A Semantic Account of Metric Preservation

Abstract

Program sensitivity measures how robust program's outputs are to changes in its input. This is a useful measure in fields like differential privacy and cyber-physical systems, in order to guarantee that input changes have a limited effect on the program behavior. A natural way to formalize program sensitivity is in terms of metrics on the input and output spaces, requiring that a k-sensitive function maps inputs that are at distance d to outputs that are at most at distance $k \cdot d$. Program sensitivity is thus an analog of Lipschitz continuity for programs.

Reed and Pierce introduced *Fuzz*, a functional language with a linear type system that can express program sensitivity. They show soundness operationally, in the form of a *metric preservation* property. Inspired by their work, we study program sensitivity and metric preservation from a denotational point of view. In particular, we introduce *metric CPOs*, a novel semantic structure for reasoning about computation on metric spaces, endowing CPOs with a compatible notion of distance. This structure is useful for reasoning about metric properties of programs, and specifically about program sensitivity. We demonstrate metric CPOs by giving a model for the deterministic fragment of Fuzz.

1. Introduction

In many applications, programs should not be too sensitive to small changes in their input. For example, in cyber-physical systems often the input has a small degree of uncertainty that must be taken into account to make sure that a program correctly responds to measurement errors; in *differential privacy* [22], a strong statistical guarantee that protects the privacy of individuals in a database, the amount of noise that one needs to add to the output of a program in order to guarantee the privacy of individual depends on how much a single individual can influence the result of the program, etc.

To measure this dependency, researchers have focused on the notion of *sensitivity* for programs. Roughly speaking, sensitivity is a measure of how much the results of the program may vary when the program is run on nearby inputs. More formally, a function $f : X \to Y$ is *r-sensitive* if $d_X(f(x), f(y)) \leq r \cdot d_Y(x, y)$ for every pair of inputs $x, y \in X$, where d_S is a function assigning a non-negative *distance* to pairs of elements of a set S. This notion of program sensitivity is an analog of Lipschitz continuity for programs. Accordingly, program sensitivity is also a natural tool for expressing notions of uniform continuity and numerical stability properties.

Due to its useful applications, several works have proposed tools for formal reasoning about sensitivity. On the one hand, some works have used standard tools from program verification to design analyses specifically tailored for reasoning about sensitivity. Two important examples are the work by Reed and Pierce [37], based on a linear type system annotated with sensitivity information, and the work by Chaudhuri et al. [14], based on a program analysis for reasoning about sensitivity of imperative programs. On the other hand, by noticing that sensitivity can be seen as a natural example of a relational property (or 2-property), a property of a set of pairs of runs of a program, other works have used tools for relational properties to directly reason about sensitivity, e.g. Barthe et al. [8] use a relational Hoare Logic, and Barthe et al. [9] use relational refinement types.

In this work we consider the formal approach to program sensitivity taken by Reed and Pierce [37] which is closely inspired by the original description in terms of *metric spaces* provided by the initial paper on differential privacy [22]. Reed and Pierce introduced *Fuzz*, a purely functional language featuring a linear indexed type system for a PCF-like language. In Fuzz, every type τ is endowed with a notion of distance, and function types $!_r \tau \multimap \sigma$ carry an index r describing the sensitivity of the program to which the type can be assigned. The soundness result, called *metric preservation* by Reed and Pierce [37], essentially says that the type system assigns correct sensitivity values to programs, i.e. a program e of type $!_r \tau \multimap \sigma$ maps two expressions $\vdash e_1, e_2 : \tau$ that are at distance k to expressions that are at distance $k \cdot r$. Metric preservation and the compositionality of the type system allow a clean, compositional analysis of sensitivity for higher-order programs.

The design of sensitivity analyses leads to interesting technical challenges. In Fuzz, reasoning about sensitivities is complicated by the expressive programming language, which allows general recursion and features recursive types. The original proof of metric preservation relies on the definition of intricate, syntactic logical relations that mix step-indexing and metric information. The logical relations are used at the same time to define the notion of distance and as the proof method to prove the soundness of the type system. This combined approach makes it difficult to reason about programs and values as objects of a metric space.

In this paper, instead of considering a syntactic logical-relations approach, we provide a denotational, domain-theoretic view of sensitivity and metric preservation. A denotational model for sensitivity based on metric spaces is insufficient to interpret a language with general recursion and recursive types in the style of Reed and Pierce, so we introduce a category of *metric CPOs* with non-expansive, Scott-continuous functions as morphisms. A metric CPO is a complete partial order endowed with a metric for which every open ball is stable under limits of ascending chains. This additional requirement is a compatibility condition between the metric structure and the CPO structure. While simple, this notion of compatibility seems to have seen lit-

tle investigation in the literature, and provides a natural extension of the notion of sensitivity to partial functions.[1]

The category of metric CPOs has rich structure that suffices to interpret the sensitivity analysis of Fuzz.[2] At the same time, by grounding our work on well-established domain-theoretic notions, we are able to reuse a vast array of standard tools from the literature to model recursive definitions of functions, types, and relations. First, we show that metric CPOs have the right structure for solving recursive domain equations, following the approach laid out by Freyd [25], Smyth and Plotkin [39], and others. Second, we prove the adequacy of the denotational semantics of Fuzz with respect to its operational semantics, by adapting a method due to Pitts [35] for constructing a family of type-indexed logical relations. Fibrational category theory plays a unifying role, initially for lifting colimits of CPOs to the metric setting, and later for defining relations on metric CPOs.

While the motivation of our work comes from Fuzz, the structure of metric CPO is richer, supporting a metric interpretation of constructs not directly definable in Fuzz. For instance, the model indicates a significantly more precise type for fixed-point combinators than had previously been considered before (cf. Lemma 4.10). Metric CPOs can also provide a useful tool to model other languages that reason about sensitivity.

The rest of the paper is organized as follows. We begin with a simplified setting that highlights the core features of sensitivity analysis *without* general recursion, reviewing basic notions of metric spaces (Section 2) and showing how they yield a model of a terminating fragment of Fuzz (Section 3). We present our main contribution in Section 4: the notion of *metric CPO*. We show how the constructions carried out for metric spaces can be naturally lifted to this setting, and how we can use these structures to interpret recursive definitions of functions and data types. We use this infrastructure to extend our model of Fuzz with recursive types in Section 5, thereby validating metric preservation. We conclude with a discussion of related work, pointing to promising directions for future work (Sections 6 and 7).

2. Metric Spaces

We begin by studying the essence of sensitivity analysis in the simplest setting, with metric spaces and total functions. Most results here are standard, and covered in more detail in other works (e.g. [27]).

Let $\mathbb{R}^{\infty}_{\geq 0} \triangleq \{r \in \mathbb{R} \mid r \geq 0\} \cup \{\infty\}$ be the set of *extended non-negative reals*. We extend addition and the order relation on $\mathbb{R}$ to $\mathbb{R}^{\infty}_{\geq 0}$ by setting

$$\infty + r = r + \infty = \infty, \qquad r \leq \infty, \qquad \text{for every } r \in \mathbb{R}_{\geq 0}.$$

An *extended pseudo-metric space* is a tuple (X, d_X), where X is a set and $d_X : X^2 \to \mathbb{R}^{\infty}_{\geq 0}$ is a *metric*: a function satisfying (i) $d_X(x,x) = 0$ (ii) $d_X(x,y) = d_X(y,x)$; and (iii) the *triangle inequality* $d_X(x,z) \leq d_X(x,y) + d_X(y,z)$.

An extended pseudo-metric space differs from the classic notion of metric space in two aspects. First, two points can be at distance 0 from each other without being equal; we don't impose the axiom $d(x,y) = 0 \Rightarrow x = y$. Second, since distances range over $\mathbb{R}^{\infty}_{\geq 0}$, pairs of points can be infinitely apart. We simplify the exposition by henceforth referring to extended pseudo-metric spaces *simply as metric spaces*. Figure 1 summarizes some useful metrics. Besides standard examples, such as the real numbers $\mathbb{R}$ under the Euclidian metric, we consider metrics that we can define on more general sets: products, sums, and functions.

Space (Carrier)	$d(a,b)$
$\mathbb{R}$	$\lvert a-b\rvert$
$\mathbf{1}$	0
$r \cdot X$ (X)	$r \cdot d_X(a,b)$
$X \,\&\, Y$ $(X \times Y)$	$\max(d_X(a_1,b_1), d_Y(a_2,b_2))$
$X \otimes Y$ $(X \times Y)$	$d_X(a_1,b_1) + d_Y(a_2,b_2)$
$X + Y$	$d_X(a,b)$ if $a,b \in X$ $d_Y(a,b)$ if $a,b \in Y$ ∞ otherwise
$\mathsf{Met}(X,Y)$	$\sup_{x \in X} d_Y(a(x), b(x))$

Figure 1. Basic metric spaces

The heart of sensitivity analysis is the notion of *non-expansiveness*. A function $f : X \to Y$ between metric spaces is non-expansive if

$$d_Y(f(x_1), f(x_2)) \leq d_X(x_1, x_2)$$

for all $x_1, x_2 \in X$. Metric spaces and non-expansive functions form a category Met with rich structure, which we develop in the remainder of this section.

Non-expansiveness subsumes the notion of function sensitivity, thanks to the metric *scaling* operation (cf. Figure 1). Unpacking definitions, an r-sensitive function $X \to Y$ is exactly a non-expansive function from the r-scaled space $r \cdot X$ to Y. It is important to point out that we will extend multiplication to $\mathbb{R}^{\infty}_{\geq 0}$ in a non-standard way, by setting:

$$r \cdot \infty \triangleq \infty \qquad\qquad \infty \cdot r \triangleq \begin{cases} 0 & \text{if } r = 0 \\ \infty & \text{otherwise.} \end{cases}$$

Note that multiplication on $\mathbb{R}^{\infty}_{\geq 0}$ is *non-commutative* but otherwise well-behaved: it is associative, monotone in both arguments, and it distributes over addition. We will later see that this treatment of ∞ is crucial for scaling to distribute over sums, and for modeling function sensitivity in the presence of non-termination.

If $f \in \mathsf{Met}(X,Y)$, then $f \in \mathsf{Met}(r \cdot X, s \cdot Y)$ for any r and s such that $r \geq s$. In categorical language, this means that scaling extends to a bifunctor $\mathbb{R}^{\infty}_{\geq 0} \times \mathsf{Met} \to \mathsf{Met}$, where $\mathbb{R}^{\infty}_{\geq 0}$ is regarded as the category arising from the order $\geq$.

Now that we have pinned down the basic definitions for metric spaces, we turn our attention to simple constructions for building spaces. These operations will be used to interpret more complex types, as usual. The first observation is that there are two natural metrics on a product space $X \times Y$, denoted $X \,\&\, Y$ and $X \otimes Y$. The first one combines distances with the max operator, whereas the second one adds them up. The two metrics on products correspond to different sensitivity analyses. For instance, addition on real numbers is a non-expansive function $\mathbb{R} \otimes \mathbb{R} \to \mathbb{R}$, but not for $X \,\&\, Y$.

From the perspective of the Met category, there are other differences between the two metrics as well. The first, $X \,\&\, Y$,

[1] Although there is a wide range of semantics based on metric spaces, these were designed with quite different applications in mind. We return to this point when discussing related work.

[2] While Fuzz allows probabilistic sampling to model algorithms from differential privacy, the probabilistic features of Fuzz are orthogonal to the sensitivity analysis. We keep the discussion focused on sensitivity analysis, leaving modeling of the probabilistic features for future work.

yields the usual notion of Cartesian product on Met: given two non-expansive functions $f : Z \to X$ and $g : Z \to Y$, the function $\langle f, g\rangle : Z \to X \times Y$, that wraps the result of both functions in a pair,

$$\langle f, g\rangle(z) \triangleq (f(z), g(z)),$$

is non-expansive for $X \,\&\, Y$. Note that the projections

$$\pi_1 : X \times Y \to X \qquad \pi_2 : X \times Y \to Y$$

are trivially non-expansive for this metric.

The second product $X \otimes Y$ also supports the projections π_i, but not pairing. Instead, it allows us to split the metric of a space: the diagonal function $\delta(x) = (x, x)$ is a non-expansive function

$$(r + s) \cdot X \to (r \cdot X) \otimes (s \cdot X).$$

Furthermore, this is the metric for which the operations of currying and function application are non-expansive. More precisely, $(\mathsf{Met}, \otimes, \mathbf{1})$ is a symmetric monoidal category, and there is an adjunction

$$(-) \otimes X \dashv \mathsf{Met}(X, -)$$

making this structure closed. Here, non-expansive functions are endowed with the supremum metric of Figure 1.

We can also define a metric on the disjoint union of two spaces, placing elements from different components infinitely far apart. Note that this metric yields a coproduct on Met: if $f : X \to Z$ and $g : Y \to Z$, then the case-analysis function $[f, g] : X + Y \to Z$, defined as

$$[f, g](\iota_1(x)) \triangleq f(x) \qquad [f, g](\iota_2(y)) \triangleq g(y),$$

is non-expansive, where $\iota_1 : X \to X + Y$ and $\iota_2 : Y \to X + Y$ are the (trivially non-expansive) canonical injections.

We conclude by remarking useful identities that relate scaling to the above constructions:

$$\begin{aligned} r \cdot (X \,\&\, Y) &= r \cdot X \,\&\, r \cdot Y \\ r \cdot (X \otimes Y) &= r \cdot X \otimes r \cdot Y \\ r \cdot (X + Y) &= r \cdot X + r \cdot Y \\ r \cdot (s \cdot X) &= (rs) \cdot X. \end{aligned}$$

The case for sums relies crucially on the fact that $0 \cdot \infty = \infty$, which guarantees that the copies of X and Y in $X + Y$ remain infinitely apart after scaling. This point was overlooked in the original Fuzz work [37], where $0 \cdot \infty$ is defined as 0. In that case, the identity only holds for $r > 0$.

3. Core Fuzz

We now show how to use the structure described in the previous section to model a fragment of Fuzz without general recursion. The syntax, described in Figure 2, is based on a λ-calculus with products and sums, with a few modifications. First, Fuzz has two pair constructors, (e_1, e_2) and $\langle e_1, e_2\rangle$, corresponding to the two product metrics of last section. The first one is eliminated using case analysis ($\mathbf{let}\ (x, y) = e\,\mathbf{in}\,e'$), whereas the second one is eliminated using the projections π_i. The ! is a constructor with trivial computational meaning: it boxes its argument, which can later be unboxed with the form $\mathbf{let}\ !x = e\,\mathbf{in}\,e'$. This constructor is used by the type system to mark where we need to scale the metric of a space. Finally, we include two kinds of constants (real numbers k and a unit $()$), and one operation, addition on real numbers.

Programs execute under a standard call-by-value big-step semantics. We write $e \hookrightarrow v$ to say that term e evaluates to value v (also a term). We omit the definition of this relation, which can be found in the original paper [37].

$$\begin{aligned} e \in E ::= {} & x \mid k \in \mathbb{R} \mid e_1 + e_2 \mid () \\ & \mid \lambda x.\, e \mid e_1\ e_2 \\ & \mid (e_1, e_2) \mid \mathbf{let}\ (x, y) = e\,\mathbf{in}\,e' \\ & \mid \langle e_1, e_2\rangle \mid \pi_i\ e \\ & \mid !\,e \mid \mathbf{let}\ !x = e\,\mathbf{in}\,e' \\ & \mid \mathbf{inl}\,e \mid \mathbf{inr}\,e \mid (\mathbf{case}\,e\,\mathbf{of}\,\mathbf{inl}\,x \Rightarrow e_l \mid \mathbf{inr}\,y \Rightarrow e_r) \\ v \in V ::= {} & k \in \mathbb{R} \mid () \mid \lambda x.\, e \\ & \mid (v_1, v_2) \mid \langle v_1, v_2\rangle \mid !\,v \mid \mathbf{inl}\,v \mid \mathbf{inr}\,v \end{aligned}$$

Figure 2. Syntax of Core Fuzz

Turning to the type system of Fuzz, terms are typed with judgments of the form $\Gamma \vdash e : \sigma$, where Γ is a typing environment and σ is a type. The complete definition is given in Figure 3. The type system is loosely inspired by bounded linear logic, but unusual in some respects that we review here.

First, judgments keep track of the sensitivity of each variable used in a term. More precisely, a binding $x :_r \sigma$ in an environment Γ means that the variable x has type σ under Γ and that terms typed under Γ are r-sensitive with respect to x. Most rules use environment scaling ($r\Gamma$) and addition ($\Gamma + \Delta$) to track sensitivities. Note that the latter operation is only defined when Γ and Δ agree on the types of all variable bindings.[3]

An abstraction $\lambda x.\, e$ can only be typed if e is 1-sensitive on x (cf. $(\multimap I)$). Functions of different sensitivities must take arguments in a scaled type $!_r\,\sigma$ and unwrap them using **let** (cf. $(!E)$).

The Fuzz type system essentially corresponds to the constructions of last section, and can be interpreted in metric spaces in a straightforward manner. Given a type σ, we define a metric space $[\![\sigma]\!]$ with the rules

$$\begin{aligned} [\![\mathbb{R}]\!] &= \mathbb{R} & [\![1]\!] &= \mathbf{1} \\ [\![\sigma \multimap \tau]\!] &= \mathsf{Met}([\![\sigma]\!], [\![\tau]\!]) & [\![\sigma \otimes \tau]\!] &= [\![\sigma]\!] \otimes [\![\tau]\!] \\ [\![\sigma \,\&\, \tau]\!] &= [\![\sigma]\!] \,\&\, [\![\tau]\!] & [\![!_r\,\sigma]\!] &= r \cdot [\![\sigma]\!]. \end{aligned}$$

Each environment Γ is interpreted as a tensor product, scaled by the corresponding sensitivities:

$$[\![\emptyset]\!] = \mathbf{1} \qquad [\![\Gamma, x :_r \sigma]\!] = [\![\Gamma]\!] \otimes (r \cdot [\![\sigma]\!])$$

We sometimes treat elements of $[\![\Gamma]\!]$ as maps from variables in Γ to elements of the denotations of their types.

We can show by a straightforward induction how this interpretation interacts with scaling and addition.

Lemma 3.1. *For every r and Γ, $[\![r\Gamma]\!] = r \cdot [\![\Gamma]\!]$. For every Γ and Δ, if $\Gamma + \Delta$ is defined, then the diagonal function $\delta(x) = (x, x)$ is a non-expansive function $[\![\Gamma + \Delta]\!] \to [\![\Gamma]\!] \otimes [\![\Delta]\!]$.*

Finally, each typing derivation $\Gamma \vdash e : \sigma$ yields a non-expansive function $[\![e]\!] : [\![\Gamma]\!] \to [\![\sigma]\!]$ by structural induction:

(Var) $[\![x]\!](a) = a(x)$.

[3] In the original paper [37], two environments Γ, Δ can be added also when a variable appears either only in Γ or only in Δ. For simplicity, here we require instead all the variables to appear both in Γ and Δ. These are essentially equivalent, since we can always assume that the sensitivity of a variable is 0.

$$r, s \in \mathbb{R}^{\infty}_{\geq 0} \qquad \sigma, \tau ::= \mathbb{R} \mid 1 \mid \sigma \multimap \tau \mid \sigma \otimes \tau \mid \sigma \,\&\, \tau \mid \sigma + \tau \mid !_r \sigma \qquad \Gamma, \Delta ::= \emptyset \mid \Gamma, x :_r \sigma$$

$$\frac{\Gamma = x_1 :_{r_1} \sigma_1, ..., x_n :_{r_n} \sigma_n}{r\Gamma = x_1 :_{r \cdot r_1} \sigma_1, ..., x_n :_{r \cdot r_n} \sigma_n} \qquad \frac{\Gamma = x_1 :_{r_1} \sigma_1, ..., x_n :_{r_n} \sigma_n \qquad \Delta = x_1 :_{s_1} \sigma_1, ..., x_n :_{s_n} \sigma_n}{\Gamma + \Delta = x_1 :_{r_1 + s_1} \sigma_1, ..., x_n :_{r_n + s_n} \sigma_n}$$

$$\frac{(x :_r \sigma) \in \Gamma \qquad r \geq 1}{\Gamma \vdash x : \sigma} \text{ (Var)} \qquad \frac{k \in \mathbb{R}}{\Gamma \vdash k : \mathbb{R}} \text{ (Const)} \qquad \frac{\Gamma \vdash e_1 : \mathbb{R} \qquad \Delta \vdash e_2 : \mathbb{R}}{\Gamma + \Delta \vdash e_1 + e_2 : \mathbb{R}} \text{ (Plus)} \qquad \frac{}{\Gamma \vdash () : 1} \ (1I)$$

$$\frac{\Gamma, x :_1 \sigma \vdash e : \tau}{\Gamma \vdash \lambda x.\, e : \sigma \multimap \tau} \ (\multimap I) \qquad \frac{\Gamma \vdash e_1 : \sigma \multimap \tau \qquad \Delta \vdash e_2 : \sigma}{\Gamma + \Delta \vdash e_1\ e_2 : \tau} \ (\multimap E) \qquad \frac{\Gamma \vdash e_1 : \sigma \qquad \Delta \vdash e_2 : \tau}{\Gamma + \Delta \vdash (e_1, e_2) : \sigma \otimes \tau} \ (\otimes I)$$

$$\frac{\Gamma \vdash e : \sigma_1 \otimes \sigma_2 \qquad \Delta, x :_r \sigma_1, y :_r \sigma_2 \vdash e' : \tau}{r\Gamma + \Delta \vdash \mathbf{let}\ (x, y) = e\ \mathbf{in}\ e' : \tau} \ (\otimes E) \qquad \frac{\Gamma \vdash e_1 : \sigma \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \langle e_1, e_2 \rangle : \sigma \,\&\, \tau} \ (\&I) \qquad \frac{\Gamma \vdash e : \sigma_1 \,\&\, \sigma_2}{\Gamma \vdash \pi_i\ e : \sigma_i} \ (\&E)$$

$$\frac{\Gamma \vdash e : \sigma}{r\Gamma \vdash e : !_r \sigma} \ (!\,I) \qquad \frac{\Gamma \vdash e_1 : !_s \sigma \qquad \Delta, x :_{rs} \sigma \vdash e_2 : \tau}{r\Gamma + \Delta \vdash \mathbf{let}\,!x = e_1\ \mathbf{in}\ e_2 : \tau} \ (!\,E)$$

$$\frac{\Gamma \vdash e : \sigma}{\Gamma \vdash \mathbf{inl}\, e : \sigma + \tau} \ (+I_l) \qquad \frac{\Gamma \vdash e : \tau}{\Gamma \vdash \mathbf{inr}\, e : \sigma + \tau} \ (+I_r) \qquad \frac{\Gamma \vdash e : \sigma_1 + \sigma_2 \qquad \Delta, x :_r \sigma_1 \vdash e_l : \tau \qquad \Delta, y :_r \sigma_2 \vdash e_r : \tau}{r\Gamma + \Delta \vdash \mathbf{case}\, e\, \mathbf{of}\, \mathbf{inl}\, x \Rightarrow e_l \mid \mathbf{inr}\, y \Rightarrow e_r : \tau} \ (+E)$$

Figure 3. Core Fuzz Typing Rules

(Const) $[\![k]\!] = k$.

(Plus) $[\![e_1 + e_2]\!] = (+) \circ ([\![e_1]\!] \otimes [\![e_2]\!]) \circ \delta$.

$(1I)$ $[\![()]\!] = \star$, where $\star$ is the unique element of the singleton **1**.

$(\multimap I)$ $[\![\lambda x.\, e]\!] = \lambda [\![e]\!]$.

$(\multimap E)$ $[\![e_1\ e_2]\!] = \varepsilon \circ ([\![e_1]\!] \otimes [\![e_2]\!]) \circ \delta$.

$(\otimes I)$ $[\![(e_1, e_2)]\!] = ([\![e_1]\!] \otimes [\![e_2]\!]) \circ \delta$.

$(\otimes E)$ $[\![\mathbf{let}\ (x, y) = e_1\ \mathbf{in}\ e_2]\!] = [\![e_2]\!] \circ (id \otimes (r \cdot [\![e_1]\!])) \circ \delta$, where r is the sensitivity of x and y in e_2.

$(\&I)$ $[\![\langle e_1, e_2 \rangle]\!] = \langle [\![e_1]\!], [\![e_2]\!] \rangle$.

$(\&E)$ $[\![\pi_i e]\!] = \pi_i [\![e]\!]$.

$(!\,I)$ $[\![!\, e]\!] = r \cdot [\![e]\!]$, where r is the corresponding scaling factor.

$(!\,E)$ $[\![\mathbf{let}\ !x = e_1\ \mathbf{in}\ e_2]\!] = [\![e_2]\!] \circ (id \otimes (r \cdot [\![e_1]\!])) \circ \delta$.

$(+I_l)$ $[\![\mathbf{inl}\, e]\!] = \iota_1 \circ [\![e]\!]$.

$(+I_r)$ $[\![\mathbf{inr}\, e]\!] = \iota_2 \circ [\![e]\!]$.

$(+E)$ $[\![\mathbf{case}\, e\, \mathbf{of}\, \mathbf{inl}\, x \Rightarrow e_l \mid \mathbf{inr}\, y \Rightarrow e_r]\!] = [[\![e_l]\!], [\![e_r]\!]] \circ (r \cdot [\![e]\!])$, where r is the sensitivity of x and y.

We'll tacitly identify the denotation of typed closed terms $\vdash e : \sigma$ with elements $[\![e]\!] \in [\![\sigma]\!]$ in what follows. This interpretation is compatible with the operational semantics. We begin with the following standard lemma, showing that the denotational semantics behaves well with respect to weakening. As usual, the proof follow by simple induction on the typing derivation.

Lemma 3.2 (Weakening)**.** *Let e be a typed term $\Gamma_1, \Gamma_2 \vdash e : \sigma$. For any other environment Δ, we have a derivation $\Gamma_1, \Delta, \Gamma_2 \vdash e : \sigma$, whose semantics is equal to $[\![e]\!] \circ \pi_\Gamma$, where $\pi_\Gamma : [\![\Gamma_1, \Delta, \Gamma_2]\!] \to [\![\Gamma_1, \Gamma_2]\!]$ discards all components corresponding to Δ.*

To state a substitution lemma, we introduce some terminology and notation. We define a *substitution* as a finite partial map from variables to values, and use $\vec{v}$ to range over them. We write $e[\vec{v}]$ for the simultaneous substitution of the values $\vec{v}(x)$ for the variables x in e. We say that a substitution $\vec{v}$ is well-typed under Γ, written $\vec{v} : \Gamma$, if for all types σ, $\vdash \vec{v}(x) : \sigma$ if and only if there exists r such that $(x :_r \sigma) \in \Gamma$. We can readily lift the semantics of terms to substitutions, by assigning well-typed substitutions to denotations $[\![\vec{v}]\!] \in [\![\Gamma]\!]$ in the obvious way. Then:

Lemma 3.3 (Substitution)**.** *Let e be a well-typed term*

$$\Gamma, \Delta \vdash e : \sigma,$$

and $\vec{v} : \Gamma$ a well-typed substitution. Then, there is a derivation of

$$\Delta \vdash e[\vec{v}] : \sigma,$$

Furthermore, the semantics of this derivation $[\![e[\vec{v}]]\!]$ is such that

$$[\![e[\vec{v}]]\!] = [\![e]\!]([\![\vec{v}]\!], -).$$

(A similar result holds for the substitution of arbitrary expressions, but we don't need the extra generality.) With this lemma, we can show:

Lemma 3.4 (Preservation)**.** *If $\vdash e : \sigma$ and $e \hookrightarrow v$, then $\vdash v : \sigma$, and the semantics of both typing judgments are equal.*

Combined, the lemmas provide a short proof of metric preservation for this simple fragment of Fuzz.

Theorem 3.5 (Metric Preservation)**.** *Suppose that we have a well-typed program*

$$\Gamma \vdash e : \sigma,$$

and well-typed substitutions $\vec{v} : \Gamma$ and $\vec{v}' : \Gamma$. Then, there are well-typed values v and v' such that

$$e[\vec{v}] \hookrightarrow v \qquad \text{and} \qquad e[\vec{v}'] \hookrightarrow v'.$$

Furthermore,

$$d_{[\![\sigma]\!]}([\![v]\!], [\![v']\!]) \leq d_{[\![\Gamma]\!]}([\![\vec{v}]\!], [\![\vec{v}']\!]).$$

Proof. By Lemma 3.3, both $e[\vec{v}]$ and $e[\vec{v}]$ have type σ under the empty environment, and their denotations are equal to

$$[\![e]\!]([\![\vec{v}]\!]) \qquad \text{and} \qquad [\![e]\!]([\![\vec{v}']\!]).$$

By non-expansiveness of $[\![e]\!]$,

$$d_{[\![\sigma]\!]}([\![e]\!]([\![\vec{v}]\!]), [\![e]\!]([\![\vec{v}']\!])) \leq d_{[\![\Gamma]\!]}([\![\vec{v}]\!], [\![\vec{v}']\!]). \tag{1}$$

We can show by standard techniques that well-typed terms normalize, and thus we find values v and v' such that

$$e[\vec{v}] \hookrightarrow v \qquad \text{and} \qquad e[\vec{v}'] \hookrightarrow v'.$$

By Lemma 3.4, both v and v' have type σ under the empty environment, and their denotations are equal to those of $e[\vec{v}]$ and $e[\vec{v}']$. Thus, (1) yields the desired result. □

4. Metric CPOs

So far, we have considered a core version of Fuzz where all expressions are terminating. We need to handle non-termination to model all of Fuzz, which includes non-terminating expressions arising from recursive types. Our main tool will be *metric CPOs*, which blend the basic principles of sensitivity analysis with the domain-theoretic notion of *complete partial orders*. We first review the basic theory of these structures, and then show how to extend them with metrics—our main contribution. We will discuss in the next section how to use metric CPOs to interpret Fuzz with recursive types.

4.1 Preliminaries

Let $(X, \sqsubseteq)$ be a poset (that is, a set endowed with a reflexive, transitive, and anti-symmetric relation). We say that X is *complete* (or a *CPO*, for short) if every ω-chain of elements of X

$$x_0 \sqsubseteq x_1 \sqsubseteq x_2 \sqsubseteq \cdots$$

has a least upper bound, denoted $\bigsqcup_i x_i$. If X possesses a least element $\bot$, we say that X is *pointed*.

A function $f : X \to Y$ between CPOs is *monotone* if $x \sqsubseteq x'$ implies $f(x) \sqsubseteq f(x')$; in particular, f maps ω-chains to ω-chains. It is *continuous* if it preserves least upper bounds: $f\left(\bigsqcup_i x_i\right) = \bigsqcup_i f(x_i)$. Continuous functions between CPOs are the morphisms of a category, CPO. Note that continuous functions also form a CPO under the point-wise order

$$f \sqsubseteq g \Leftrightarrow \forall x.\, f(x) \sqsubseteq g(x),$$

with least upper bounds of chains given by

$$\left(\bigsqcup_i f_i\right)(x) = \bigsqcup_i f_i(x).$$

If the codomain is pointed, then the CPO is pointed as well, with the constant function that returns $\bot$ as the least element.

Continuous functions are useful because they allow us to interpret recursive definitions as fixed points.

Theorem 4.1 (Kleene)**.** *Let X be a pointed CPO. Every continuous function $f : X \to X$ has a least fixed point, given by*

$$\mathrm{fix}(f) = \bigsqcup_i f^i(\bot).$$

That is, $\mathrm{fix}(f) = f(\mathrm{fix}(f))$*, and* $\mathrm{fix}(f) \sqsubseteq x$ *for every x such that $x = f(x)$. The fixed point induces a continuous function* $\mathrm{fix} : \mathsf{CPO}(X, X) \to X$.

We use CPOs to represent outcomes of a computation. Any set X can be regarded as a CPO under the trivial discrete order

$$x \sqsubseteq x' \Leftrightarrow x = x'.$$

We use this order for sets of first-order values, such as $\mathbb{R}$ or $\mathbb{B}$. If X and Y are CPOs then so is $X \times Y$, with ordering

$$(x, y) \sqsubseteq (x', y') \Leftrightarrow x \sqsubseteq x' \wedge y \sqsubseteq y',$$

and the disjoint union $X + Y$, with ordering

$$\iota_i(x) \sqsubseteq \iota_j(x') \Leftrightarrow i = j \wedge x \sqsubseteq x'.$$

These constructions, with the obvious projections and injections, yield categorical products and sums in CPO. The singleton set $\mathbf{1}$ is a terminal object in this category. Currying and uncurrying continuous functions yields an adjunction

$$(-) \times X \dashv \mathsf{CPO}(X, -),$$

making CPO into a cartesian-closed category.

We represent computations that may run forever with pointed CPOs of the form $X_\bot$, constructed by adjoining a distinguished least element $\bot$ to a CPO X. The copy of X in $X_\bot$ represents computations that terminate successfully, whereas $\bot$ represents diverging ones. This construction extends to a functor on CPO in the obvious way. This functor has the structure of a monad, where the unit $\eta : X \to X_\bot$ injects X into $X_\bot$, and the multiplication $X_{\bot\bot} \to X_\bot$ collapses the two bottom elements into a single one. We write $\mathsf{CPO}_\bot$ for the Kleisli category of this monad. Its morphisms are continuous functions $X \to Y_\bot$, and composition of two arrows $g : Y \to Z_\bot$ and $f : X \to Y_\bot$ is given by $g^\dagger f$, where $g^\dagger : Y_\bot \to Z_\bot$ is the Kleisli lifting of g:

$$g^\dagger(\bot) = \bot$$
$$g^\dagger(y) = g(y) \qquad \text{if } y \neq \bot.$$

Note that there is a natural transformation $t : X_\bot \times Y_\bot \to (X \times Y)_\bot$, corresponding to forcing a pair of computations:

$$t(x, y) = \begin{cases} (x, y) & \text{if } x \neq \bot \text{ and } y \neq \bot \\ \bot & \text{otherwise.} \end{cases} \tag{2}$$

This, along with the unit $\eta_{\mathbf{1}} : \mathbf{1} \to \mathbf{1}_\bot$, makes $(-)_\bot$ into a lax symmetric monoidal functor. We use arrows in $\mathsf{CPO}_\bot$ to model programs in a call-by-value discipline, which take fully computed values as inputs and may either terminate or run forever.

4.2 Adding Metrics

If we want to mimic the sensitivity analysis of Section 2 on CPOs, we need to find a category of CPOs with metrics that is similar to Met in structure. In particular, we would like non-expansive functions to correspond to objects in this category, and to be closed under least upper bounds so that they can form a CPO.

Let's think about how this might hold. Suppose that we have an ω-chain $(f_i)_{i \in \mathbb{N}}$ of non-expansive continuous functions $X \to Y$, where both X and Y are CPOs endowed with metrics. To show that the limit $\bigsqcup_i f_i$ is non-expansive, we must show that for any pair of inputs x and x',

$$d\left(\bigsqcup_i f_i(x), \bigsqcup_i f_i(x')\right) \leq d(x, x'),$$

assuming that $d(f_i(x), f_i(x')) \leq d(x, x')$ for every $i \in \mathbb{N}$. Unfortunately, this does not hold in general. For instance, let $\mathbb{N}_\infty$ be the CPO of natural numbers extended with infinity, ordered in the usual way. We can define a metric on the disjoint union $X = \mathbb{N}_\infty + \mathbb{N}_\infty$ by setting

$$d(\iota_1(n), \iota_2(n)) = \begin{cases} 1 & \text{if } n = \infty \\ 0 & \text{otherwise,} \end{cases}$$

and stipulating that all other pairs of distinct points are infinitely apart. Then, the functions $f_n : X \to X$ ($n \in \mathbb{N}$), defined by

$$f_n(\iota_k(m)) \triangleq \iota_k(n),$$

are non-expansive and form an ω-chain, but do not satisfy the above properties, since at the limit we have

$$\begin{aligned} d\left(\bigsqcup_n f_n(\iota_1(0)), \bigsqcup_n f_n(\iota_2(0))\right) &= d(\iota_1(\infty), \iota_2(\infty)) \\ &= 1 \not\leq d(\iota_1(0), \iota_2(0)). \end{aligned}$$

These pathological cases force us to impose additional restrictions on the metrics we consider:

Definition 4.2. A *pre-metric CPO* is a CPO X endowed with a metric. We say that X is a *metric CPO* if its metric is compatible with the underlying partial order, in the following sense. Let $r \in \mathbb{R}^{\infty}_{\geq 0}$, and $(x_i)_{i\in\mathbb{N}}$ and $(x'_i)_{i\in\mathbb{N}}$ be two ω-chains on X, such that $d(x_i, x'_i) \leq r$ for all i. Then

$$d\left(\bigsqcup_i x_i, \bigsqcup_i x'_i\right) \leq r.$$

Metric CPOs and continuous, non-expansive functions between them form a category MetCPO.

All CPO constructions from the last section can be lifted to metric CPOs.[4] For instance, any discrete CPO with a metric is a metric CPO. Another simple case is sums.

Lemma 4.3. *If X and Y are metric CPOs, then so are $X+Y$ and $X_\perp$, under the sum metric of Section 2. Furthermore, $X + Y$ and the canonical injections give a coproduct on* MetCPO.

Since $\perp$ is infinitely apart from every other point, any morphism $f : X \to Y_\perp$ has the same termination behavior for any pair of inputs that are at finite distance. Just as in the previous section, we can extend $(-)_\perp$ to a monad on MetCPO, yielding a corresponding Kleisli category $\mathsf{MetCPO}_\perp$ representing potentially non-terminating computations.

We can also lift the cartesian product on Met to MetCPO.

Lemma 4.4. *Let X and Y be metric CPOs. The product metric $X \,\&\, Y$, with the standard CPO structure over $X \times Y$, is a metric CPO. The projections $\pi_1 : X \,\&\, Y \to X$ and $\pi_2 : X \,\&\, Y \to Y$ are non-expansive continuous functions, and make $X \,\&\, Y$ a cartesian product in* MetCPO.

Dealing with the tensor product and its additive metric requires more care. The following characterization of metric CPOs comes in handy.

Lemma 4.5. *A pre-metric CPO X is a metric CPO if and only if for every pair of ω-chains on X, $(x_i)_{i\in\mathbb{N}}$ and $(x'_i)_{i\in\mathbb{N}}$, we have*

$$d\left(\bigsqcup_i x_i, \bigsqcup_i x'_i\right) \leq \liminf_i d(x_i, x'_i).$$

Proof. ($\Rightarrow$) Consider an arbitrary $r > \liminf_i d(x_i, x'_i)$. There exists an infinite set $I \subseteq \mathbb{N}$ such that

$$\forall i \in I.\, d(x_i, x'_i) \leq r.$$

Since I is infinite, we get ω-chains $(x_i)_{i\in I}$ and $(x'_i)_{i\in I}$, and because X is a metric CPO, we find

$$d\left(\bigsqcup_{i\in\mathbb{N}} x_i, \bigsqcup_{i\in\mathbb{N}} x'_i\right) = d\left(\bigsqcup_{i\in I} x_i, \bigsqcup_{i\in I} x'_i\right) \leq r.$$

Since r can be arbitrarily close to $\liminf_i d(x_i, x'_i)$, we conclude

$$d\left(\bigsqcup_{i\in\mathbb{N}} x_i, \bigsqcup_{i\in\mathbb{N}} x'_i\right) \leq \liminf_{i\in\mathbb{N}} d(x_i, x'_i).$$

($\Leftarrow$) Suppose that

$$d\left(\bigsqcup_{i\in\mathbb{N}} x_i, \bigsqcup_{i\in\mathbb{N}} x'_i\right) \leq \liminf_{i\in\mathbb{N}} d(x_i, x'_i).$$

Suppose furthermore that there exists r such that

$$\forall i.\, d(x_i, x'_i) \leq r.$$

This implies $\liminf_i d(x_i, x'_i) \leq r$, from which we conclude. □

Lemma 4.6. *Let X and Y be metric CPOs. The space $X \otimes Y$ is a metric CPO over the standard product CPO.*

Proof. We have to show that the above metric is compatible with the order on $X \times Y$. By Lemma 4.5, it suffices to show that for every pair of ω-chains $(p_i)_{i\in\mathbb{N}}$ and $(p'_i)_{i\in\mathbb{N}}$,

$$d\left(\bigsqcup_i p_i, \bigsqcup_i p'_i\right) \leq \liminf_i d(p_i, p'_i).$$

By definition, this is equivalent to

$$\begin{aligned} &d\left(\bigsqcup_i x_i, \bigsqcup_i x'_i\right) + d\left(\bigsqcup_i y_i, \bigsqcup_i y'_i\right) \\ &\leq \liminf_i \left(d(x_i, x'_i) + d(y_i, y'_i)\right), \end{aligned}$$

where $p_i = (x_i, y_i)$ and $p'_i = (x'_i, y'_i)$. Since X and Y are metric CPOs, it suffices to show that

$$\begin{aligned} &\liminf_i d(x_i, x'_i) + \liminf_i d(y_i, y'_i) \\ &\leq \liminf_i \left(d(x_i, x'_i) + d(y_i, y'_i)\right), \end{aligned}$$

which always holds. □

As previously, this metric yields a symmetric monoidal category $(\mathsf{MetCPO}, \otimes, \mathbf{1})$ whose tensor unit is the terminal object. Note that the forcing natural transformation $t : X_\perp \times Y_\perp \to (X \times Y)_\perp$ of (2) is compatible with this metric, as well as the metric from Lemma 4.4:

$$\begin{aligned} t &: X_\perp \otimes Y_\perp \to (X \otimes Y)_\perp \\ t &: X_\perp \,\&\, Y_\perp \to (X \,\&\, Y)_\perp. \end{aligned}$$

Morphisms of metric CPOs form a metric CPO.

Lemma 4.7. *Let X and Y be metric CPOs. The set of morphisms $\mathsf{MetCPO}(X, Y)$ forms a metric CPO, inheriting its partial order from $\mathsf{CPO}(X, Y)$ and its metric structure from $\mathsf{Met}(X, Y)$.*

Proof. First, we must show that $\mathsf{MetCPO}(X, Y)$ is a pre-metric CPO, for which it suffices to show that it is closed under least upper bounds. We can then conclude by showing that this structure satisfies the metric CPO axiom.

We prove both properties with the following auxiliary result. Consider two chains $(f_i)_{i\in\mathbb{N}}$ and $(g_i)_{i\in\mathbb{N}}$ in $\mathsf{MetCPO}(X, Y)$, and two elements $x_1, x_2 \in X$. Pose $f = \bigsqcup_i f_i$

[4] We will later see in Section 4.3 how to lift much of the structure of CPO to MetCPO in a principled way, via a general fibrational construction.

and $g = \bigsqcup_i g_i$. Suppose that there exists r such that $d(f_i, g_i) \leq r$ for every $i \in \mathbb{N}$. Since each f_i and g_i is non-expansive, we get $d(f_i(x_1), g_i(x_2)) \leq r + d(x_1, x_2)$ for every $i \in \mathbb{N}$. We then conclude

$$d(f(x_1), g(x_2)) = d\left(\bigsqcup_i f_i(x_1), \bigsqcup_i g_i(x_2)\right) \leq r + d(x_1, x_2).$$

Now, we can see that $\mathsf{MetCPO}(X, Y)$ is closed under least upper bounds by taking $g_i = f_i$ and $r = 0$, since then $d(f(x_1), f(x_2)) \leq d(x_1, x_2)$. Furthermore, by taking $x_2 = x_1$, we find $d(f(x_1), g(x_1)) \leq r + 0$ and, since x_1 is arbitrary, we conclude $d(f, g) \leq r$ and that $\mathsf{MetCPO}(X, Y)$ is indeed a metric CPO. □

As expected, this allows us to curry and apply functions:

Lemma 4.8. *Let X be a metric CPO. The cartesian-closed structure of* CPO *induces an adjunction in* MetCPO*:*

$$(-) \otimes X \dashv \mathsf{MetCPO}(X, -),$$

making it a symmetric monoidal closed category.

Metric CPOs also support scaling.

Lemma 4.9. *Let X be a metric CPO and $r \in \mathbb{R}^{\infty}_{\geq 0}$. Then $r \cdot X$ is also a metric domain, under the same order as X.*

Proof. We just need to show that the new metric is compatible with the domain order. Suppose that we are given two chains on X, (x_i) and (x'_i), and that there is $r' \in \mathbb{R}^{\infty}_{\geq 0}$ such that $r \cdot d(x_i, x'_i) \leq r'$ for every i; we must show that $r \cdot d\left(\bigsqcup_i x_i, \bigsqcup_i x'_i\right) \leq r'$. If $r = 0$ or $r' = \infty$, the inequality becomes trivial and we're done. If $r \notin \{0, \infty\}$, then $d(x_i, x'_i) \leq r'/r$ for every i, hence $d\left(\bigsqcup_i x_i, \bigsqcup_i x'_i\right) \leq r'/r$ and we're done. The remaining case is when $r = \infty$ and $r' < \infty$. It must be the case that $d(x_i, x'_i) = 0$ for every i, so $d\left(\bigsqcup_i x_i, \bigsqcup_i x'_i\right) = 0$ and we are done. □

All the scaling identities of Section 2 remain valid, with the addition of

$$r \cdot X_\perp = (r \cdot X)_\perp.$$

Similarly to Section 2, we have inclusions

$$\mathsf{MetCPO}(X, Y) \subseteq \mathsf{MetCPO}(r \cdot X, s \cdot Y)$$
$$\mathsf{MetCPO}_\perp(X, Y) \subseteq \mathsf{MetCPO}_\perp(r \cdot X, s \cdot Y)$$

whenever $r \geq s$. Thus, scaling extends once again to a functor on both categories.

Finally, we interpret recursion by adding sensitivity information to the Kleene fixed-point combinator of Theorem 4.1:

Lemma 4.10. *Let X be a pointed metric CPO, and $r \in \mathbb{R}^{\infty}_{\geq 0}$. The* fix *combinator is a morphism $s \cdot \mathsf{MetCPO}(r \cdot X, X) \to X$, where*

$$s = \begin{cases} \frac{1}{1-r} & \text{if } r < 1 \\ \infty & \text{otherwise.} \end{cases}$$

Proof. Let f and g be two morphisms $r \cdot X \to X$. We can show by induction that

$$d(f^i(\perp), g^i(\perp)) \leq \left(\sum_{j<i} r^j\right) \cdot d(f, g). \tag{3}$$

Furthermore, when $r < 1$, we have

$$\sum_{j<i} r^j = \frac{1 - r^i}{1 - r}.$$

Therefore, the right-hand side of (3) is bounded by $s \cdot d(f, g)$ for every i. Since X is a metric CPO, we find that $d(\mathrm{fix}(f), \mathrm{fix}(g)) \leq s \cdot d(f, g)$ and conclude. □

4.3 Domain Equations

Many languages allow users to define data types with recursive structure. When giving a semantics to such languages, we are then led to solve the following problem: given an operator F that maps types to types, find a type μF such that $F(\mu F) = \mu F$. In domain-theoretic semantics, these are usually known as *domain equations*, and are elegantly described by the framework of *algebraic compactness* [24, 25]. After a short review of this framework, we show how it applies to $\mathsf{MetCPO}_\perp$, allowing us to solve many domain equations in that category and preparing the way to model recursive types in Fuzz in the next section.

Solutions to domain equations usually exploit existing CPO structure on the arrows of a category—a special case of *enriched category theory*. A CPO-category is a category whose hom-sets are CPOs and whose composition is continuous. There are many examples of such categories, including CPO and $\mathsf{CPO}_\perp$, but also MetCPO and $\mathsf{MetCPO}_\perp$ by Lemma 4.7. Additionally, CPO-categories are closed under products and opposites: in the first case, the order on arrows is just the product order, while in the second it is the same as the original category.

We are interested in solving domain equations for type operators F that can be extended to CPO-*functors*: these are functors between CPO-categories whose action on morphisms is continuous. This includes identity functors, constant functors, and all type operators that we have considered in this section, such as &, ⊗, etc. Since CPO-functors are closed under composition, they can model many recursive data types. For instance, the operator

$$F(X) = \mathbf{1} + \mathbb{R} \otimes X \tag{4}$$

is a CPO-functor $\mathsf{MetCPO} \to \mathsf{MetCPO}$ describing lists of real numbers. By construction, the distance between two lists of same length is the sum of the distances of corresponding pairs of numbers, and lists of different length are infinitely apart.

We say that a CPO-category $\mathcal{C}$ is *algebraically compact* if, for every CPO-functor $F : \mathcal{C} \to \mathcal{C}$, there exists an object μF and an isomorphism

$$i : F(\mu F) \cong \mu F \tag{5}$$

such that i is an initial algebra and i^{-1} is a final coalgebra. This means that μF is a valid solution for our domain equation, while the initiality and finality properties of i imply that μF is a canonical solution, characterized up to isomorphism. It turns out that algebraic compactness also provides canonical solutions to a larger class of domain equations on $\mathcal{C}$, given in terms of *mixed-variance* CPO-functors

$$F : \mathcal{C}^\star \to \mathcal{C},$$

where $\mathcal{C}^\star \triangleq \mathcal{C}^{op} \times \mathcal{C}$. More precisely, when $\mathcal{C}$ is algebraically compact, we can find an isomorphism

$$i : F(\mu F, \mu F) \cong \mu F \tag{6}$$

for some object $\mu F \in \mathcal{C}$. This solution is also characterized by a universal property [35, Theorem 3.4], although we won't need this fact. Solving domain equations for mixed-variance functors allows us to consider type operators involving exponentials $\mathcal{C}(-, -)$, which cannot be modeled directly as covariant functors as was done for (4).

Our first goal is to show that $\mathsf{MetCPO}_\perp$ is algebraically compact. We will use the following result.

Theorem 4.11 (Smyth and Plotkin [39])**.** *Let $\mathcal{C}$ be a* CPO*-category with a terminal object. Suppose that $\mathcal{C}(X,Y)$ is pointed for every X and Y, and that $f \circ \perp = \perp$ for every f. Suppose furthermore that $\mathcal{C}$ has colimits of ω-chains of embeddings; that is, of diagrams of the form*

$$X_0 \longrightarrow X_1 \longrightarrow X_2 \longrightarrow \cdots,$$

where every arrow e has an arrow $e^\#$ such that $e^\# e = id$ and $e e^\# \sqsubseteq id$. Then, $\mathcal{C}$ is algebraically compact.

All of these conditions hold of $\mathsf{MetCPO}_\perp$. (Its terminal object is the empty metric CPO $\mathbf{0}$.) The only thing remaining to check is that $\mathsf{MetCPO}_\perp$ has colimits of ω-chains of embeddings. For this purpose, we introduce a fibrational construction that will let us lift colimits in CPO to MetCPO, where it is easy to transfer colimits to $\mathsf{MetCPO}_\perp$. Later, we will reuse the fibrational machinery to show that the denotational semantics of Fuzz is adequate.

Let $F : \mathcal{E} \to \mathcal{D}$ be a functor. The *fiber category* over an object $X \in \mathcal{D}$ is the subcategory $\mathcal{E}_X$ of $\mathcal{E}$ consisting of objects and morphisms that are mapped to X and id_X by F, respectively. We say that F is a $\mathsf{CLat}_\wedge$*-fibration*[5] over $\mathcal{D}$ if it is a *posetal fibration with fibered limits*, or, more explicitly, if it satisfies the following properties.

1. For each $X \in \mathcal{D}$, the fiber category $\mathcal{E}_X$ is a poset. This property and the next one imply the faithfulness of F. We think of elements of $\mathcal{E}_X$ as abstract predicates or relations over X. For $A, B \in \mathcal{E}$, by $f : A \supset B$ we mean that $f \in \mathcal{D}(FA, FB)$, and there exists a (necessarily unique) morphism $f' : A \to B$ in $\mathcal{E}$ such that $f = Ff'$. We think such f as taking elements related by A to elements related by B.
2. For each arrow $f : X \to Y$ in $\mathcal{D}$ and $B \in \mathcal{E}_Y$, there is a (necessarily unique) element $f^*B \in \mathcal{E}_X$ (called the *inverse image of B by f*) such that

$$g : A \supset f^*B \qquad \text{iff} \qquad fg : A \supset B.$$

3. Each fibre poset $\mathcal{E}_X$ has arbitrary meets (denoted by $\bigcap S$), and each inverse image functor $A \mapsto f^*A$ preserves these meets.

If $\mathcal{D}$ is also a CPO-category, it is useful to require more structure of F. An object $B \in \mathcal{E}$ is called *admissible* [35, Definition 4.3] if $\mathcal{E}(A, B)$ is closed under limits of ω-chains for every A. We say that F itself is admissible if every object in $\mathcal{E}$ is admissible. The terminology is reminiscent of Pitts' work on relational properties of domains [35]. In fact, $\mathsf{CLat}_\wedge$-fibrations correspond exactly to his notion of normal relational structure with inverse images and intersections.

One example of $\mathsf{CLat}_\wedge$-fibration is the canonical forgetful functor $p : \mathsf{Met} \to \mathsf{Set}$. Each fiber Met_X corresponds to the poset of metrics on X, ordered by

$$d \leq d' \Leftrightarrow \forall x, x' \in X.\, d(x,x') \geq d'(x,x').$$

Thus, the intersection of a family of metrics $\{d_i\}_{i \in I}$ on a set is just their point-wise supremum $(\sup_i d_i)(x,y) = \sup_i d_i(x,y)$, and the inverse image of a metric d by a function f is given by $f^*d(x,y) = d(f(x), f(y))$.

Besides allowing us to lift structure across categories, as alluded to earlier, $\mathsf{CLat}_\wedge$-fibrations are stable under various category-theoretical constructions:

Lemma 4.12.

1. $\mathsf{CLat}_\wedge$*-fibrations preserve and create limits and colimits.*
2. $\mathsf{CLat}_\wedge$*-fibrations are closed under products, opposites, and pullbacks along any functor. The same conclusions hold for admissible* $\mathsf{CLat}_\wedge$*-fibrations over* CPO*-categories, restricting pullbacks along* CPO*-functors.*
3. *Let $\mathcal{D}$ be a* CPO*-category. Admissible objects in a* $\mathsf{CLat}_\wedge$*-fibration $F : \mathcal{E} \to \mathcal{D}$ are closed under inverse images and intersections [35, Lemma 4.14]. In particular, restricting F to the full subcategory $\mathcal{E}_{adm}$ of admissible objects of $\mathcal{E}$ yields an admissible* $\mathsf{CLat}_\wedge$*-fibration.*

To apply this result to MetCPO, we characterize it as the full subcategory of admissible objects of $\mathsf{CPO} \times_{\mathsf{Set}} \mathsf{Met}$, the category of premetric CPOs and non-expansive, continuous functions. The latter arises as the following pullback of functors, and r below is a $\mathsf{CLat}_\wedge$-fibration:

$$\begin{array}{ccc} \mathsf{CPO} \times_{\mathsf{Set}} \mathsf{Met} & \longrightarrow & \mathsf{Met} \\ \downarrow{\scriptstyle r} & & \downarrow{\scriptstyle p} \\ \mathsf{CPO} & \xrightarrow{U} & \mathsf{Set} \end{array}$$

Proposition 4.13. $(\mathsf{CPO} \times_{\mathsf{Set}} \mathsf{Met})_{adm} = \mathsf{MetCPO}$.

Proof. That every metric CPO is admissible follows by an argument analogous to Lemma 4.7. To see the converse, we can observe that a pre-metric CPO is a metric CPO if and only if the set of continuous, non-expansive functions $\mathbb{B}_r \to X$ is closed under least upper bounds for every $r \in \mathbb{R}^\infty_{\geq 0}$, where $\mathbb{B}_r$ is the discrete metric CPO consisting of two points at distance r. □

Corollary 4.14. *The forgetful functor $q : \mathsf{MetCPO} \to \mathsf{CPO}$ is an admissible* $\mathsf{CLat}_\wedge$*-fibration, and* MetCPO *is cocomplete.*

Proof. Appeal to Lemma 4.12. □

To conclude, we just need to show that ω-colimits of embeddings in $\mathsf{MetCPO}_\perp$ can be transferred from MetCPO. The key observation is that every embedding is the image of a morphism by the left adjoint $J : \mathsf{MetCPO} \to \mathsf{MetCPO}_\perp$ associated to the Kleisli category.

Lemma 4.15. *For any embedding $e \in \mathsf{MetCPO}_\perp(X,Y)$, there exists a unique morphism $m \in \mathsf{MetCPO}(X,Y)$ such that $e = Jm$.*

Proof. We write K for a right adjoint of J. Let $e \in \mathsf{MetCPO}_\perp(X,Y)$ be an embedding. Since it is a split monomorphism, $Ke = e^\dagger \in \mathsf{MetCPO}(X_\perp, Y_\perp)$ is also a (split) monomorphism. Moreover, $Ke(\perp) = \perp$; therefore, there exists a unique $m \in \mathsf{MetCPO}(X,Y)$ such that $e^\dagger = (m)_\perp$. By composing the unit η of the lifting monad, we conclude $e = \eta_Y \circ m = Jm$. □

Theorem 4.16. *The colimits of ω-chains of embeddings in* $\mathsf{MetCPO}_\perp$ *exist.*

Proof. From Lemma 4.15, every ω-chain (X_i, e_i) of embeddings in $\mathsf{MetCPO}_\perp$ is the J-image of an ω-chain (X_i, m_i) in MetCPO. Moreover, J preserves any colimits. Therefore the

[5] The name $\mathsf{CLat}_\wedge$-fibration stems from the fact that these structures correspond uniquely (via the Grothendieck construction) to a functor $\mathcal{D}^{op} \to \mathsf{CLat}_\wedge$, where the codomain is the category of complete lattices and meet-preserving functions between them.

$$\sigma, \tau ::= \cdots \mid \alpha \in T \qquad e ::= \cdots \mid \mathbf{fold}\, e \mid \mathbf{unfold}\, e$$
$$\Phi ::= (\alpha \mapsto \Phi(\alpha))_{\alpha \in T} \qquad v ::= \cdots \mid \mathbf{fold}\, v$$

$$\frac{\Gamma \vdash e : \Phi(\alpha)}{\Gamma \vdash \mathbf{fold}\, e : \alpha}\ (\mu I) \qquad \frac{\Gamma \vdash e : \alpha}{\Gamma \vdash \mathbf{unfold}\, e : \Phi(\alpha)}\ (\mu E)$$

Figure 4. Fuzz Recursive Types

J-image of a colimiting cone over (X_i, m_i), which exists by Corollary 4.14, gives a colimiting cone over (X_i, e_i). □

This result was the only condition missing for us to apply Theorem 4.11, and we have thus shown that $\mathsf{MetCPO}_\perp$ is algebraically compact.

5. Modeling Recursive Types

Now that we have shown algebraic compactness of $\mathsf{MetCPO}_\perp$, we are ready to model full Fuzz with recursive types (Figure 4). We will extend the basic setup of Section 3 and prove a metric preservation property, analogous to Theorem 3.5.

The full Fuzz language is parameterized by a finite set T of *type identifiers*, and a *definition environment* Φ mapping identifiers α to type expressions $\Phi(\alpha)$, which may themselves contain identifiers.[6] Identifiers behave as *iso-recursive types*: programs can freely cast between α and $\Phi(\alpha)$ by using the **fold** and **unfold** constructors (cf. (μE) and (μI)).

To interpret types, we first define a family of mixed-variance CPO-functors

$$F_\sigma : (\mathsf{MetCPO}_\perp^T)^\star \to \mathsf{MetCPO}_\perp$$

by induction on σ. The definition essentially follows that of $[\![\sigma]\!]$ in Section 3, but interpreting type identifiers as projections, and by switching the variance when traversing an arrow to the left:

$$F_\alpha(X, Y) \triangleq Y(\alpha)$$
$$F_{\sigma \multimap \tau}(X, Y) \triangleq \mathsf{MetCPO}_\perp(F_\sigma(Y, X), F_\tau(X, Y))$$

Finally, following Section 4.3, we use F to interpret a type σ as the metric CPO

$$[\![\sigma]\!] = F_\sigma(\mu F_\Phi, \mu F_\Phi),$$

where F_Φ is the lifting of F to Φ:

$$F_\Phi : (\mathsf{MetCPO}_\perp^T)^\star \to \mathsf{MetCPO}_\perp^T$$

$$F_\Phi(X, Y)(\alpha) \triangleq F_{\Phi(\alpha)}(X, Y).$$

With this definition, all the equations describing the interpretation of types for Core Fuzz carry over to $\mathsf{MetCPO}_\perp$. We think of μF_Φ as mapping each identifier α to its interpretation $[\![\alpha]\!] = \mu F_\Phi(\alpha)$. By unfolding definitions, the universal property of μF_Φ corresponds to a family of isomorphisms

$$i_\alpha : [\![\Phi(\alpha)]\!] \cong [\![\alpha]\!],$$

which give recursive types their intended semantics.

The interpretation of environments Γ remains the same: an iterated tensor product of scaled metric CPOs. As before, we scale and split environments with an analog of Lemma 3.1:

$$[\![r\Gamma]\!] = r \cdot [\![\Gamma]\!] \qquad \delta : [\![\Gamma + \Delta]\!] \to [\![\Gamma]\!] \otimes [\![\Delta]\!].$$

[6] This is slightly different from the original presentation of Fuzz, which has anonymous recursive types $\mu\alpha.\, \sigma$ instead of globally defined ones.

The biggest difference with respect to Core Fuzz is that the new semantics is monadic, in order to accommodate the presence of non-termination in a call-by-value discipline. Judgments $\Gamma \vdash e : \sigma$ now correspond to Kleisli arrows $[\![e]\!] : [\![\Gamma]\!] \to [\![\sigma]\!]_\perp$ in MetCPO, defined by adapting the semantics of Section 3. More concretely:

- we embed terminating computations from MetCPO (e.g. real number constants) into $\mathsf{MetCPO}_\perp$ with the canonical embedding $J : \mathsf{MetCPO} \to \mathsf{MetCPO}_\perp$; and
- we replace composition in MetCPO with its Kleisli counterpart.

For instance, consider the rule $(\& I)$: we want to interpret a typed term

$$\Gamma \vdash \langle e_1, e_2 \rangle : \sigma \,\&\, \tau,$$

where

$$\Gamma \vdash e_1 : \sigma \qquad \Gamma \vdash e_2 : \tau.$$

Since $[\![e_1]\!] : [\![\Gamma]\!] \to [\![\sigma]\!]_\perp$ and $[\![e_2]\!] : [\![\Gamma]\!] \to [\![\tau]\!]_\perp$, we must define $[\![\langle e_1, e_2 \rangle]\!]$ as the composition

$$[\![\Gamma]\!] \xrightarrow{\langle [\![e_1]\!], [\![e_2]\!] \rangle} [\![\sigma]\!]_\perp \,\&\, [\![\tau]\!]_\perp \xrightarrow{\ t\ } ([\![\sigma]\!] \,\&\, [\![\tau]\!])_\perp,$$

where t is the forcing morphism from (2). Additionally, we interpret the **fold** and **unfold** using the isomorphisms provided by algebraic compactness:

(μI) $[\![\mathbf{fold}\, e]\!] = i_\alpha \circ [\![e]\!]$

(μE) $[\![\mathbf{unfold}\, e]\!] = i_\alpha^{-1} \circ [\![e]\!]$

The basic semantic properties of Core Fuzz (Lemmas 3.2 to 3.4) generalize accordingly. As in other call-by-value languages, we also obtain:

Lemma 5.1. *Let $\vdash v : \sigma$ be a value. Then $[\![v]\!] = \eta(x)$ for some $x \in [\![\sigma]\!]$.*

Thanks to this result, we can treat the denotation of a value $\vdash v : \sigma$ as an element $[\![v]\!] \in [\![\sigma]\!]$. Once again, these properties lead to our main soundness result:

Theorem 5.2 (Metric Preservation)**.** *Suppose that we have a well-typed program*

$$\Gamma \vdash e : \sigma,$$

and well-typed substitutions $\vec{v} : \Gamma$ and $\vec{v}' : \Gamma$. Then

$$d_{[\![\sigma]\!]_\perp}([\![e[\vec{v}]]\!], [\![e[\vec{v}']]\!]) \le d_{[\![\Gamma]\!]}([\![\vec{v}]\!], [\![\vec{v}']\!]).$$

Unlike the previous statement of metric preservation, this result doesn't allow us to conclude anything about the termination behavior of the programs $e[\vec{v}]$ and $e[\vec{v}']$. To do so, we need the following property to connect the domain-theoretic and operational views of termination:

Lemma 5.3 (Adequacy)**.** *Let $\vdash e : \sigma$ be a well-typed term. If $[\![e]\!] \neq \bot$, there exists a value $\vdash v : \sigma$ such that $e \hookrightarrow v$.*

Then, we can recover that programs $e[\vec{v}]$ and $e[\vec{v}']$ from metric preservation indeed have the same termination behavior if $d_{[\![\Gamma]\!]}(\vec{v}, \vec{v}') < \infty$. By metric preservation, we find that

$$d_{[\![\sigma]\!]_\perp}([\![e[\vec{v}]]\!], [\![e[\vec{v}']]\!]) < \infty.$$

Now, imagine that $e[\vec{v}]$ terminates in a value v. By preservation, $[\![e[\vec{v}]]\!] = [\![v]\!] \neq \bot$. This implies that $[\![e[\vec{v}']]\!] \neq \bot$, because $d([\![v]\!], \bot) = \infty$. Finally, by adequacy, we are able to find v' such that $e[\vec{v}'] \hookrightarrow v'$.

Thus, we just need to prove adequacy (Lemma 5.3). Following an approach due to Plotkin [36], we generalize the

$$
\begin{aligned}
\hat{F}_{\mathbb{R}}(A,B) &= \{(k,k) \mid k \in \mathbb{R}\} \\
\hat{F}_{1}(A,B) &= \{(\star,())\} \\
\hat{F}_{\sigma\otimes\tau}(A,B) &= \{((x,y),(v_x,v_y)) \mid (x,v_x) \in \hat{F}_{\sigma}(A,B), (y,v_y) \in \hat{F}_{\tau}(A,B)\} \\
\hat{F}_{\sigma\&\tau}(A,B) &= \{((x,y),\langle v_x,v_y\rangle) \mid (x,v_x) \in \hat{F}_{\sigma}(A,B), (y,v_y) \in \hat{F}_{\tau}(A,B)\} \\
\hat{F}_{\sigma+\tau}(A,B) &= \{(\iota_1(x), \mathbf{inl}\, v) \mid (x,v) \in \hat{F}_{\sigma}(A,B)\} \cup \{(\iota_2(y), \mathbf{inl}\, v) \mid (y,v) \in \hat{F}_{\tau}(A,B)\} \\
\hat{F}_{\sigma\multimap\tau}(A,B) &= \{(f, \lambda x.\, e) \mid \forall (x,v) \in \hat{F}_{\sigma}(B,A).\, (f(x), e[x \mapsto v]) \in \hat{F}_{\tau}(A,B)^{\perp}\} \ ((-)^{\perp} \text{ is as in (8)}) \\
\hat{F}_{!\sigma}(A,B) &= \{(x, !\, v) \mid (x,v) \in \hat{F}_{\sigma}(A,B)\} \\
\hat{F}_{\alpha}(A,B) &= \{(x, \mathbf{fold}\, v) \mid (x,v) \in B(\alpha)\} \\
\hat{F}_{\Phi}(A,B)(\alpha) &= \hat{F}_{\Phi(\alpha)}(A,B)
\end{aligned}
$$

Figure 5. Definition of logical relations for adequacy

adequacy lemma by constructing, for each type σ, a logical relation $S_\sigma \subseteq [\![\sigma]\!] \times V$ satisfying the following property: if $\Gamma \vdash e : \sigma$, $a \in [\![\Gamma]\!]$, and $\vec{v} : \Gamma$, then

$$(\forall (x :_r \tau) \in \Gamma.\, (a(x), \vec{v}(x)) \in S_\tau) \Rightarrow ([\![e]\!](a), e[\vec{v}]) \in S_\sigma^{\perp}, \quad (7)$$

where $S_\sigma^{\perp}$ is a relation between $[\![\sigma]\!]_\perp$ and expressions E, defined as

$$S_\sigma^{\perp} \triangleq \{(x,e) \mid x \neq \perp \Rightarrow \exists v.\, e \hookrightarrow v \wedge (x,v) \in S_\sigma\}. \quad (8)$$

Adequacy follows from (7) by instantiating Γ with the empty environment. Our goal is to define S_σ so that (7) is strong enough to be established by a simple induction on the typing derivation. This property almost completely determines how S_σ should be defined; it must satisfy equations including

$$S_{\mathbb{R}} = \{(k,k) \mid k \in \mathbb{R}\} \quad (9)$$

$$S_{\sigma\&\tau} = \{((x,y),\langle v_x,v_y\rangle) \mid (x,v_x) \in S_\sigma, (y,v_y) \in S_\tau\} \quad (10)$$

$$S_\alpha = \{(i_\alpha(x), \mathbf{fold}\, v) \mid (x,v) \in S_{\Phi(\alpha)}\}. \quad (11)$$

However, the last equation points to an obstacle: this logical relation cannot be defined by induction on σ since the defining equation for S_α mentions $S_{\Phi(\alpha)}$, and $\Phi(\alpha)$ is *not* smaller than α. To overcome this circularity, we use a method due to Pitts [35, Theorem 4.16], originally stated in terms of his relational structures and adapted here to $\mathsf{CLat}_\wedge$-fibrations.

Theorem 5.4. *Let $\mathcal{D}$ be a CPO-category that is algebraically compact, $F : \mathcal{D}^\star \to \mathcal{D}$ be a CPO-functor, and $G : \mathcal{E} \to \mathcal{D}$ be an admissible $\mathsf{CLat}_\wedge$-fibration. Suppose we can lift F to $\mathcal{E}$, in the sense that there exists a functor $\hat{F} : \mathcal{E}^\star \to \mathcal{E}$ such that the following diagram commutes:*

$$\begin{array}{ccc} \mathcal{E}^\star & \xrightarrow{\hat{F}} & \mathcal{E} \\ \downarrow{\scriptstyle G^\star} & & \downarrow{\scriptstyle G} \\ \mathcal{D}^\star & \xrightarrow{F} & \mathcal{D} \end{array}$$

Then, we can construct $\mu\hat{F} \in \mathcal{E}_{\mu F}$ such that

$$\mu\hat{F} = (i^{-1})^* \hat{F}(\mu\hat{F}, \mu\hat{F}),$$

where $i : F(\mu F, \mu F) \cong \mu F$ is the isomorphism given by algebraic compactness, as in (6).

By unpacking definitions, the lifted functor $\hat{F}$ simply maps a pair of relations $A \in \mathcal{E}_X$ and $B \in \mathcal{E}_Y$ to a relation $\hat{F}(A,B) \in \mathcal{E}_{F(X,Y)}$, in a way that depends covariantly on Y and contravariantly on X. Intuitively, we'll use $\hat{F}$ to express the logical relations S_σ as the solution of fixed point equations, similarly to how we interpret recursive functions and types.

To apply Theorem 5.4, we must first find an admissible $\mathsf{CLat}_\wedge$-fibration over $\mathsf{MetCPO}_\perp$ whose fibers are relations between metric CPOs and values. This is given by the following category Rel_V.

1. Objects are pairs (X,P), where X is a metric CPO, and $P \subseteq X \times V$ is a relation such that
$$(\forall i.\, (x_i, v) \in P) \Rightarrow \left(\bigsqcup_i x_i, v\right) \in P,$$
for all ω-chains (x_i) in X and $v \in V$.
2. Morphisms $(X,P) \to (Y,Q)$ are continuous, non-expansive functions $f : X \to Y_\perp$ such that, whenever $(x,v) \in P$ and $f(x) \neq \perp$, we have $(f(x), v) \in Q$.

We let R denote the forgetful functor $\mathsf{Rel}_V \to \mathsf{MetCPO}_\perp$; this results in an admissible $\mathsf{CLat}_\wedge$-fibration. Intersections are given by intersections of relations, and the inverse image of $(X,P) \in \mathsf{Rel}_V$ along $f \in \mathsf{MetCPO}_\perp(Y,X)$ is given by

$$f^*(X,P) \triangleq (Y, \{(x,v) \mid (f(x),v) \in P \vee f(x) = \perp\}).$$

Now that we have an appropriate $\mathsf{CLat}_\wedge$-fibration, we just need to lift F_σ and F_Φ across R; the complete definition is given in Figure 5. We have all the ingredients to apply Theorem 5.4 and compute the fixed point $\mu\hat{F}_\Phi$. We then pose $S_\sigma \triangleq \hat{F}_\sigma(\mu\hat{F}_\Phi, \mu\hat{F}_\Phi)$. By folding this definition, and using the characterization of $\mu\hat{F}_\Phi$ in Theorem 5.4, we can read off simple recursive equations describing each relation S_σ, including (9) to (11). With these equations, proving (7) (and thus Lemma 5.3) is a simple argument by induction on the typing derivation.

Remark 5.5. Alternatively, we could have also characterized Rel_V reusing the machinery of Lemma 4.12, specifically by pulling back $\mathsf{SubCPO}_\perp$, the category of admissible subobjects of $\mathsf{CPO}_\perp$, as depicted below.

$$\begin{array}{ccccc} \mathsf{Rel}_V & & \longrightarrow & & \mathsf{SubCPO}_\perp \\ \downarrow{\scriptstyle R} & & & & \downarrow \\ \mathsf{MetCPO}_\perp & \xrightarrow{q_\perp} & \mathsf{CPO}_\perp & \xrightarrow{(-)\times V} & \mathsf{CPO}_\perp \end{array}$$

In this diagram, by $I \times V$ we mean the *coproduct* of V-many copies of I in $\mathsf{CPO}_\perp$, which can be automatically lifted from CPO via the Kleisli adjunction.

6. Related works

Since the seminal works of Arnold and Nivat [5], and de Bakker and Zucker [18], several authors have used metric spaces as a foundation for denotational semantics. The technical motivations are often similar to those for order-based structures, such as CPOs, since the well-known *Banach fixed-point theorem* yields a natural tool for interpreting recursive definitions of functions and types.

A recurrent theme in these approaches is the use of *ultrametric* spaces, where the triangle inequality is replaced with the stronger variant

$$d(x,z) \leq \max(d(x,y), d(y,z)).$$

Typically, ultrametrics express that two objects (e.g., execution traces, sets of terms, etc.) are equal up to a finite approximation: the bigger the approximation, the closer the two objects are. For instance, we can define an ultrametric on the set of sequences of program states by posing $d(\vec{s}_1, \vec{s}_2) = 2^{-c(\vec{s}_1,\vec{s}_2)}$, where $c(\vec{s}_1, \vec{s}_2)$ is the length of the largest common prefix of $\vec{s}_1$ and $\vec{s}_2$.

Ultrametrics on traces and trees appear in much of the earlier work on the subject, where they provide a natural setting for discussing language features such as nondeterminism and concurrency [4, 18, 31, 32]. See van Breugel [40] for a good introduction to the subject, and Baier and Majster-Cederbaum [7], Majster-Cederbaum and Zetzsche [33] for a comparison between the metric approaches and their order-based counterparts, discussing the relative pros and cons. A similar use of ultrametric spaces appears in a denotational model of PCF given by Escardó [23]. In this model, the metric structure describes intensional temporal aspects of PCF programs, and its extensional collapse allows us to recover the standard Scott model. Such intensional uses contrast with our metric CPOs, where the metrics describe mostly extensional aspects of programs.

A different use of ultrametrics emerged for modeling recursive types in functional languages, starting with MacQueen et al. [30], and continuing with Abadi and Plotkin [1], Abadi et al. [2], Amadio [3]; see also Chroboczek [16] for a similar approach based on game semantics. An interesting aspect of these models is that the metric structure is often used in conjunction with the CPO structure. These approaches have been extended recently to model more advanced language features (e.g. references), providing a semantic framework for investigating guardedness, step-indexing and Kripke's world semantics. Works in this direction include those by Birkedal et al. [10, 11, 12], Schwinghammer et al. [38]. In the model presented in these works the metric structure is used to express the convergence properties that are behind some of the syntactic structures that are used in languages with guarded definitions, e.g. Nakano's recursion modality [34]. A similar approach has also been used by Krishnaswami and Benton [29] in the context of reactive and event-based programming, which models interactive programs as operating on streams; stream functions are contractive maps in their model. Our model differs from the ones used in this context, i.e. contractivity plays a different role and again the requirement we have on the domain structure is a sort of compatibility. However we plan to explore whether our model can be used for similar goals in future works.

Another line of work, this time not based on ultrametrics, is given by Kozen [28], who uses Banach lattices—a special kind of metric space—and non-expansive linear operators between them to model probabilistic programs. Spaces of subprobability distributions over a set of values are modeled as Banach lattices. Although this is closer in spirit to our use of metrics, there is still a crucial conceptual difference: Kozen uses non-expansiveness to model the loss of mass of a distribution as a program executes, due to the possibility of non-termination. Indeed, he shows how non-expansiveness in this setting corresponds to the usual monotonicity of domain-theoretic functions.

7. Conclusion

In this work we have studied a semantical structure suitable to describe the notion of program sensitivity in higher-order languages with recursive types and non-termination. We have shown the effectiveness of our approach by interpreting the deterministic fragment of *Fuzz* [37].

As future work, we plan to extend our approach to cover the probability monad of *Fuzz*. While metric interpretation of probabilities are widespread, e.g. [6, 19–21, 28], these use a different metric on distributions than the one proposed by Reed and Pierce [37], a non-standard metric directly inspired by the definition of differential privacy. Interpreting this metric could also hint at how to interpret a larger class of metrics called *f-divergences* [17]. An orthogonal direction is to study an interpretation of *DFuzz* [26], a dependently typed version of *Fuzz* that allows proving differential privacy for programs whose privacy depends on values provided at runtime. This may require an extension of our framework to accommodate their use of sized types.

The semantical structure we consider could also give meaning to the program analysis studied by Chaudhuri et al. [14, 15]. Their notion of *robustness* is analogous to the notion of sensitivity we consider in this paper. However, their program analysis is based on previous work by the same authors for analyzing *program continuity* [13]. Considering restrictions or relaxations of metric CPOs that would enable describing these notions of continuity and robustness is also an interesting avenue for future work.

References

[1] M. Abadi and G. D. Plotkin. A per model of polymorphism and recursive types. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science (LICS '90), Philadelphia, Pennsylvania, USA, June 4-7, 1990*, pages 355–365, 1990.

[2] M. Abadi, B. C. Pierce, and G. D. Plotkin. Faithful ideal models for recursive polymorphic types. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), Pacific Grove, California, USA, June 5-8, 1989*, pages 216–225, 1989.

[3] R. M. Amadio. Recursion over realizability structures. *Inf. Comput.*, 91(1):55–85, 1991.

[4] P. America and J. J. M. M. Rutten. Solving reflexive domain equations in a category of complete metric spaces. In *Mathematical Foundations of Programming Language Semantics, 3rd Workshop, Tulane University, New Orleans, Louisiana, USA, April 8-10, 1987, Proceedings*, pages 254–288, 1987.

[5] A. Arnold and M. Nivat. Metric interpretations of infinite trees and semantics of non deterministic recursive programs. *Theor. Comput. Sci.*, 11:181–205, 1980.

[6] C. Baier and M. Z. Kwiatkowska. Domain equations for probabilistic processes. *Electr. Notes Theor. Comput. Sci.*, 7:34–54, 1997.

[7] C. Baier and M. E. Majster-Cederbaum. Denotational semantics in the cpo and metric approach. *Theoretical Computer Science*, 135(2):171 – 220, 1994.

[8] G. Barthe, B. Köpf, F. Olmedo, and S. Z. Béguelin. Probabilistic relational reasoning for differential privacy. In *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012*, pages 97–110, 2012.

[9] G. Barthe, M. Gaboardi, E. J. Gallego Arias, J. Hsu, A. Roth, and P.-Y. Strub. Higher-order approximate relational refinement types for mechanism design and differential privacy. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pages 55–68. ACM, 2015.

[10] L. Birkedal, K. Støvring, and J. Thamsborg. Realizability semantics of parametric polymorphism, general references, and recursive types. In *Foundations of Software Science and Computational Structures, 12th International Conference, FOSSACS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, pages 456–470, 2009.

[11] L. Birkedal, K. Støvring, and J. Thamsborg. The category-theoretic solution of recursive metric-space equations. *Theor. Comput. Sci.*, 411(47):4102–4122, 2010.

[12] L. Birkedal, R. E. Møgelberg, J. Schwinghammer, and K. Støvring. First steps in synthetic guarded domain theory: Step-indexing in the topos of trees. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada*, pages 55–64, 2011.

[13] S. Chaudhuri, S. Gulwani, and R. Lublinerman. Continuity analysis of programs. In *Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010, Madrid, Spain, January 17-23, 2010*, pages 57–70, 2010.

[14] S. Chaudhuri, S. Gulwani, R. Lublinerman, and S. Navidpour. Proving programs robust. In *SIGSOFT/FSE'11 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-19) and ESEC'11: 13rd European Software Engineering Conference (ESEC-13), Szeged, Hungary, September 5-9, 2011*, pages 102–112, 2011.

[15] S. Chaudhuri, S. Gulwani, and R. Lublinerman. Continuity and robustness of programs. *Commun. ACM*, 55(8):107–115, 2012.

[16] J. Chroboczek. Subtyping recursive games. In *TLCA*, pages 61–75, 2001.

[17] I. Csiszár and P. Shields. Information theory and statistics: A tutorial. *Foundations and Trends® in Communications and Information Theory*, 1(4):417–528, 2004.

[18] J. W. de Bakker and J. I. Zucker. Denotational semantics of concurrency. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 153–158, 1982.

[19] E. P. de Vink and J. J. M. M. Rutten. Bisimulation for probabilistic transition systems: A coalgebraic approach. *Theor. Comput. Sci.*, 221(1-2):271–293, 1999.

[20] J. den Hartog, E. P. de Vink, and J. W. de Bakker. Metric semantics and full abstractness for action refinement and probabilistic choice. *Electr. Notes Theor. Comput. Sci.*, 40: 72–99, 2000.

[21] J. Desharnais, R. Jagadeesan, V. Gupta, and P. Panangaden. The metric analogue of weak bisimulation for probabilistic processes. In *17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22-25 July 2002, Copenhagen, Denmark, Proceedings*, pages 413–422, 2002.

[22] C. Dwork, F. McSherry, K. Nissim, and A. D. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, pages 265–284, 2006.

[23] M. H. Escardó. A metric model of PCF, 1999. Presented at the Workshop on Realizability Semantics and Applications associated to the Federated Logic Conference, held in Trento, June 29-July 12, 1999.

[24] M. P. Fiore and G. D. Plotkin. An axiomatization of computationally adequate domain theoretic models of FPC. In *LICS*, pages 92–102. IEEE Computer Society, 1994.

[25] P. Freyd. Algebraically complete categories. pages 95–104.

[26] M. Gaboardi, A. Haeberlen, J. Hsu, A. Narayan, and B. C. Pierce. Linear dependent types for differential privacy. In *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*, pages 357–370, 2013.

[27] D. Hofmann, G. J. Seal, and W. Tholen, editors. *Monoidal Topology*. Cambridge University Press, 2014.

[28] D. Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 22(3):328 – 350, 1981.

[29] N. R. Krishnaswami and N. Benton. Ultrametric semantics of reactive programs. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada*, pages 257–266, 2011.

[30] D. B. MacQueen, G. D. Plotkin, and R. Sethi. An ideal model for recursive polymorphic types. In *Conference Record of the Eleventh Annual ACM Symposium on Principles of Programming Languages, Salt Lake City, Utah, USA, January 1984*, pages 165–174, 1984.

[31] M. E. Majster-Cederbaum. On the uniqueness of fixed points of endofunctors in a category of complete metric spaces. *Inf. Process. Lett.*, 29(6):277–281, 1988.

[32] M. E. Majster-Cederbaum and F. Zetzsche. Towards a foundation for semantics in complete metric spaces. *Inf. Comput.*, 90(2):217–243, 1991.

[33] M. E. Majster-Cederbaum and F. Zetzsche. The comparison of a cpo-based semantics with a cms-based semantics for CSP. *Theor. Comput. Sci.*, 124(1):1–40, 1994.

[34] H. Nakano. A modality for recursion. In *15th Annual IEEE Symposium on Logic in Computer Science, Santa Barbara, California, USA, June 26-29, 2000*, pages 255–266, 2000.

[35] A. Pitts. *Relational Properties of Domains*. Computer Laboratory Cambridge: Technical report. University of Cambridge, Computer Laboratory, 1993.

[36] G. Plotkin. Lectures on predomains and partial functions. Notes for a course given at the Center for the Study of Language and Information, Stanford, 1985.

[37] J. Reed and B. C. Pierce. Distance makes the types grow stronger: A calculus for differential privacy. In *Proceedings of the 15th ACM SIGPLAN International Conference on Functional Programming*, ICFP '10, pages 157–168, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-794-3.

[38] J. Schwinghammer, L. Birkedal, and K. Støvring. A step-indexed kripke model of hidden state via recursive properties on recursively defined metric spaces. In *Foundations of Software Science and Computational Structures - 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, pages 305–319, 2011.

[39] M. B. Smyth and G. D. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM J. Comput.*, 11(4):761–783, 1982.

[40] F. van Breugel. An introduction to metric semantics: operational and denotational models for programming and specification languages. *Theor. Comput. Sci.*, 258(1-2):1–98, 2001.