{f}(\mu)\ \overline{\mathcal{S}}^\kappa\ \overline{g}(\nu)$.

(2) If $f : A \to \mathsf{D}^\kappa A'$ *and* $g : B \to \mathsf{D}^\forall B'$ *satisfy* $f(a)\ \overline{\mathcal{S}}^\kappa\ g(b)$ *whenever* $a\,\mathcal{R}\,b$, *then for all* $\mu : \mathsf{D}^\kappa A$ *and* $\nu : \mathsf{D}^\forall B$ *satisfying* $\mu\ \overline{\mathcal{R}}^\kappa\ \overline{\nu}$, *also* $\overline{f}(\mu)\ \overline{\mathcal{S}}^\kappa\ \overline{g}(\nu)$.

Proof (sketch). The first statement is by induction on the coupling $\rho : \mathsf{Cpl}_{\mathcal{R}}(\mu, \nu)$.

For the second statement, suppose that $\mu\ \overline{\mathcal{R}}^\kappa\ \nu$. By the definition of $\overline{\mathcal{R}}^\kappa$, there are three cases to consider. We only show the first case here. The second case is by guarded recursion, and the third case follows from the first and second cases.

For the first case, we assume that $\mu : \mathcal{D}A$, that there exist $\nu_1 : \mathcal{D}B$ such that $\nu \rightsquigarrow^\approx \nu_1$, and that there exists a $\rho : \mathsf{Cpl}_{\mathcal{R}}(\mu, \nu_1)$. What we want is to show that $\overline{f}(\mu)\ \overline{\mathcal{S}}^\kappa\ \overline{g}(\nu)$.

Notice that since the $\rightsquigarrow^\approx$ relation is preserved by homomorphisms (Lemma 4.18), $\nu \rightsquigarrow^\approx \nu_1$ implies $\overline{g}(\nu) \rightsquigarrow^\approx \overline{g}(\nu_1)$. By Lemma 7.6 it is therefore enough to show that $\overline{f}(\mu)\ \overline{\mathcal{S}}^\kappa\ \overline{g}(\nu_1)$, which follows directly from the first statement of this lemma.

□

Value relation

$$\frac{}{n \preceq^{\kappa,\mathsf{Val}}_{\mathsf{Nat}} \underline{n}} \qquad \frac{}{\star \preceq^{\kappa,\mathsf{Val}}_{1} \langle\rangle} \qquad \frac{v \preceq^{\kappa,\mathsf{Val}}_{\sigma} V \qquad w \preceq^{\kappa,\mathsf{Val}}_{\tau} W}{(v,w) \preceq^{\kappa,\mathsf{Val}}_{\sigma\times\tau} \langle V, W\rangle} \qquad \frac{v \preceq^{\kappa,\mathsf{Val}}_{\sigma} V}{\mathbf{inl}\, v \preceq^{\kappa,\mathsf{Val}}_{\sigma+\tau} \mathsf{inl}\, V}$$

$$\frac{v \preceq^{\kappa,\mathsf{Val}}_{\tau} V}{\mathbf{inr}\, v \preceq^{\kappa,\mathsf{Val}}_{\sigma+\tau} \mathsf{inr}\, V} \qquad \frac{\forall w, V. w \preceq^{\kappa,\mathsf{Val}}_{\sigma} V \to v(w) \preceq^{\kappa,\mathsf{Tm}}_{\tau} \mathsf{eval}(M[V/x])}{v \preceq^{\kappa,\mathsf{Val}}_{\sigma\to\tau} \mathsf{lam}\, x.M}$$

$$\frac{\triangleright(\alpha:\kappa).(v[\alpha] \preceq^{\kappa,\mathsf{Val}}_{\tau[\mu X.\tau/X]} V)}{v \preceq^{\kappa,\mathsf{Val}}_{\mu X.\tau} \mathsf{fold}\, V}$$

Expression relation

$$\mu \preceq^{\kappa,\mathsf{Tm}}_{\sigma} d \triangleq \mu \overline{\preceq^{\kappa,\mathsf{Val}}_{\sigma}}^{\kappa} d$$

Fig. 6. Logical Relation.

8 Relating Syntax and Semantics

In this section we prove that our denotational semantics is adequate with respect to the operational semantics. The overall approach is standard in that we define type-indexed logical relations between semantic denotations and terms of the operational semantics, and we define both value relations and expression (aka term or computation) relations:

$$\begin{aligned} \preceq^{\kappa,\mathsf{Val}}_{\sigma} &: [\![\sigma]\!]^{\kappa} \to \mathsf{Val}_{\sigma} \to \mathsf{Prop} \\ \preceq^{\kappa,\mathsf{Tm}}_{\sigma} &: \mathsf{D}^{\kappa}[\![\sigma]\!]^{\kappa} \to \mathsf{D}^{\forall}(\mathsf{Val}_{\sigma}) \to \mathsf{Prop} \end{aligned}$$

There are two things to note, however. The first is that traditionally it is tricky to define such logical relations for a programming language with recursive types; see, e.g., [34]. Here we follow [29] and simply use guarded recursion (and induction on types), see Figure 6. Notice in particular the case for recursive types, where guarded recursion is used. The second point to note is that the definition of the term relation is novel: it is defined as the *lifting* (in the sense of the previous section) of the value relation. This allows us to use the reasoning principles associated with lifting (e.g., Lemmas 7.5,7.8 and 7.9) in proofs and examples.

We extend the expression relation to open terms in the standard way by using related environments and closing substitutions:

Definition 8.1. For $M, N : \mathsf{Tm}^{\Gamma}_{\sigma}$ we define

$$M \preceq^{\kappa,\Gamma}_{\sigma} N \triangleq \left(\forall \rho, \delta. (\rho \preceq^{\kappa,\mathsf{Val}}_{\Gamma} \delta) \to [\![M]\!]^{\kappa}_{\rho} \preceq^{\kappa,\mathsf{Tm}}_{\sigma} \mathsf{eval}(N[\delta])\right)$$

Here δ is a closing substitution for Γ and $\rho : [\![\Gamma]\!]^{\kappa}$ an environment, and $\rho \preceq^{\kappa,\mathsf{Val}}_{\Gamma} \delta$ denotes that for every variable x of Γ the corresponding semantic and syntactic value in ρ and δ are related.

One can now show that the logical relation is a congruence and that it is reflexive (the fundamental lemma holds). Both these are proved by induction on C and M respectively.

Lemma 8.2 (Congruence Lemma). *For any terms $M, N : \mathsf{Tm}^{\Gamma}_{\sigma}$ and every context $C : (\Gamma \vdash \sigma) \Rightarrow (\Delta \vdash \tau)$ we get that*

$$M \preceq^{\kappa,\Gamma}_{\sigma} N \to C[M] \preceq^{\kappa,\Delta}_{\tau} C[N]$$

Lemma 8.3 (Fundamental Lemma). *For all $M : \mathsf{Tm}^{\Gamma}_{\sigma}$ we have that $M \preceq^{\kappa,\Gamma}_{\sigma} M$.*

To prove soundness of the logical relation for contextual equivalence, it only remains to relate it to termination probability at the unit type. To begin, we use our logical relation to prove Theorem 6.1 presented in Section 6:

Proof of Theorem 6.1. Recall that our goal is to show that $\mathsf{PT}_{(.)}(\mathsf{eval}\,M) =_{\lim} \mathsf{PT}_{(.)}(\llbracket M \rrbracket)$ for any $M : \mathsf{Tm}_1$. One direction follows directly from Lemma 8.3: Since $\forall\kappa.\llbracket M \rrbracket^\kappa \ \preceq_1^{\kappa,\mathsf{Tm}}\ \mathsf{eval}\,M$, by Lemma 7.4 also $\mathsf{PT}_{(.)}(\llbracket M \rrbracket) \leq_{\lim} \mathsf{PT}_{(.)}(\mathsf{eval}\,M)$. For the other direction, we define an alternative denotational semantics $(\!|-|\!)$, which agrees with $\llbracket - \rrbracket$ on types and most terms, but which uses the same steps as the operational semantics, so for example,

$$(\!|MN|\!)^\kappa_\rho \triangleq (\!|M|\!)^\kappa_\rho \gg\!=^\kappa \lambda f.(\!|N|\!)^\kappa_\rho \gg\!=^\kappa \lambda v.\mathsf{step}^\kappa(\lambda(\alpha\!:\!\kappa).fv)$$

and similarly for the interpretation of elimination for sum types. Since the steps agree, it is easy to show the following soundness result for $(\!|-|\!)$: For any well typed closed term $\cdot \vdash M : \sigma$ we have

$$\mathsf{D}^\kappa\Big((\!|-|\!)^{\mathsf{Val},\kappa}\Big)\,(\mathsf{eval}^\kappa M) \equiv (\!|M|\!)^\kappa$$

and as a consequence $\mathsf{PT}_{(.)}(\mathsf{eval}\,M) = \mathsf{PT}_{(.)}((\!|M|\!))$. Finally, we construct a logical relation on the denotation of types to show that $\mathsf{PT}_{(.)}((\!|M|\!)) \leq_{\lim} \mathsf{PT}_{(.)}(\llbracket M \rrbracket)$. □

As a consequence, the logical relation implies relatedness of the termination probabilities.

Corollary 8.4. *For any* $M, N : \mathsf{Tm}_1$ *we have that*

$$(\forall\kappa.\llbracket M \rrbracket^\kappa \ \preceq_1^{\kappa,\mathsf{Tm}}\ \mathsf{eval}(N)) \to \mathsf{PT}_{(.)}(\mathsf{eval}(M)) \leq_{\lim} \mathsf{PT}_{(.)}(\mathsf{eval}(N))$$

Proof. By Lemma 7.4, $\mathsf{PT}_{(.)}(\llbracket M \rrbracket) \leq_{\lim} \mathsf{PT}_{(.)}(\mathsf{eval}(N))$, so the result follows by Theorem 6.1. □

Combining this with the Congruence Lemma, we get that the logical relation is sound with respect to contextual refinement:

Theorem 8.5. *For any terms* $\Gamma \vdash M : \sigma$ *and* $\Gamma \vdash N : \sigma$ *we have*

$$(\forall\kappa.M \preceq_\sigma^{\kappa,\Gamma} N) \to M \preceq_{\mathsf{Ctx}} N$$

As a corollary, we obtain computational adequacy (using the notation $\llbracket M \rrbracket \triangleq \Lambda\kappa.\llbracket M \rrbracket^\kappa$):

Theorem 8.6 (Adequacy). *Let* M, N *be terms of the same type. If* $\llbracket M \rrbracket = \llbracket N \rrbracket$ *then* $M \equiv_{\mathsf{Ctx}} N$.

Proof. By Lemma 8.3 we have $\forall\kappa.N \preceq_\sigma^{\kappa,\Gamma} N$, since κ is free in the statement of the lemma. So, since $\llbracket M \rrbracket = \llbracket N \rrbracket$ also $\forall\kappa.M \preceq_\sigma^{\kappa,\Gamma} N$. By Theorem 8.5 also $M \preceq_{\mathsf{Ctx}} N$. We conclude by symmetry. □

9 Examples

We now give a series of examples to illustrate how Theorem 8.5 can be used for reasoning about $\mathsf{FPC}_\oplus$. All the examples presented are variants of similar examples in the literature [1, 9]. Our point is to show how these examples can be done in constructive type theory, and to illustrate the simplicity of these in our abstract viewpoint. We choose to work directly with the logical relation $\preceq_\sigma^{\kappa,\Gamma}$ between syntax and semantics. An alternative could be to use a relation on denotational semantics, as the one used in the proof of Theorem 6.1. One could have also defined a relation directly from syntax to syntax, but the additional steps in the operational semantics would have cluttered the proofs. We return to this point in Remark 3.

We start by showing that the reduction relation $\rightsquigarrow$ is contained in contextual equivalence.

Theorem 9.1. *Let* M *and* N *be closed terms of the same type* σ.

(1) If $\mathsf{eval}\,M \rightsquigarrow \mathsf{eval}\,N$*, then* $M \equiv_{\mathsf{Ctx}} N$.
(2) If $\mathsf{eval}\,M \rightsquigarrow^{\approx} \mathsf{eval}\,N$*, then* $N \preceq_{\mathsf{Ctx}} M$.

Proof. For (1), by Lemma 7.5, $\mu \preceq_{\sigma}^{\kappa,\mathsf{Val}} \mathsf{eval}\, M$ if and only if $\mu \preceq_{\sigma}^{\kappa,\mathsf{Val}} \mathsf{eval}\, N$. By Lemma 8.3 $\forall\kappa.[\![M]\!]^{\kappa} \preceq_{\sigma}^{\kappa,\mathsf{Val}} \mathsf{eval}\, M$, so $\forall\kappa.[\![M]\!]^{\kappa} \preceq_{\sigma}^{\kappa,\mathsf{Val}} \mathsf{eval}\, N$, and so by Theorem 8.5 also $M \preceq_{\mathsf{Ctx}} N$. The other way is similar. The second statement is proved similarly using Lemma 7.6. □

For example, since $\mathsf{eval}(\mathsf{Y} f\, V) \rightsquigarrow \mathsf{eval}(f(\mathsf{Y} f) V)$, also $\mathsf{Y} f\, V \equiv_{\mathsf{Ctx}} f(\mathsf{Y} f)\, V$. Similarly, since

$$\mathsf{eval}((\mathsf{lam}\; x.M)\; V) = \mathsf{step}^{\forall}(\mathsf{eval}(M[V/x])) \rightsquigarrow \mathsf{eval}(M[V/x])$$
$$\mathsf{eval}(\mathsf{unfold}\; (\mathsf{fold}\; V)) = \mathsf{step}^{\forall}(V) \rightsquigarrow V$$

for closed values V and $\mathsf{lam}\; x.M$, we get the usual call-by-value β rules up to contextual equivalence.

9.1 A Hesitant Identity Function

For any type σ and rational number $p : (0,1)$, define the hesitant identity function $\mathsf{hid_p} : \sigma \to \sigma$ as

$$\mathsf{hid_p} \triangleq \mathsf{lam}\; z.\mathsf{Y}\, \mathsf{hid'_p}\, z \qquad \text{where}$$
$$\mathsf{hid'_p} \triangleq \mathsf{lam}\; f.\mathsf{lam}\; x.\mathsf{choice}^{p}(x, f x)$$

Given an x, $\mathsf{hid_p}$ makes a probabilistic choice between immediately returning x and calling itself recursively. We will show that $\mathsf{hid_p} \equiv_{\mathsf{Ctx}} \mathsf{id}$, starting with $\mathsf{id} \preceq_{\mathsf{Ctx}} \mathsf{hid_p}$.

Since $\mathsf{hid_p}$ is a value, it suffices to show that

$$\delta^{\kappa}(v) \preceq_{\sigma}^{\kappa,\mathsf{Tm}} \mathsf{eval}(\mathsf{hid_p}\; V) \tag{9}$$

for all v, V satisfying $v \preceq_{\sigma}^{\kappa,\mathsf{Val}} V$. Now, $\mathsf{eval}(\mathsf{hid_p}\; V) \rightsquigarrow \mathsf{eval}(\mathsf{Y}\, \mathsf{hid'_p}\; V)$, and by definition of Y, $\mathsf{Y}\, \mathsf{hid'_p}$ reduces in one step to a value, which we shall simply write $W_{\mathsf{hid,p}}$ for. To prove (9) it therefore suffices to prove

$$\delta^{\kappa}(v) \preceq_{\sigma}^{\kappa,\mathsf{Tm}} \mathsf{eval}(W_{\mathsf{hid,p}}\; V) \tag{10}$$

Since the reduction $\mathsf{eval}(\mathsf{Y}\, \mathsf{hid'_p}\; V) \rightsquigarrow \mathsf{eval}(\mathsf{hid'_p}\; (\mathsf{Y}\, \mathsf{hid'_p})\; V)$ factors through $W_{\mathsf{hid,p}}\; V$ it follows that

$$\begin{aligned}
\mathsf{eval}(W_{\mathsf{hid,p}}\; V) &\rightsquigarrow \mathsf{eval}(\mathsf{hid'_p}\; (\mathsf{Y}\, \mathsf{hid'_p})\; V)\\
&\rightsquigarrow \mathsf{eval}(\mathsf{hid'_p}\; W_{\mathsf{hid,p}}\; V)\\
&\rightsquigarrow \mathsf{eval}(\mathsf{choice}^{p}(V, W_{\mathsf{hid,p}}\; V))\\
&= \delta^{\forall} V \oplus_{p}^{\forall} \mathsf{eval}(W_{\mathsf{hid,p}}\; V)
\end{aligned}$$

By Example 4.16 then $\mathsf{eval}(W_{\mathsf{hid,p}}\; V) \rightsquigarrow^{\approx} \delta^{\forall} V$, so that (10) follows from Lemma 7.6.

To prove $\mathsf{hid_p} \preceq_{\mathsf{Ctx}} \mathsf{id}$, since $\mathsf{hid_p}$ is a value, it suffices to show that

$$[\![\mathsf{hid_p}]\!]^{\mathsf{Val},\kappa}(v) \preceq_{\sigma}^{\kappa,\mathsf{Tm}} \delta^{\forall}(V) \tag{11}$$

for all v, V satisfying $v \preceq_{\sigma}^{\kappa,\mathsf{Val}} V$. Using (8) we compute

$$\begin{aligned}
[\![\mathsf{hid_p}]\!]^{\mathsf{Val},\kappa}(v) &= [\![\mathsf{Y}(\mathsf{hid'_p})\; z]\!]^{\kappa}_{z \mapsto v}\\
&= \Delta^{\kappa} [\![\mathsf{hid'_p}(\mathsf{Y}(\mathsf{hid'_p}))\; z]\!]^{\kappa}_{z \mapsto v}\\
&= \Delta^{\kappa} [\![\mathsf{choice}^{p}(x, f x)]\!]^{\kappa}_{x \mapsto v, f \mapsto [\![W_{\mathsf{hid,p}}]\!]^{\mathsf{Val},\kappa}}\\
&= \Delta^{\kappa} (\delta^{\kappa} v \oplus_{p}^{\kappa} [\![\mathsf{hid_p}]\!]^{\mathsf{Val},\kappa}(v))
\end{aligned}$$

so that (11) is equivalent to

$$\triangleright^{\kappa}(\delta^{\kappa} v \oplus_{p}^{\kappa} [\![\mathsf{hid_p}]\!]^{\mathsf{Val},\kappa}(v) \preceq_{\sigma}^{\kappa,\mathsf{Tm}} \delta^{\forall}(V)) \tag{12}$$

and can therefore be easily proved by guarded recursion using Lemma 7.7.

REMARK 3. *Note the benefit of having the denotational semantics on the left hand side of the relation in the example above: Unfolding (11) only leads to a single (outer) $\triangleright^\kappa$ in (12). Had we used operational semantics on the left hand side, the unfolding would have been cluttered by additional steps intertwined with probabilistic choices. On the right hand side, the additional steps disappear by the use of $\rightsquigarrow$.*

9.2 A Fair Coin from an Unfair Coin

Define bool $\triangleq 1+1$ referring to the elements as tt and ff. A *coin* is a program of the form $\mathsf{choice}^p(\mathsf{tt},\mathsf{ff}) : \mathsf{bool}$ for $p:(0,1)$. A coin is called *fair* if $p=\frac{1}{2}$.

Given a coin, one can encode a fair coin as follows: Toss the coin twice. If the results are different, return the first result, otherwise try again. This idea can be encoded as the $\mathsf{FPC}_\oplus$ term $\mathsf{efair_p}$:

$$\begin{aligned}\mathsf{efair_p} &\triangleq \mathsf{Y}(\mathsf{efair'_p}) : 1 \to \mathsf{bool}\\ \mathsf{efair'_p} &\triangleq \mathsf{lam}\ g.\mathsf{lam}\ z.\mathsf{let}\ x = \mathsf{choice}^p(\mathsf{tt},\mathsf{ff})\\ &\qquad\qquad\qquad \mathsf{let}\ y = \mathsf{choice}^p(\mathsf{tt},\mathsf{ff})\\ &\qquad\qquad\qquad \mathsf{if}\ \mathsf{eqbool}(x,y)\ \mathsf{then}\ g(z)\ \mathsf{else}\ x\end{aligned}$$

We now prove that this indeed gives a fair coin.

THEOREM 9.2. $\mathsf{efair_p}\langle\rangle$ *is contextually equivalent to* $\mathsf{choice}^{\frac{1}{2}}(\mathsf{tt},\mathsf{ff})$

PROOF. Let $W_{\mathsf{efair,p}}$ be the value that $\mathsf{Y}(\mathsf{efair'_p})$ unfolds to. Unfolding definitions shows that

$$\begin{aligned}\mathsf{eval}(W_{\mathsf{efair,p}}\langle\rangle) &\rightsquigarrow (\mathsf{eval}(W_{\mathsf{efair,p}}\langle\rangle) \oplus_p^\forall \delta^\forall \mathsf{tt}) \oplus_p^\forall (\delta^\forall \mathsf{ff} \oplus_p^\forall \mathsf{eval}(W_{\mathsf{efair,p}}\langle\rangle))\\ &= (\delta^\forall \mathsf{tt} \oplus_{\frac{1}{2}}^\forall \delta^\forall \mathsf{ff}) \oplus_{2p(1-p)}^\forall \mathsf{eval}(W_{\mathsf{efair,p}}\langle\rangle)\end{aligned}$$

From this, it follows that $\mathsf{eval}(W_{\mathsf{efair,p}}\langle\rangle) \rightsquigarrow^\approx \delta^\forall(\mathsf{tt}) \oplus_{\frac{1}{2}}^\forall \delta^\forall(\mathsf{ff})$ as in Example 4.16, and since $\mathsf{eval}(\mathsf{efair_p}\langle\rangle) \rightsquigarrow \mathsf{eval}(W_{\mathsf{efair,p}}\langle\rangle)$, also $\mathsf{choice}^{\frac{1}{2}}(\mathsf{tt},\mathsf{ff}) \preceq_{\mathsf{Ctx}} \mathsf{efair_p}\langle\rangle$ follows from Theorem 9.1.

On the other hand, to show $\mathsf{efair_p}\langle\rangle \preceq_{\mathsf{Ctx}} \mathsf{choice}^{\frac{1}{2}}(\mathsf{tt},\mathsf{ff})$, since $\mathsf{efair_p} \rightsquigarrow W_{\mathsf{efair,p}}$, it suffices to show that $[\![W_{\mathsf{efair,p}}\langle\rangle]\!]^\kappa \preceq_{\mathsf{bool}}^{\kappa,\mathsf{Tm}} \mathsf{eval}(\mathsf{choice}^{\frac{1}{2}}(\mathsf{tt},\mathsf{ff}))$. Computing

$$\begin{aligned}[\![W_{\mathsf{efair,p}}\langle\rangle]\!]^\kappa &= \Delta^\kappa[\![\mathsf{efair'_p}\ (\mathsf{Y}\ \mathsf{efair'_p})\langle\rangle]\!]^\kappa\\ &= \Delta^\kappa[\![\mathsf{efair'_p}\ W_{\mathsf{efair,p}}\ \langle\rangle]\!]^\kappa\\ &= \Delta^\kappa\left(\left([\![W_{\mathsf{efair,p}}\ \langle\rangle]\!]^\kappa \oplus_p^\kappa \delta^\kappa(\mathsf{tt})\right) \oplus_p^\kappa \left(\delta^\kappa(\mathsf{ff}) \oplus_p^\kappa ([\![W_{\mathsf{efair,p}}\ \langle\rangle]\!]^\kappa)\right)\right)\\ &= \Delta^\kappa\left(\left(\delta^\kappa(\mathsf{tt}) \oplus_{\frac{1}{2}}^\kappa \delta^\kappa(\mathsf{ff})\right) \oplus_{2p(1-p)}^\kappa [\![W_{\mathsf{efair,p}}\ \langle\rangle]\!]^\kappa\right)\end{aligned}$$

So that $[\![W_{\mathsf{efair,p}}\langle\rangle]\!]^\kappa \preceq_{\mathsf{bool}}^{\kappa,\mathsf{Tm}} \mathsf{eval}(\mathsf{choice}^{\frac{1}{2}}(\mathsf{tt},\mathsf{ff}))$ is equivalent to

$$\triangleright^\kappa\left((\delta^\kappa(\mathsf{tt}) \oplus_{\frac{1}{2}}^\kappa \delta^\kappa(\mathsf{ff})) \oplus_{2p(1-p)}^\kappa [\![W_{\mathsf{efair,p}}\ \langle\rangle]\!]^\kappa \preceq_{\mathsf{bool}}^{\kappa,\mathsf{Tm}} \mathsf{eval}(\mathsf{choice}^{\frac{1}{2}}(\mathsf{tt},\mathsf{ff}))\right)$$

and can therefore be proved by guarded recursion. □

9.3 Random Walk

The next example, inspired by [1], is an equivalence of two random walks. We represent a random walk by its trace, which is a potentially infinite list of integers. In a call-by-value language such terms are best implemented as *lazy lists*. We define an $\mathsf{FPC}_\oplus$ type of these as

$$\mathsf{LazyL} \triangleq \mu X.1 + \mathsf{Nat} \times (1 \to X)$$

as well as a familiar interface for programming with these in Figure 7.

$$\mathsf{nil} : \mathsf{LazyL}$$
$$\mathsf{nil} \triangleq \mathsf{fold}\,(\mathsf{inl}\,(\langle\rangle))$$

$$\mathsf{tl} : \mathsf{LazyL} \to \mathsf{LazyL}$$
$$\mathsf{tl} \triangleq \mathsf{lam}\, l.\mathsf{case}(\mathsf{unfold}\, l, x.\mathsf{nil}, (n, f).f(\langle\rangle))$$

$$\mathsf{hd} : \mathsf{LazyL} \to \mathsf{Nat} + 1$$
$$\mathsf{hd} \triangleq \mathsf{lam}\, l.\mathsf{case}(\mathsf{unfold}\, l, x.\mathsf{inr}\,\langle\rangle, (n, f).\mathsf{inl}\,\underline{n})$$

$$- :: - : \mathsf{Nat} \to (1 \to \mathsf{LazyL}) \to \mathsf{LazyL}$$
$$- :: - \triangleq \mathsf{lam}\, n.\mathsf{lam}\, f.\mathsf{fold}\,(\mathsf{inr}\,(n, f))$$

Fig. 7. Lazy List Interface

Using these we can define a lazy list processing function $\mathsf{2nd} : \mathsf{LazyL} \to \mathsf{LazyL}$, returning a lazy list that contains every second element of the argument. Define $\mathsf{2nd} \triangleq \mathsf{Y}(G_{\mathsf{2nd}})$ where

$$G_{\mathsf{2nd}} : (\mathsf{LazyL} \to \mathsf{LazyL}) \to \mathsf{LazyL} \to \mathsf{LazyL}$$
$$G_{\mathsf{2nd}} \triangleq \mathsf{lam}\, g.\mathsf{lam}\, l.\mathsf{case}(\mathsf{unfold}\,(l), x.\mathsf{nil}, (n, f).n :: \mathsf{lam}\, y.g(\mathsf{tl}(f(\langle\rangle))))$$

For this example, we consider two random walks: a classical random walk $\mathsf{R}_1 : \mathsf{Nat} \to \mathsf{LazyL}$, which in each step takes a step up or down, each with probability $\frac{1}{2}$, and a sped-up version $\mathsf{R}_2 : \mathsf{Nat} \to \mathsf{LazyL}$, which stays put with probability $\frac{1}{2}$, and otherwise takes two steps either up or down. Define these as

$$\mathsf{R}_1 \triangleq \mathsf{Y}(G_{\mathsf{R}_1}) \qquad \mathsf{R}_2 \triangleq \mathsf{Y}(G_{\mathsf{R}_2})$$

where

$$G_{\mathsf{R}_1} \triangleq \mathsf{lam}\, g.\mathsf{lam}\, n.n :: \mathsf{lam}\, y.\mathsf{ifz}(n, \mathsf{nil}, \mathsf{choice}^{\frac{1}{2}}(g(n-1), g(n+1)))$$
$$G_{\mathsf{R}_2} \triangleq \mathsf{lam}\, g.\mathsf{lam}\, n.n :: \mathsf{lam}\, y.\mathsf{ifz}(n, \mathsf{nil}, \mathsf{choice}^{\frac{1}{2}}(g(n), \mathsf{choice}^{\frac{1}{2}}(g(n-2), g(n+2))))$$

Intuitively, R_2 is just R_1 at double speed. The following theorem makes this intuition precise.

THEOREM 9.3. *For all $n : \mathbb{N}$ we have that* $\mathsf{2nd}(\mathsf{R}_1(\underline{2n})) \equiv_{\mathsf{Ctx}} \mathsf{R}_2(\underline{2n})$.

PROOF (SKETCH). Let V_{R_1}, V_{R_2} and V_{2nd} be the values that R_1, R_2 and $\mathsf{2nd}$ reduce to. It suffices to show that $V_{\mathsf{2nd}}(V_{\mathsf{R}_1}(\underline{2n})) \equiv_{\mathsf{Ctx}} V_{\mathsf{R}_2}(\underline{2n})$. We just show $[\![V_{\mathsf{2nd}}(V_{\mathsf{R}_1}(\underline{2n}))]\!]^\kappa \preceq^{\kappa,\mathsf{Tm}}_{\mathsf{LazyL}} \mathsf{eval}(V_{\mathsf{R}_2}(\underline{2n}))$ by guarded recursion, omitting the other direction, which can be proved similarly. We focus just on the case of $n > 0$, which is the harder one. Unfolding definitions shows that

$$[\![V_{\mathsf{2nd}}(V_{\mathsf{R}_1}(\underline{2n}))]\!]^\kappa = (\Delta^\kappa)^3(\delta^\kappa(\mathsf{next}^\kappa(\mathbf{inr}\,(\underline{2n}, \lambda_.W_{\mathsf{R}_1}))))$$

where

$$W_{\mathsf{R}_1} = \left([\![V_{\mathsf{2nd}}(\mathsf{tl}(V_{\mathsf{R}_1}(\underline{2n-1})))]\!]^\kappa \oplus^\kappa_{\frac{1}{2}} [\![V_{\mathsf{2nd}}(\mathsf{tl}(V_{\mathsf{R}_1}(\underline{2n+1})))]\!]^\kappa\right)$$

Similarly,

$$\begin{aligned}&\mathsf{eval}(V_{\mathsf{R}_2}(\underline{2n}))\\ &\rightsquigarrow \mathsf{fold}\,(\mathsf{inr}\,(\underline{2n}, \mathsf{lam}\, y.\mathsf{ifz}(\underline{2n}, \mathsf{nil}, \mathsf{choice}^{\frac{1}{2}}(V_{\mathsf{R}_2}(\underline{2n}), \mathsf{choice}^{\frac{1}{2}}(V_{\mathsf{R}_2}(\underline{2n-2}), V_{\mathsf{R}_2}(\underline{2n+2}))))))\end{aligned}$$

So showing $[\![V_{\mathsf{2nd}}(V_{\mathsf{R}_1}(\underline{2n}))]\!]^\kappa \preceq^{\kappa,\mathsf{Tm}}_{\mathsf{LazyL}} \mathsf{eval}(V_{\mathsf{R}_2}(\underline{2n}))$ easily reduces to showing

$$\triangleright^\kappa(W_{\mathsf{R}_1} \preceq^{\kappa,\mathsf{Tm}}_{\mathsf{LazyL}} \mathsf{eval}(\mathsf{choice}^{\frac{1}{2}}(V_{\mathsf{R}_2}(\underline{2n}), \mathsf{choice}^{\frac{1}{2}}(V_{\mathsf{R}_2}(\underline{2n-2}), V_{\mathsf{R}_2}(\underline{2n+2}))))) \tag{13}$$

Since $2n - 1 > 0$, unfolding definitions gives

$$\llbracket V_{\mathsf{2nd}}(\mathsf{tl}(V_{\mathsf{R}_1}(\underline{2n{-}1})))\rrbracket^\kappa = (\Delta^\kappa)^2 \left(\llbracket V_{\mathsf{2nd}}(V_{\mathsf{R}_1}(\underline{2n{-}2}))\rrbracket^\kappa \oplus^\kappa_{\frac{1}{2}} \llbracket V_{\mathsf{2nd}}(V_{\mathsf{R}_1}(\underline{2n}))\rrbracket^\kappa\right)$$

$$\llbracket V_{\mathsf{2nd}}(\mathsf{tl}(V_{\mathsf{R}_1}(\underline{2n{+}1})))\rrbracket^\kappa = (\Delta^\kappa)^2 \left(\llbracket V_{\mathsf{2nd}}(V_{\mathsf{R}_1}(\underline{2n}))\rrbracket^\kappa \oplus^\kappa_{\frac{1}{2}} \llbracket V_{\mathsf{2nd}}(V_{\mathsf{R}_1}(\underline{2n{-}2}))\rrbracket^\kappa\right)$$

so by Lemma 7.7 (13) is equivalent to

$$(\triangleright^\kappa)^3 \left(\begin{array}{l}\llbracket V_{\mathsf{2nd}}(V_{\mathsf{R}_1}(\underline{2n}))\rrbracket^\kappa \oplus^\kappa_{\frac{1}{2}} \left(\llbracket V_{\mathsf{2nd}}(V_{\mathsf{R}_1}(\underline{2n{-}2}))\rrbracket^\kappa \oplus^\kappa_{\frac{1}{2}} \llbracket V_{\mathsf{2nd}}(V_{\mathsf{R}_1}(\underline{2n{+}2}))\rrbracket^\kappa\right) \\ \quad \preceq^{\kappa,\mathsf{Tm}}_{\mathsf{LazyL}} \ \mathsf{eval}(V_{\mathsf{R}_2}(\underline{2n})) \oplus^\forall_{\frac{1}{2}} \left(\mathsf{eval}(V_{\mathsf{R}_2}(\underline{2n{-}2})) \oplus^\forall_{\frac{1}{2}} \mathsf{eval}(V_{\mathsf{R}_2}(\underline{2n{+}2}))\right)\end{array}\right)$$

which follows from the guarded induction hypothesis and Lemma 7.8. □

Remark 4. *Note again how working with denotational semantics simplifies this proof. This enables us to take two steps in the walk defined by V_{R_1}, collect all outcomes, and couple them with the outcomes of a single step of V_{R_2}. Other coupling-based logics like [1] require specialized rules to deal with these two-to-one couplings but in our setting it is just a simple consequence of our definitions.*

10 Related Work

We have discussed some related work in the Introduction and throughout the paper; here we discuss additional related work.

In this paper, we have shown how to use synthetic guarded domain theory (SGDT) to model $\mathsf{FPC}_\oplus$. SGDT has been used in earlier work to model PCF [33], a call-by-name variant of FPC [29], FPC with general references [36, 37], untyped lambda calculus with nondeterminism [30], guarded interaction trees [18], and gradual typing [19]. Thus the key new challenge addressed in this paper is the modelling of probabilistic choice, in combination with recursion, which led us to introduce (guarded) convex delay algebras.

In previous works on SGDT models of PCF and FPC, the logical relation between syntax and semantics required a 1-to-1 correspondence between the steps on either side. The previous work on nondeterminism [30] is in some ways closer to our work: To model the combination of nondeterminism and recursion in the case of may-simulation, it uses a monad defined similarly to D^κ, but for the finite powerset monad, rather than distributions. It also uses a logical relation between syntax and semantics, which similarly to ours is a refinement relation. The theory developed for D^κ here, such as the $\rightsquigarrow$ relation, and the lifting of relations using couplings is new. The applications are also different: Here we use the relation to reason about contextual refinement whereas in *op. cit.* it is used for proving congruence of an applicative simulation relation defined on operational semantics.

The use of step-indexed logical relations to model the combination of probabilistic choice, recursive types, polymorphism and other expressive language features has also been explored in [2, 9, 20]. These papers use an explicit account of step-indexing, and rely on non-constructive mathematics to define their operational semantics and to prove their soundness theorems. Our approach is constructive, and our use of SGDT eliminates the need for explicit manipulation of step indices. Moreover, their logical relations are defined purely in operational terms.

Applicative bisimulation for a probabilistic, call-by-value version of PCF is studied in [13]. This work defines an approximation-based operational semantics, where a term evaluates to a family of finite distributions over values, each element of the family corresponding to the values observed after a given finite number of steps. This is reminiscent of our semantics, although in *op. cit.* the approximation semantics is then used to define a limit semantics using suprema.

Probabilistic couplings and liftings have been a popular technique in recent years to reason about probabilistic programs [5, 6], since they provide a compositional way to lift relations from base types to distributions over those types. To the best of our knowledge, our presentation is the first that uses constructive mathematics in the definition of couplings and the proofs of their properties, e.g. the bind lemma. Our definition of couplings is also asymmetric to account for the fact that the distribution on the left uses guarded recursion. This is similar in spirit to the left-partial coupling definition of [20], where asymmetry is used to account for step indexing.

In this paper, we model the probabilistic effects of $\mathsf{FPC}_{\oplus}$ using guarded / co-inductive types. In spirit, this is similar to how effects are being modeled in both coinductive and guarded interaction trees [18, 44]. Coinductive interaction trees are useful for giving denotational semantics for first-order programs and are defined using ordinary type theory, whereas guarded interaction trees can also be applied to give denotational semantics for higher-order programs, but are defined using a fragment of guarded type theory. Interaction trees have recently been extended to account for nondeterminism [10] but, to the best of our knowledge, interaction trees have so far not been extended to account for probabilistic effects.

11 Conclusion and Future Work

We have developed a notion of (guarded) convex delay algebras and shown how to use it to define and relate operational and denotational semantics for $\mathsf{FPC}_{\oplus}$ in guarded type theory. To the best of our knowledge, this is the first constructive type theoretic account of the semantics of $\mathsf{FPC}_{\oplus}$. The denotational semantics can be viewed as a shallow embedding of $\mathsf{FPC}_{\oplus}$ in constructive type theory, which can therefore be used directly as a probabilistic programming language. Our examples show how to use the relation between syntax and semantics for proving contextual equivalence of $\mathsf{FPC}_{\oplus}$ programs. The use of denotational semantics for these simplifies proofs by using much fewer steps than the operational semantics.

Future work includes combining and extending the present work with the account of nondeterminism in [30] and to compare the resulting model with the recent classically defined operationally-based logical relation in [2].

Acknowledgments

This work was supported in part by the Independent Research Fund Denmark grant number 2032-00134B, in part by a Villum Investigator grant (no. 25804), Center for Basic Research in Program Verification (CPV), from the VILLUM Foundation, and in part by the European Union (ERC, CHORDS, 101096090). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

References

[1] Alejandro Aguirre, Gilles Barthe, Lars Birkedal, Ales Bizjak, Marco Gaboardi, and Deepak Garg. 2018. Relational Reasoning for Markov Chains in a Probabilistic Guarded Lambda Calculus. In *Programming Languages and Systems - 27th European Symposium on Programming, ESOP 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 10801)*, Amal Ahmed (Ed.). Springer, 214–241. https://doi.org/10.1007/978-3-319-89884-1_8

[2] Alejandro Aguirre and Lars Birkedal. 2023. Step-Indexed Logical Relations for Countable Nondeterminism and Probabilistic Choice. *Proc. ACM Program. Lang.* 7, POPL (2023), 33–60. https://doi.org/10.1145/3571195

[3] Robert Atkey and Conor McBride. 2013. Productive coprogramming with guarded recursion. In *ACM SIGPLAN International Conference on Functional Programming, ICFP'13, Boston, MA, USA - September 25 - 27, 2013*, Greg Morrisett and Tarmo Uustalu (Eds.). ACM, 197–208. https://doi.org/10.1145/2500365.2500597

[4] Patrick Bahr, Hans Bugge Grathwohl, and Rasmus Ejlers Møgelberg. 2017. The clocks are ticking: No more delays!. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*. IEEE Computer Society, 1–12. https://doi.org/10.1109/LICS.2017.8005097

[5] Gilles Barthe, Thomas Espitau, Benjamin Grégoire, Justin Hsu, Léo Stefanesco, and Pierre-Yves Strub. 2015. Relational Reasoning via Probabilistic Coupling. In *Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings (Lecture Notes in Computer Science, Vol. 9450)*, Martin Davis, Ansgar Fehnker, Annabelle McIver, and Andrei Voronkov (Eds.). Springer, 387–401. https://doi.org/10.1007/978-3-662-48899-7_27

[6] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. 2009. Formal certification of code-based cryptographic proofs. In *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009*, Zhong Shao and Benjamin C. Pierce (Eds.). ACM, 90–101. https://doi.org/10.1145/1480881.1480894

[7] Lars Birkedal, Ranald Clouston, Bassel Mannaa, Rasmus Ejlers Møgelberg, Andrew M. Pitts, and Bas Spitters. 2020. Modal dependent type theory and dependent right adjoints. *Math. Struct. Comput. Sci.* 30, 2 (2020), 118–138. https://doi.org/10.1017/S0960129519000197

[8] Lars Birkedal and Rasmus Ejlers Møgelberg. 2013. Intensional Type Theory with Guarded Recursive Types qua Fixed Points on Universes. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*. IEEE Computer Society, 213–222. https://doi.org/10.1109/LICS.2013.27

[9] Ales Bizjak and Lars Birkedal. 2015. Step-Indexed Logical Relations for Probability. In *Foundations of Software Science and Computation Structures - 18th International Conference, FoSSaCS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings (Lecture Notes in Computer Science, Vol. 9034)*, Andrew M. Pitts (Ed.). Springer, 279–294. https://doi.org/10.1007/978-3-662-46678-0_18

[10] Nicolas Chappe, Paul He, Ludovic Henrio, Yannick Zakowski, and Steve Zdancewic. 2023. Choice Trees: Representing Nondeterministic, Recursive, and Impure Programs in Coq. *Proc. ACM Program. Lang.* 7, POPL (2023), 1770–1800. https://doi.org/10.1145/3571254

[11] Ranald Clouston. 2018. Fitch-Style Modal Lambda Calculi. In *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 10803)*, Christel Baier and Ugo Dal Lago (Eds.). Springer, 258–275. https://doi.org/10.1007/978-3-319-89366-2_14

[12] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. 2017. Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom. *FLAP* 4, 10 (2017), 3127–3170. http://collegepublications.co.uk/ifcolog/?00019

[13] Raphaëlle Crubillé and Ugo Dal Lago. 2014. On Probabilistic Applicative Bisimulation and Call-by-Value λ-Calculi. In *Programming Languages and Systems - 23rd European Symposium on Programming, ESOP 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings (Lecture Notes in Computer Science, Vol. 8410)*, Zhong Shao (Ed.). Springer, 209–228. https://doi.org/10.1007/978-3-642-54833-8_12

[14] Ryan Culpepper and Andrew Cobb. 2017. Contextual Equivalence for Probabilistic Programs with Continuous Random Variables and Scoring. In *Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings (Lecture Notes in Computer Science, Vol. 10201)*, Hongseok Yang (Ed.). Springer, 368–392. https://doi.org/10.1007/978-3-662-54434-1_14

[15] Fredrik Dahlqvist and Dexter Kozen. 2020. Semantics of higher-order probabilistic programs with conditioning. *Proc. ACM Program. Lang.* 4, POPL (2020), 57:1–57:29. https://doi.org/10.1145/3371125

[16] Thomas Ehrhard, Michele Pagani, and Christine Tasson. 2018. Full Abstraction for Probabilistic PCF. *J. ACM* 65, 4 (2018), 23:1–23:44. https://doi.org/10.1145/3164540

[17] Marcelo P. Fiore. 1994. *Axiomatic domain theory in categories of partial maps.* Ph. D. Dissertation. University of Edinburgh, UK. https://hdl.handle.net/1842/406

[18] Dan Frumin, Amin Timany, and Lars Birkedal. 2024. Modular Denotational Semantics for Effects with Guarded Interaction Trees. *Proc. ACM Program. Lang.* 8, POPL (2024), 332–361. https://doi.org/10.1145/3632854

[19] Eric Giovannini, Tingting Ding, and Max S. New. 2025. Denotational Semantics of Gradual Typing using Synthetic Guarded Domain Theory. *Proc. ACM Program. Lang.* 9, POPL (2025). https://doi.org/10.1145/3704863

[20] Simon Oddershede Gregersen, Alejandro Aguirre, Philipp G. Haselwarter, Joseph Tassarotti, and Lars Birkedal. 2024. Asynchronous Probabilistic Couplings in Higher-Order Separation Logic. *Proc. ACM Program. Lang.* 8, POPL, Article 26 (jan 2024), 32 pages. https://doi.org/10.1145/3632868

[21] Chris Heunen, Ohad Kammar, Sam Staton, and Hongseok Yang. 2017. A convenient category for higher-order probability theory. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*. IEEE Computer Society, 1–12. https://doi.org/10.1109/LICS.2017.8005137

[22] Bart Jacobs. 2010. Convexity, Duality and Effects. In *Theoretical Computer Science - 6th IFIP TC 1/WG 2.2 International Conference, TCS 2010, Held as Part of WCC 2010, Brisbane, Australia, September 20-23, 2010. Proceedings (IFIP Advances in Information and Communication Technology, Vol. 323)*, Cristian S. Calude and Vladimiro Sassone (Eds.). Springer, 1–19. https://doi.org/10.1007/978-3-642-15240-5_1

[23] Patricia Johann, Alex Simpson, and Janis Voigtländer. 2010. A Generic Operational Metatheory for Algebraic Effects. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*. IEEE Computer Society, 209–218. https://doi.org/10.1109/LICS.2010.29

[24] C. Jones and Gordon D. Plotkin. 1989. A Probabilistic Powerdomain of Evaluations. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), Pacific Grove, California, USA, June 5-8, 1989*. IEEE Computer Society, 186–195. https://doi.org/10.1109/LICS.1989.39173

[25] Shin-ya Katsumata and Tetsuya Sato. 2013. Preorders on Monads and Coalgebraic Simulations. In *Foundations of Software Science and Computation Structures*, David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, and Frank Pfenning (Eds.). Vol. 7794. Springer Berlin Heidelberg, Berlin, Heidelberg, 145–160. https://doi.org/10.1007/978-3-642-37075-5_10

[26] Magnus Baunsgaard Kristensen, Rasmus Ejlers Møgelberg, and Andrea Vezzosi. 2022. Greatest HITs: Higher inductive types in coinductive definitions via induction under clocks. In *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, Christel Baier and Dana Fisman (Eds.). ACM, 42:1–42:13. https://doi.org/10.1145/3531130.3533359

[27] Ugo Dal Lago, Davide Sangiorgi, and Michele Alberti. 2014. On coinductive equivalences for higher-order probabilistic functional programs. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, Suresh Jagannathan and Peter Sewell (Eds.). ACM, 297–308. https://doi.org/10.1145/2535838.2535872

[28] T. Lindvall. 2002. *Lectures on the Coupling Method*. Dover Publications, Incorporated.

[29] Rasmus Ejlers Møgelberg and Marco Paviotti. 2019. Denotational semantics of recursive types in synthetic guarded domain theory. *Math. Struct. Comput. Sci.* 29, 3 (2019), 465–510. https://doi.org/10.1017/S0960129518000087

[30] Rasmus Ejlers Møgelberg and Andrea Vezzosi. 2021. Two Guarded Recursive Powerdomains for Applicative Simulation. In *Proceedings 37th Conference on Mathematical Foundations of Programming Semantics, MFPS 2021, Hybrid: Salzburg, Austria and Online, 30th August - 2nd September, 2021 (EPTCS, Vol. 351)*, Ana Sokolova (Ed.). 200–217. https://doi.org/10.4204/EPTCS.351.13

[31] Rasmus Ejlers Møgelberg and Maaike Zwart. 2024. What Monads Can and Cannot Do with a Bit of Extra Time. In *32nd EACSL Annual Conference on Computer Science Logic, CSL 2024, February 19-23, 2024, Naples, Italy (LIPIcs, Vol. 288)*, Aniello Murano and Alexandra Silva (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 39:1–39:18. https://doi.org/10.4230/LIPICS.CSL.2024.39

[32] Hiroshi Nakano. 2000. A Modality for Recursion. In *15th Annual IEEE Symposium on Logic in Computer Science, Santa Barbara, California, USA, June 26-29, 2000*. IEEE Computer Society, 255–266. https://doi.org/10.1109/LICS.2000.855774

[33] Marco Paviotti, Rasmus Ejlers Møgelberg, and Lars Birkedal. 2015. A Model of PCF in Guarded Type Theory. In *The 31st Conference on the Mathematical Foundations of Programming Semantics, MFPS 2015, Nijmegen, The Netherlands, June 22-25, 2015 (Electronic Notes in Theoretical Computer Science, Vol. 319)*, Dan R. Ghica (Ed.). Elsevier, 333–349. https://doi.org/10.1016/J.ENTCS.2015.12.020

[34] Andrew M. Pitts. 1996. Relational Properties of Domains. *Inf. Comput.* 127, 2 (1996), 66–90. https://doi.org/10.1006/INCO.1996.0052

[35] Gordon D Plotkin. 1985. Denotational semantics with partial functions. *Lecture at CSLI Summer School* (1985).

[36] Jonathan Sterling, Daniel Gratzer, and Lars Birkedal. 2022. Denotational semantics of general store and polymorphism. *CoRR* abs/2210.02169 (2022). https://doi.org/10.48550/ARXIV.2210.02169 arXiv:2210.02169

[37] Jonathan Sterling, Daniel Gratzer, and Lars Birkedal. 2023. Free theorems from univalent reference types. *CoRR* abs/2307.16608 (2023). https://doi.org/10.48550/ARXIV.2307.16608 arXiv:2307.16608

[38] Hermann Thorisson. 2000. *Coupling, stationarity, and regeneration*. Springer-Verlag, New York. xiv+517 pages.

[39] The Univalent Foundations Program. 2013. *Homotopy Type Theory: Univalent Foundations of Mathematics*. https://homotopytypetheory.org/book, Institute for Advanced Study.

[40] Matthijs Vákár, Ohad Kammar, and Sam Staton. 2019. A domain theory for statistical probabilistic programming. *Proc. ACM Program. Lang.* 3, POPL (2019), 36:1–36:29. https://doi.org/10.1145/3290349

[41] Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. 2019. Cubical agda: a dependently typed programming language with univalence and higher inductive types. *Proc. ACM Program. Lang.* 3, ICFP (2019), 87:1–87:29. https://doi.org/10.1145/3341691

[42] C. Villani. 2008. *Optimal Transport: Old and New*. Springer Berlin Heidelberg.

[43] Mitchell Wand, Ryan Culpepper, Theophilos Giannakopoulos, and Andrew Cobb. 2018. Contextual equivalence for a probabilistic language with continuous random variables and recursion. *Proc. ACM Program. Lang.* 2, ICFP (2018), 87:1–87:30. https://doi.org/10.1145/3236782

[44] Li-yao Xia, Yannick Zakowski, Paul He, Chung-Kil Hur, Gregory Malecha, Benjamin C. Pierce, and Steve Zdancewic. 2020. Interaction trees: representing recursive and impure programs in Coq. *Proc. ACM Program. Lang.* 4, POPL (2020), 51:1–51:32. https://doi.org/10.1145/3371119

[45] Yizhou Zhang and Nada Amin. 2022. Reasoning about "reasoning about reasoning": semantics and contextual equivalence for probabilistic programs with nested queries and recursion. *Proc. ACM Program. Lang.* 6, POPL (2022), 1–28. https://doi.org/10.1145/3498677

Received 2024-07-11; accepted 2024-11-07