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

Victor Sannier
Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRISTAL, F-59000 Lille, France

Patrick Baillot
Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRISTAL, F-59000 Lille, France

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 $L^\infty$), 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 graded linear type system – inspired by Bunched Fuzz [27] – 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.

2012 ACM Subject Classification Theory of computation → Type theory; Theory of computation → Linear logic; Security and privacy → Logic and verification

Keywords and phrases type system, linear logic, sensitivity, vector metrics, differential privacy, lambda-calculus, functional programming, denotational semantics

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.12

Related Version Full Version: https://hal.science/hal-04514677

Acknowledgements The authors would like to thank Arthur Azevedo de Amorim for his valuable comments, in particular on denotational semantics.

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 smallest positive real $s$ such that $d_Y(f(x), f(x')) \leq s \cdot d_X(x, x')$, 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 [9, 12]. In particular, sensitivity is a key notion for differential privacy [14, 15], 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 dataset by perturbing 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, some approaches in the programming languages community have been developed to assist programmers. They can be categorised into two classes: those based on Hoare logics [6, 7, 5], which are interactive and suitable for verifying mechanism implementations, and those based on type systems [23, 16], which are automatisable and well-suited for verifying functional programs that compose mechanisms. Moreover recent works [21, 24] 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 typing 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 [23] has shown how ideas from linear logic [18, 19] 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 $L^\infty$ 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, in the literature on optimisation and statistical applications [10, 20], $L^p$ distances with $1 \leq p \leq +\infty$ have been used to advantage. For this reason, wunder et al. [27] have 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 typing judgements inspired by the logic of Bunched Implications [22], where typing contexts have 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 will discuss why Bunched Fuzz does not satisfy the desired properties with respect to an operational semantics, and we will design a type system for $L^p$ metrics inspired by Bunched Fuzz 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 keep the same type language as Bunched Fuzz, but we consider a system of rules that uses standard judgements with list contexts, we call this system Plurimetric Fuzz. As an additional benefit, we will define (partial) translation mappings 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 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 enjoys the subject reduction property (Theorem 5.2), and that it is sound with respect to its denotational semantics (Theorem 4.15). 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 the 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, or metric space for short, is a pair $(X, d)$ where $X$ is a set and $d : X \times X \rightarrow [0, \infty]$ is a function such that for all $x, y, z \in X$:

1. $d(x, y) = 0$ if $x = y$;
2. $d(x, y) = d(y, x)$.

Note that we do not require the triangle inequality to hold.

In this paper, we are interested in a family of metrics over $\mathbb{R}^d$, which are defined as follows, and related by the inequalities of Lemma 2.3.

► Definition 2.2. For all parameter $p \geq 1$ and for all vectors $x = (x_1, \ldots, x_d)$ and $y = (y_1, \ldots, y_d)$ in $\mathbb{R}^d$, we define the $L^p$-distance or the vector metric of parameter $p$ between $x$ and $y$ by $d_p(x, y) = (\sum_{i=1}^{d} |x_i - y_i|^p)^{1/p}$.

► Lemma 2.3. For all parameters $p$ and $q$ such that $1 \leq p, q \leq \infty$, let $c(p, q) = 2^{1/p - 1/q}$. If $p \leq q$, then 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 2.4. A map $f$ between two metric spaces $(X, d_X)$ and $(Y, d_Y)$ is said to be $s$-sensitive, or $s$-Lipschitz continuous, for $s \in [0, \infty]$ if for all points $x$ and $x'$ in $X$, we have $d_Y(f(x), f(x')) \leq s \cdot d_X(x, x')$. 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 2.5. To perform operations on sensitivities, we extend addition to possibly infinite reals in a straightforward way and multiplication in the same way as [3, Section 2], that is such that $s \cdot \infty$ equals $\infty$, and $\infty \cdot s$ equals 0 if $s = 0$ and $\infty$ otherwise. Note that this operation is not commutative, see [24, 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 2.6. Let $f$ be a differentiable function from $\mathbb{R}$ to $\mathbb{R}$ such that for all $x \in \mathbb{R}$, 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 [14, 15] 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 2.7. A probabilistic algorithm $A$ endowed with an adjacency relation is said to be $(\epsilon, \delta)$-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 codom($A$), 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 [25].

In practice, $X$ will 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 $\mathbb{R}^d$ and endowed with a vector metric such as the Manhattan distance ($p = 1$) or the Euclidean distance ($p = 2$).
In order to guarantee differential privacy, 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 statements.

Let $f$ be a vector-valued function from a metric space $(X,d_X)$ to $\mathbb{R}^d$. 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 $x \mapsto \frac{1}{2b} \cdot e^{-|x|/b}$, for $x$ in $\mathbb{R}$.

12.4 Plurimetric Fuzz

**Theorem 2.8** (Laplace Mechanism [15, Theorem 3.6]). If $\Delta_1f$ is finite, then for all positive real number $\epsilon$, the function $x \mapsto f(x) + (\text{Lap}(\Delta_1f/\epsilon), \ldots, \text{Lap}(\Delta_1f/\epsilon))$ is $\epsilon$-differentially private.

However, in some cases, we may prefer to add Gaussian noise instead of Laplace noise. This way, the noise added to protect privacy is of the same type as other sources of perturbation in the original data. 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 [15, Section 3.5.3]. To do so, we need to bound the $L^2$-sensitivity of $f$.

**Theorem 2.9** (Gaussian mechanism [15, Theorem 3.22]). If $\Delta_2f$ is finite, then for all positive real numbers $\epsilon$ and $\delta$, if $\sigma > \sqrt{2\ln(5/\delta)} \cdot \Delta_2f/\epsilon$, then the function $x \mapsto f(x) + (\mathcal{N}(0,\sigma^2), \ldots, \mathcal{N}(0,\sigma^2))$ is $(\epsilon, \delta)$-differentially private.

**Remark 2.10.** In this paper, we will only consider discrete probability distributions, but the above two theorems can be adapted to this setting [17, 11].

### 2.3 Type Systems for Bounding Sensitivity

Reed and Pierce have introduced the Fuzz type system [23] based on the fact that $L^1$-sensitivity can be viewed as an affine resource (in the sense of linear logic [18, 19]). 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:

$$
\Gamma \vdash a : A \quad \Delta \vdash b : B \\
\Gamma + \Delta \vdash (a,b) : A \otimes B \\
\frac{\Delta \vdash e : A \otimes B}{\Gamma + s\Delta \vdash (\text{let } (x,y) = e \text{ in } c) : C}
$$

where the sum of two contexts is the result of adding the sensitivities of the involved variables.

It was subsumed by Bunched Fuzz [27], 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 $(-\circ_p)_{p \in [1,\infty]}$. Moreover contexts $\Gamma$ are no longer represented as lists, but as trees (or bunches).

We reproduce the tensor rules below:

$$
\Gamma \vdash a : A \quad \Delta \vdash b : B \\
\Gamma_p \Delta \vdash (a,b) : A \circ_p B \\
\frac{\Delta \vdash e : A \circ_p B}{\Gamma(s\Delta) \vdash (\text{let } (x,y) = e \text{ in } c) : C}
$$

where $\Gamma(\Delta)$ denotes a composite bunch formed by substituting the bunch $\Delta$ into another bunch $\Gamma(s)$, which features a unique, distinguished hole $*$. Let us give more precise statements.

**Remark 2.11.** We write $b[a/x]$ 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,

$$
\frac{\Gamma(\Delta_p \Delta') \vdash a : A \quad \Delta \approx \Delta'}{\Gamma(\text{Contr}(p; \Delta; \Delta')) \vdash a[\text{vars } \Delta/ \text{vars } \Delta'] : A}
$$
where Contr($p; \Gamma; \Delta$) is defined by induction on the structure of $\Gamma$ by the following equations, and where we write $\cdot$ for the sensitivity scaling operation.

$$\text{Contr}(p; 0; 0) \equiv 0$$

$$\text{Contr}(p; [x : A], [y : A]_s) \equiv [x : A]_{\psi(p) + s}$$

$$\text{Contr}(p; \Gamma_1 \otimes \Gamma_2; \Delta_1 \otimes \Delta_2) \equiv 2^{1/p - 1/q} \cdot (\text{Contr}(p; \Gamma_1; \Delta_1) \otimes \text{Contr}(p; \Gamma_2; \Delta_2))$$

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

**Theorem 2.12 ([27, Theorem 7]).** Given a derivation $\pi$ proving $\Gamma \vdash a : A$, the function $[\pi] : (\Gamma) \to [A]$ is non-expansive.

However, 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 $\Delta(s\Gamma) \not\vdash b[a/x] : B$.

**Proof.** To see this, let us look at the following example. Here, we use the algorithmic approach to the rules, which means we systematically apply a contraction after each typing rule:

$$\begin{array}{c}
\text{var} \\
\var \\
\cap I
\end{array}
\quad
\begin{array}{c}
\text{var} \\
\var \\
\otimes I
\end{array}
\quad
\begin{array}{c}
\text{var} \\
\var \\
\ominus I
\end{array}
$$

If we could substitute $(+)(a, b)$ for $x$ in the derivation
$$\begin{array}{c}
[x : \text{Nat}]_{s_2} \vdash x : \text{Nat}
\end{array}
$$
we would obtain $[a : \text{Nat}]_{\sqrt{s_2}} \vdash (+(a, b), (+(a, b)) : \text{Nat} \otimes \text{Nat}$, which is not derivable. Indeed, a derivation of this judgement would have the following shape:

$$\begin{array}{c}
[a : \text{Nat}]_{r_a} \vdash (a, b) : \text{Nat}
\end{array}
$$

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

**Claim 2.13.** Bunched Fuzz doesn’t meet the subject reduction property when it is given a standard operational semantics similar to that of Fuzz (see Figure 3).

In addition, the failure to satisfy the substitution property implies that we cannot meaningfully state certain properties regarding denotational semantics (see Section 5 for the metatheoretical properties our type system enjoys). This includes the assertion, using the notations above, that $[b[a/x]]$ is equal to $[b]$ when partially applied to $a$, as the first term may not have a valid derivation, and therefore a well-defined interpretation.

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

In a nutshell we will consider the terms of Fuzz, that is to say an extended $\lambda$-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: $A, B, \cdots ::= \text{Unit} \mid A \oplus B \mid \mu\alpha. \ A \mid \Box A \mid !\alpha. \ A \mid A \otimes_p B \mid A \sim_p B$. We write $\text{Bool}$ for the type $\text{Unit} \oplus \text{Unit}$; $\text{List}_p(A)$ for the iso-recursive type $\mu\alpha. \text{Unit} \oplus (A \otimes_p \alpha)$; and $\bigotimes_{p=1}^{d} A = A \oplus_p \cdots \oplus_p A$.

On the other hand, the terms of the language are defined by the following grammar, for $c \in \text{Const}$, $x, y \in \text{Var}$ and $A \in \text{Typ}$:

\[
\begin{align*}
  &a, b, c, d, e, f, \ldots ::= \ast \mid c \mid x \mid (a, b) \mid \text{let } (x, y) = e \mid \pi_i e \mid \lambda x. e \mid f e \\
  &\mid \text{inj}_e \mid \text{inj}_2 e \mid \text{case } e \text{ of } x \Rightarrow a \text{ or } y \Rightarrow b \mid e \mid \text{let } !x = e \mid \text{in } b \\
  &\mid \text{fold}_A e \mid \text{unfold}_e \mid \text{return } e \mid \text{let } \bigcirc x = e \mid \text{in } b
\end{align*}
\]

\[\text{Lemma 3.3.} \quad \text{If } \Gamma + \Delta \text{ then } \Gamma \leq \Delta \]

3.2 Precontexts and Contexts

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

\[\text{Definition 3.2.} \quad \text{Two precontexts } \Gamma \text{ and } \Delta \text{ are said to be compatible if they do not assign different types to the same variable. The } p\text{-contraction of two compatible precontexts } \Gamma \text{ and } \Delta \text{ is defined by induction on the structure of } \Gamma \text{ by the following equations:}
\]

\[
\begin{align*}
  &C^p (\emptyset; \Delta) \overset{\text{def}}{=} \Delta \\
  &C^p ([x : A]_r, \Gamma; \Delta) \overset{\text{def}}{=} [x : A]_r C^p (\Gamma; \Delta) \quad \text{if } x \notin \Delta \\
  &C^p ([x : A]_r, \Gamma; [x : A]_s, \Delta) \overset{\text{def}}{=} [x : A] [\varphi_{r \leftarrow s}] p, C^p (\Gamma; \Delta)
\end{align*}
\]

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

\[\text{Lemma 3.3.} \quad \text{For all precontexts } \Gamma \text{ and } \Delta, \text{ and all parameters } p:
\]

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

\[\text{Definition 3.4.} \quad \text{For any two precontexts } \Gamma \text{ and } \Delta, \text{ we write } \Gamma \leq \Delta \text{ if every variable of } \Gamma \text{ also occurs in } \Delta, \text{ and with greater or equal sensitivity.}
\]

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 $(p, \Gamma)$ written $(p) \Gamma$.
this reason, we may simply write by choosing other values of Fuzz rules when Observe that, as range over contexts, We omit the type constructor, and encode it with (see Lemma 2.3). In addition, as a particular case W constructor can be encoded by For (see Figure 1 Typing Rules for Plurimetric Fuzz.

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 \( \rightarrow_p \) type constructor, and encode it with \( \Rightarrow_p \) and \( !_\infty \) as follows: \( A \rightarrow_p B \equiv !_\infty A \Rightarrow_p B \). Similarly, the & constructor can be encoded by \( \otimes_\infty \) like in [27, 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 \( c(p, q) = 2^{1/p - 1/q} \) (see Lemma 2.3). In addition, as a particular case of these rules, for all parameters \( p \) and \( q \), from \( (p) \) \( \emptyset \vdash a : A \) we can derive \( (q) \emptyset \vdash a : A \). For this reason, we may simply write \( \vdash a : A \).
Set \otimes 12:8 Plurimetric Fuzz

\[ \vdash n : \text{Nat} \quad \vdash r : \text{Real} \]

\[ (p) \quad \Gamma, x : N \vdash c(1, p) : C^p (\Gamma; \Delta) \vdash x + y : N \]

\[ (p) \quad k \Gamma, k \in N \quad N \in \{ \text{Nat}, \text{Real} \} \quad \vdash x : A \]

\[ (p) \quad \infty \cdot \Gamma \vdash \{ x \} : \text{Set}(A) \]

\[ \vdash \text{setfilter} : (A \to_p \text{Bool}) \to_p \text{Set}(A) \to_p \text{Set}(A) \]

\[ \vdash \text{setmap} : (A \to_p B) \to_p \text{Set}(A) \to_p \text{Set}(B) \]

\[ \vdash \text{setfold} : (A \to_p B \to_p B) \to_p \text{Set}(A) \to_p \text{Set}(B) \]

Figure 2 Typing Rules for Primitive Operations.

Remark 3.5. As shown in [23, 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:

\[ Y \overset{\text{def}}{=} \lambda x. \lambda a. f (\langle \text{unfold}_A x \rangle a) \langle \text{fold}_A x \rangle a \]

where \( A_0 \overset{\text{def}}{=} \mu \alpha (\alpha \to_p (A \to_p B)) \), and \( Y : ((A \to_p B) \to_p (A \to_p B)) \to_p (A \to_p B) \).

We can extend the type system to handle primitive operations on natural and real numbers, as well as on sets, having extending the syntax of types and terms accordingly in Figure 2.

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 \( (p) \quad \Gamma \vdash e : A \otimes_p B \), we can derive \( (q) \quad \Gamma \vdash e : A \otimes_q B \).

\[ \vdash e : A \otimes_p B \]

\[ \vdash (\text{let } (x, y) = e \text{ in } (x, y)) : A \otimes_p B \]

Example 3.6. As an example, say we want to compose a function \( f : \text{Real} \to_1 \text{Real} \otimes_2 \text{Real} \) with a function \( g : \text{Real} \otimes_1 \text{Real} \to_1 \text{Real} \), both typable in an empty context. We can first apply \( f \) to an input \( x \), and then coerce the result to obtain the judgement \( (1) \quad [x : \text{Real}]_{\sqrt{2}} \vdash e : \text{Real} \otimes_3 \text{Real} \) for some term \( e \) which is semantically equivalent to the term \( f(x) \). At this point, we can apply \( g \) to \( e \), and use the \((\text{II})\) and \((\text{I})\) rules to derive the judgement \( (1) \quad \emptyset \vdash h : !\sqrt{2} \text{Real} \to_1 \text{Real} \) for some term \( h \) which behaves like \( g \circ f \).
We also introduce a denotational semantics by interpreting types as metric spaces and type operations on metric spaces. We will see in the Section 5 that this semantics enjoys the desired properties such as subject reduction (also known as type preservation).

### 4.2 Denotational semantics

We also introduce a denotational semantics by interpreting types as metric spaces and type derivations as non-expansive maps, following the denotational semantics of Bunched Fuzz [27].

#### Operations on metric spaces

**Definition 4.1.** 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 $!_s X \overset{def}{=} (X, s \cdot d_X)$. Moreover, the $p$-tensor product $X \otimes_p Y$ of $X$ and $Y$ is the set $X \times Y$ endowed with

$$
d_{X \otimes_p Y}((x,y),(x',y')) \overset{def}{=} \sqrt{d_X(x,x')^p + d_Y(y,y')^p}.
$$

(3)

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

$$
d_{X \rightarrow_p Y}(f,f') \overset{def}{=} \inf \left\{ r \geq 0 \mid \forall x,x' \in X, d_Y(f(x),f'(x'))^p \leq r^p + d_X(x,x')^p \right\};
$$

(4)

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

$$
d_{A_1 \uplus A_2}(e,e') \overset{def}{=} \begin{cases} d_e(e,e') & \text{if } e,e' \in A_1 \\ \infty & \text{otherwise}. \end{cases}
$$

(5)

Given two maps $f : X \rightarrow Z$ and $g : Y \rightarrow Z$, the coproduct $[f,g] : X \uplus Y \rightarrow Z$ of $f$ and $g$ is the map defined by $[f,g](i_1(x)) \overset{def}{=} f(x)$, and $[f,g](i_2(y)) \overset{def}{=} g(y)$.  

---

**Figure 3** Evaluation rules for (Plurimetric) Fuzz.
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 \( \text{Ev}: (X \rightarrow_p Y) \otimes_p X \rightarrow Y \) is non-expansive. Moreover, we have \( (X \otimes_p Y) \otimes_p Z \simeq (X \otimes_p Z) \oplus (Y \otimes_p Z) \).

We also define probability distributions over metric spaces.

**Definition 4.2.** A discrete probability distribution over a metric space with countable support \( X \) is a function \( \mu : X \rightarrow [0, 1] \) such that \( \sum_{x \in X} \mu(x) = 1 \). We write \( \text{Dist}(X) \) for the set of such distributions endowed with the following distance, parametrised by a positive real \( \epsilon_0 \):

\[
\max \text{div}(\mu, \mu') \overset{\text{def}}{=} \frac{1}{\epsilon_0} \max_{x \in X} \left| \ln \frac{\mu(x)}{\mu'(x)} \right|
\]

with the convention that \( 0/0 \overset{\text{def}}{=} 1 \) and \( |\ln(0|x)\| \overset{\text{def}}{=} |\ln(x/0)| \overset{\text{def}}{=} \infty \) for all \( x > 0 \). We write \( \delta_x \) for the Dirac distribution at \( x \). Recall that \( \delta_x(x) = 1 \) and \( \delta_x(x') = 0 \) for all \( x \neq x' \).

**Lemma 4.3.** A non-expansive map from \( X \) to \( \text{Dist}(Y) \) is exactly an \( \epsilon_0 \)-differentially private random map from \( X \) to \( Y \).

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

**Definition 4.4.** The Kleisli extension \( f^\dagger : \text{Dist}(X) \rightarrow \text{Dist}(Y) \) of a map \( f : X \rightarrow \text{Dist}(Y) \) is defined by the following formula: \( f(\mu)(y) \overset{\text{def}}{=} \sum_{x \in X} \mu(x) f(x)(y) \).

**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:

\[
\begin{align*}
\text{[Unit]} & \overset{\text{def}}{=} \{*, 0\} ; \\
\text{[Nat]} & \overset{\text{def}}{=} (\mathbb{N}, (m, n) \mapsto |m - n|) ; \\
\text{[Real]} & \overset{\text{def}}{=} (\mathbb{R}, (x, y) \mapsto |x - y|) ; \\
\text{[
\text{A}_p B]} & \overset{\text{def}}{=} \text{Dist}(\mathbb{R}) ; \\
\text{[A] } & \overset{\text{def}}{=} \{ P_{\text{finite}}(\mathbb{N}), \text{card}(- \Delta -) \},
\end{align*}
\]

where \( \Delta \) is the symmetric difference on sets. Next, we define the interpretation of contexts: \( [(p) \emptyset] \overset{\text{def}}{=} \{ \ast \} \) and \( [(p) \Gamma, x : A] \overset{\text{def}}{=} [(p) \Gamma] \otimes_p [A] \). Derivations are seen as non-expansive maps between metric spaces. More precisely, if \( \pi \) is a derivation whose last rule is \( R \), we write \( \pi_{\epsilon} \) for its premise whose conclusion has term \( \epsilon \), and \( \Gamma \) and \( \Delta \) for the contexts involved. Moreover, we write \( \dot{\rightarrow} \) for the currying map; \( \text{Ev} \) 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 \( I_q \) for the natural map from \( [(p) \Gamma] \) to \( [(q) \Gamma] \) when \( R \) is binary or ternary; \( D_p \) for the diagonal-like map from \( [(p) C^p (\Gamma; \Delta)] \) to \( [(p) \Gamma] \otimes_p [(p) \Delta] \). We omit isomorphisms when they are clear from the context.

\[
\begin{align*}
\text{(Unit)} & \quad \pi \overset{\text{def}}{=} \text{ Const}; \\
\text{(var)} & \quad \pi \overset{\text{def}}{=} \text{Id}_{[A]} ; \\
\text{(\otimes I)} & \quad \pi \overset{\text{def}}{=} (\pi_a \times \pi_b) \circ D_p \\
\text{(\otimes E)} & \quad \pi \overset{\text{def}}{=} \text{Ev} \circ (\pi_j \times \pi_a) \circ D_p \\
\text{($\otimes$ E)} & \quad \pi \overset{\text{def}}{=} \text{Id} \times r \cdot [\pi_{\epsilon}] \circ D_p
\end{align*}
\]
Soundness of Core Plurimetric Fuzz

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

\begin{itemize}
  \item **Proposition 4.5 (Soundness).** If \( \pi \) is a Core Plurimetric Fuzz derivation of \( (p) \Gamma \vdash a : A \), then \( \llbracket \pi \rrbracket \) is a non-expansive map from \( \llbracket (p) \Gamma \rrbracket \) to \( \llbracket A \rrbracket \).

Properties we use repeatedly in the proof of this result are summarised in the following lemmata. See Appendix A.1 for more details.

\begin{itemize}
  \item **Lemma 4.6.** For all precontexts \( \Gamma \), and reals \( p \geq 1 \) and \( s \geq 0 \), we have \( \llbracket (p) s \Gamma \rrbracket = s \llbracket (p) \Gamma \rrbracket \).
  \item **Lemma 4.7.** For all metric spaces \( X_1, X_2, Y_1, \) and \( Y_2 \), and parameters \( p \), if \( f : X_1 \to Y_1 \) and \( g : X_2 \to Y_2 \) are non-expansive maps, then so is \( f \times g : X_1 \otimes_p X_2 \to Y_1 \otimes_p Y_2 \).
  \end{itemize}

\begin{proof}
The function \((x, y) \mapsto e^{-p(x+y)}\) is increasing in both arguments over \( \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \).
\end{proof}

\begin{itemize}
  \item **Lemma 4.8 ([27, Proposition 6]).** 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: \( X \otimes_p Y \to X \otimes_q Y \) and \( !_{\llbracket (p, q) \rrbracket} \llbracket X \otimes_p Y \rrbracket \to \llbracket X \otimes_q Y \rrbracket \).
  \end{itemize}

\begin{proof}
By induction on \( \Gamma \), using the fact that for all metric spaces \( X \), and sensitivities \( r \) and \( s \), we have a non-expansive map from \( !_{\llbracket \Delta \rrbracket} \llbracket X \rrbracket \) to \( !_{\llbracket \Delta \rrbracket} \llbracket X \otimes_p X \rrbracket \) given by \( x \mapsto (x, x) \).
\end{proof}

\begin{itemize}
  \item **Lemma 4.9.** For all compatible precontexts \( \Gamma \) and \( \Delta \), the diagonal-like map \( D_p \) from \( \llbracket (p) C_p \rrbracket_\Gamma \Delta \) to \( \llbracket (p) \rrbracket_\Gamma \rrbracket \rrbracket \Delta \) is non-expansive. Moreover, if \( \Gamma \leq \Delta \), then the inclusion map \( I_p : \llbracket (p) \rrbracket_\Gamma \to \llbracket (p) \rrbracket_\Delta \) is non-expansive.
  \end{itemize}

\begin{proof}
By induction on \( \Gamma \), using the fact that for all metric spaces \( X \), and sensitivities \( r \) and \( s \), we have a non-expansive map from \( !_{\llbracket \Gamma \rrbracket} \llbracket X \rrbracket \) to \( !_{\llbracket \Delta \rrbracket} \llbracket X \otimes_p X \rrbracket \).
\end{proof}

\begin{itemize}
  \item **Lemma 4.10.** The bind map defined by \( \text{bind}(f, \mu) \overset{\text{df}}{=} \text{bind}(f, \mu) \) is non-expansive from \( \text{Dist} Y \otimes_1 (Y \to_1 \text{Dist} X) \) to \( \text{Dist} X \).
  \end{itemize}

See Appendix A.2 for a proof of the soundness of the typing rule for the primitive operations. Note that the types of the higher-order primitives setmap, setfilter, and setfold are not sound when the functional is not guaranteed to converge. Some solutions to this problem are discussed in [23, Section 3.5].

Recursive types and Recursive functions

In this section, we will show that the introduction of a denotational semantics for interpreting recursive definitions of both data types and functions for the fragment of Fuzz that does not include probability distributions [3] can be generalised to our setting. Note that the interpretation is parametrised by a finite set of type identifiers, that behave as iso-recursive types, and by a definition environment. This approach slightly diverges from what precedes.

\begin{itemize}
  \item **Definition 4.11.** A metric complete partial order is a complete partial order \( X \) endowed with a metric \( d \) such that for all \((x_i)_{i \in \mathbb{N}}\) and \((x'_i)_{i \in \mathbb{N}}\) two \( \omega \)-chains in \( X \), if \( d_X(x_i, x'_i) \leq r \) for all \( i \in \mathbb{N} \), then \( d_X(\bigcup_{i \in \mathbb{N}} x_i, \bigcup_{i \in \mathbb{N}} x'_i) \leq r \).
  \end{itemize}
Equivalently, we may ask that \(d(\bigcup_{i \in \mathbb{N}} x_i, \bigcup_{i \in \mathbb{N}} x_i') \leq \liminf_{i \to \infty} d(x_i, x_i')\) \cite[Lemma 4.5]{3}.

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 \(\bot\).

**Theorem 4.12** \cite[Theorem 4.15]{3}. MetCPO\(\bot\) is an algebraically compact CPO-category, that is for every CPO-endofunctor \(F\), there exists an object \(\mu F\) and an isomorphism \(\iota : F(\mu F) \simeq \mu F\) such that \(\iota\) is an initial algebra and \(\iota^{-1}\) is a final coalgebra.

We have to show that MetCPO\(\bot\) is closed under the tensor and arrow constructors.

**Lemma 4.13.** If \(X\) and \(Y\) are two metric complete partial orders, then so is \(X \otimes_p Y\).

**Proof.** Let \((f_i)_{i \in \mathbb{N}}\) and \((g_i)_{i \in \mathbb{N}}\) be two \(\omega\)-chains in \(X \otimes_p Y\) such that for all \(i \in \mathbb{N}\), we have \(d_{X \otimes_p Y}(f_i, g_i) \leq r\). Let \(x_1\) and \(x_2\) in \(X\), and \(y_1\) and \(y_2\) in \(Y\).

\[
d_Y(f_i(x_1), g_i(x_2)) \leq d_{X \otimes_p Y}(f_i, g_i) + d_X(x_1, x_2) \leq d_{X \otimes_p Y}(f_i, g_i) + d_X(x_1, x_2) \leq r + d_X(x_1, x_2)
\]

therefore, as the function \(x \mapsto x^p\) is increasing, we have

\[
d_X \left( \bigcup_{i \in \mathbb{N}} x_i, \bigcup_{i \in \mathbb{N}} x_i' \right)^p \leq \liminf_{i \to \infty} d_X(x_i, x_i')^p + \liminf_{i \to \infty} d_Y(y_i, y_i')^p \leq \liminf_{i \to \infty} d_X(x_i, x_i')^p + \liminf_{i \to \infty} d_Y(y_i, y_i')^p = \liminf_{i \to \infty} d_X(x_i, x_i')^p + \liminf_{i \to \infty} d_Y(y_i, y_i')^p \leq \liminf_{i \to \infty} d_{X \otimes_p Y}(p_i, p_i').
\]

From Proposition 4.5 and what preceeds, we can deduce the soundness of the deterministic fragment of Plurimetric Fuzz, which features recursive types and functions.

**Theorem 4.15** (Soundness). If \(\pi\) is a derivation in the deterministic fragment of Plurimetric Fuzz of \((p) \Gamma \vdash a : A\), then \(\llbracket \pi \rrbracket\) is a non-expansive map from \(\llbracket (p) \Gamma \rrbracket\) to \(\llbracket A \rrbracket\).

**Remark 4.16.** One may notice that if we use recursive types to define \(\text{Nat}\) as \(\mu a\). Unit \(\oplus\) \(a\), then the following implementation of \((+)\) is non-expansive for all \(p\) rather than \(c(1, p)\)-sensitive:

```latex
\begin{verbatim}
let rec (+) n m = match n with injl () -> m | injr k -> injr (k + m)
\end{verbatim}
```

However, in this setting, the sensitivity is calculated with respect to the following distance:
\(d_{\text{Nat}}(m, n)\) equals 0 if \(m = n\), and \(\infty\) otherwise, rather than the usual distance on \(\mathbb{N}\). This justifies the introduction of \(\text{Nat}\) as a primitive data type and of \((+)\) as a primitive operation.
5 Metatheoretical properties

Unless otherwise stated, $[-]$ will refer to one of the two closely-related denotational semantics we have defined. More specifically, $[A]$ may be either a metric space or a metric complete partial order (CPO). Furthermore, the terms will be drawn from the appropriate fragment.

In the same way as [3], we call a substitution a finite partial map from variables to values. We write $S(e)$ for the simultaneous substitution of $x$ by $S(x)$ in $e$ for all $x$ in $\text{dom}(S) \cap \text{FV}(e)$. A substitution $S$ is said to be well-typed by a precontext $\Gamma$ and we write $S : \Gamma$ when the following two assertions are equivalent: $\vdash S(x) : A$ and $[x : A]_s \in \Gamma$ for some $s \geq 0$. Finally, for all parameters $p$, we naturally define $[[S]]$ as an element of $[[((p) \ \Gamma)]]$.

Lemma 5.1 (Substitution). For all derivations $\pi$ of $(p) \ \Gamma, \Delta \vdash a : A$ and for all well-typed substitutions $S : \Gamma$, there exists a derivation $\pi'$ of $(p) \ \Delta \vdash S(a) : A$. Moreover, $[[\pi']] = [\pi][[S]], -$.

Types and denotation are preserved by the operational semantics.

Theorem 5.2 (Preservation). For all derivations $\pi_{a}$ of $\vdash a : A$, if $a \Downarrow v$, then there exists a derivation $\pi_{v}$ of $\Downarrow v : A$. Moreover, $[[\pi_{a}]] = [[\pi_{v}]]$.

Let us now state the main result of this section, that is the metric preservation theorem.

Theorem 5.3 (Metric preservation for Core Plurimetric Fuzz). For all derivations $\pi$ of $(p) \ \Gamma \vdash a : A$ and for all well-typed substitutions $S, S' : \Gamma$, there exists well-typed values $v$ and $v'$ such that $S(a) \Downarrow v$ and $S'(a) \Downarrow v'$ and $d_{[A]}([v]), [v']) \leq d_{[\Gamma]}([S]), [S'])$.

Theorem 5.4 (Metric preservation for Plurimetric Fuzz). For all derivations $\pi$ of $(p) \ \Gamma \vdash a : A$ and for all well-typed substitutions $S, S' : \Gamma$, we have $d_{[A]}([S(a)), [S'(a)]) \leq d_{[\Gamma]}([S]), [S'])$.

By itself, the second formulation of the metric preservation theorem does not constrain the termination behaviour of the two terms $S(a)$ and $S'(a)$. However, the following lemma connects termination from both the operational and denotational perspectives. More details on the implications of this result are given in [3, Section 5].

Lemma 5.5 (Adequacy for Plurimetric Fuzz). If $\emptyset \vdash a : A$ and $[[a]] \neq \bot$, then there exists a value $v$ such that $a \Downarrow v$.

The proofs are similar to the one given in [3], given our soundness results (Proposition 4.5 and Theorem 4.15).

6 Expressive power and Precision

Let us now illustrate the usage of our type system with three examples.

Example: Functions with Multiple Arguments

Let us consider the term $\lambda c. \ (\text{let } (x, y) = c \text{ in } f(\langle x, y \rangle) + g(x, !y))$ where $f : !\mathbb{R} \otimes_2 \mathbb{R} - \odot_2 \mathbb{R}$ and $g : \mathbb{R} \otimes_2 !\mathbb{R} - \odot_2 \mathbb{R}$, that is the same example as in [27, Section 5].

\[
\begin{align*}
(2) \ & \ [x : \mathbb{R}]_2, [y : \mathbb{R}]_1 \vdash f(\langle x, y \rangle) : \mathbb{R} \quad (2) \ & \ [x : \mathbb{R}]_1, [y : \mathbb{R}]_2 \vdash g(x, !y) : \mathbb{R} \\
(2) \ & \ \sqrt{2} \cdot (\langle x : \mathbb{R} \rangle, \langle y : \mathbb{R} \rangle) \vdash f(\langle x, y \rangle) + g(x, !y) : \mathbb{R}
\end{align*}
\]

This typing derivation shows that, 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 naive extension of Fuzz would overestimate it to 4 [27, Section 5].
Example: Suboptimal sensitivity analysis

We can without difficulty find a term \( e \) and a type \( A \) such that \( \vdash e : A \) in Bunched Fuzz, but \( \not\vdash e : A \) in Plurimetric Fuzz. For example, let \( e = \lambda x. ((x, x), *) \) and \( A = \text{Unit} \rightarrow \text{Real} \). (Plurimetric Fuzz would require the exponential constructor to be annotated with at least \( 2\sqrt{2} \).) However, such cases do not seem to appear in practical programs.

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 \( \text{weight} : x \mapsto 1 - 1/(1 + e^{-4(x-r)}) \).

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 \( \text{Real} \rightarrow \text{Real} \).

A row of the database is represented by the following type: \( \text{Row} = \text{Point} \oplus \text{Label} \) where \( \text{Point} = \text{Real} \oplus \text{Real} \), and \( \text{Label} = \text{Unit} \oplus \cdots \oplus \text{Unit} \). 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 \( \infty \)-sensitive primitive):

<table>
<thead>
<tr>
<th>Listing 2</th>
<th>Implementation of the neighbour classification algorithm.</th>
</tr>
</thead>
<tbody>
<tr>
<td>let get_pos (r : row) : point = let (pos, _) = r in pos</td>
<td></td>
</tr>
<tr>
<td>let get_label (r : row) : label = let (_, label) = r in label</td>
<td></td>
</tr>
<tr>
<td>let score (l : label) (db : database) : real = db</td>
<td></td>
</tr>
<tr>
<td>&gt; setfilter (fun r -&gt; get_label r = 1)</td>
<td></td>
</tr>
<tr>
<td>&gt; setmap (fun r -&gt; distance (0, 0) (get_pos r))</td>
<td></td>
</tr>
<tr>
<td>&gt; setfold (fun acc x -&gt; acc + weight x) 0</td>
<td></td>
</tr>
<tr>
<td>let predict (db : database) : label = exp_noise labels score db</td>
<td></td>
</tr>
</tbody>
</table>

Informally, \( \text{score} \) 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 \( \text{distance} \) is non-expansive on elements of type \( \text{Point} \)); (3) computing the sum of the weights of the points. Moreover, \( \text{exp_noise} \) is a specialised version of the exponential mechanism presented in [16, Equation 1] for the case \( s = 1 \) and \( \epsilon = 1 \), which has type \( \text{Set} \langle \text{Label} \rangle \rightarrow_1 (\text{Label} \rightarrow_1 \text{Database} \rightarrow_0 \text{Real}) \rightarrow_1 \text{Set} \langle \text{Row} \rangle \rightarrow_0 \text{Unit} \).

We can derive the following types for the above functions: \( \text{score} : \text{Label} \rightarrow_1 \text{Database} \rightarrow_0 \text{Real} \), and \( \text{predict} : \text{Set} \langle \text{Row} \rangle \rightarrow_0 \text{Unit} \). First, by applying the tensor-elimination and arrow-introduction rules, we can show that the helper functions \( \text{get_pos} \) and \( \text{get_label} \) have type \( \text{Row} \rightarrow_0 \text{Point} \) and \( \text{Row} \rightarrow_0 \text{Bool} \) respectively. Then we type the three anonymous functions that appear in our implementation:

- \( \text{fun} \ r \rightarrow \text{get_label} \ r = 1 \) has type \( \text{Row} \rightarrow_0 \text{Bool} \) in the context \( [l : \text{Label}]_{\infty} \);
- \( \text{fun} \ r \rightarrow \text{distance} (0, 0) (\text{get_pos} \ r) \) has type \( \text{Row} \rightarrow_0 \text{Real} \) as the Euclidean distance \( \text{distance} \) has type \( \text{Point} \rightarrow_0 \text{Point} \rightarrow_0 \text{Real} \);
- \( \text{fun} \ \text{acc} \ x \rightarrow \text{acc} + \text{weight} \ x \) has type \( \text{Real} \rightarrow_0 \text{Real} \rightarrow_0 \text{Real} \).

This way, we show that \( \text{score} \) has the following type \( \text{Label} \rightarrow_1 \text{Database} \rightarrow_0 \text{Real} \) and we can apply the \( \text{exp_noise} \) function to conclude.

In particular, by Lemma 4.3, this 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 B.

Let \(\text{Der}(\text{Fuzz})\) be the set of derivations in Fuzz endowed with the following partial order: for all derivations \(\pi \vdash a : A \) and \(\pi' \vdash b : B\), we have \(\pi \leq \pi'\) iff \(\Gamma \leq \Delta\) (see Definition 3.4) and \((a, A) = (b, B)\). Similarly, we define \(\text{Der}(\text{PFuzz})\) for Plurimetric Fuzz.

7.1 Translation from Fuzz to Plurimetric Fuzz

For all parameters \(p\), we define a mapping \(P^p\) from Fuzz types to Plurimetric Fuzz types by structural induction as follows:

\[
\begin{align*}
P^p_{\text{type}}(\text{Unit}) &= \text{Unit} \\
P^p_{\text{type}}(A \oplus B) &= P^p_{\text{type}}(A) \oplus P^p_{\text{type}}(B) \\
P^p_{\text{type}}(A \sqcup B) &= P^p_{\text{type}}(A) \sqcup P^p_{\text{type}}(B) \\
P^p_{\text{type}}(A \otimes B) &= P^p_{\text{type}}(A) \otimes P^p_{\text{type}}(B) \\
\end{align*}
\]

Note that Fuzz lists are mapped to \(p\)-lists in Plurimetric Fuzz, i.e., for all type \(A\), we have \(P^p_{\text{type}}(\text{List}(A)) = \text{List}_p(P^p_{\text{type}}(A))\). The distance on the latter type is given by:

\[
d_{\text{List}_p}(A(l, l')) = \sqrt{\sum_{i=1}^n d_A(l_i, l'_i)^p} \text{ if length}(l) = \text{length}(l') = n, \text{ and } \infty \text{ otherwise.}
\]

We also define a mapping \(P^p_{\text{ctx}}\) from Fuzz contexts to Plurimetric Fuzz precontexts by \(P^p_{\text{ctx}}(\emptyset) = 0\), and \(P^p_{\text{ctx}}(\Gamma, [x : A]_a) = P^p_{\text{ctx}}(\Gamma), [x : P^p_{\text{type}}(A)]_s/p\), and a mapping \(P^p_{\text{der}}\) on derivations. For unary and binary rules, we have for instance:

\[
\begin{align*}
P^p_{\text{der}}([x : A]_1 \vdash x : A)_{\text{var}} &= (p) [x : P^p_{\text{type}}(A)]_1 \vdash x : P^p_{\text{type}}(A)_{\text{var}} \\
&P^p_{\text{der}}(\pi_a \vdash \Delta \vdash a : B)_{\otimes I} = (p) P^p_{\text{ctx}}(\pi_a) \vdash a : P^p_{\text{type}}(A)_{\otimes I} \quad (p) P^p_{\text{ctx}}(\Delta) \vdash b : P^p_{\text{type}}(B)_{\otimes I} \\
&P^p_{\text{ctx}}(\Gamma + \Delta) \vdash (a, b) : P^p_{\text{type}}(A) \otimes P^p_{\text{type}}(B)_{\otimes I} = W
\end{align*}
\]

Definition 7.1. A derivable judgement \(\Gamma \vdash e : A\) is said to be minimal in a (Plurimetric) Fuzz if for all contexts \(\Delta\) 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 B for a proof).

Lemma 7.2. For all precontexts \(\Gamma\) and \(\Delta\), sensitivities \(s\), and parameters \(p\), we have the following equality: \(C^p(P^p_{\text{ctx}}(\Gamma); s^{1/p} : P^p_{\text{ctx}}(\Delta)) = P^p_{\text{ctx}}(\Gamma + s\Delta)\).

Corollary 7.3. For all parameters \(p\), the image by \(P^p_{\text{der}}\) of the derivation \(\pi\) 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 [23] 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 [23, 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, Corollary 7.3 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.

\textbf{Claim 7.4 (No miracle).} We cannot soundly extend $P_{\text{der}}$ to the derivations involving primitive operations such as addition on numbers.

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

\[
\begin{array}{c}
\Gamma \vdash a : \text{Real} \\
\Delta \vdash b : \text{Real}
\end{array}
\quad \begin{array}{c}
P_{\text{der}}^2 \quad (2) \\
P_{\text{ctx}}^2(\Gamma) \vdash a : \text{Real} \\
P_{\text{ctx}}^2(\Delta) \vdash b : \text{Real}
\end{array}
\quad \begin{array}{c}
\Gamma + \Delta \vdash a + b : \text{Real}
\end{array}
\]

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

\section*{7.2 Translation from Plurimetric Fuzz to Fuzz}

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

\[
\begin{align*}
F_{\text{type}}^p(A \otimes_q B) &= F_{\text{type}}^p(A) \otimes F_{\text{type}}^p(B) & \text{if } q \leq p \\
F_{\text{type}}^p(A \rightarrow_q B) &= F_{\text{type}}^p(A) \rightarrow F_{\text{type}}^p(B) & \text{if } q \leq p \\
F_{\text{type}}^p(!sA) &= !sF_{\text{type}}^p(A) \\
F_{\text{ctx}}^p([x : A]_s, \Gamma) &= [x : F_{\text{type}}^p(A)]_s, F_{\text{ctx}}^p(\Gamma) \\
F_{\text{der}}^p(q) [x : A]_1 \vdash x : A & \text{ var} \quad [x : F_{\text{type}}^p(A)]_1 \vdash x : F_{\text{type}}^p(A) & \text{ var} & \text{if } q \leq p
\end{align*}
\]

\textbf{Lemma 7.5.} For all precontexts $\Gamma$ and $\Delta$, for all sensitivities $s$, we have the following inequality: $s^p F_{\text{ctx}}^p(\Gamma) + F_{\text{ctx}}^p(\Delta) \leq F_{\text{ctx}}^p(C^p(\Gamma; s\Delta))$.

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

\textbf{Corollary 7.6.} For all parameters $p$, if the image by the mapping $F_{\text{der}}^p$ of a derivation $\pi$ in Plurimetric Fuzz is defined, then it is a valid derivation in Fuzz.

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

\textbf{Theorem 7.7.} For all parameters $p$, we have $F_{\text{der}}^p \circ F_{\text{der}}^p = \text{Id}_{\text{Der(Fuzz)}}$. In other words, the following diagram commutes:

\[
\begin{array}{ccc}
\text{Id} & \xrightarrow{F_{\text{der}}^p} & \text{Der(Fuzz)} \\
\xrightarrow{\text{Der(Fuzz)}} & & \xrightarrow{F_{\text{der}}^p} \text{Der(PFuzz)}
\end{array}
\]
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 typing 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.

Type checking and type inference for systems based on linear logic have been the object of several works, e.g., [4, 1, 13]. While type checking for Fuzz is straightforward (for DFuzz [16], which is a variant of Fuzz that incorporates dependent types, see [2]), we anticipate that type checking for Plurimetric Fuzz will be significant more challenging to the non-linear nature of the sensitivity constraints. If solved, it would allow us to replace Fuzz by Plurimetric Fuzz in [26], 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.

Finally, the question of whether one can combine recursive types and functions with probability distributions is still open, both in the case of Fuzz and of its extensions like Plurimetric Fuzz.

References


Appendix

A Semantics

A.1 Proof of the Soundness of the Typing Rules

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

Proof. Let $(x_1, x_2)$ and $(x'_1, x'_2)$ be two elements of $X_1 \otimes_p X_2$.

\[ d_{Y_1 \otimes_p Y_2}((f \times g)(x_1, x_2), (f \times g)(x'_1, x'_2)) = d_{Y_1 \otimes_p Y_2}(f(x_1), g(x_2)), (f(x'_1), g(x'_2))) \]

\[ = \sqrt{d_{Y_1}(f(x_1), f(x'_1))^p + d_{Y_2}(g(x_2), g(x'_2))^p} \]

\[ \leq \sqrt{d_{X_1}(x_1, x'_1)^p + d_{X_2}(x_2, x'_2)^p} \]

\[ = d_{X_1 \otimes_p X_2}((x_1, x_2), (x'_1, x'_2)) \]

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

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

\[ \sum_{i=1}^{n} \frac{x_i}{y_i} \leq \max_{1 \leq i \leq n} \frac{x_i}{y_i} \quad \text{and therefore} \quad \ln \left( \sum_{i=1}^{n} \frac{x_i}{y_i} \right) \leq \max_{1 \leq i \leq n} \ln \frac{x_i}{y_i}. \]

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

\[ \sum_{i=1}^{n} \frac{x_i}{y_i} = \frac{x_1}{y_1} + \frac{x_2}{y_2} = \frac{1}{\frac{1}{y_1} + \frac{1}{y_2}} \cdot \frac{x_1}{y_1} + \frac{1}{\frac{1}{y_1} + \frac{1}{y_2}} \cdot \frac{x_2}{y_2} \]

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

\[ \sum_{i=1}^{n} \frac{x_i}{y_i} \leq u \cdot \max \left\{ \frac{x_1}{y_1}, \frac{x_2}{y_2} \right\} + v \cdot \max \left\{ \frac{x_1}{y_1}, \frac{x_2}{y_2} \right\} = (u + v) \cdot \max \left\{ \frac{x_1}{y_1}, \frac{x_2}{y_2} \right\} \]

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

$$\sum_{i=1}^{n} x_i = \frac{x_1 + \sum_{i=2}^{n} x_i}{y_1 + \sum_{i=2}^{n} y_i} \leq \max \left\{ \frac{x_i}{y_i}, \frac{\sum_{i=2}^{n} x_i}{\sum_{i=2}^{n} y_i} \right\}$$

$$\leq \max \left\{ \frac{x_i}{y_i}, \frac{\sum_{i=2}^{n} x_i}{\sum_{i=2}^{n} y_i} \right\} = \max \frac{x_i}{2 \leq i \leq n}. \frac{y_i}{y_i}$$

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 $X - Y = \ln(\sum_{i=1}^{n} x_i) - \ln(\sum_{i=1}^{n} y_i) = \ln(\sum_{i=1}^{n} x_i/\sum_{i=1}^{n} y_i)$.

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

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

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

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

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

Lemma A.3. The following map is non-expansive:

$$\text{bind} : \text{Dist} \ Y \oplus_{1} (Y \to_{1} \text{Dist} \ X) \to \text{Dist} \ X$$

$$(\mu, f) \mapsto t \mapsto \sum_{s \in Y} f(s)(t) \mu(t).$$

We present an elementary proof of this result (which also follows from the work of Barthe and Olmedo [8]).

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

For all $t \in X$, we have

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

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

Therefore, we have

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

which is equivalent to the desired inequality:

$$d_{\text{Dist} \ X}(f^\uparrow(\mu), f'^\uparrow(\mu')) \leq d_{\text{Dist} \ Y}(\mu, \mu') + \max_{s \in Y} d_{\text{Dist} \ X}(f(s), f'(s))$$

$$= d_{\text{Dist} \ Y}(\mu, \mu') + d_{Y \to_{1} \text{Dist} \ X}(f, f')$$

$$= d_{\text{Dist} \ Y \oplus_{1} (Y \to_{1} \text{Dist} \ X)}((\mu, f), (\mu', f')).$$
A.2 Proof of the Soundness of the Rules for Primitive Operations

Lemma A.4. The following rules are sound with respect to the denotational semantics:

\[
\begin{align*}
(p) & \quad \Gamma \vdash x : A & (p) & \quad \Gamma \vdash \{x\} : \text{Set}(A) \\
(p) & \quad \Gamma \vdash n : \text{Nat} & (p) & \quad 2\Gamma \vdash \{n\} : \text{Set}(\text{Nat})
\end{align*}
\]

Proof. For all set \(X\), the sensitivity of the map \(x \mapsto \{x\}\) is bounded by \(\infty\). Therefore, on can soundly introduce the following rule:

\[
\vdash \lambda x. \{x\} : !\infty A \rightarrow \text{Set}(A)
\]

which is equiderivable with the \((\text{Set})\) rule.

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

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

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

\[
\vdash \lambda n. \{n\} : !p \mathbb{N} \rightarrow \text{Set}(\mathbb{Nat})
\]

which is equiderivable with the \((\text{Set}_{\mathbb{Nat}})\) rule.

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

B. Translation mappings

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

\[
\begin{align*}
(p) & \quad \Gamma \vdash a : A & (p) & \quad \Gamma \vdash b : B \\
(p) & \quad \Gamma \vdash (a, b) : A \& B & (p) & \quad \Gamma \vdash c : A \& B \\
(p) & \quad \Gamma \vdash \pi_1(c) : A & (p) & \quad \Gamma \vdash \pi_2(c) : B
\end{align*}
\]

Lemma B.1. For all precontexts \(\Gamma\), sensitivities \(s\) and parameters \(p\), we have:

\[
\begin{align*}
\text{(i)} & \quad s^{1/p} \cdot P^{P^p_{\text{ctx}}(\Delta)} = P^{P^p_{\text{ctx}}(s\Delta)} \\
\text{(ii)} & \quad s^p \cdot P^{P^p_{\text{ctx}}(\Delta)} = P^{P^p_{\text{ctx}}(s\Delta)}
\end{align*}
\]

Proof. By induction on the structure of the \(\Delta\).

Lemma B.2. For all precontexts \(\Gamma\) and \(\Delta\), for all sensitivities \(s\) and parameters \(p\), we have the following equality:

\[
P^{P^p_{\text{ctx}}(\Gamma)}(s^{1/p} \cdot P^{P^p_{\text{ctx}}(\Delta)}) = P^{P^p_{\text{ctx}}(\Gamma + s\Delta)}
\]

Proof. Let us prove this equality by induction on the structure of \(\Gamma\).

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

\[
\begin{align*}
C^p & \left( P^{P^p_{\text{ctx}}(\Gamma)}; s^{1/p} \cdot P^{P^p_{\text{ctx}}(\Delta)} \right) = C^p \left( P^{P^p_{\text{ctx}}(\Gamma_0, [x : A]_{|\Gamma})}; s^{1/p} \cdot P^{P^p_{\text{ctx}}(\Delta_0, [x : A]_{|\Gamma})} \right) \\
& = C^p \left( P^{P^p_{\text{ctx}}(\Gamma_0, [x : A]_{|\Gamma})}; P^{P^p_{\text{ctx}}(\Delta_0, [x : A]_{|\Gamma})} \right) \\
& = C^p \left( P^{P^p_{\text{ctx}}(\Gamma_0)}; [x : P^{P^p_{\text{ctx}}(\Delta_0)}]_{|\Gamma + s\Delta} \right) \\
& = P^{P^p_{\text{ctx}}(\Gamma_0 + s\Delta)}([x : P^{P^p_{\text{ctx}}(\Delta)}]_{|\Gamma + s\Delta})
\end{align*}
\]

as desired.
Lemma B.3. For all precontexts $\Gamma$ and $\Delta$, for all sensitivities $s$, for all parameters $p$ and $q$ such that $p \geq q$, we have the following inequality: $s^p F^p_{\text{ctx}}(\Gamma) + F^p_{\text{ctx}}(\Delta) \leq F^p_{\text{ctx}}(C^q(\Gamma; s\Delta))$.

Proof. Let us prove this inequality by induction on the structure of $\Gamma$.

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

\[
\begin{align*}
&\quad\quad\quad s^p F^p_{\text{ctx}}(\Gamma) + F^p_{\text{ctx}}(\Delta) \\
&= s^p F^p_{\text{ctx}}(\Gamma_0, [x : A]_t) + F^p_{\text{ctx}}(\Delta_0, [x : A]_t) \\
&= s^p \left( F^p_{\text{ctx}}(\Gamma_0, [x : F^p_{\text{type}}(A)]_{rs}) + F^p_{\text{ctx}}(\Delta_0, [x : F^p_{\text{type}}(A)]_{tp}) \right) \\
&\leq F^p_{\text{ctx}}(C^q(\Gamma_0; s\Delta_0), [x : F^p_{\text{type}}(A)]_{(rs)q + tp}) \\
&\leq F^p_{\text{ctx}}(C^q(\Gamma_0; s\Delta_0), [x : F^p_{\text{type}}(A)]_{(rs)q + tp})^{p/q} \\
&\leq F^p_{\text{ctx}}(C^q(\Gamma_0; s\Delta_0), [x : F^p_{\text{type}}(A)]_{(rs)q + tp})^{1/q} \\
&\leq F^p_{\text{ctx}}(C^q(\Gamma; s\Delta))
\end{align*}
\]

as desired.