A Semantic Account of Metric Preservation

Arthur Azevedo de Amorim
University of Pennsylvania, USA

Marco Gaboardi
University at Buffalo,
The State University of New York, USA

Justin Hsu
University of Pennsylvania, USA

Shin-ya Katsumata
Research Institute for Mathematical Sciences,
Kyoto University, Japan

Ikram Cherigui
École Normale Supérieure Paris, France

Abstract

Program sensitivity measures how robust a program is to small changes in its input, and is a fundamental notion in domains ranging from differential privacy to cyber-physical systems. A natural way to formalize program sensitivity is in terms of metrics on the input and output spaces, requiring that an r-sensitive function map inputs that are at distance d to outputs that are at distance at most $r \cdot d$. Program sensitivity is thus an analogue 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, by 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.

Categories and Subject Descriptors F.3.2 [*Logics and Meaning of Programs*]: Semantics of Programming Languages

Keywords domain theory, program sensitivity, metric spaces, Lipschitz continuity

1. Introduction

In many applications, programs should not be too sensitive to small variations in their inputs. For example, cyber-physical systems must cope with measurement errors from the outside world, whereas differential privacy [22] tries to protect the privacy of individuals in a database by bounding the influence that the presence of each individual has on the result of database queries. *Program sensitivity* (or *Lipschitz continuity*) has recently emerged as a useful tool for reasoning about such requirements. 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.

Motivated by its useful applications, many techniques have been proposed for reasoning about program sensitivity formally, including static analyses for imperative programs [14], relational program logics [9], and relational refinement types [8]. In this work, we focus on the approach proposed by Reed and Pierce [38] in the *Fuzz* programming language.[1] Fuzz is a purely functional PCF-like language that provides a clean, compositional sensitivity analysis for higher-order programs. This analysis is implemented as a linear indexed type system: every Fuzz type τ is endowed with a notion of distance, and function types $!_r \, \tau \multimap \sigma$ carry a numeric index r describing their sensitivity.

Establishing soundness for Fuzz is challenging due to the presence of general recursive functions and types. The central technical result, *metric preservation* [38], relied on the definition of intricate, syntactic logical relations that mixed step-indexing and metric information. The logical relations were used for two purposes: to define distances, and to prove soundness. This mixed approach obscures the connection between Fuzz programs and the theory of metric spaces.

In this paper, we propose an alternative, domain-theoretic treatment of sensitivity and metric preservation in the presence of general recursion. Our main contribution is a new notion of *metric CPO*, which is a complete partial order endowed with a compatible metric, in the sense that every open ball is stable under limits of ω-chains. While simple, this notion of compatibility provides a natural extension of the notion of sensitivity to partial functions and has received little attention in the literature. We use metric CPOs to build a model of Fuzz that validates metric preservation.[2] This model helps clarify some aspects of the analysis of Fuzz; for instance, a result on least fixed points on metric CPOs gave us a much more precise encoding of recursive functions in Fuzz (cf. Lemma 4.9 and Section 5.3).

By grounding our work on well-established domain-theoretic notions, we can leverage a vast array of tools to model recursive functions and types. Technically, we first show that metric CPOs

[1] The language did not have a name at first; "Fuzz" was only introduced later (e.g. [26, 27]).

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

have the appropriate structure for solving recursive domain equations, following the approach laid out by Freyd [25], Smyth and Plotkin [40], and others. Then, we prove the adequacy of the denotational semantics of Fuzz with respect to its operational semantics by adapting a method due to Pitts [36] for constructing a family of type-indexed logical relations. We use fibrational category theory as a key technical ingredient, for smoothly lifting colimits of CPOs to the metric setting and for defining relations on metric CPOs.

While our work is primarily motivated by Fuzz, we believe that metric CPOs can provide useful guidance for studying metric aspects of programs. For instance, differential privacy is a form of non-expansiveness [38, Proposition 4.1], but that result applies to total functions, and it is not clear what it means to partial ones. Another intriguing question is evaluating what constructs from the theory of metric spaces could be incorporated in the design of languages and libraries. For instance, the Banach fixed-point theorem, a central tool in analysis, has a constructive interpretation that permits approximating a fixed point up to arbitrary precision, but it requires reasoning about the sensitivity of programs. We plan to investigate these and other directions in future work.

Outline. We will 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). Then, we introduce metric CPOs in Section 4 and demonstrate how the constructions in the terminating fragment 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 these tools to extend our model of Fuzz with recursive types and to prove metric preservation in Section 5. We conclude with a discussion of related work and some 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. [28]).

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 respects. 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. In additional to standard metric spaces, such as the real numbers $\mathbb{R}$ under the Euclidean metric, we will consider metrics defined on products, sums, and functions; Figure 1 summarizes these constructions.

The essence of sensitivity analysis lies in 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.

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

Figure 1. Basic metric spaces

To define scaling by r, we extend multiplication to $\mathbb{R}^{\infty}_{\geq 0}$:

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

It is important to point out that multiplication on $\mathbb{R}^{\infty}_{\geq 0}$ is *non-commutative* since $0 \cdot \infty = \infty$ and $\infty \cdot 0 = 0$. Otherwise, it is 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 $X \mathbin{\&} Y$ and $X \otimes Y$. The first one combines distances by taking the maximum, while the second one adds them up. These two metrics 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 the signature $\mathbb{R} \mathbin{\&} \mathbb{R} \to \mathbb{R}$.

Categorically speaking, there are other differences between the metrics. The first, $X \mathbin{\&} Y$ 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 $\langle f, g\rangle : Z \to X \times Y$ defined by

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

is non-expansive for $X \mathbin{\&} Y$. Furthermore, note that the projections

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

are trivially non-expansive for this metric.

The second, product $X \otimes Y$ also supports the non-expansive 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, currying and function application are non-expansive under this metric. 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 on functions defined on 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 with several 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 [38], where $0 \cdot \infty$ is defined as 0. In that case, the identity only holds for $r > 0$ strictly.

3. Core Fuzz

We now show how to model a fragment of Fuzz without general recursion. The syntax, summarized 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 products. The first one is eliminated using case analysis (**let** $(x, y) = e$ **in** e'), whereas the second one is eliminated using the projections π_i. The ! constructor boxes its argument, which can later be unboxed with the form **let** $!x = e$ **in** e'. This constructor marks where we need to scale the metric of a space. For concreteness we will include real numbers k and a unit $()$ value, and addition on real numbers.

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

Figure 2. Syntax of Core Fuzz

Fuzz programs run 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 [38].

The type system is more interesting. 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 inspired by bounded linear logic, with a few idiosyncratic points. First, judgments track 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] Second, 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

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

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

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

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) $[\![x]\!](a) \triangleq a(x)$.

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

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

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

($\multimap I$) $[\![\lambda x.\, e]\!] \triangleq \lambda [\![e]\!]$, where λ denotes currying.

($\multimap E$) $[\![e_1\; e_2]\!] \triangleq \varepsilon \circ ([\![e_1]\!] \otimes [\![e_2]\!]) \circ \delta$, where ε denotes function application.

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

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

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

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

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

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

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

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

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

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

Lemma 3.2 (Weakening)**.** *Let e be a typed term such that $\Gamma_1, \Gamma_2 \vdash e : \sigma$. 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*

[3] In the original paper [38], 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.

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

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

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

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

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

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

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

Figure 3. Core Fuzz Typing Rules

$\pi_\Gamma : [\![\Gamma_1, \Delta, \Gamma_2]\!] \to [\![\Gamma_1, \Gamma_2]\!]$ *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,[4] 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 $\sigma, \vdash \vec{v}(x) : \sigma$ 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 $\vec{v} : \Gamma$ be a well-typed substitution. Then, there is a derivation of

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

Furthermore, this derivation has semantics

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

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.*

Together, the lemmas provide a short proof of metric preservation for our 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 $[\![e]\!]([\![\vec{v}]\!])$ and $[\![e]\!]([\![\vec{v}']\!])$. 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 $e[\vec{v}] \hookrightarrow v$ and $e[\vec{v}'] \hookrightarrow v'$. 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

While metric spaces suffice for the core fragment of Fuzz studied so far, they lack the structure needed to model the full language with non-terminating expressions and recursive types. To handle these features, we will use the domain-theoretic notion of *complete partial order*. We first review the basic theory of these structures, and then show how to refine them into *metric CPOs*, which enable sensitivity analysis in the presence of general recursion.

4.1 Preliminaries

Let $(X, \sqsubseteq)$ be a poset (i.e., a set 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

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

[4] A similar result holds for the substitution of arbitrary expressions, but we will not need this generality.

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 mapping $f \mapsto \mathrm{fix}(f)$ defines 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 makes CPO a cartesian-closed category.

As it is typical, 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 $X_\bot$ models computations that terminate successfully, whereas $\bot$ models divergence. 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:

$$\begin{aligned} g^\dagger(\bot) &= \bot \\ g^\dagger(y) &= g(y) && \text{if } y \neq \bot. \end{aligned}$$

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

In order to extend the sensitivity analysis of Section 2 on CPOs, we seek to define 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 with the usual linear (not flat) order, extended with a greatest element ∞. 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 by 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

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

So, we 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, which we call MetCPO.

All CPO constructions from the last section can be lifted to metric CPOs.[5] 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 $X_\bot$, under the sum metric of Section 2. Furthermore, $X + Y$ and the canonical injections give a coproduct on* MetCPO.

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

$$d\left(\bigsqcup_i x_i, \bigsqcup_i x'_i\right) \leq \mathit{lim\,inf}_i\, d(x_i, x'_i).$$

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

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

[5] 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.

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

$$d\left(\bigsqcup_i x_i, \bigsqcup_i x'_i\right) + d\left(\bigsqcup_i y_i, \bigsqcup_i y'_i\right)$$
$$\leq \liminf_i \left(d(x_i, x'_i) + d(y_i, y'_i)\right),$$

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

$$\liminf_i d(x_i, x'_i) + \liminf_i d(y_i, y'_i)$$
$$\leq \liminf_i \left(d(x_i, x'_i) + d(y_i, y'_i)\right),$$

which always holds. □

As before, 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:

$$t : X_\perp \otimes Y_\perp \to (X \otimes Y)_\perp$$
$$t : X_\perp \mathbin{\&} Y_\perp \to (X \mathbin{\&} Y)_\perp.$$

Morphisms of metric CPOs form a metric CPO, as shown in the next result. As expected, currying and function application have a similar than in Met.

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 $\mathsf{Met}(X, Y)$. The cartesian-closed structure of CPO induces an adjunction in MetCPO:*

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

making it a symmetric monoidal closed category.

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. Showing that the monoidal structure is closed is standard.

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 $f = \bigsqcup_i f_i$ 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 setting x_1 and x_2 to the same value, 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. □

Metric CPOs also support scaling.

Lemma 4.8. *Let X be a metric CPO and $r \in \mathbb{R}^\infty_{\geq 0}$. Then $r \cdot X$ is also a metric CPO, under the same order as X.*

Proof. We just need to show that the new metric is compatible with the CPO 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 can interpret recursion by adding sensitivity information to the Kleene fixed-point combinator of Theorem 4.1:

Lemma 4.9. *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*

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

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

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

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

Fuzz allows users to define data types recursively. To give a semantics to these types, we must solve the following problem: given an operator F that maps types to types, find a type μF such that $F(\mu F) \cong \mu F$. The theory of *algebraic compactness* [24, 25, 40] provides an elegant framework for studying these so called *domain equations*. After a short review of this framework, we show how it applies to $\mathsf{MetCPO}_\perp$, 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 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 one it is the same as in 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 the composition of CPO-functors, as well as all the type operators that we have considered in this section (&, ⊗, etc.). Thus, CPO-functors can describe many recursive data types. For instance, the functor $F : \mathsf{MetCPO}_\perp \to \mathsf{MetCPO}_\perp$ defined as

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

is a CPO-functor, and the solution of the corresponding domain equation is a metric CPO of 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. As usual, this universal property of i translates into powerful induction and coinduction principles [36] that characterize the solution μF up to isomorphism. However, it does not play a major role in our analysis, so we will not worry about it in what follows.

Two basic facts about algebraic compactness will be useful later on. First, if $\mathcal{C}$ is algebraically compact and T is a finite set, then the product $\mathcal{C}^T$ is also algebraically compact. This allows us to describe *mutually recursive types* as solutions to domain equations of the form $F(X_1, ..., X_n) \cong (X_1, ..., X_n)$.

Second, algebraic compactness also provides solutions to domain equations given in terms of *mixed-variance* CPO-functors. More precisely, suppose $\mathcal{C}$ is algebraically compact, and $F : \mathcal{C}^\star \to \mathcal{C}$ is a CPO-functor, where $\mathcal{C}^\star \triangleq \mathcal{C}^{op} \times \mathcal{C}$. Then we can find $\mu F \in \mathcal{C}$ and an isomorphism

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

Such domain equations allow us to consider type operators involving exponentials $\mathcal{C}(-,-)$, which cannot be modeled directly as covariant functors as was done for (4).

The following classic result provides useful sufficient conditions for showing that $\mathsf{MetCPO}_\perp$ is algebraically compact.

Theorem 4.10 (Smyth and Plotkin [40])**.** *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 $ee^\# \sqsubseteq id$. Then, $\mathcal{C}$ is algebraically compact.

Most of these conditions can be easily checked. (The terminal object in $\mathsf{MetCPO}_\perp$ is the empty metric CPO $\mathbf{0}$.) The most difficult one is showing 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 they can be easily transferred to $\mathsf{MetCPO}_\perp$. Later (Section 5), we will reuse this 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. If $A, B \in \mathcal{E}$, we write $f : A \supset B$ to mean that there exists $f' : A \to B$ such that $f = Ff'$. We say that F is a $\mathsf{CLat}_\wedge$-*fibration*[6] 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, and every subset $S \subseteq \mathcal{E}_X$ has a meet, denoted by $\bigcap S$.
2. For each arrow $f : X \to Y$ in $\mathcal{D}$ and $B \in \mathcal{E}_Y$, there is a element $f^*B \in \mathcal{E}_X$ (called the *inverse image of* B *by* f) such that
$$g : A \supset f^*B \Leftrightarrow fg : A \supset B \tag{7}$$
for all arrows g. Furthermore, $f^*\left(\bigcap S\right) = \bigcap\{f^*B \mid B \in S\}$ for any set $S \subseteq \mathcal{E}_X$.

Intuitively, we use elements of $\mathcal{E}_X$ to represent abstract predicates or relations over X, with the partial order of $\mathcal{E}_X$ corresponding to logical implication. We think of an arrow $f : A \supset B$ as taking elements related by A to elements related by B. Note that the above properties imply that F is a faithful functor, and that each inverse image f^*B is the unique element satisfying (7).

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 $f^*d(x, y) = d(f(x), f(y))$. In terms of the relational intuition above, each metric d on X yields a family of relations $\{R_r\}_{r\in\mathbb{R}^{\infty}_{\geq 0}}$, defined by $(x, x') \in R_r \Leftrightarrow d(x, x') \leq r$. Non-expansiveness then simply means that elements related at distance r are mapped to elements related at distance r.

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* [36, Definition 4.3] if the image of $\mathcal{E}(A, B)$ under F 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; this gives $\mathcal{E}$ a canonical structure of CPO-category.[7] Alternatively, F is admissible if both $\mathcal{E}$ and $\mathcal{D}$ are CPO-categories and F is a CPO-functor.

The following summarizes useful facts about $\mathsf{CLat}_\wedge$-fibrations.

Lemma 4.11.

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*

[6] 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.

[7] The terminology is reminiscent of Pitts' work on relational properties of domains [36]. In fact, $\mathsf{CLat}_\wedge$-fibrations are closely related to his notion of normal relational structure with inverse images and intersections.

admissible $\mathsf{CLat}_\wedge$*-fibrations over* CPO*-categories, restricting pullbacks along* CPO*-functors.*

3. *Let $\mathcal{D}$ be a* CPO*-category, and $F : \mathcal{E} \to \mathcal{D}$ a* $\mathsf{CLat}_\wedge$*-fibration. Admissible objects of F are closed under inverse images and intersections [36, 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.*

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

$$\begin{tikzcd}
\mathsf{CPO} \times_{\mathsf{Set}} \mathsf{Met} \arrow[r] \arrow[d, "r"] & \mathsf{Met} \arrow[d, "p"] \\
\mathsf{CPO} \arrow[r, "U"] & \mathsf{Set}
\end{tikzcd}$$

Proposition 4.12. $(\mathsf{CPO} \times_{\mathsf{Set}} \mathsf{Met})_{adm} = \mathsf{MetCPO}$.

Proof. Every metric CPO is admissible, 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 4.13. *The forgetful functor q :* $\mathsf{MetCPO} \to \mathsf{CPO}$ *is an admissible* $\mathsf{CLat}_\wedge$*-fibration, and* MetCPO *is cocomplete.*

Proof. By Lemma 4.11. □

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 4.14. *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 be an embedding in $\mathsf{MetCPO}_\perp(X, Y)$. 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 4.15. *The category* $\mathsf{MetCPO}_\perp$ *has colimits of ω-chains of embeddings.*

Proof. From Lemma 4.14, 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 colimit. Therefore the J-image of a colimiting cone over (X_i, m_i), which exists by Corollary 4.13, gives a colimiting cone over (X_i, e_i). □

Having checked this result, we can apply Theorem 4.10 to show that $\mathsf{MetCPO}_\perp$ is algebraically compact.

5. Full Fuzz

Now, 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.[8] Identifiers behave as *iso-recursive types*: programs can freely cast between α and $\Phi(\alpha)$ with the **fold** and **unfold** operators (cf. (μE) and (μI)).

$$\sigma, \tau ::= \cdots \mid \alpha \in T \qquad e ::= \cdots \mid \mathbf{fold}\, e \mid \mathbf{unfold}\, e$$

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

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

Figure 4. Fuzz Recursive Types

5.1 Adapting the Model

Ideally, we would like to extend the interpretation of types in Section 3 by setting

$$[\![\alpha]\!] \triangleq [\![\Phi(\alpha)]\!]. \tag{8}$$

Since $\Phi(\alpha)$ is not smaller than α, this definition is not well-founded. However, we can still give it a formal meaning by appealing to algebraic compactness.

The first step, following Section 4.3, is to express the interpretation of recursive types as the solution of a system of domain equations

$$i : F_\Phi(\mu F_\Phi, \mu F_\Phi) \cong \mu F_\Phi, \tag{9}$$

where $F_\Phi : (\mathsf{MetCPO}_\perp^T)^\star \to \mathsf{MetCPO}_\perp^T$, and $\mu F_\Phi \in \mathsf{MetCPO}_\perp^T$ maps each recursive type α to its interpretation $\mu F_\Phi(\alpha)$. To define F_Φ, we assign to each σ a mixed-variance CPO-functor $F_\sigma : (\mathsf{MetCPO}_\perp^T)^\star \to \mathsf{MetCPO}_\perp$ defined by recursion on σ:

$$F_\alpha(X, Y) \triangleq Y(\alpha)$$

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

The other cases essentially follow the definition of $[\![-]\!]$ in Section 3, and are omitted for brevity. We can now define

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

Since $\mathsf{MetCPO}_\perp$ is algebraically compact, so is $\mathsf{MetCPO}_\perp^T$, implying that a solution to (9) exists. With this solution in hand, we can finally interpret types as

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

All the equations describing the interpretation of types for Core Fuzz carry over to this definition. Additionally, the isomorphism i of (9) corresponds to a family of isomorphims

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

which give recursive types their intended semantics.

Now that we know how to interpret types, we can proceed with the rest of the 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:

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

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 recursively by adapting the semantics of Section 3. For instance, consider the rule $(\& I)$: we want to interpret a typed term

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

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

$$\widehat{F}_\sigma : (\mathsf{Rel}_V^T)^\star \to \mathsf{Rel}_V$$

$$\frac{k \in \mathbb{R}}{(k, k) \in \widehat{F}_{\mathbb{R}}(A, B)} \qquad \frac{(a, v) \in \widehat{F}_\sigma(A, B)}{(\iota_1(a), \textbf{inl}\, v) \in \widehat{F}_{\sigma+\tau}(A, B)}$$

$$\frac{}{(\star, ()) \in \widehat{F}_1(A, B)} \qquad \frac{(b, v) \in \widehat{F}_\tau(A, B)}{(\iota_2(b), \textbf{inr}\, v) \in \widehat{F}_{\sigma+\tau}(A, B)}$$

$$\frac{\bullet \in \{\otimes, \times\} \qquad (a, v_a) \in \widehat{F}_\sigma(A, B) \qquad (b, v_b) \in \widehat{F}_\tau(A, B)}{((a, b), (v_a, v_b)) \in \widehat{F}_{\sigma\bullet\tau}(A, B)}$$

$$\frac{\forall (a, v) \in \widehat{F}_\sigma(B, A).\, (f(a), e[x \mapsto v]) \in \widehat{F}_\tau(A, B)^\perp \text{ (as in (12))}}{(f, \lambda x.\, e) \in \widehat{F}_{\sigma\multimap\tau}(A, B)}$$

$$\frac{(a, v) \in \widehat{F}_\sigma(A, B)}{(a, !v) \in \widehat{F}_{!\sigma}(A, B)} \qquad \frac{(a, v) \in B(\alpha)}{(a, \textbf{fold}\, v) \in \widehat{F}_\alpha(A, B)}$$

Figure 5. Relational lifting of the F_σ functors. We implicitly use an object $(X, P) \in \mathsf{Rel}_V$ to denote the relation $P \subseteq X \times V$, so that $\widehat{F}_\sigma(A, B)$ stands for a relation between $F_\sigma(R^T A, R^T B)$ and V.

given interpretations for both subterms, $[\![e_1]\!] : [\![\Gamma]\!] \to [\![\sigma]\!]_\perp$ and $[\![e_2]\!] : [\![\Gamma]\!] \to [\![\tau]\!]_\perp$. We define $[\![\langle e_1, e_2\rangle]\!]$ as the composite

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

where t is the forcing morphism from (2). The interpretation of other term constructors of Core Fuzz is adapted to this new setting analogously. To conclude, we interpret **fold** and **unfold** using the isomorphisms provided by algebraic compactness:

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

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

5.2 Metatheory

The basic properties of Core Fuzz (Lemmas 3.2 to 3.4) generalize without difficulty to this new setting. 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 $[\![v]\!] \in [\![\sigma]\!]$. 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 $e[\vec{v}\,']$. For that we need the following property, which connects the domain-theoretic and operational views of termination:

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

Adequacy implies that programs $e[\vec{v}]$ and $e[\vec{v}\,']$ in the statement of Theorem 5.2 have the same termination behavior if $d_{[\![\Gamma]\!]}(\vec{v}, \vec{v}\,') < \infty$. Indeed, supposing that the inputs are at finite distance, metric preservation yields

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

Now, imagine that $e[\vec{v}]$ terminates in a value v. By preservation, $[\![e[\vec{v}]]\!] = [\![v]\!] \ne \perp$. This implies $[\![e[\vec{v}\,']]\!] \ne \perp$, because $d([\![v]\!], \perp) = \infty$. Finally, by adequacy, we find v' such that $e[\vec{v}\,'] \hookrightarrow v'$. The symmetric case follows similarly.

Following Plotkin [37], we prove Lemma 5.3 by constructing, for each type σ, a logical relation $S_\sigma \subseteq [\![\sigma]\!] \times V$ such that if $\Gamma \vdash e : \sigma$, $\vec{a} \in [\![\Gamma]\!]$, and $\vec{v} : \Gamma$, then

$$(\vec{a}, \vec{v}) \in S_\Gamma \Rightarrow ([\![e]\!](\vec{a}), e[\vec{v}]) \in S_\sigma^\perp, \tag{10}$$

where

$$(\vec{a}, \vec{v}) \in S_\Gamma \Leftrightarrow (\forall (x :_r \tau) \in \Gamma.\, (\vec{a}(x), \vec{v}(x)) \in S_\tau) \tag{11}$$

$$(a, e) \in S_\sigma^\perp \Leftrightarrow (a \ne \perp \Rightarrow \exists v.\, e \hookrightarrow v \wedge (a, v) \in S_\sigma). \tag{12}$$

Adequacy follows from (10) by instantiating Γ with the empty environment. Our goal is to define S_σ so that (10) is strong enough to be established by a simple induction on the typing derivation. This almost completely determines how S_σ should be defined; it must satisfy equations including

$$S_{\mathbb{R}} = \{(k, k) \mid k \in \mathbb{R}\} \tag{13}$$

$$S_{\sigma\&\tau} = \{((a, b), \langle v_a, v_b\rangle) \mid (a, v_a) \in S_\sigma, (b, v_b) \in S_\tau\} \tag{14}$$

$$S_\alpha = \{(i_\alpha(a), \textbf{fold}\, v) \mid (a, v) \in S_{\Phi(\alpha)}\}. \tag{15}$$

Once again, we cannot define S by structural recursion, since (15) expresses S_α in terms of $S_{\Phi(\alpha)}$. To overcome this circularity, we use a method due to Pitts [36, 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 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* $\widehat{F} : \mathcal{E}^\star \to \mathcal{E}$ *such that the following diagram commutes:*

$$\begin{tikzcd}
\mathcal{E}^\star \arrow[r, "\hat{F}"] \arrow[d, "G^\star"] & \mathcal{E} \arrow[d, "G"] \\
\mathcal{D}^\star \arrow[r, "F"] & \mathcal{D}
\end{tikzcd}$$

Suppose furthermore that the hom sets of $\mathcal{E}$ *and* $\mathcal{D}$ *are pointed, and that* G *preserves these least elements. Then, we can construct* $\mu\widehat{F} \in \mathcal{E}_{\mu F}$ *such that* $\mu\widehat{F} = (i^{-1})^* \widehat{F}(\mu\widehat{F}, \mu\widehat{F})$*, where* $i : F(\mu F, \mu F) \cong \mu F$ *is the isomorphism given by algebraic compactness, as in* (6).

Analogously to our interpretation of types, we will use $\widehat{F}$ to express the logical relations S_α as the solution of fixed-point equations, and then define the other logical relations S_σ in terms of these solutions. To apply Theorem 5.4, we use 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

$$(\forall i.\, (x_i, v) \in P) \Rightarrow \left(\bigsqcup_i x_i, v\right) \in P, \tag{16}$$

for all ω-chains (x_i) in X and $v \in V$.

2. Arrows $(X, P) \to (Y, Q)$ are continuous, non-expansive functions $f : X \to Y_\perp$ such that, whenever $(x, v) \in P$ and $f(x) \ne \perp$, 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

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

Furthermore, both $\mathsf{MetCPO}_\bot$ and Rel_V have pointed hom sets, and R preserves least elements.

We build the logical relations $(S_\alpha \subseteq \mu F_\Phi(\alpha) \times V)_{\alpha \in T}$ by building an object $(\mu F_\Phi, S_\alpha)_{\alpha \in T}$ in the fiber of R^T over $\mu F_\Phi \in \mathsf{MetCPO}_\bot^T$. Since R^T is also an admissible $\mathsf{CLat}_\wedge$-fibration, we just need to lift F_Φ across R^T and apply Theorem 5.4. It suffices to find, for each type σ, a functor $\widehat{F}_\sigma : (\mathsf{Rel}_V^T)^\star \to \mathsf{Rel}_V$ such that

$$R \circ \widehat{F}_\sigma = F_\sigma \circ R^T, \tag{17}$$

and then set $\widehat{F}_\Phi(A, B)(\alpha) \triangleq \widehat{F}_{\Phi(\alpha)}(A, B)$; the complete definition is in Figure 5. With the fixed point $\mu\widehat{F}_\Phi$, we can finally define the logical relations S_σ as (the relation component of) $\widehat{F}_\sigma(\mu\widehat{F}_\Phi, \mu\widehat{F}_\Phi)$. With the definition in Figure 5, and the characterization of $\mu\widehat{F}_\Phi$ in Theorem 5.4, we can validate all the properties needed for proving (10) (and thus Lemma 5.3) by induction, including (13) to (15).

Remark 5.5. Alternatively, we could have characterized Rel_V reusing the machinery of Lemma 4.11, specifically by pulling back $\mathsf{SubCPO}_\bot$, the category of admissible subobjects of $\mathsf{CPO}_\bot$, as depicted below.

$$\begin{tikzcd}
\mathsf{Rel}_V \arrow[rr] \arrow[d, "R"] & & \mathsf{SubCPO}_\bot \arrow[d] \\
\mathsf{MetCPO}_\bot \arrow[r, "q_\bot"] & \mathsf{CPO}_\bot \arrow[r, "(-)\times V"] & \mathsf{CPO}_\bot
\end{tikzcd}$$

In this diagram, by $I \times V$ we mean the *coproduct* of V-many copies of I in $\mathsf{CPO}_\bot$, which is inherited from CPO via the Kleisli adjunction.

5.3 A Remark on Recursive Functions

Now that we have interpreted the full version of Fuzz, we show how our semantics gives a different perspective on fixed points. Using a standard encoding based on recursive types, Reed and Pierce [38] showed how to type the call-by-value Y combinator in Fuzz as follows:

$$\begin{aligned}
&Y : !_\infty(!_\infty(\tau \multimap \sigma) \multimap \tau \multimap \sigma) \multimap \tau \multimap \sigma \\
&Y \triangleq \lambda F.\, \textbf{let}\, f : \alpha = \lambda f x.\, F\,(f\,f)\,x\, \textbf{in}\, f\,f,
\end{aligned}$$

where α is a recursive type defined as $!_\infty\alpha \multimap \tau \multimap \sigma$. (To improve readability, we have elided the wrapping and unwrapping of recursive and scaled types, and we use a derived **let** form.) With this combinator, we can construct the fixed-point expression $\textbf{fix}\, f.\, e \triangleq Y(\lambda f.\, e)$, and derive a corresponding typing rule.

$$\frac{\Gamma, f :_\infty \tau \multimap \sigma \vdash e : \tau \multimap \sigma}{\infty\Gamma \vdash \textbf{fix}\, f.\, e : \tau \multimap \sigma}$$

This rule makes it possible to define functions of finite sensitivity by recursion. It places little restrictions on how the recursive function calls itself, since it allows the body e to be infinitely sensitive on f; however, it also requires scaling the typing environment by infinity. Reed and Pierce [38] justified this by arguing that "we can't [...] establish any bound on how sensitive the overall function is from just one call to it".

Somewhat surprisingly, Lemma 4.9 allows us to define fixed points directly on metric CPOs with a more precise sensitivity than the one above. This suggests that we might be able to improve the encoding of Y if we assume that its argument F is a finitely sensitive function (i.e., if the body e is finitely sensitive on f). After some thought, we obtain

$$\begin{aligned}
&Y_r : !_{1/(1-r)}(!_r(\tau \multimap \sigma) \multimap \tau \multimap \sigma) \multimap \tau \multimap \sigma \\
&Y_r \triangleq \lambda F.\, \textbf{let}\, f : \alpha_r = \lambda f x.\, F\,(f\,f)\,x\, \textbf{in}\, f\,f,
\end{aligned}$$

where $r < 1$, and α_r is now defined as $!_{r/(1-r)}\,\alpha_r \multimap \tau \multimap \sigma$. This leads to the following typing rule:

$$\frac{\Gamma, f :_r \tau \multimap \sigma \vdash e : \tau \multimap \sigma \qquad r < 1}{\dfrac{1}{1-r}\Gamma \vdash \textbf{fix}_r f.\, e : \tau \multimap \sigma}$$

where $\textbf{fix}_r\, f.\, e \triangleq Y_r(\lambda f.\, e)$. We see that the scaling factor $1/(1-r)$ is unbounded as r approaches 1, when we recover the original rule.

One situation where this fixed point can be useful is for typing functions where recursive calls are guarded by a scaling factor smaller than 1. For instance, suppose that we define a type of lists with exponentially decaying distances:

$$\mathsf{list}\,\tau \triangleq () + \tau \otimes !_r\, \mathsf{list}\,\tau$$

If $r < 1$, we can type the *map* function with a finite sensitivity on its function argument:

$$\begin{aligned}
&\mathit{map} : !_{1/(1-r)}(\tau \multimap \sigma) \multimap \mathsf{list}\,\tau \multimap \mathsf{list}\,\sigma \\
&\mathit{map} = \lambda f.\, \textbf{fix}_r m.\, \lambda l. \\
&\qquad\qquad \textbf{case}\, l\, \textbf{of} \\
&\qquad\qquad \mid \textbf{inl}() \Rightarrow \textbf{inl}() \\
&\qquad\qquad \mid \textbf{inr}(x, l') \Rightarrow \textbf{inr}(f\,x, m\,l')
\end{aligned}$$

This stands in contrast to the typical *map* function, which has infinite sensitivity on its function argument. Exploring applications of this new, more precise type for the fixed point is an intriguing direction for future work.

6. Related Work

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 *Banach fixed-point theorem* yields a natural interpretation of recursive functions and types.

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

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

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 can model language features such as non-determinism and concurrency [4, 18, 32, 33]. (See van Breugel [41] for a good introduction to the subject, and Baier and Majster-Cederbaum [7], Majster-Cederbaum and Zetzsche [34] for a comparison between the metric approaches and their order-based counterparts.) A similar use of ultrametric spaces appears in a denotational model of PCF given by Escardó [23], where the metric structure describes intensional temporal aspects of PCF programs, and its extensional collapse recovers 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. [31], 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 possible-world semantics. Works in this direction include those by Birkedal et al. [10, 11, 12], Schwinghammer et al. [39]. In these works, the metric structure expresses convergence properties that underlie syntactic structures used in languages with guarded definitions, e.g. Nakano's recursion modality [35]. A similar approach has also been used by Krishnaswami and Benton [30] 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 these works, e.g. contractivity plays a different role and our requirement 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 work.

In a separate line of work, unrelated to ultrametrics, Kozen [29] 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 similar 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 introduced a domain-theoretic structure for studying 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 [38].

As future work, we plan to extend our approach to cover the probability monad of Fuzz. While metric interpretations of probabilities are widespread in the programming-languages literature, e.g. [6, 19–21, 29], we are not aware of any similar work that models the metric of Reed and Pierce [38], used for reasoning about differential privacy. Interpreting this metric could also hint at how to interpret a larger class of metric-like functions called *f-divergences* [17]. An orthogonal direction is to study an interpretation of *DFuzz* [26], a dependently typed version of Fuzz for 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.

Metric CPOs 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 for describing these notions of continuity and robustness is also an interesting avenue for future work.

Acknowledgments

We thank the anonymous reviewers for their detailed comments, which improved earlier versions of this work. This work was partially supported by NSF grants TC-1065060, TWC-1513694, TWC-1565365 and TWC-1513854, a grant from the Simons Foundation (#360368 to Justin Hsu), and JSPS KAKENHI Grant Number JP15K00014 (to Shin-ya Katsumata).

References

[1] M. Abadi and G. D. Plotkin. A PER model of polymorphism and recursive types. In *IEEE Symposium on Logic in Computer Science (LICS), Philadelphia, Pennsylvania*, pages 355–365, 1990.

[2] M. Abadi, B. C. Pierce, and G. D. Plotkin. Faithful ideal models for recursive polymorphic types. In *IEEE Symposium on Logic in Computer Science (LICS), Asilomar, California*, pages 216–225, 1989.

[3] R. M. Amadio. Recursion over realizability structures. *Information and Computation*, 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 *Workshop on the Mathematical Foundations of Programming Semantics (MFPS), New Orleans, Louisiana*, volume 298 of *Lecture Notes in Computer Science*, pages 254–288. Springer-Verlag, 1987.

[5] A. Arnold and M. Nivat. Metric interpretations of infinite trees and semantics of non-deterministic recursive programs. *Theoretical Computer Science*, 11(2):181–205, 1980.

[6] C. Baier and M. Z. Kwiatkowska. Domain equations for probabilistic processes. *Electronic Notes in Theoretical Computer Science*, 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. Zanella Béguelin. Probabilistic relational reasoning for differential privacy. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Philadelphia, Pennsylvania*, 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 *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Mumbai, India*, pages 55–68, 2015.

[10] L. Birkedal, K. Støvring, and J. Thamsborg. Realizability semantics of parametric polymorphism, general references, and recursive types. In *International Conference on Foundations of Software Science and Computation Structures (FoSSaCS), York, England*, volume 5504 of *Lecture Notes in Computer Science*, pages 456–470. Springer-Verlag, 2009.

[11] L. Birkedal, K. Støvring, and J. Thamsborg. The category-theoretic solution of recursive metric-space equations. *Theoretical Computer Science*, 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 *IEEE Symposium on Logic in Computer Science (LICS), Toronto, Ontario*, pages 55–64, 2011.

[13] S. Chaudhuri, S. Gulwani, and R. Lublinerman. Continuity analysis of programs. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Madrid, Spain*, pages 57–70, 2010.

[14] S. Chaudhuri, S. Gulwani, R. Lublinerman, and S. NavidPour. Proving programs robust. In *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), Szeged, Hungary*, pages 102–112, 2011.

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

[16] J. Chroboczek. Subtyping recursive games. In *International Conference on Typed Lambda Calculi and Applications (TLCA), Kraków, Poland*, volume 2044 of *Lecture Notes in Computer Science*, pages 61–75. Springer-Verlag, 2001.

[17] I. Csiszár and P. C. 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 *ACM SIGACT Symposium on Theory of Computing (STOC), San Francisco, California*, pages 153–158, 1982.

[19] E. P. de Vink and J. J. M. M. Rutten. Bisimulation for probabilistic transition systems: A coalgebraic approach. *Theoretical Computer Science*, 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. *Electronic Notes in Theoretical Computer Science*, 40:72–99, 2000.

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

[22] C. Dwork, F. McSherry, K. Nissim, and A. D. Smith. Calibrating noise to sensitivity in private data analysis. In *IACR Theory of Cryptography Conference (TCC), New York, New York*, volume 3876 of *Lecture Notes in Computer Science*, pages 265–284. Springer-Verlag, 2006.

[23] M. H. Escardó. A metric model of PCF, 1999. Workshop on Realizability Semantics and Applications, Trento, Italy.

[24] M. P. Fiore and G. D. Plotkin. An axiomatization of computationally adequate domain theoretic models of FPC. In *IEEE Symposium on Logic in Computer Science (LICS), Paris, France*, pages 92–102, 1994.

[25] P. Freyd. Algebraically complete categories. In *International Category Theory Conference (CT), Como, Italy*, volume 1488 of *Lecture Notes in Mathematics*, pages 95–104. Springer-Verlag, 1990. ISBN 978-3-540-46435-8.

[26] M. Gaboardi, A. Haeberlen, J. Hsu, A. Narayan, and B. C. Pierce. Linear dependent types for differential privacy. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Rome, Italy*, pages 357–370, 2013.

[27] A. Haeberlen, B. C. Pierce, and A. Narayan. Differential privacy under fire. In *USENIX Security Symposium, San Francisco, USA*, 2011.

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

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

[30] N. R. Krishnaswami and N. Benton. Ultrametric semantics of reactive programs. In *IEEE Symposium on Logic in Computer Science (LICS), Toronto, Ontario*, pages 257–266, 2011.

[31] D. B. MacQueen, G. D. Plotkin, and R. Sethi. An ideal model for recursive polymorphic types. In *ACM Symposium on Principles of Programming Languages (POPL), Salt Lake City, Utah*, pages 165–174, 1984.

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

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

[34] M. E. Majster-Cederbaum and F. Zetzsche. The comparison of a CPO-based semantics with a CMS-based semantics for CSP. *Theoretical Computer Science*, 124(1):1–40, 1994.

[35] H. Nakano. A modality for recursion. In *IEEE Symposium on Logic in Computer Science (LICS), Santa Barbara, California*, pages 255–266, 2000.

[36] A. M. Pitts. Relational properties of domains. *Information and Computation*, 127(2):66–90, 1996.

[37] 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.

[38] J. Reed and B. C. Pierce. Distance makes the types grow stronger: A calculus for differential privacy. In *ACM SIGPLAN International Conference on Functional Programming (ICFP), Baltimore, Maryland*, pages 157–168, 2010. ISBN 978-1-60558-794-3.

[39] J. Schwinghammer, L. Birkedal, and K. Støvring. A step-indexed Kripke model of hidden state via recursive properties on recursively defined metric spaces. In *International Conference on Foundations of Software Science and Computation Structures (FoSSaCS), Saarbrücken, Germany*, volume 6604 of *Lecture Notes in Computer Science*, pages 305–319. Springer-Verlag, 2011.

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

[41] F. van Breugel. An introduction to metric semantics: operational and denotational models for programming and specification languages. *Theoretical Computer Science*, 258(1–2):1–98, 2001.