A Linear Type System for Lp -Metric Sensitivity Analysis

Victor Sannier, Patrick Baillot

A Linear Type System for L^p-Metric Sensitivity Analysis

Victor Sannier[1] and Patrick Baillot[1]

[1]Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL, F-59 000 Lille, France

March 21, 2024

Abstract

When working in optimisation or privacy protection, one may need to estimate the sensitivity of computer programs, i.e., the maximum multiplicative increase in the distance between two inputs and the corresponding two outputs. In particular, differential privacy is a rigorous and widely used notion of privacy that is closely related to sensitivity. Several type systems for sensitivity and differential privacy based on linear logic have been proposed in the literature, starting with the functional language Fuzz. However, they are either limited to certain metrics (L^1 and L^∞), and thus to the associated privacy mechanisms, or they rely on a complex notion of type contexts that does not interact well with operational semantics. We therefore propose a linear type system called Plurimetric Fuzz that handles L^p vector metrics (for $1 \leq p \leq +\infty$), uses standard type contexts, gives reasonable bounds on sensitivity, and has good metatheoretical properties. We also provide a denotational semantics in terms of metric complete partial orders, and translation mappings from and to Fuzz.

1 Introduction

The sensitivity of a program is a measure of how much the result of the computation depends on its inputs, and is defined with respect to some metrics on data. Concretely if d_X and d_Y are metrics on the input and output spaces respectively, and if f is the function computed by the program, its sensitivity is the minimal positive real s such that $d_Y\big(f(x), f(x')\big) \leq s \cdot d_X(x, x')$, for any pairs of inputs (x, x'). This notion is important for analysing the stability of some machine-learning algorithms or the privacy properties of a program [7, 10]. In particular sensitivity is a key notion for *differential privacy* [11, 12], a popular approach to the protection of sensitive data, like medical records, that provides mathematically-based, rigorous and composable guarantees. The intuition behind differential privacy is that one can hide the information about whether or not a given individual is included in the input data-set by perturbating the result of the function. In practice, one adds a well-calibrated amount of random noise to the result, by means of specific *mechanisms*: one should not be able to deduce from the output whether the individual belongs to the input or not, but the result should still be accurate enough.

As the analysis of sensitivity and the implementation of differential privacy are delicate and error-prone tasks, approaches in the programming languages community have been developed to assist programmers. They can be categorised into two classes: those based on Hoare logics [4, 5, 3], which are interactive and suitable for verifying mechanism implementations, and those based on type systems [19, 13], which are automatisable and well-suited for verifying functional programs that compose mechanisms. Moreover recent works [17, 20] on type systems have suggested that the analysis of sensitivity and privacy in these systems could be handled essentially separately, by using two different classes of type judgements.

In this paper, we are interested in the type systems for the analysis of sensitivity. The seminal work on the Fuzz language by Reed and Pierce [19] has shown how ideas from linear logic [15] can be used to

design a type system for a functional language which statically bounds the sensitivity of a program by providing connectives which can express two metrics on vectors: the L^1 and the L^∞ metrics. However depending on the applications some other metrics on vectors are relevant. For instance, for many geometric algorithms one is interested in the Euclidean distance L^2, and more generally, some L^p distances with $1 \leq p \leq +\infty$ have been used in literature for optimisation and statistical applications [8, 16]. For this reason [23] has introduced an extension of Fuzz, called Bunched Fuzz, which features connectives allowing to handle L^p-metrics ($1 \leq p \leq +\infty$) on vectors. The derivations of this system use generalised type judgements inspired by the logic of Bunched Implications (BI) [18], where typing contexts do not have a list or multiset structure but instead a tree structure. The authors established a soundness result analogous to that of Fuzz, showing that the functions computed by well-typed programs admit a certain sensitivity property.

In the following, we address the question of the properties of the type system with respect to an operational semantics. In particular, we will discuss why Bunched Fuzz does not satisfy the desired properties, and we will design a type system for L^p metrics with the following expectations: (i) sensitivity soundness property, (ii) substitution and subject-reduction property, (iii) subtyping property, and (iv) expressiveness. Requirement (iii) refers to the fact that for all p and q, the L^p and L^q-metrics are related by two inequalities that can be used for coercions between data-types convenient for composing functions. As to (iv), we mean that we want the system to be able to type some meaningful examples.

Concretely, we propose to keep the same type language as Bunched Fuzz but to consider a system of rules that uses traditional judgements with list contexts, we call this system Plurimetric Fuzz. As an additional benefit, we will define some (partial) translations from Fuzz to Plurimetric Fuzz and vice versa, that we think shed some light on how the new system refines Fuzz.

1.1 Summary of Contributions

We introduce Plurimetric Fuzz, a type system with recursive types (see Section 5.4) and a form of subtyping (see Section 3.4) for bounding the L^p-sensitivity of vector-valued functions, which subsumes Fuzz ($p = 1$). We show that Plurimetric Fuzz is type-safe (Theorem 4.2) and sound with respect to its denotational semantics (Theorem 5.12). We also show that it gives significantly lower bounds on sensitivity compared to a naïve extension of Fuzz, and that it is expressive enough to prove a classification algorithm $(\epsilon, 0)$-private (see Section 6).

2 Background

We first give an overview of notions and results about sensitivity, differential privacy and type systems that will be needed in the paper.

2.1 Metric Spaces and Sensitivity

Definition 2.1. An *extended pseudosemimetric space* is a pair (X, d) where X is a set and $d : X \times X \to [0, \infty]$ is a function such that for all $x, y, z \in X$: (1) $d(x, y) = 0$ if $x = y$; and (2) $d(x, y) = d(y, x)$. For simplicity, we will call such a space a *metric space*.

In this paper, we are interested in a family of metrics over $\mathbf{R}^d$, which are defined as follows, and related by the inequalities of Lemma 2.1.

Definition 2.2. For all parameter $p \geq 1$ and for all vectors $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_d)$ and $\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_d)$ in $\mathbf{R}^d$, we define the *L^p-distance* or the *vector metric of parameter p* between $\mathbf{x}$ and $\mathbf{y}$ by $d_p(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{d} |\mathbf{x}_i - \mathbf{y}_i|^p\right)^{1/p}$.

Lemma 2.1. *For all parameters p and q such that $1 \leq p \leq q \leq \infty$, let $c(p, q) = 2^{1/p - 1/q}$. We have $d_p \geq d_q \geq c(p, q) \cdot d_p$.*

The *sensitivity* of a map between metric spaces is a measure of how much its output changes when its input changes. This notion is useful for analysing the privacy guarantees of probabilistic algorithms, as we will see in Section 2.2.

Definition 2.3. A map f between two metric spaces (X, d_X) and (Y, d_Y) is said to be *s-sensitive* for $s \in [0, \infty]$ if for all points x and x' in X, we have $d_Y\big(f(x), f(x')\big) \leq s \cdot d_X(x, x')$. The *sensitivity* of f is the least real s such that f is s-sensitive. When it is bounded by 1, we say that f is *non-expansive*.

Remark 1. To perform operations on sensitivities, we extend addition to possibly infinite reals in a straightforward way and multiplication in the same way as [2, Section 2], that is $s \cdot \infty = \infty$ and $\infty \cdot s$ equals 0 if $s = 0$ and ∞ otherwise. Note that this operation is not commutative, see [20, Section 4.2] for a discussion on the soundness of this choice.

For differentiable real functions, sensitivity is related to the magnitude of the derivative.

Lemma 2.2. *Let f be a differentiable function from $\mathbf{R}$ to $\mathbf{R}$ such that for all $x \in \mathbf{R}$, we have $|f'(x)| \leq s$. Then f is s-sensitive.*

2.2 Differential Privacy

A strong motivation for studying sensitivity lies in the field of privacy-preserving data analysis. Informally, differential privacy [12] is a strong statistical notion of privacy, probably the most widely used and studied, which requires that the outcome of a computation should not depend too much on the presence or absence of a single record in the input database.

Definition 2.4. A probabilistic algorithm A over a domain X endowed with an adjacency relation is said to be *(ϵ, δ)-differentially private* for some $\epsilon \geq 0$ and $\delta \in [0, 1]$ if, for all adjacent inputs x and x', and all subsets S of X, we have $\Pr[A(x) \in S] \leq e^{\epsilon} \Pr[A(x') \in S] + \delta$.

Remark. Differential privacy can also be defined in terms of a hypothesis-testing problem, where an adversary attempts to distinguish between two adjacent inputs by observing the outcome of the algorithm [21].

In practice, X is often be the set of databases, and two databases will be adjacent if one can be obtained from the other by adding or removing a single record. Moreover, the codomain will often be of the form $\mathbf{R}^d$ and endowed with a vector metric such as the Manhattan distance ($p = 1$) or the Euclidean distance ($p = 2$).

To achieve this, it is enough to add noise to the computation, as long as the noise is sufficiently large compared to the sensitivity of the function being computed. Let us give more precise definitions and results.

Let f be a vector-valued function from a metric space (X, d_X) to $\mathbf{R}^d$. We write $\Delta_p f$ for the sensitivity of f when the codomain is endowed with the L^p-distance, that is for the L^p-sensitivity of f. Recall that the *Laplace distribution* of parameter $b > 0$ is the probability distribution with density function $x \mapsto 1/2b \cdot e^{-|x|/b}$ for x in $\mathbf{R}$.

Theorem 2.3 (Laplace Mechanism)**.** *If $\Delta_1 f$ is finite, then for all positive real number ϵ, the function $x \mapsto f(x) + \mathrm{Lap}(\Delta_1 f/\epsilon)$ is ϵ-differentially private.*

However, in some cases, we may prefer to add Gaussian noise instead of Laplace noise. This way, the noise added for privacy is of the same type as other sources of noise. Moreover, the effects of the privacy mechanism on the statistical analysis may be easier to account for given that the sum of normally distributed random variables is itself normally distributed [12, Section 3.5.3]. To do so, we need to bound the L^2-sensitivity of f.

Theorem 2.4 (Gaussian mechanism)**.** *If $\Delta_2 f$ is finite, then for all positive real numbers ϵ and δ, if $\sigma > \sqrt{2 \ln(5/4\delta)} \cdot \Delta_2 f/\epsilon$, then the function $x \mapsto f(x) + \mathrm{Gauss}(0, \sigma^2)$ is (ϵ, δ)-differentially private.*

Remark 2. In this paper, we will only consider discrete probability distributions, but the above two theorems can be adapted to this setting [14, 9].

2.3 Type Systems for Bounding Sensitivity

Reed and Pierce have introduced the Fuzz type system [19] based on the fact that L^1-sensitivity can be viewed as an affine resource (in the sense of linear logic [15]). For example, the judgement $[x : A]_2 \vdash (x, x) : A \otimes A$ means that the map $x \mapsto (x, x)$ is 2-sensitive (for the L^1-distance). See the following tensor rules for an example of Fuzz typing rules:

$$\frac{\Gamma \vdash a : A \qquad \Delta \vdash b : B}{\Gamma + \Delta \vdash (a, b) : A \otimes B} \otimes I \qquad \frac{\Delta \vdash e : A \otimes B \qquad \Gamma, [x : A]_s, [y : B]_s \vdash c : C}{\Gamma + s\Delta \vdash (\textbf{let } (x, y) = e \textbf{ in } c) : C} \otimes E$$

It was subsumed by Bunched Fuzz [23], which allows for the analysis of L^p-sensitivity for $p \in [1, \infty]$ by the introduction of a family of tensor products $(\otimes_p)_{p\in[1,\infty]}$ and affine arrows $(\multimap_p)_{p\in[1,\infty]}$. Moreover contexts are no longer represented as lists, but as trees (or *bunches*). We reproduce the tensor rules below:

$$\frac{\Gamma \vdash a : A \qquad \Delta \vdash b : B}{\Gamma \,,_p\, \Delta \vdash (a, b) : A \otimes_p B} \otimes I \qquad \frac{\Delta \vdash e : A \otimes_p B \qquad \Gamma\big([x : A]_s \,,_p\, [y : B]_s\big) \vdash c : C}{\Gamma(s\Delta) \vdash (\textbf{let } (x, y) = e \textbf{ in } c) : C} \otimes E$$

Remark 3. We write $b[x \mapsto a]$ for the capture-avoiding substitution of b for x in a.

The contraction rule enables the identification of variables with the same type in two different subtrees of a context,

$$\frac{\Gamma(\Delta \,,_p\, \Delta') \vdash a : A \qquad \Delta \approx \Delta'}{\Gamma\big(\mathrm{Contr}(p; \Delta'; \Delta)\big) \vdash a[\mathrm{vars}\,\Delta \mapsto \mathrm{vars}\,\Delta'] : A}\ \mathrm{Contr}$$

where $\mathrm{Contr}(p; \Gamma; \Delta)$ is defined by induction on the structure of Γ by the following equations:

$$\begin{aligned}
\mathrm{Contr}(p; \emptyset; \emptyset) &\equiv \emptyset \\
\mathrm{Contr}(p; [x : A]_r; [y : A]_s) &\equiv [x : A]_{\sqrt[p]{r^p + s^p}} \\
\mathrm{Contr}(p; \Gamma_1 \,,_q\, \Gamma_2; \Delta_1 \,,_q\, \Delta_2) &\equiv 2^{|1/p - 1/q|} \cdot \mathrm{Contr}(p; \Gamma_1; \Delta_1) \,,_q\, \mathrm{Contr}(p; \Gamma_2; \Delta_2)
\end{aligned}$$

The authors have proved that if types and contexts are interpreted as metric spaces, then the derivations correspond to non-expansive functions.

Theorem 2.5 ([23, Theorem 4.9])**.** *Given a derivation π proving $\Gamma \vdash a : A$, the function $[\![\pi]\!] : [\![\Gamma]\!] \to [\![A]\!]$ is non-expansive.*

However, the syntactic complexity induced by the use of bunches comes at the cost of the loss of the substitution property: there exists derivations $\Gamma \vdash a : A$ and $\Delta([x : A]_s) \vdash b : B$ such that $\Delta(s\Gamma) \not\vdash b[x \mapsto a] : B$.

Proof. To see this, consider the following derivation, where we apply the algorithmic version of the rules:

$$\dfrac{\dfrac{}{\emptyset \vdash (+) : \mathsf{Nat} \otimes_1 \mathsf{Nat} \multimap_1 \mathsf{Nat}} + \qquad \dfrac{\dfrac{}{[a : \mathsf{Nat}]_1 \vdash a : \mathsf{Nat}}\ \mathrm{var} \qquad \dfrac{}{[b : \mathsf{Nat}]_1 \vdash b : \mathsf{Nat}}\ \mathrm{var}}{[a : \mathsf{Nat}]_1 \,,_1\, [b : \mathsf{Nat}]_1 \vdash (a, b) : \mathsf{Nat} \otimes_1 \mathsf{Nat}} \otimes I}{[a : \mathsf{Nat}]_1 \,,_1\, [b : \mathsf{Nat}_1] \vdash (+)(a, b) : \mathsf{Nat}} \multimap E$$

If we could substitute $(+)(a, b)$ for x in the derivation

$$\dfrac{\dfrac{}{[x : \mathsf{Nat}]_1 \vdash x : \mathsf{Nat}}\ \mathrm{var} \qquad \dfrac{}{[x : \mathsf{Nat}]_1 \vdash x : \mathsf{Nat}}\ \mathrm{var}}{[x : \mathsf{Nat}]_{\sqrt{2}} \vdash (x, x) : \mathsf{Nat} \otimes_2 \mathsf{Nat}} \otimes I$$

we would obtain $[a : \mathsf{Nat}]_{\sqrt{2}} ,_1 [b : \mathsf{Nat}]_{\sqrt{2}} \vdash \big((+)(a,b),(+)(a,b)\big) : \mathsf{Nat} \otimes_2 \mathsf{Nat}$, which is not derivable. Indeed, a derivation of this judgement would have the following shape:

$$\dfrac{\dfrac{\begin{array}{cc}\vdots & \vdots \\ [a : \mathsf{Nat}]_{r_a} ,_1 [b : \mathsf{Nat}]_{r_b} \vdash a + b : \mathsf{Nat} & [a : \mathsf{Nat}]_{s_a} ,_1 [b : \mathsf{Nat}]_{s_b} \vdash a + b : \mathsf{Nat}\end{array}}{c(1,2) \cdot \big([a : \mathsf{Nat}]_1 ,_1 [b : \mathsf{Nat}]_1\big) \vdash \big((+)(a,b),(+)(a,b)\big) : \mathsf{Nat} \otimes_2 \mathsf{Nat}} \otimes I}{[a : \mathsf{Nat}]_{\sqrt{2}} ,_1 [b : \mathsf{Nat}]_{\sqrt{2}} \vdash \big((+)(a,b),(+)(a,b)\big) : \mathsf{Nat} \otimes_2 \mathsf{Nat}} =$$

where r_a, r_b, s_a and s_b would be real numbers such that $r_a^2 + s_a^2 = 1$ and $r_b^2 + s_b^2 = 1$, therefore such that $\min\{r_a, s_a\} \leq \sqrt{2}/2 < 1$, which is absurd as $(a,b) \mapsto a + b$ is 1-sensitive for the L^1-distance. □

Claim 2.6. *Bunched Fuzz is not type-safe with respect to its operational semantics.*

In conclusion, the flexibility provided by representing contexts as trees is offset by the loss of important syntactic and operational properties.

3 Syntax

In a nutshell we will consider the terms of Fuzz, that is to say an extended λ-calculus, with the types of Bunched Fuzz but with a new notion of typing context.

3.1 Types and Terms

Types are defined by the following context-free grammar where s and p range over $[0,\infty]$ and $[1,\infty]$ respectively: $A ::= \mathsf{Unit} \mid A \oplus A \mid \mu\alpha.\, A \mid \bigcirc A \mid !_s A \mid A \otimes_p A \mid A \multimap_p A \mid A \to_p A$. We write Bool for the type $\mathsf{Unit} \oplus \mathsf{Unit}$; $\mathsf{List}_p(A)$ for the iso-recursive type $\mu\alpha.\, \mathsf{Unit} \oplus (A \otimes_p \alpha)$; and $\bigotimes_p^d A = A \otimes_p \cdots \otimes_p A$.

On the other hand, the terms of the language are defined by the following grammar:

$$\begin{aligned} e ::= {} & * \mid x && \text{for } x \in \mathrm{Var} \\ & \mid \mathbf{inj}_1\, e \mid \mathbf{inj}_2\, e \mid (e,e) \mid \mathbf{case}\ e\ \mathbf{of}\ x \Rightarrow e\ \mathbf{or}\ x \Rightarrow e \\ & \mid \lambda x.\, e \mid e\ e \mid \mathbf{let}\ x = e\ \mathbf{in}\ e \\ & \mid \underset{A}{\mathbf{fold}}\, e \mid \underset{A}{\mathbf{unfold}}\, e \mid \mathbf{return}\, e \mid \mathbf{let}\ \bigcirc x = e\ \mathbf{in}\ e && \text{for } \alpha \in \mathrm{Var}\,. \end{aligned} \tag{1}$$

They are written using lowercase roman letters (a, b, c, etc.) in the following.

Remark 4. In examples, we write terms in an ML-like syntax instead of the one described in Section 3. In particular, we may write `x |> f` for `f x`, we may use pattern matching and `let` bindings (**let** $x = e$ **in** b is syntactic sugar for $(\lambda x.\, b)\, e$), and we may omit the Y combinator when defining recursive functions.

3.2 Precontexts and Contexts

We refer to elements of the set defined by grammar $\Gamma ::= \emptyset \mid [x : A]_s, \Gamma$, where s, x and A range over $[0,\infty]$, Var and Type respectively, as *precontexts*. In addition, we define the scaling $s\Gamma$ of a precontext Γ by a sensitivity s by $s \cdot \emptyset \equiv \emptyset$ and $s \cdot ([x : A]_r, \Gamma) \equiv [x : A]_{rs}, s\Gamma$ for all $s \in [0,\infty]$.

Definition 3.1. Two precontexts Γ and Δ are said to be *compatible* if they do not assign different types to the same variable. The *p-contraction* of two compatible precontexts Γ and Δ is defined by induction on the structure of Γ by the following equations:

$$\begin{aligned} C^p\left(\emptyset; \Delta\right) &\equiv \Delta \\ C^p\left([x : A]_r, \Gamma; \Delta\right) &\equiv [x : A]_r, C^p\left(\Gamma; \Delta\right) && \text{if } x \notin \Gamma \\ C^p\left([x : A]_r, \Gamma; [x : A]_s, \Delta\right) &\equiv [x : A]_{\sqrt[p]{r^p + s^p}}, C^p\left(\Gamma; \Delta\right) \end{aligned} \tag{2}$$

We write $\Gamma + \Delta$ for $C^1(\Gamma; \Delta)$.

Lemma 3.1. *For all precontexts Γ and Δ, and all parameters p:*

- $C^p(\Gamma; \Delta) = C^p(\Delta; \Gamma)$;
- *if* $C^p(\Gamma; \Delta) = \emptyset$, *then* $\Gamma = \emptyset$ *and* $\Delta = \emptyset$;
- *for all sensitivity s, we have* $s \cdot C^p(\Gamma; \Delta) = C^p(s\Gamma; s\Delta)$.

Definition 3.2. For any two precontexts Γ and Δ, we write $\Gamma \leq \Delta$ if every variable of Γ also occurs in Δ, and with greater or equal sensitivity.

Finally, we define a *context* as a pair of a parameter and a precontext, which can be seen as a Bunched Fuzz context where all parameters are equal, and which can therefore be flattened into a list. More precisely, a *context* is a pair (p, Γ) written $(p)\ \Gamma$ where $p \in [1, \infty]$ and Γ is a precontext.

3.3 Typing Rules

The typing rules and typing rules schemas for Plurimetric Fuzz are given in Figure 1 where Γ and Δ range over contexts, A, B, and C range over types, etc.

We omit the $\to_p$ type constructor, and encode them with $\multimap_p$ and $!_\infty$ as follows: $A \to_p B \equiv\ !_\infty A \multimap_p B$. Similarly, the & constructor can be encoded by $\otimes_\infty$ as in [23, Section 3]. Observe that, as $C^1(\Gamma; \Delta) = \Gamma + \Delta$, all rules but the last two, $(\geq W)$ and $(\leq W)$, correspond to Fuzz rules when $p = 1$ (by identifying connectives of parameter 1 with the corresponding Fuzz ones). So all Fuzz type derivations can be seen as Plurimetric Fuzz type derivations (up to the encoding of &). We will see in Section 7 other ways of translating Fuzz derivations, by choosing other values of p.

Also note that the weakening rules $(\geq W)$ and $(\leq W)$ are the only ones that make the parameter of the judgement change. One direction $(\geq W)$ is direct, but the other one $(\leq W)$ requires a coefficient $c(p, q)$ (see Section 2.1). In addition, as a particular case of these rules, for all parameters p and q, from $(p)\ \emptyset \vdash a : A$ we can derive $(q)\ \emptyset \vdash a : A$.

Remark 5. As shown in [19, Section 3.1], recursive types let us encode a fix-point combinator for any two types A and B, and parameters p without a specific rule:

$$Y \equiv \lambda f.\ \Big(\lambda x.\ \lambda a.\ f\big((\mathbf{unfold}_{A_0}\ x)x\big)a\Big)\Big(\mathbf{fold}_{A_0}\Big(\lambda x.\ \lambda a.\ f\big((\mathbf{unfold}_{A_0}\ x)x\big)a\Big)\Big)$$

where $A_0 \equiv \mu\alpha\ \big(\alpha \to_p (A \multimap_p B)\big)$, and $Y : \big((A \multimap_p B) \to_p (A \multimap_p B)\big) \to_p (A \multimap_p B)$.

We can supplement the type system with rules for primitive operations on natural and real numbers, and sets (see Fig. 2), after extending the syntax of types and terms accordingly.

3.4 Subtyping

For all p and q, the L^p and L^q-metrics are related by two inequalities that can be used for coercions between data types: if $p \leq q$, from $(p)\ \Gamma \vdash e : A \otimes_p B$, we can derive $(p)\ \Gamma \vdash \big(\mathbf{let}\ (x, y) = e\ \mathbf{in}\ (x, y)\big) : A \otimes_q B$; and similarly from $(q)\ \Gamma \vdash e : A \otimes_q B$, we can derive $(q)\ c(p, q) \cdot \Gamma \vdash \big(\mathbf{let}\ (x, y) = e\ \mathbf{in}\ (x, y)\big) : A \otimes_p B$. Let us give the derivation of the first case:

$$\dfrac{(p)\ \Gamma \vdash e : A \otimes_p B \qquad \dfrac{\dfrac{\dfrac{}{(p)\ [x : A]_1 \vdash x : A}\ \text{var}}{(q)\ [x : A]_1 \vdash x : A}\ \geq W \qquad \dfrac{\dfrac{\dfrac{}{(p)\ [y : B]_1 \vdash y : B}\ \text{var}}{(q)\ [y : B]_1 \vdash y : B}\ \geq W}{(q)\ [x : A]_1, [y : B]_1 \vdash (x, y) : A \otimes_q B}\ \otimes I}{(p)\ [x : A]_1, [y : B]_1 \vdash (x, y) : A \otimes_q B}\ \geq W}{(p)\ \Gamma \vdash \big(\mathbf{let}\ (x, y) = e\ \mathbf{in}\ (x, y)\big) : A \otimes_q B}\ \otimes E$$

$$\frac{}{(p)\ [x : A]_1 \vdash x : A}\ \text{var} \qquad \frac{}{(p)\ \emptyset \vdash * : \mathsf{Unit}}\ \mathsf{Unit}$$

$$\frac{(p)\ \Gamma \vdash a : A \qquad (p)\ \Delta \vdash b : B}{(p)\ C^p\,(\Gamma;\Delta) \vdash (a,b) : A \otimes_p B}\ \otimes I$$

$$\frac{(p)\ \Gamma \vdash e : A \otimes_p B \qquad (p)\ \Delta, [x : A]_s, [y : B]_s \vdash c : C}{(p)\ C^p\,(s\Gamma;\Delta) \vdash (\mathbf{let}\ \ (x,y) = e\ \ \mathbf{in}\ \ c) : C}\ \otimes E$$

$$\frac{(p)\ \Gamma \vdash a : A}{(p)\ \Gamma \vdash \mathbf{inj}_1\, a : A \oplus B}\ \oplus I_{\triangleleft} \qquad \frac{(p)\ \Gamma \vdash b : B}{(p)\ \Gamma \vdash \mathbf{inj}_2\, b : A \oplus B}\ \oplus I_{\triangleright}$$

$$\frac{(p)\ \Gamma \vdash e : A \oplus B \qquad (p)\ \Delta, [x : A]_s \vdash c_1 : C \qquad (p)\ \Delta, [y : B]_s \vdash c_2 : C}{(p)\ C^p\,(s\Gamma;\Delta) \vdash (\mathbf{case}\ \ e\ \ \mathbf{of}\ \ x \Rightarrow c_1\ \ \mathbf{or}\ \ y \Rightarrow c_2) : C}\ \oplus E$$

$$\frac{(p)\ \Gamma \vdash a : A}{(p)\ s\Gamma \vdash\, !a : !_s A}\ !I \qquad \frac{(p)\ \Gamma \vdash e : !_r A \qquad (p)\ \Delta, [x : A]_{rs} \vdash c : C}{(p)\ C^p\,(s\Gamma;\Delta) \vdash (\mathbf{let}\ \ !x = e\ \ \mathbf{in}\ \ c) : C}\ !E$$

$$\frac{(p)\ \Gamma, [x : A]_1 \vdash b : B}{(p)\ \Gamma \vdash (\lambda x.\ b) : A \multimap_p B}\ \multimap I \qquad \frac{(p)\ \Gamma \vdash f : A \multimap_p B \qquad (p)\ \Delta \vdash a : A}{(p)\ C^p\,(\Gamma;\Delta) \vdash f(a) : B}\ \multimap E$$

$$\frac{(p)\ \Gamma \vdash e : A[\alpha \mapsto \mu\alpha.\ A]}{(p)\ \Gamma \vdash \underset{\mu\alpha.\ A}{\mathbf{fold}}\, e : A}\ \mu I \qquad \frac{(p)\ \Gamma \vdash a : A}{(p)\ \Gamma \vdash \underset{\mu\alpha.\ A}{\mathbf{unfold}}\, a : A[\alpha \mapsto \mu\alpha.\ A]}\ \mu E$$

$$\frac{(1)\ \Gamma \vdash a : A}{(1)\ \infty \cdot \Gamma \vdash \mathbf{return}\, a : \bigcirc A}\ \bigcirc I \qquad \frac{(1)\ \Gamma \vdash e : \bigcirc A \qquad (1)\ \Delta, [x : A]_\infty \vdash b : \bigcirc B}{(1)\ \Gamma + \Delta \vdash (\mathbf{let}\ \ \bigcirc x = e\ \ \mathbf{in}\ \ b) : \bigcirc B}\ \bigcirc E$$

$$\frac{(p)\ \Gamma \vdash a : A \qquad \Gamma \leq \Delta \qquad p \geq q}{(q)\ \Delta \vdash a : A}\ \geq W \qquad \frac{(p)\ \Gamma \vdash a : A \qquad \Gamma \leq \Delta \qquad p \leq q}{(q)\ c(p,q) \cdot \Delta \vdash a : A}\ \leq W$$

Figure 1: Typing Rules for Plurimetric Fuzz

Example. As an example, say we want to compose a function $f\colon \mathsf{Real} \multimap_1 \mathsf{Real} \otimes_2 \mathsf{Real}$ with a function $g\colon \mathsf{Real} \otimes_1 \mathsf{Real} \multimap_1 \mathsf{Real}$, both typable in an empty context. We can first apply f to an input x, and then coerce the result to obtain the judgement $(1)\ [x : \mathsf{Real}]_{\sqrt{2}} \vdash e : \mathsf{Real} \otimes_1 \mathsf{Real}$ for some term e which is semantically equivalent to the term $f(x)$. At that point, we can apply g to e, and use the $(!I)$ and $(\multimap I)$ rules to derive the judgement $(1)\ \emptyset \vdash h : !_{\sqrt{2}}\mathsf{Real} \multimap_1 \mathsf{Real}$ for some term h which behaves like $g \circ f$.

4 Operational Semantics

We want now to examine the properties of our type system with respect to operational semantics. We consider for that the same big-step operational semantics as for Fuzz [19]. First, values are given by the following grammar:

$$v ::= * \mid (v,v) \mid \lambda x.\ b \mid\, !v \mid \mu \mid \underset{A}{\mathbf{fold}}\ \ v \mid \mathbf{inj}_1\, v \mid \mathbf{inj}_2\, v\,. \tag{3}$$

$$\frac{}{(p)\ \emptyset \vdash n : \mathsf{Nat}}\ \mathsf{Nat} \qquad \frac{}{(p)\ \emptyset \vdash r : \mathsf{Real}}\ \mathsf{Real}$$

$$\frac{(p)\ \Gamma \vdash x : N \qquad (p)\ \Delta \vdash y : N \qquad N \in \{\mathsf{Nat}, \mathsf{Real}\}}{(p)\ c(1,p) \cdot C^p\,(\Gamma; \Delta) \vdash x + y : N}\ +$$

$$\frac{(p)\ \Gamma \vdash x : N \qquad k \in N \qquad N \in \{\mathsf{Nat}, \mathsf{Real}\}}{(p)\ k\Gamma \vdash k \times x : N}\ \times$$

$$\frac{(p)\ \Gamma \vdash x : A}{(p)\ \infty \cdot \Gamma \vdash \{x\} : \mathsf{Set}(A)}\ \mathsf{Set} \qquad \frac{(p)\ \Gamma \vdash e : \mathsf{Set}(A)}{(p)\ \Gamma \vdash \mathrm{card}(e) : \mathsf{Nat} \oplus \mathsf{Unit}}\ \mathrm{card}$$

$$\frac{}{(p)\ \emptyset \vdash \mathrm{setfilter} : (A \to_p \mathsf{Bool}) \to_p \mathsf{Set}(A) \multimap_p \mathsf{Set}(A)}\ \mathrm{setfilter}$$

$$\frac{}{(p)\ \emptyset \vdash \mathrm{setmap} : (A \to_p B) \to_p \mathsf{Set}(A) \multimap_p \mathsf{Set}(B)}\ \mathrm{setmap}$$

$$\frac{}{(p)\ \emptyset \vdash \mathrm{setfold} : (A \multimap_p B \multimap_p B) \to_p B \to_p \mathsf{Set}(A) \multimap_p B}\ \mathrm{setfold}$$

Figure 2: Typing Rules for Primitive Operations

where μ ranges over multisets of probability-value pairs. For the sake of brevity, we do not reproduce the entirety of the evaluation rules here, but only two illustrative cases:

$$\frac{f \Downarrow \lambda x.\ b \qquad a \Downarrow v_a \qquad b[x \mapsto v_a] \Downarrow v}{f(a) \Downarrow v} \qquad \frac{a \Downarrow v_a \qquad b \Downarrow v_b}{(a,b) \Downarrow (v_a, v_b)}$$

They satisfy the desired metatheoretical properties, namely substitution and subject reduction (also known as type preservation).

Lemma 4.1 (Substitution)**.** *If we have* $(p)\ \Gamma, [x : A]_s \vdash b : B$ *and* $(p)\ \Delta \vdash a : A$*, then* $(p)\ C^p\,(\Gamma; s\Delta) \vdash b[x \mapsto a] : B$*.*

See Appendix A for the complete set of evaluation rules, which can be extended with rules for primitive operations, and for a partial proof of the substitution lemma, which requires to prove a slightly more general statement.

Theorem 4.2 (Subject Reduction)**.** *Plurimetric Fuzz is type-safe with respect to its operational semantics. This means that if we have* $(p)\ \emptyset \vdash a : A$ *and* $a \Downarrow v$*, then* $(p)\ \emptyset \vdash v : A$*.*

Proof. By induction on the shortest evaluation tree of $a \Downarrow v$, inversing the typing rules, and for function application, **let** and **case** expressions, by using Lemma 4.1. □

5 Denotational Semantics

We will show how to interpret types and type derivations in the denotational semantics of metric spaces and non-expansive maps.

5.1 Operations on Metric Spaces

Definition 5.1. Let (X, d_X), (Y, d_Y) and (Z, d_Z) be three metric spaces, p be a parameter, and s be a sensitivity. The *scaling* of (X, d) by s is the metric space $!_s X \equiv (X, s \cdot d_X)$. Moreover, the *p-tensor*

product $X \otimes_p Y$ of X and Y is the set $X \times Y$ endowed with

$$d_{X \otimes_p Y}\big((x,y),(x',y')\big) \equiv \sqrt[p]{d_X(x,x')^p + d_Y(y,y')^p}\,; \tag{4}$$

the *p-affine arrow* $X \multimap_p Y$ from X to Y is the set Y^X endowed with

$$d_{X \multimap_p Y}(f,f') \equiv \inf\left\{r \geq 0 : \forall x,x' \in X, d_Y\big(f(x),f'(x')\big)^p \leq r^p + d_X(x,x')^p\right\}\,; \tag{5}$$

and the *disjoint union* $X \oplus Y$ of X and Y is the set $X \sqcup Y$ endowed with

$$d_{A \oplus B}(e,e') \equiv \begin{cases} d_A(e,e') & \text{if } e,e' \in [\![A]\!] \\ d_B(e,e') & \text{if } e,e' \in [\![B]\!] \\ \infty & \text{otherwise.} \end{cases} \tag{6}$$

Given two maps $f\colon X \oplus Y$ and $g\colon Y \oplus Z$, the *coproduct* $[f,g]\colon X \oplus Y \oplus Z$ of f and g is the map defined by $[f,g]\big(i_1(x)\big) \equiv f(x)$, and $[f,g]\big(i_2(y)\big) \equiv g(y)$.

Note that it follows directly from the definitions above that for all p, the operation $\otimes_p$ is commutative and associative up to isomorphism, and that the evaluation map $\mathrm{Ev}\colon (X \multimap_p Y) \otimes_p X \multimap_p Y$ is non-expansive. Moreover, we have $(X \oplus Y) \otimes_p Z \simeq (X \otimes_p Z) \oplus (Y \otimes_p Z)$.

We also define probability distributions over metric spaces.

Definition 5.2. A *discrete probability distribution* over a countable metric space X is a function $\mu : X \to [0,1]$ such that $\sum_{x \in X} \mu(x) = 1$. We write $\mathrm{Dist}(X)$ for the set of discrete probability distributions over X endowed with the following distance, which depends on a given positive real number ϵ_0:

$$\max \mathrm{div}(\mu,\mu') \equiv \frac{1}{\epsilon_0} \max_{x \in X} \left| \ln \frac{\mu(x)}{\mu'(x)} \right| \tag{7}$$

with the convention that $0/0 \equiv 1$ and $|\ln(0/x)| \equiv |\ln(x/0)| \equiv \infty$ for all $x > 0$. We write δ_x for the Dirac distribution at x, i.e., the function such that $\delta_x(x) = 1$ and $\delta_x(x') = 0$ for all $x' \neq x$.

Recall that the *support* $\operatorname{supp}\mu$ of a distribution μ over a set X is the set of elements of X with non-zero probability, and note that if μ and μ' are two discrete distributions over the same set X, then $\max \mathrm{div}(\mu,\mu')$ is finite if and only if $\operatorname{supp}\mu = \operatorname{supp}\mu'$. This distance is "carefully chosen" [19, Section 4.2] to ensure that the following lemma holds.

Lemma 5.1. *A non-expansive map from X to $\mathrm{Dist}(Y)$ is exactly an ϵ_0-differentially private random map from X to Y.*

To compose probabilistic programs, we define the Kleisli extension of a map.

Definition 5.3. The *Kleisli extension* $f^\dagger : \mathrm{Dist}(X) \to \mathrm{Dist}(Y)$ of a map $f : X \to \mathrm{Dist}(Y)$ is defined by the following formula: $f(\mu)(y) \equiv \sum_{x \in X} \mu(x) \cdot f(x)(y)$.

5.2 Interpretation of Types, Contexts and Derivations

Let Core Plurimetric Fuzz be the fragment of Plurimetric Fuzz without recursive types. We interpret its types inductively as metric spaces:

- $[\![\mathsf{Unit}]\!] \equiv (\{*\}, 0)$;
- $[\![\mathsf{Nat}]\!] \equiv (\mathbf{N}, (m,n) \mapsto |m-n|)$;
- $[\![\mathsf{Real}]\!] \equiv (\mathbf{R}, (x,y) \mapsto |x-y|)$;
- $[\![!_s A]\!] \equiv !_s [\![A]\!]$;
- $[\![A \otimes_p B]\!] \equiv [\![A]\!] \otimes_p [\![B]\!]$;
- $[\![A \multimap_p B]\!] \equiv [\![A]\!] \multimap_p [\![B]\!]$;

- $[\![A \oplus B]\!] \equiv [\![A]\!] \oplus [\![B]\!]$;
- $[\![\bigcirc A]\!] \equiv \mathrm{Dist}\,[\![A]\!]$;
- $[\![\mathsf{Set}(A)]\!] \equiv \big(\mathcal{P}([\![A]\!]), \mathrm{card}(\cdot \mathbin{\triangle} \cdot)\big)$.

where $\triangle$ is the symmetric difference on sets. Next, we define the interpretation of contexts: $[\![(p)\ \emptyset]\!] \equiv \{*\}$ and $[\![(p)\ \Gamma, x : A]\!] \equiv [\![A]\!] \otimes_p (p)\ \Gamma$. Derivations are seen as non-expansive maps between metric spaces. More precisely, if π is a derivation whose last rule is R, we write π_e for its premise whose conclusion has term e, and Γ and Δ for the contexts involved. Moreover, we write $\hat{\cdot}$ for the currying map; Ev for the evaluation map; if R is a weakening rule, I_p for the inclusion map from $[\![(p)\ \Delta]\!]$ to $[\![(p)\ \Gamma]\!]$ when $\Gamma \leq \Delta$, and I_p^q for the natural map from $[\![(p)\ \Gamma]\!]$ to $[\![(q)\ \Gamma]\!]$; and if R is binary or ternary, D_p for the diagonal map from $[\![(p)\ C^p\,(\Gamma; \Delta)]\!]$ to $[\![(p)\ \Gamma]\!] \otimes_p [\![(p)\ \Delta]\!]$. We omit isomorphisms when they are clear from the context.

$$
\begin{array}{ll}
(\mathsf{Unit}) & [\![\pi]\!] \equiv \mathrm{Const}_* \\
(\mathbf{var}) & [\![\pi]\!] \equiv \mathrm{Id}_{[\![A]\!]} \\
(\otimes I) & [\![\pi]\!] \equiv \big([\![\pi_a]\!] \times [\![\pi_b]\!]\big) \circ D_p \\
(\otimes E) & [\![\pi]\!] \equiv [\![\pi_c]\!] \circ (\mathrm{Id} \times r \cdot [\![\pi_e]\!]) \circ D_p \\
(\multimap I) & [\![\pi]\!] \equiv \widehat{[\![\pi_b]\!]} \\
(\multimap E) & [\![\pi]\!] \equiv \mathrm{Ev} \circ \big([\![\pi_f]\!] \times [\![\pi_a]\!]\big) \circ D_p \\
(\oplus I_\triangleleft) & [\![\pi]\!] \equiv i_1 \circ [\![\pi_a]\!] \\
(\oplus I_\triangleright) & [\![\pi]\!] \equiv i_2 \circ [\![\pi_b]\!]
\end{array}
\qquad
\begin{array}{ll}
(\oplus E) & [\![\pi]\!] \equiv [[\![\pi_{c_1}]\!], [\![\pi_{c_2}]\!]] \circ (\mathrm{Id} \times s \cdot [\![\pi_e]\!]) \circ D_p \\
(!I) & [\![\pi]\!] \equiv s \cdot [\![\pi_a]\!] \\
(!E) & [\![\pi]\!] \equiv [\![\pi_c]\!] \circ \big(\mathrm{Id} \times r \cdot [\![\pi_e]\!]\big) \circ D_p \\
(\bigcirc I) & [\![\pi]\!] \equiv \delta \circ \infty \cdot [\![\pi_a]\!] \\
(\bigcirc E) & [\![\pi]\!] \equiv \mathrm{Ev} \circ \big((\infty \cdot [\![\pi_e]\!]) \times (\cdot^\dagger \circ \widehat{[\![\pi_b]\!]})\big) \circ D_1 \\
(\leq W) & [\![\pi]\!] \equiv c(p,q) \cdot I_q^p \circ I_q \circ [\![\pi_a]\!] \\
(\geq W) & [\![\pi]\!] \equiv I_q^p \circ I_q \circ [\![\pi_a]\!]
\end{array}
$$

5.3 Soundness of Core Plurimetric Fuzz

Let Core Plurimetric Fuzz be the fragment of Plurimetric Fuzz without recursive types.

Proposition 5.2 (Soundness)**.** *If π is a Core Plurimetric Fuzz derivation of $(p)\ \Gamma \vdash a : A$, then $[\![\pi]\!]$ is a non-expansive map from $[\![(p)\ \Gamma]\!]$ to $[\![A]\!]$.*

Properties we use repeatedly in the proof of this result are summarised in the following lemmata. See Appendix B and Appendix C for more details.

Lemma 5.3. *For all precontexts Γ, and reals $p \geq 1$ and $s \geq 0$, we have $[\![(p)\ s\Gamma]\!] = !_s[\![(p)\ \Gamma]\!]$.*

Proof. By induction on Γ, using the fact that for all metric spaces X and Y, we have $!_sX \otimes_p !_sY = !_s(X \otimes_p Y)$. □

Lemma 5.4. *For all metric spaces X_1, X_2, Y_1, and Y_2, and parameters p, if $f\colon X_1 \to Y_1$ and $g\colon X_2 \to Y_2$ are non-expansive maps, then so is $f \times g\colon X_1 \otimes_p X_2 \to Y_1 \otimes_p Y_2$.*

Proof. The function $(x, y) \mapsto \sqrt[p]{x^p + y^p}$ is increasing in both arguments over $\mathbf{R}_{\geq 0} \times \mathbf{R}_{\geq 0}$. □

Lemma 5.5 ([23, Proposition 4.1])**.** *For all metric spaces X and Y, and parameters p and q such that $p \leq q$, the identity map on pairs belongs to the following spaces: $X \otimes_p Y \multimap X \otimes_q Y$ and $!_{c(p,q)}(X \otimes_q Y) \multimap X \otimes_p Y$.*

Lemma 5.6. *For all compatible precontexts Γ and Δ, the diagonal map $D_p\colon [\![(p)\ C^p\,(\Gamma; \Delta)]\!] \to [\![(p)\ \Gamma]\!] \otimes_p [\![(p)\ \Delta]\!]$ is non-expansive. Moreover, if $\Gamma \leq \Delta$, then the inclusion map $I_p\colon [\![(p)\ \Gamma]\!] \to [\![(p)\ \Delta]\!]$ is non-expansive.*

Proof. By induction on Γ, using the fact that for all metric spaces X, and sensitivities r and s, we have a non-expansive map from $!_{\sqrt[p]{r^p + s^p}}X$ to $!_rX \otimes_p !_sX$ given by $x \mapsto (x, x)$. □

Lemma 5.7. *The* bind *map defined by* $\mathrm{bind}(f,\mu) \equiv f^\dagger(\mu)$ *is non-expansive from* $\mathrm{Dist}\, Y \otimes_1 (Y \to_1 \mathrm{Dist}\, X)$ *to* $\mathrm{Dist}\, X$.

5.4 Recursive types

In order to handle iso-recursive types, we extend the semantics from metric spaces to metric complete partial orders introduced for Fuzz in [2].

Definition 5.4. A *metric complete partial order* is a complete partial order X endowed with a metric d such that for all $(x_i)_{i\in\mathbf{N}}$ and $(x'_i)_{i\in\mathbf{N}}$ two ω-chains in X, if $d_X(x_i,x'_i) \le r$ for all $i \in \mathbf{N}$, then $d_X\big(\bigsqcup_{i\in\mathbf{N}} x_i, \bigsqcup_{i\in\mathbf{N}} {x'}_i\big) \le r$.

Equivalently, we may ask that $d\big(\bigsqcup_{i\in\mathbf{N}} x_i, \bigsqcup_{i\in\mathbf{N}} {x'}_i\big) \le \liminf_{i\to\infty} d(x_i,x'_i)$.

This framework allows to describe Plurimetric Fuzz recursive types as solutions to domain equations of the form $F(X) = X$, and to describe divergence by the least element $\bot$. See Appendix C for an explanation of why we don't use recursive types to define the type of natural numbers and arithmetic operations.

Theorem 5.8 ([2, Theorem 4.15])**.** $\mathsf{MetCPO}_\bot$ *is an algebraically compact* CPO*-category, that is for every* CPO*-endofunctor* F*, there exists an object* μF *and an isomorphism* $i : F(\mu F) \simeq \mu F$ *such that* i *is an initial algebra and* i^{-1} *is a final coalgebra.*

We have to show that $\mathsf{MetCPO}_\bot$ is closed under the tensor and arrow constructors.

Lemma 5.9. *Let* X *and* Y *be two metric complete partial orders, and let* $(f_i)_{i\in\mathbf{N}}$ *and* $(g_i)_{i\in\mathbf{N}}$ *be two* ω*-chains in* $X \multimap_p Y$ *such that* $d_{X\multimap_p Y}(f_i,g_i) \le r$ *for all* $i \in \mathbf{N}$*. Then, for all* x_1 *and* x_2 *in* X*, and* $i \in \mathbf{N}$*, we have* $d_Y\big(f_i(x_1), g_i(x_2)\big) \le r + d_X(x_1,x_2)$.

Proof.
$$\begin{aligned} d_Y\big(f_i(x_1),g_i(x_2)\big) &\le d_{X\multimap_1 Y}(f_i,g_i) + d_X(x_1,x_2) && \text{by Equation 5}\\ &\le d_{X\multimap_p Y}(f_i,g_i) + d_X(x_1,x_2) && \text{by [23, Theorem 4.3]}\\ &\le r + d_X(x_1,x_2) \end{aligned}$$
□

Corollary 5.10. *If* X *and* Y *are two metric complete partial orders, then so is* $X \multimap_p Y$.

Lemma 5.11. *If* X *and* Y *are two metric complete partial orders, then so is* $X \otimes_p Y$.

Proof. Let $(p_i)_{i\in\mathbf{N}}$ and $(p'_i)_{i\in\mathbf{N}}$ be two ω chains in $X \times Y$. For all $i \in \mathbf{N}$ we write $p_i = (x_i, y_i)$ and $p'_i = (x'_i, y'_i)$. Since X and Y are metric complete partial orders, we have

$$d_X\left(\bigsqcup_{i\in\mathbf{N}} x_i, \bigsqcup_{i\in\mathbf{N}} {x'}_i\right) \le \liminf_{i\to\infty} d_X(x_i,x'_i) \quad\text{and}\quad d_Y\left(\bigsqcup_{i\in\mathbf{N}} y_i, \bigsqcup_{i\in\mathbf{N}} {y'}_i\right) \le \liminf_{i\to\infty} d_Y(y_i,y'_i)\,.$$

Therefore, as the function $x \mapsto x^p$ is increasing, we have

$$\begin{aligned} d_X\left(\bigsqcup_{i\in\mathbf{N}} x_i, \bigsqcup_{i\in\mathbf{N}} {x'}_i\right)^p + d_Y\left(\bigsqcup_{i\in\mathbf{N}} y_i, \bigsqcup_{i\in\mathbf{N}} {y'}_i\right)^p &\le \liminf_{i\to\infty} d_X(x_i,x'_i)^p + \liminf_{i\to\infty} d_Y(y_i,y'_i)^p\\ &\le \liminf_{i\to\infty} \big(d_X(x_i,x'_i)^p + d_Y(y_i,y'_i)^p\big)\\ &= \liminf_{i\to\infty} d_X(x_i,x'_i)^p + d_Y(y_i,y'_i)^p \end{aligned}$$

and by taking the p-th root of both sides, we obtain $d_{X\otimes_p Y}\big(\bigsqcup_{i\in\mathbf{N}} p_i, \bigsqcup_{i\in\mathbf{N}} {p'}_i\big) \le \liminf_{i\to\infty} d_{X\otimes_p Y}(p_i,p'_i)$. □

From Lemma 5.2 and what precedes, we can deduce the soundness of Plurimetric Fuzz.

Theorem 5.12 (Soundness)**.** *If* π *is a Plurimetric Fuzz derivation of* $(p)\ \Gamma \vdash a : A$*, then* $[\![\pi]\!]$ *is a non-expansive map from* $[\![(p)\ \Gamma]\!]$ *to* $[\![A]\!]$.

6 Expressive power and Precision

Let us now illustrate the usage of our type system.

Example: Functions with Multiple Arguments. Let us take the same example as in [23, Section 5]: consider the term $\lambda c.\ \big(\mathbf{let}\,(x,y) = c\ \mathbf{in}\ f(!x,y) + g(x,!y)\big)$ where $f : !_2\mathsf{Real} \otimes_2 \mathsf{Real} \multimap_2 \mathsf{Real}$ and $g : \mathsf{Real} \otimes_2 !_2\mathsf{Real} \multimap_2 \mathsf{Real}$. The following derivation is valid in Plurimetric Fuzz:

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ (2)\ [x:\mathsf{Real}]_2, [y:\mathsf{Real}]_1 \vdash f(!x,y) : \mathsf{Real} & (2)\ [x:\mathsf{Real}]_1, [y:\mathsf{Real}]_2 \vdash g(x,!y) : \mathsf{Real}\end{array}}{(2)\ \underbrace{\sqrt{2}\cdot([x:\mathsf{Real}]_{\sqrt{5}}, [y:\mathsf{Real}]_{\sqrt{5}})}_{=\sqrt{10}\cdot([x:\mathsf{Real}]_1, [y:\mathsf{Real}]_1)} \vdash f(!x,y) + g(x,!y) : \mathsf{Real}}\ +$$

Therefore, by using Plurimetric Fuzz, we manage to obtain the same sensitivity as with Bunched Fuzz, that is to say $\sqrt{10} \approx 3 + 1/6$, while a naïve extension of Fuzz would overestimate it to 4 [23, Section 5].

Example: Neighbour Classification. Let us consider an example using the Euclidean distance L^2. Say that given a database of labelled points in the Euclidean plane, we want to predict the label of a new point x by a majority vote weighted by the distance d to its neighbours (approximately 1 when $d < r$ for a given radius r, and 0 otherwise). Here, we choose the function $\mathrm{weight} : x \mapsto 1 - 1/(1 + e^{-4(x-r)})$. This is the complement of a shifted and scaled function (widely used as an activation function in machine learning), which can be soundly added to the language as a primitive of type $\mathsf{Real} \multimap_1 \mathsf{Real}$ (see Lemma 2.2).

A row of the database is represented by the following type: $\mathsf{Row} = \mathsf{Point} \otimes_1 \mathsf{Label}$ where $\mathsf{Point} = \mathsf{Real} \otimes_2 \mathsf{Real}$, and $\mathsf{Label} = \mathsf{Unit} \oplus \cdots \oplus \mathsf{Unit}$. We assume that the coordinates are precise enough so that no two different points have the same coordinates, and that $x = (0,0)$ (we lose nothing in generality by doing this, since translation is a non-expansive operation on the Euclidean plane).

The algorithm is implemented as follows (where = is an ∞-sensitive primitive):

Listing 1: Implementation of the neighbour classification algorithm

```
let get_pos   (r : row) : point = let (pos, _)   = r in pos
let get_label (r : row) : label = let (_, label) = r in label

let score (l : label) (db : database) : real = db
        |> setfilter (fun r -> get_label r = l)
        |> setmap (fun r -> distance (0, 0) (get_pos r))
        |> setfold (fun acc x -> acc + weight x) 0

let predict (db : database) : label = exp_noise labels score db
```

Informally, `score` computes the score of a label by: (1) filtering the database to keep only the points with the given label; (2) computing the distance of each point to the origin (the Euclidean distance `distance` is non-expansive on elements of type Point); (3) computing the sum of the weights of the points. Moreover, `exp_noise` is a specialised version of the exponential mechanism presented in [13, Equation 1] for the case $s = 1$ and $\epsilon = 1$, which has type $\mathsf{Set}(\mathsf{Label}) \to_1 (\mathsf{Label} \to_1 \mathsf{Database} \multimap_1 \mathsf{Real}) \to_1 \mathsf{Set}(\mathsf{Row}) \multimap_1 \bigcirc\mathsf{Label}$.

We can therefore derive the following types for the above functions (see Appendix D for a sketch of the derivations): score : $\mathsf{Label} \to_1 \mathsf{Database} \multimap_1 \mathsf{Real}$, and predict : $\mathsf{Set}(\mathsf{Row}) \multimap_1 \bigcirc\mathsf{Label}$. In particular, by Lemma 5.1, this implementation of the neighbour classification algorithm is 1-differentially private.

7 Translation Mappings

In order to better understand the relationships between Fuzz and Plurimetric Fuzz we will now investigate some translations between the two systems. We consider a presentation of Fuzz with a weakening rule (W), rather than axioms with an arbitrary context. Moreover, we extend Plurimetric Fuzz by adding a & type constructor to simplify the presentation. Its introduction and elimination rules are given in Appendix E.

Let Der(Fuzz) be the set of derivations in Fuzz endowed with the following partial order: for all derivations π of $\Gamma \vdash a : A$ and π' of $\Delta \vdash b : B$, we have $\pi \leq \pi'$ iff $\Gamma \leq \Delta$ (see Definition 3.2) and $(a, A) = (b, B)$. Similarly, we define Der(PFuzz) for Plurimetric Fuzz.

7.1 Translation from Fuzz to Plurimetric Fuzz

For all parameters p, we define a mapping P^p_{type} from Fuzz types to Plurimetric Fuzz types by structural induction as follows:

$$\begin{aligned}
P^p_{\text{type}}(\mathsf{Unit}) &= \mathsf{Unit} & P^p_{\text{type}}(A \multimap B) &= P^p_{\text{type}}(A) \multimap_p P^p_{\text{type}}(B) \\
P^p_{\text{type}}(A \oplus B) &= P^p_{\text{type}}(A) \oplus P^p_{\text{type}}(B) & P^p_{\text{type}}(!_s A) &= !_{s^{1/p}} P^p_{\text{type}}(A) \\
P^p_{\text{type}}(A \,\&\, B) &= P^p_{\text{type}}(A) \,\&\, P^p_{\text{type}}(B) & P^p_{\text{type}}(\bigcirc A) &= \bigcirc P^p_{\text{type}}(A) \\
P^p_{\text{type}}(A \otimes B) &= P^p_{\text{type}}(A) \otimes_p P^p_{\text{type}}(B) & P^p_{\text{type}}(\mu\alpha.\, A) &= \mu\alpha.\, P^p_{\text{type}}(A)
\end{aligned} \tag{8}$$

Note that Fuzz lists are mapped to p-lists in Plurimetric Fuzz, i.e., for all type A, we have $P^p_{\text{type}}\big(\mathsf{List}(A)\big) = \mathsf{List}_p\big(P^p_{\text{type}}(A)\big)$. The distance on the latter type is given by $d_{\mathsf{List}_p(A)}(l, l') = \sqrt[p]{\sum_{i=1}^{n} d_A(l_i, l'_i)^p}$ if $\text{length}(l) = \text{length}(l')$, and ∞ otherwise.

We also define a mapping P^p_{ctx} from Fuzz contexts to Plurimetric Fuzz precontexts by $P^p_{\text{ctx}}(\emptyset) = \emptyset$, and $P^p_{\text{ctx}}(\Gamma, [x : A]_s) = P^p_{\text{ctx}}(\Gamma), [x : P^p_{\text{type}}(A)]_{s^{1/p}}$, and a mapping P^p_{der} on derivations. For unary rules, we have for instance:

$$P^p_{\text{der}}\left(\dfrac{}{[x : A]_1 \vdash x : A}\ \text{var}\right) = \dfrac{}{(p)\ [x : P^p_{\text{type}}(A)]_1 \vdash x : P^p_{\text{type}}(A)}\ \text{var}$$

and similarly for binary and ternary rules:

$$P^p_{\text{der}}\left(\dfrac{\begin{matrix}\vdots\ \pi_a \\ \Gamma \vdash a : A\end{matrix} \qquad \begin{matrix}\vdots\ \pi_b \\ \Delta \vdash b : B\end{matrix}}{\Gamma + \Delta \vdash (a, b) : A \otimes B}\ \otimes I\right) = \dfrac{\dfrac{\begin{matrix}\vdots\ P^p_{\text{der}}(\pi_a) \\ (p)\ P^p_{\text{ctx}}(\Gamma) \vdash a : P^p_{\text{type}}(A)\end{matrix} \qquad \begin{matrix}\vdots\ P^p_{\text{der}}(\pi_b) \\ (p)\ P^p_{\text{ctx}}(\Delta) \vdash b : P^p_{\text{type}}(B)\end{matrix}}{(p)\ C^p\left(P^p_{\text{ctx}}(\Gamma); P^p_{\text{ctx}}(\Delta)\right) \vdash (a, b) : P^p_{\text{type}}(A) \otimes_p P^p_{\text{type}}(B)}\ \otimes I}{(p)\ P^p_{\text{ctx}}(\Gamma + \Delta) \vdash (a, b) : P^p_{\text{type}}(A \otimes B)} = W$$

Definition 7.1. A derivable judgement $\Gamma \vdash e : A$ is said to be *minimal* in a (Plurimetric) Fuzz if for all contexts Δ such that $\Delta \vdash e : A$, we have $\Gamma \leq \Delta$.

We can now prove the main result of this section, that is to say that the translation of a valid derivation is valid (see Appendix E for a proof).

Lemma 7.1. *For all precontexts Γ and Δ, sensitivities s, and parameters p, we have the following equality: $C^p\left(P^p_{\text{ctx}}(\Gamma); s^{1/p} \cdot P^p_{\text{ctx}}(\Delta)\right) = P^p_{\text{ctx}}(\Gamma + s\Delta)$.*

Corollary 7.2. *For all parameters p, the image by P^p_{der} of the derivation π of a (minimal) judgement $\Gamma \vdash a : A$ in Fuzz is a valid derivation of a (minimal) judgement $(p)\ P^p_{\text{ctx}}(\Gamma) \vdash a : P^p_{\text{type}}(A)$ in Plurimetric Fuzz.*

In particular, all examples in [19] that only use structural and logical rules can be translated to Plurimetric Fuzz for any parameter p. This includes elementary operations on lists such as binary and iterated concatenation, length, but also higher-order combinators such as map, foldl, foldr (see [19, Section 3.2]). More generally, this means that the L^1 sensitivity properties obtained by typing in Fuzz for these programs can be for free transposed into L^p sensitivity properties obtained by typing in Plurimetric Fuzz. However, Theorem 7.2 does not extend to primitive operations (the ones presented in Figure 2), which do not behave uniformly with respect to the metric chosen on the pairs and functions.

Claim 7.3 (No miracle)**.** *We cannot soundly extend P_{der} to the derivations involving primitive operations such as addition on numbers.*

Proof. For instance, we cannot soundly translate the following $(+)$ rule for $p = 2$:

$$\frac{\Gamma \vdash a : \mathsf{Real} \qquad \Delta \vdash b : \mathsf{Real}}{\Gamma + \Delta \vdash a + b : \mathsf{Real}}\,+ \quad \xrightarrow{P^2_{\mathrm{der}}} \quad \frac{(2)\ P^2_{\mathrm{ctx}}(\Gamma) \vdash a : \mathsf{Real} \qquad (2)\ P^2_{\mathrm{ctx}}(\Delta) \vdash b : \mathsf{Real}}{(2)\ P^2_{\mathrm{ctx}}(\Gamma + \Delta) \vdash a + b : \mathsf{Real}}\,+$$

as the following function is not non-expansive: $(+) \colon \mathbf{R} \otimes_2 \mathbf{R} \to \mathbf{R}$ (its sensitivity is $\sqrt{2}$). □

7.2 Translation from Plurimetric Fuzz to Fuzz

Conversely, we can define *partial* mappings F^p_{type}, F^p_{ctx} and F^p from Plurimetric Fuzz to Fuzz. We only give the most interesting cases:

$$\begin{aligned}
F^p_{\mathrm{type}}(A \otimes_q B) &= F^p_{\mathrm{type}}(A) \otimes F^p_{\mathrm{type}}(B) && \text{if } q \le p \\
F^p_{\mathrm{type}}(A \multimap_q B) &= F^p_{\mathrm{type}}(A) \multimap F^p_{\mathrm{type}}(B) && \text{if } q \le p \\
F^p_{\mathrm{type}}(!_s A) &= !_{s^p} F^p_{\mathrm{type}}(A) \\
F^p_{\mathrm{ctx}}([x : A]_s, \Gamma) &= [x : F^p_{\mathrm{type}}(A)]_{s^p}, F^p_{\mathrm{ctx}}(\Gamma) \\
F^p_{\mathrm{der}}\left(\overline{(q)\ [x : A]_1 \vdash x : A}^{\ \mathrm{var}}\right) &= \overline{[x : F^p_{\mathrm{type}}(A)]_1 \vdash x : F^p_{\mathrm{type}}(A)}^{\ \mathrm{var}} && \text{if } q \le p
\end{aligned}$$

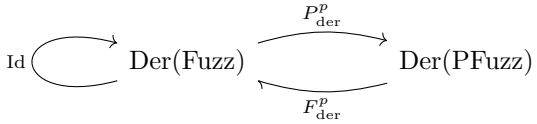
Lemma 7.4. *For all precontexts Γ and Δ, for all sensitivities s, we have the following inequality: $s^p F^p_{\mathrm{ctx}}(\Gamma) + F^p_{\mathrm{ctx}}(\Delta) \le F^p_{\mathrm{ctx}}\big(C^p(\Gamma; s\Delta)\big)$.*

It follows from the definition above that the image $F^p_{\mathrm{der}}(\pi)$ of a Plurimetric Fuzz derivation π is defined iff any parameter q occurring in a judgement of π is less than or equal to p.

Corollary 7.5. *For all parameters p, if the image by the mapping F^p_{der} of a derivation π in Plurimetric Fuzz is defined, then it is a valid derivation in Fuzz.*

Finally, we obtain the following property relating the two translations:

Theorem 7.6. *For all parameters p, we have $F^p_{\mathrm{der}} \circ P^p_{\mathrm{der}} = \mathrm{Id}_{\mathrm{Der(Fuzz)}}$. In other words, the following diagrams commute:*



8 Conclusion and Future Work

We have shown that Plurimetric Fuzz extends the Fuzz language by handling L^p distance, using the types of Bunched Fuzz but with classical type judgements. This system can be seen as a subsystem of Bunched Fuzz which satisfies type safety. Among its other benefits are the facts that it includes

subtyping which relates distances L^p and L^q, and it supports recursive types. We have also investigated translations between Plurimetric Fuzz and Fuzz.

We leave as future work the problem of type checking, which we expect to be significantly more difficult than for Fuzz [1] due to the non-linearity of the constraints on sensitivities. If solved, it would allow us to replace Fuzz by Plurimetric Fuzz in [22], and obtain a type system for adaptive differential privacy with respect to vector metrics.

In addition, one may work on improving the sensitivity obtained by typing in Plurimetric Fuzz. One the one hand, we do not know whether a generalisation of the monad elimination rule to any parameter p, which would be finer than the one presented in this paper, is sound. On the other hand, it should be straightforward to extend Plurimetric Fuzz to dependent types, by adapting the DFuzz system presented in [13].

References

[1] Arthur Azevedo de Amorim, Marco Gaboardi, Emilio Jesús Gallego Arias, and Justin Hsu. Really natural linear indexed type checking. In *Proceedings of the 26nd 2014 International Symposium on Implementation and Application of Functional Languages*. Association for Computing Machinery, October 2014. doi:10.1145/2746325.2746335.

[2] Arthur Azevedo de Amorim, Marco Gaboardi, Justin Hsu, Shin ya Katsumata, and Ikram Cherigui. A semantic account of metric preservation. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*. Association for Computing Machinery, January 2017. doi:10.1145/3009837.3009890.

[3] Gilles Barthe, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. Proving differential privacy via probabilistic couplings. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 749–758. ACM, 2016. doi:10.1145/2933575.2934554.

[4] Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. Probabilistic relational reasoning for differential privacy. In John Field and Michael Hicks, editors, *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. ACM, 2012. doi:10.1145/2103656.2103670.

[5] Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. Probabilistic relational reasoning for differential privacy. *ACM Trans. Program. Lang. Syst.*, 35(3):9:1–9:49, 2013. doi:10.1145/2492061.

[6] Gilles Barthe and Federico Olmedo. Beyond differential privacy: Composition theorems and relational logic for f-divergences between probabilistic programs. In *Automata, Languages, and Programming*, pages 49–60. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-39212-2_8.

[7] Olivier Bousquet and André Elisseeff. Stability and generalization. *J. Mach. Learn. Res.*, 2:499–526, 2002. URL: http://jmlr.org/papers/v2/bousquet02a.html.

[8] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, March 2004.

[9] Clément L. Canonne, Gautam Kamath, and Thomas Steinke. The discrete gaussian for differential privacy. In *Advances in Neural Information Processing Systems*, volume 33, pages 15676–15688. Curran Associates, Inc., 2020.

[10] Swarat Chaudhuri, Sumit Gulwani, Roberto Lublinerman, and Sara NavidPour. Proving programs robust. In Tibor Gyimóthy and Andreas Zeller, editors, *SIGSOFT/FSE'11 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-19) and ESEC'11: 13th European Software Engineering Conference (ESEC-13), Szeged, Hungary, September 5-9, 2011*, pages 102–112. ACM, 2011. doi:10.1145/2025113.2025131.

[11] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2006. doi:10.1007/11681878_14.

[12] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3–4):211–407, August 2013. doi:10.1561/0400000042.

[13] Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin C. Pierce. Linear dependent types for differential privacy. *ACM SIGPLAN Notices*, 48(1):357–370, January 2013. doi:10.1145/2480359.2429113.

[14] Arpita Ghosh, Tim Roughgarden, and Mukund Sundararajan. Universally utility-maximizing privacy mechanisms. *SIAM Journal on Computing*, 41(6):1673–1693, 2012. doi:10.1137/09076828X.

[15] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[16] René Gonin and Arthur H. Money. *Nonlinear Lp-Norm Estimation*. Marcel Dekker, Inc., USA, 1989.

[17] Joseph P. Near, David Darais, Chike Abuah, Tim Stevens, Pranav Gaddamadugu, Lun Wang, Neel Somani, Mu Zhang, Nikhil Sharma, Alex Shan, and Dawn Song. Duet: an expressive higher-order language and linear type system for statically enforcing differential privacy. In *Proceedings of the ACM on Programming Languages*, volume 3, pages 1–30, 2019.

[18] Peter W. O'Hearn and David J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, June 1999. doi:10.2307/421090.

[19] Jason Reed and Benjamin 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*. Association for Computing Machinery, September 2010. doi:10.1145/1863543.1863568.

[20] Matías Toro, David Darais, Chike Abuah, Joseph P. Near, Damián Árquez, Federico Olmedo, and Éric Tanter. Contextual linear types for differential privacy. *ACM Transactions on Programming Languages and Systems*, 45(2):1–69, May 2023. doi:10.1145/3589207.

[21] Larry Wasserman and Shuheng Zhou. A statistical framework for differential privacy. *Journal of the American Statistical Association*, 105(489):375–389, March 2010. doi:10.1198/jasa.2009.tm08651.

[22] Daniel Winograd-Cort, Andreas Haeberlen, Aaron Roth, and Benjamin C. Pierce. A framework for adaptive differential privacy. In *Proceedings of the ACM on Programming Languages*, volume 1, pages 1–29. Association for Computing Machinery, August 2017. doi:10.1145/3110254.

[23] June Wunder, Arthur Azevedo de Amorim, Patrick Baillot, and Marco Gaboardi. Bunched fuzz: Sensitivity for vector metrics. In Thomas Wies, editor, *Programming Languages and Systems: ESOP 2023*, pages 451–478, Cham, April 2023. Springer Nature Switzerland. doi:10.1007/978-3-031-30044-8_17.

Appendices

A Operational Semantics

$$\frac{}{* \Downarrow *} \qquad \frac{}{\lambda x.\, b \Downarrow \lambda x.\, b} \qquad \frac{f \Downarrow \lambda x.\, b \quad a \Downarrow v_a \quad b[x \mapsto v_a] \Downarrow v}{f(a) \Downarrow v}$$

$$\frac{a \Downarrow v_a \quad b \Downarrow v_b}{(a, b) \Downarrow (v_a, v_b)} \qquad \frac{c \Downarrow (v_a, v_b) \quad e[x \mapsto v_a][y \mapsto v_b] \Downarrow v}{(\mathbf{let}\ \ (x, y) = c\ \ \mathbf{in}\ \ e) \Downarrow v}$$

$$\frac{e \Downarrow v}{\mathbf{inj}_1\, e \Downarrow \mathbf{inj}_1\, v} \qquad \frac{e \Downarrow \mathbf{inj}_i\, v \quad e_i[x \mapsto v] \Downarrow v_i}{(\mathbf{case}\ \ e\ \ \mathbf{of}\ \ x \mapsto e_1\ \ \mathbf{or}\ \ x \mapsto e_2) \Downarrow v_i}$$

$$\frac{e \Downarrow v}{!e \Downarrow !v} \qquad \frac{b \Downarrow !v_b \quad e[x \mapsto v_b] \Downarrow v}{(\mathbf{let}\ \ !x = b\ \ \mathbf{in}\ \ e) \Downarrow v}$$

$$\frac{e \Downarrow v}{\mathbf{fold}_A\, e \Downarrow \mathbf{fold}_A\, v} \qquad \frac{e \Downarrow \mathbf{fold}_A\, v}{\mathbf{unfold}_A\, e \Downarrow v}$$

$$\frac{e \Downarrow v}{\mathbf{return}\, e \Downarrow (1, v)} \qquad \frac{d \Downarrow (p_i, v_i)_{i \in I} \quad \dfrac{i \in I}{b[x \mapsto v_i] \Downarrow (q_{ij}, w_{ij})_{j \in J}}}{\mathbf{let}\ \ \bigcirc x = d\ \ \mathbf{in}\ \ b \Downarrow (p_i q_{ij}, w_{ij})_{i \in I, j \in J}}$$

Figure 3: Evaluation rules for (Plurimetric) Fuzz

Let us give a more general version of Lemma 4.1, which is easier to prove.

Lemma A.1 (Substitution)**.** *If we have* $(p)\ \Gamma, [x : A]_s \vdash b : B$ *and* $(q)\ \Delta \vdash a : A$*, then the following judgements are derivable:*

- $\big(\min(p, q)\big)\ C^{\min(p,q)}\,(\Gamma; s\Delta) \vdash b[x \mapsto a] : B$*,*
- $\big(\max(p, q)\big)\ c(p, q) \cdot C^{\max(p,q)}\,(\Gamma; s\Delta) \vdash b[x \mapsto a] : B$*.*

Proof. We proceed by induction on the height n of a derivation π of minimal height of the judgement $(p)\ \Gamma, [x : A]_s \vdash b : B$. Let R be the last non-weakening rule applied in π.

Case $n = 1$ **and** $R = (\text{var})$ We have $\Gamma = \emptyset$, $s = 1$ and $A = B$. The term we have to derive a type for is $x[a \mapsto x] = a$. Consider the following two trees:

$$\frac{(q)\ \Delta \vdash a : A}{\big(\min(p, q)\big)\ \Delta \vdash a : A} \geq W \qquad \frac{(q)\ \Delta \vdash a : A}{\big(\max(p, q)\big)\ c(p, q) \cdot \Delta \vdash a : A} \leq W$$

Case $n > 1$ **and** $R = (\text{var})$ The last rule applied in the derivation of π is necessarily a weakening rule. If it is $(\geq W)$, then for some parameter $p_0 \geq p$, some precontext Γ, and some sensitivity $s \geq 1$,

we have

$$\pi = \dfrac{\dfrac{}{(p_0)\ [x : A]_1 \vdash x : A}\ \text{var}}{(p)\ \Gamma, [x : A]_s \vdash x : A} \geq W$$

By what precedes, we can derive the following judgements:

- $\big(\min(p_0, q)\big)\ \Delta \vdash a : A$,
- $\big(\max(p_0, q)\big)\ c(p_0, q) \cdot \Delta \vdash a : A$,

and since $\min(p_0, q) \geq \min(p, q)$, $\max(p_0, q) \geq \max(p, q)$, and $c(p_0, q) \leq c(p, q)$ we can apply the $(\geq W)$ rule to conclude.

If the last rule of π is $(\leq W)$, then we can proceed similarly.

Case $n > 1$ **and** $R = (\otimes I)$ The last rule applied in π is either $(\geq W)$, $(\leq W)$, or $(\otimes I)$. Let us consider the first case. For some parameter $p_0 \geq p$, some precontexts Θ_0, Ξ_0, and Δ_0 such that $C^{p_0}(\Theta_0; \Xi_0) = \Gamma_0$, and some sensitivities s_{01} and s_{02} such that ${s_{01}}^p + {s_{02}}^p = s^p$, we have

$$\pi = \dfrac{\dfrac{\vdots\ \pi_1 \qquad\qquad \vdots\ \pi_2}{(p_0)\ \Theta_0, [x : A]_{s_{01}} \vdash b_1 : B_1 \qquad (p_0)\ \Xi_0, [x : A]_{s_{02}} \vdash b_2 : B_2}}{\dfrac{(p_0)\ \Gamma_0, [x : A]_{s_0} \vdash (b_1, b_2) : B_1 \otimes_p B_2}{(p)\ \Gamma, [x : A]_s \vdash (b_1, b_2) : B_1 \otimes_p B_2} \geq W}$$

There exists two derivations of minimal height π_1' and π_2' with the same conclusion as π_1 and π_2 respectively. Moreover, we have $h(\pi_1') \leq h(\pi_1) < n$, and $h(\pi_2') \leq h(\pi_2) < n$. By induction hypothesis, we can derive the following judgements:

- $\big(\min(p_0, q)\big)\ C^{\min(p_0,q)}(\Theta_0; s_{01}\Delta_0) \vdash b_1[x \mapsto a] : B_1$,
- $\big(\min(p_0, q)\big)\ C^{\min(p_0,q)}(\Xi_0; s_{02}\Delta_0) \vdash b_2[x \mapsto a] : B_2$.

which we combine using the $(\otimes I)$ rule to obtain:

$$\big(\min(p_0, q)\big)\ C^{\min(p_0,q)}\Big(C^{\min(p_0,q)}(\Theta_0; s_{01}\Delta_0)\,; C^{\min(p_0,q)}(\Xi_0; s_{02}\Delta_0)\Big) \vdash (b_1[x \mapsto a], b_2[x \mapsto a]) : B_1 \otimes_p B_2$$

which simplifies to:

$$\big(\min(p_0, q)\big)\ C^{\min(p_0,q)}(\Gamma_0; s_0\Delta_0) \vdash (b_1, b_2)[x \mapsto a] : B_1 \otimes_p B_2$$

and, using the $(\geq W)$ rule, we conclude that the following judgement is derivable:

$$\big(\min(p, q)\big)\ C^{\min(p,q)}(\Gamma; s\Delta) \vdash (b_1, b_2)[x \mapsto a] : B_1 \otimes_p B_2\,.$$

Moreover, by using the rest of the induction hypothesis, we obtain in a similar manner

$$\big(\max(p, q)\big)\ c(p, q) \cdot C^{\max(p,q)}(\Gamma; s\Delta) \vdash (b_1, b_2)[x \mapsto a] : B_1 \otimes_p B_2\,.$$

Other cases The other cases are similar. □

B Proof of the Soundness of the Typing Rules

Lemma B.1. *For all metric spaces X_1, X_2, Y_1, and Y_2, and parameters p, if $f\colon X_1 \to Y_1$ and $g\colon X_2 \to Y_2$ are non-expansive maps, then so is $f \times g\colon X_1 \otimes_p X_2 \to Y_1 \otimes_p Y_2$.*

Proof. Let (x_1, x_2) and (x_1', x_2') be two elements of $X_1 \otimes_p X_2$.

$$\begin{aligned} d_{Y_1 \otimes_p Y_2}\big((f \times g)(x_1, x_2), (f \times g)(x_1', x_2')\big) &= d_{Y_1 \otimes_p Y_2}\big((f(x_1), g(x_2)), (f(x_1'), g(x_2'))\big) \\ &= \sqrt[p]{d_{Y_1}\big(f(x_1), f(x_1')\big)^p + d_{Y_2}\big(g(x_2), g(x_2')\big)^p} \\ &\leq \sqrt[p]{d_{X_1}(x_1, x_1')^p + d_{X_2}(x_2, x_2')^p} \\ &= d_{X_1 \otimes_p X_2}\big((x_1, x_2), (x_1', x_2')\big) \square \end{aligned}$$

Let us show that bind is non-expansive. We first need the following lemmata.

Lemma B.2. *For all finite sequences of positive real numbers $(x_i)_{1 \leq i \leq n}$ and $(y_i)_{1 \leq i \leq n}$, we have*

$$\frac{\sum_{i=1}^n x_i}{\sum_{i=1}^n y_i} \leq \max_{1 \leq i \leq n} \frac{x_i}{y_i}$$

and therefore

$$\left| \ln \frac{\sum_{i=1}^n x_i}{\sum_{i=1}^n y_i} \right| \leq \max_{1 \leq i \leq n} \left| \ln \frac{x_i}{y_i} \right|.$$

Proof. Let us show the first inequality by induction on n.

- If $n = 1$, then the inequality becomes $x_1/y_1 \leq \max\{x_1/y_1\}$ which is true.
- If $n = 2$, then we have

$$\begin{aligned} \frac{\sum_{i=1}^n x_i}{\sum_{i=1}^n y_i} &= \frac{x_1}{y_1 + y_2} + \frac{x_2}{y_1 + y_2} \\ &= \frac{1}{1 + \frac{y_2}{y_1}} \cdot \frac{x_1}{y_1} + \frac{1}{1 + \frac{y_1}{y_2}} \cdot \frac{x_2}{y_2} \\ &= \frac{\frac{1}{y_1}}{\frac{1}{y_1} + \frac{1}{y_2}} \cdot \frac{x_1}{y_1} + \frac{\frac{1}{y_2}}{\frac{1}{y_1} + \frac{1}{y_2}} \cdot \frac{x_2}{y_2}. \end{aligned}$$

  Let $u = \frac{\frac{1}{y_1}}{\frac{1}{y_1} + \frac{1}{y_2}}$ and $v = \frac{\frac{1}{y_2}}{\frac{1}{y_1} + \frac{1}{y_2}}$. We have

$$\begin{aligned} \frac{\sum_{i=1}^n x_i}{\sum_{i=1}^n y_i} &\leq u \cdot \max\left\{\frac{x_1}{y_1}, \frac{x_2}{y_2}\right\} + v \cdot \max\left\{\frac{x_1}{y_1}, \frac{x_2}{y_2}\right\} \\ &= (u + v) \cdot \max\left\{\frac{x_1}{y_1}, \frac{x_2}{y_2}\right\} \end{aligned}$$

  which is the desired inequality since $u + v = 1$.

- If $n \geq 3$ and if the result has been proved up to $n-1$, then we write

$$\begin{aligned}\frac{\sum_{i=1}^n x_i}{\sum_{i=1}^n y_i} &= \frac{x_1 + \sum_{i=2}^n x_i}{y_1 + \sum_{i=2}^n y_i} \\ &\leq \max\left\{\frac{x_1}{y_1}, \frac{\sum_{i=2}^n x_i}{\sum_{i=2}^n y_i}\right\} \\ &\leq \max\left\{\frac{x_1}{y_1}, \max_{2\leq i\leq n} \frac{x_i}{y_i}\right\} \\ &= \max_{1\leq i\leq n} \frac{x_i}{y_i}\,.\end{aligned}$$

Now, let us show the second inequality. Let $X = \ln(\sum_{i=1}^n x_i)$ and $Y = \ln(\sum_{i=1}^n y_i)$ so that we have

$$X - Y = \ln\left(\sum_{i=1}^n x_i\right) - \ln\left(\sum_{i=1}^n y_i\right) = \ln \frac{\sum_{i=1}^n x_i}{\sum_{i=1}^n y_i}\,.$$

- If $X \geq Y$, then $|X-Y| = X - Y$. Moreover, by the first inequality, we have

$$X - Y = \ln \frac{\sum_{i=1}^n x_i}{\sum_{i=1}^n y_i} \leq \ln\left(\max_{1\leq i\leq n} \frac{x_i}{y_i}\right) = \max_{1\leq i\leq n}\left(\ln \frac{x_i}{y_i}\right) \leq \max_{1\leq i\leq n}\left|\ln \frac{x_i}{y_i}\right|.$$

- If $X \leq Y$, then $|X-Y| = Y - X$, and we have

$$Y - X = \ln \frac{\sum_{i=1}^n y_i}{\sum_{i=1}^n x_i} \leq \max_{1\leq i\leq n}\left|\ln \frac{y_i}{x_i}\right| = \max_{1\leq i\leq n}\left|\ln \frac{x_i}{y_i}\right|.$$

In both cases, we get $|X-Y| \leq \max_{1\leq i\leq n} |\ln(x_i/y_i)|$ as desired. □

Lemma B.3. *The following map is non-expansive:*

$$\begin{array}{rccl}\text{bind} & : & \operatorname{Dist} Y \otimes_1 (Y \to_1 \operatorname{Dist} X) & \longrightarrow & \operatorname{Dist} X \\ & & (\mu, f) & \longmapsto & t \mapsto \sum_{s\in Y} f(s)(t)\,\mu(t)\,.\end{array}$$

We present an elementary proof of this result (which also follows from the work of Barthe and Olmedo [6]).

Proof. Let μ and μ' be two distributions over Y and let f and f' be two maps from Y to $\operatorname{Dist} X$.

For all $t \in X$, we have

$$\begin{aligned}\left|\ln \frac{\sum_{s\in Y} f(s)(t)\mu(s)}{\sum_{s\in Y} f'(s)(t)\mu'(s)}\right| &\leq \max_{s\in Y}\left|\ln \frac{f(s)(t)\mu(s)}{f'(s)(t)\mu'(s)}\right| \\ &\leq \max_{s\in Y}\left(\left|\ln \frac{\mu(s)}{\mu'(s)}\right| + \left|\ln \frac{f(s)(t)}{f'(s)(t)}\right|\right).\end{aligned}$$

Therefore, we have

$$\max_{t\in X}\left|\ln \frac{\sum_{s\in Y} f(s)(t)\mu(s)}{\sum_{s\in Y} f'(s)(t)\mu'(s)}\right| \leq \max_{s\in Y}\left|\ln \frac{\mu(s)}{\mu'(s)}\right| + \max_{s\in Y}\max_{t\in X}\left|\ln \frac{f(s)(t)}{f'(s)(t)}\right|$$

which is equivalent to the desired inequality:

$$\begin{aligned}d_{\operatorname{Dist} X}\big(f^\dagger(\mu), f'^\dagger(\mu')\big) &\leq d_{\operatorname{Dist} Y}(\mu,\mu') + \max_{s\in Y} d_{\operatorname{Dist} X}(f(s), f'(s)) \\ &= d_{\operatorname{Dist} Y}(\mu,\mu') + d_{Y\to_1 \operatorname{Dist} X}(f, f') \\ &= d_{\operatorname{Dist} Y\otimes_1 (Y\to_1 \operatorname{Dist} X)}\big((\mu, f), (\mu', f')\big)\,.\end{aligned}$$ □

C Proof of the Soundness of the Rules for Primitive Operations

Remark 6. One may notice that if we use recursive types to define Nat as $\mu\alpha.\ \mathsf{Unit} \oplus \alpha$, then the following implementation of $(+)$ is non-expansive for all p rather than $c(1,p)$-sensitive:

Listing 2: Non-expansive implementation of the addition

```
let rec (+) n m = match n with injl () -> m | injr k -> injr (k + m)
```

However, in this setting, the sensitivity is calculated with respect to the following distance: $d_{\mathsf{Nat}}(m,n)$ equals 0 if $m = n$, and ∞ otherwise, rather than the usual distance on $\mathbf{N}$. This justifies the introduction of Nat as a primitive type and of $(+)$ as a primitive.

Lemma C.1. *The following rules are sound with respect to the denotational semantics:*

$$\frac{(p)\ \Gamma \vdash x : A}{(p)\ \infty \cdot \Gamma \vdash \{x\} : \mathsf{Set}(A)}\ \mathsf{Set} \qquad \frac{(p)\ \Gamma \vdash n : \mathsf{Nat}}{(p)\ 2\Gamma \vdash \{n\} : \mathsf{Set}(\mathsf{Nat})}\ \mathsf{Set}_{\mathsf{Nat}}$$

Proof. For all set X, the sensitivity of the map $x \mapsto \{x\}$ is bounded by ∞. Therefore, on can soundy introduce the following rule:

$$\frac{}{(p)\ \emptyset \vdash \lambda x.\ \{x\} : !_\infty A \multimap_1 \mathsf{Set}(A)}$$

which is equiderivable with the (Set) rule.

If $X = \mathbf{N}$, then the map $n \mapsto \{n\}$ is 2-sensitive. Indeed for n and n' in $\mathbf{N}$,

- if $n = n'$, then $\{n\} = \{n'\}$ and therefore $d_{\mathsf{Set}(\mathbf{N})}(\{n\},\{n'\}) = 0$;
- otherwise, we have $d_{\mathsf{Set}(\mathbf{N})}(\{n\},\{n'\}) = 2$, and $d_{\mathbf{N}}(n,n') \geq 1$.

In both cases, we get the inequality $d_{\mathsf{Set}(\mathbf{N})}(\{n\},\{n'\}) \leq 2 \cdot d_{\mathbf{N}}(n,n')$, and for all parameters p, we can soundly introduce the following rule:

$$\frac{}{(p)\ \emptyset \vdash \lambda n.\ \{n\} : !_2\mathsf{Nat} \multimap_p \mathsf{Set}(\mathsf{Nat})}$$

which is equiderivable with the $(\mathsf{Set}_{\mathsf{Nat}})$ rule. □

Remark. Given a metric space X containing at least one limit point (such as $\mathbf{R}$ with the usual distance), the map $x \mapsto \{x\}$ is ∞-sensitive, and the factor ∞ above is optimal.

D Derivation of the Type of the Neighbour Classification Algorithm

First, by applying the tensor-elimination and arrow-introduction rules, we can show that the helper functions `get_pos` and `get_label` have type $\mathsf{Row} \multimap_1 \mathsf{Point}$ and $\mathsf{Row} \multimap_1 \mathsf{Bool}$ respectively.

Then we type the three anonymous functions that appear in our implementation:

- `fun r -> get_label r = l` has type $\mathsf{Row} \to_1 \mathsf{Bool}$ in the context $[l : \mathsf{Label}]_\infty$;
- `fun r -> distance (0, 0) (get_pos r)` has type $\mathsf{Row} \multimap_1 \mathsf{Real}$ as the Euclidean distance `distance` has type $\mathsf{Point} \multimap_1 \mathsf{Point} \multimap_1 \mathsf{Real}$;
- `fun acc x -> acc + weight x` has type $\mathsf{Real} \multimap_1 \mathsf{Real} \multimap_1 \mathsf{Real}$.

This way, we can show that `score` has the following type $\mathsf{Label} \to_1 \mathsf{Database} \multimap_1 \mathsf{Real}$ and we can apply the `exp_noise` function to conclude.

E Translation mappings

Below are the typing rules for the & connective that we use in the translation of Fuzz to Plurimetric Fuzz.

$$\frac{(p)\ \Gamma \vdash a : A \qquad (p)\ \Gamma \vdash b : B}{(p)\ \Gamma \vdash (a, b) : A \,\&\, B}\ \&I \qquad \frac{(p)\ \Gamma \vdash c : A \,\&\, B}{(p)\ \Gamma \vdash \pi_1(c) : A}\ \&E_{\triangleleft} \qquad \frac{(p)\ \Gamma \vdash c : A \,\&\, B}{(p)\ \Gamma \vdash \pi_2(c) : B}\ \&E_{\triangleright}$$

Lemma E.1. *For all precontexts Γ, sensitivities s and parameters p, we have:*

- $s^{1/p} \cdot P^p_{\mathrm{ctx}}(\Delta) = P^p_{\mathrm{ctx}}(s\Delta)$;
- $s^p \cdot F^p_{\mathrm{ctx}}(\Delta) = F^p_{\mathrm{ctx}}(s\Delta)$.

Proof. By induction on the structure of Δ. □

Lemma E.2. *For all precontexts Γ and Δ, for all sensitivities s and parameters p, we have the following equality: $C^p\left(P^p_{\mathrm{ctx}}(\Gamma); s^{1/p} \cdot P^p_{\mathrm{ctx}}(\Delta)\right) = P^p_{\mathrm{ctx}}(\Gamma + s\Delta)$.*

Proof. Let us prove this equality by induction on the structure of Γ.

- If $\Gamma = \emptyset$, then the equality becomes $s^{1/p} \cdot P^p_{\mathrm{ctx}}(\Delta) = P^p_{\mathrm{ctx}}(s\Delta)$.
- If $\Gamma = \Gamma_0, [x : A]_r$, then we write $\Delta = \Delta_0, [x : A]_t$ (with t being possibly zero) and we have

$$\begin{aligned}
C^p\left(P^p_{\mathrm{ctx}}(\Gamma); s^{1/p} \cdot P^p_{\mathrm{ctx}}(\Delta)\right) &= C^p\left(P^p_{\mathrm{ctx}}(\Gamma_0, [x : A]_r); s^{1/p} \cdot P^p_{\mathrm{ctx}}(\Delta_0, [x : A]_t)\right) \\
&= C^p\left(P^p_{\mathrm{ctx}}(\Gamma_0, [x : A]_r); P^p_{\mathrm{ctx}}(s\Delta_0, [x : A]_{st})\right) \\
&= C^p\left(P^p_{\mathrm{ctx}}(\Gamma_0), [x : P^p_{\mathrm{type}}(A)]_{r^{1/p}}; P^p_{\mathrm{ctx}}(s\Delta_0), [x : P^p_{\mathrm{type}}(A)]_{(st)^{1/p}}\right) \\
&= C^p\left(P^p_{\mathrm{ctx}}(\Gamma_0); s^{1/p} \cdot P^p_{\mathrm{ctx}}(\Delta_0)\right), [x : P^p_{\mathrm{type}}(A)]_{\sqrt[p]{r+st}} \\
&= P^p_{\mathrm{ctx}}(\Gamma_0 + s\Delta_0), [x : P^p_{\mathrm{type}}(A)]_{\sqrt[p]{r+st}} \\
&= P^p_{\mathrm{ctx}}(\Gamma_0 + s\Delta_0, [x : A]_{r+st}) \\
&= P^p_{\mathrm{ctx}}(\Gamma + s\Delta)
\end{aligned}$$

  as desired. □

Lemma E.3. *For all precontexts Γ and Δ, for all sensitivities s, for all parameters p and q such that $p \geq q$, we have the following inequality: $s^p F^p_{\mathrm{ctx}}(\Gamma) + F^p_{\mathrm{ctx}}(\Delta) \leq F^p_{\mathrm{ctx}}\big(C^q(\Gamma; s\Delta)\big)$.*

Proof. Let us prove this inequality by induction on the structure of Γ.

- If $\Gamma = \emptyset$, then the inequality becomes $s^p F^p_{\mathrm{ctx}}(\Delta) \leq F^p_{\mathrm{ctx}}(s\Delta)$.
- If $\Gamma = \Gamma_0, [x : A]_r$, then we write $\Delta = \Delta_0, [x : A]_t$ (with t being possibly zero) and we have

$$\begin{aligned}
s^p F^p_{\mathrm{ctx}}(\Gamma) + F^p_{\mathrm{ctx}}(\Delta) &= s^p F^p_{\mathrm{ctx}}(\Gamma_0, [x : A]_r) + F^p_{\mathrm{ctx}}(\Delta_0, [x : A]_t) \\
&= s^p\big(F^p_{\mathrm{ctx}}(\Gamma_0), [x : F^p_{\mathrm{type}}(A)]_{r^p}\big) + F^p_{\mathrm{ctx}}(\Delta_0), [x : F^p_{\mathrm{type}}(A)]_{t^p} \\
&= s^p F^p_{\mathrm{ctx}}(\Gamma_0) + F^p_{\mathrm{ctx}}(\Delta_0), [x : F^p_{\mathrm{type}}(A)]_{(rs)^p + t^p} \\
&\leq F^p_{\mathrm{ctx}}\big(C^q(\Gamma_0; s\Delta_0)\big), [x : F^p_{\mathrm{type}}(A)]_{(rs)^p + t^p} \\
&\leq F^p_{\mathrm{ctx}}\big(C^q(\Gamma_0; s\Delta_0)\big), [x : F^p_{\mathrm{type}}(A)]_{\left((rs)^q + t^q\right)^{p/q}} \\
&\leq F^p_{\mathrm{ctx}}\left(C^q(\Gamma_0; s\Delta_0), [x : F^p_{\mathrm{type}}(A)]_{\left((rs)^q + t^q\right)^{1/q}}\right) \\
&\leq F^p_{\mathrm{ctx}}\big(C^q(\Gamma; s\Delta)\big)
\end{aligned}$$

  as desired. □