From linear types to relational higher-order logic for analyzing program sensitivity*

Artur Szafarczyk, Patrick Baillot

Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL, 59000 Lille, France

1 Introduction

Differential privacy (DP) is an approach for privacy-preserving data analysis with rigorous mathematical guarantees [DR14]. Its idea is to permit queries or computation of aggregated results on a database while ensuring limits on the information that can be extracted about a given individual. It has been widely used in areas such as medical or demographic databases.

It turns out that checking that a large program is DP is a difficult task. For this reason, some programming language approaches for verifying this property have been explored. One such approach is based on types: Fuzz is a linear type system for a functional programming language ensuring that well-typed programs are ϵ-differentially private [RP10] (see also [SB24, SB26] for some recent variants). A second approach is that of proof systems, either in the form of Hoare logics for imperative programs [BKOB12], or higher-order logics for functional ones [Agu20, HAG$^+$26]. While the type system approach is appealing for compositional building of large programs and for automatization, the proof systems are more expressive and can be used to check the DP of programming primitives. It would thus be desirable to use these two techniques in combination. The paper [ZRH$^+$19] has proposed an approach of this kind for imperative programs, but this does not allow to take advantage of higher-order functional programming. In this work we want to make a first step in this direction for functional computation.

A key notion for DP is that of program *sensitivity*, which is a bound on how much the distance between two values can be multiplicatively increased by applying the function. The techniques for making a program differentially private are based on the idea of adding probabilistic noise to the output, calibrated by the sensitivity of the program [DR14]. The Fuzz language can be seen as a type system that allows to derive statically a sensitivity bound on a program. The problem we address here is to consider the deterministic fragment of Fuzz (that is to say, without probabilistic noise primitives), which we will call Fuzz_0, and to provide a compositional translation of it into a relational higher-order logic. The target logic we consider is the relational logic RHOL proposed in

*Work supported by the French National Agency for Research as part of HOPR project (ANR-24-CE48-5521-01)

[Agu20], an expressive system that admits a probabilistic extension. In [dAGH+17] the soundness of Fuzz was proved using a metric denotational semantics. In the present paper, we will instead represent metrics in RHOL and interpret Fuzz programs by means of the translation in RHOL.

2 Preliminaries and definition of metrics in RHOL

We start by giving a background on Fuzz [RP10] and RHOL [Agu20]. Fuzz is a type system for proving sensitivity of programs expressed in an extended λ-calculus. Sensitivity expresses a bound on the amount of scaling that a program can make. More precisely, in the semantics of Fuzz, types τ are interpreted as metric spaces with metric d_τ and sensitivity is expressed in the following way.

Definition 2.1 (sensitivity)**.** *Let e be a program of type σ depending on one variable x of type τ. We say that e is r-*sensitive *in x, if for any values a, b of type τ, and v, w such that $e[a/x]$ (resp. $e[b/x]$) evaluates to v (resp. w) we have $d_\sigma(v,w) \leq r \cdot d_\tau(a,b)$.*

This definition is extended to programs depending on n variables x_i of type τ_i by placing the requirement $d_\sigma(v,w) \leq \sum_{i=1}^n r_i \cdot d_{\tau_i}(a_i,b_i)$ (*), where $e[a_i/x_i]$ (resp $e[b_i/x_i]$) evaluates to v (resp. w).

The terms of Fuzz are denoted with letters t, e and its types are denoted with letters τ, σ. Fuzz judgments and rules are as in linear logic, except that variables in the context come annotated with a real number, which expresses sensitivity of the program with respect to that variable. Hence, the judgment $x_1 :_{r_1} \tau_1, \dots x_n :_{r_n} \tau_n \vdash e : \tau$ expresses that e is r_i-sensitive with respect to x_i for all i (property (*)).

Every type comes equipped with a metric. By using linear logic types, Fuzz allows for multiple metrics on the same underlying set. For instance, there are two simple metrics on the product set: $d_{\tau_1 \otimes \tau_2}(\langle x_1, x_2\rangle, \langle y_1, y_2\rangle) = d_{\tau_1}(x_1,y_1) + d_{\tau_2}(x_2,y_2)$ and $d_{\tau_1 \,\&\, \tau_2}(\langle x_1, x_2\rangle, \langle y_1, y_2\rangle) = \max(d_{\tau_1}(x_1,y_1), d_{\tau_2}(x_2,y_2)$. In addition to the metrics on product types, Fuzz enriches the ! modality by adding an indexed connective $!_s$, where $s \in \mathbb{R}_{>0}$. Its purpose is to scale the metric of a type by s, that is $d_{!_s\tau}(x,y) = s \cdot d_\tau(x,y)$. The last type that we are concerned with is the linear arrow $\multimap$. It is supposed to represent 1-sensitive functions, and its metric is $d_{\tau_1 \multimap \tau_2}(x,y) = \sup_{z:\tau_1} d_{\tau_2}(xz, yz)$. Derivation rules are analogous to those in linear logic, but with an appropriate treatment of sensitivity annotations, see Fig.1.

There are other types in Fuzz, notably for dealing with probabilities, but we omit them here and focus on the reduced deterministic version of Fuzz described above, which we call Fuzz_0.

We will now present RHOL. First, let us denote by $\Gamma \Vdash t : \tau$ typing judgments for (ordinary) λ-terms with simple types. HOL is a higher-order logic, which proves formulas on simply typed λ-terms t; its basic predicates are of the form $t = t'$. Its judgments are of the form $\Gamma \mid \Psi \vdash \phi$, where Γ is a typing environment for the λ-terms, Ψ is a set of formulas and ϕ is a formula. Its intuitive

$$\frac{r \geq 1}{\Gamma, x :_r \tau \vdash x : \tau}\ var \qquad \frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \langle e_1, e_2 \rangle : \tau_1 \,\&\, \tau_2}\ \&I \qquad \frac{\Gamma \vdash e : \tau_1 \,\&\, \tau_2}{\Gamma \vdash \pi_i e : \tau_i}\ \&E$$

$$\frac{\Delta \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Delta + \Gamma \vdash (e_1, e_2) : \tau_1 \otimes \tau_2}\ \otimes I \qquad \frac{\Gamma \vdash e : \tau_1 \otimes \tau_2 \quad \Delta, x :_r \tau_1, y :_r \tau_2 \vdash e' : \tau'}{\Delta + r\Gamma \vdash \mathrm{let}(x, y) = e \,\mathrm{in}\, e' : \tau'}\ \otimes E$$

$$\frac{\Gamma, x :_1 \tau \vdash e : \tau'}{\Gamma \vdash \lambda x.e : \tau \multimap \tau'}\ \multimap I \qquad \frac{\Delta \vdash e_1 : \tau \multimap \tau' \quad \Gamma \vdash e_2 : \tau}{\Delta + \Gamma \vdash e_1 e_2 : \tau'}\ \multimap E$$

$$\frac{\Gamma \vdash e : \tau}{s\Gamma \vdash !e :!_s \tau}\ !I \qquad \frac{\Gamma \vdash e :!_s \tau \quad \Delta, x :_{rs} \tau \vdash e' : \tau'}{\Delta + r\Gamma \vdash \mathrm{let}!x = e \,\mathrm{in}\, e' : \tau'}\ !E$$

Figure 1: Typing rules of Fuzz_0

meaning is that in the typing context Γ, the conjunction of Ψ implies ϕ. Its rules are standard and we omit them here because of space limitations.

RHOL is a relational logic, that is, a logic proving properties relating two terms (programs) t_1 and t_2. It is based on HOL. Its judgments are of the form $\Gamma \mid \Psi \vdash t_1 : \tau_1 \sim t_2 : \tau_2\{\phi\}$, where, as before, Γ is a typing environment, Ψ is a set of HOL formulas, ϕ is a HOL formula with two special free variables v_1, v_2 and t_i $(i = 1, 2)$ is a λ-term of type τ_i in environment Γ. The meaning of the judgment is that, in the typing environment Γ, if the conjunction of Ψ holds (preconditions), then so does $\phi[t_1/v_1][t_2/v_2]$ (postcondition). So, for instance, if ϕ is the formula $v_1 = v_2$, then the proved property is $t_1 = t_2$.

RHOL allows us to reason both by the structure of the postcondition (using HOL) and the structure of one or both programs. To give an idea of the rules of RHOL, we present some in Fig. 2 (acting on λ-terms). Structural rules, which only act on formulas, are omitted here. Observe, for instance, in Fig. 2 rule VAR, which deduces from a HOL judgment an RHOL one.

RHOL features two-sided rules, which require the two programs t_1 and t_2 to have the same structure, as well as one-sided rules (omitted here), which only modify one of the programs.

3 Translating Fuzz_0 types and defining metrics in RHOL

Our translation of Fuzz_0 into RHOL starts with a translation $\overline{(\cdot)}$ of Fuzz_0 terms into (ordinary) λ-terms and Fuzz_0 types into simple types. If t is a Fuzz_0 term, we denote its translation as $\bar{t}$. It is rather standard, in particular $\overline{!t} = \bar{t}$. The translation of types is also very natural and given below.

Definition 3.1. *We define inductively the* translation $\overline{\tau}$ *of Fuzz$_0$ types τ into simple types:*

(1) Base types of Fuzz$_0$ are translated to base simple types, (2) $\overline{\tau_1 \otimes \tau_2} = \overline{\tau_1 \,\&\, \tau_2} = \overline{\tau_1} \times \overline{\tau_2}$, *(3)* $\overline{\tau_1 \multimap \tau_2} = \overline{\tau_1} \to \overline{\tau_2}$, *(4)* $\overline{!_s \tau} = \overline{\tau}$.

Now, in order to speak about distances in RHOL, it would be natural to extend HOL with function symbols for each Fuzz_0 type to represent the metrics. However, such approach would not

$$\frac{\Gamma \Vdash x_1 : \sigma_1 \quad \Gamma \Vdash x_2 : \sigma_2 \quad \Gamma \mid \Psi \vdash_{HOL} \phi[x_1/v_1][x_2/v_2]}{\Gamma \mid \Psi \vdash x_1 : \sigma_1 \sim x_2 : \sigma_2\{\phi\}}\ VAR$$

$$\frac{\Gamma, x_1 : \tau_1, x_2 : \tau_2 \mid \Psi, \phi' \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2\{\phi\}}{\Gamma \mid \Psi \vdash \lambda x_1 : \tau_1.t_1 : \tau_1 \to \sigma_1 \sim \lambda x_2 : \tau_2.t_2 : \tau_2 \to \sigma_2\{\forall x_1, x_2.\phi' \Rightarrow \phi[v_1x_1/v_1][v_2x_2/v_2]\}}\ ABS$$

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash t_1 : \tau_1 \to \sigma_1 \sim t_2 : \tau_2 \to \sigma_2\{\forall x_1, x_2.\phi'[x_1/v_1][x_2/v_2] \Rightarrow \phi[v_1x_1/v_1][v_2x_2/v_2]\} \\ \Gamma \mid \Psi \vdash u_1 : \tau_1 \sim u_2\tau_2\{\phi'\}\end{array}}{\Gamma \mid \Psi \vdash t_1u_1 : \sigma_1 \sim t_2u_2 : \sigma_2\{\phi[u_1/x_1][u_2/x_2]\}}\ APP$$

Figure 2: Selected two-sided rules of RHOL

suffice, because the typing of a term with the linear arrow type conveys more information than just the distance; it also tells about the sensitivity of the programs. Because of that, we extend the HOL logic with metric predicates, starting with base types.

Definition 3.2. *For each base type of Fuzz$_0$ A, we add a basic predicate d_A on $\overline{A} \times \overline{A} \times \mathbb{R}_{\geq 0}$ with the following axioms:*

(1) $d_A(x,x) \leq 0$, *(2)* $d_A(x,y) \leq u \Leftrightarrow d_A(y,x) \leq u$,

(3) $d_A(x,y) \leq u_1 \wedge d_A(y,z) \leq u_2 \Rightarrow d_A(x,z) \leq u_1 + u_2$, *(4)* $d_A(x,y) \leq u \wedge u \leq v \Rightarrow d_A(x,y) \leq v$.

Metrics on higher types will be defined inductively by the structure of the Fuzz$_0$ types. For each Fuzz$_0$ type τ, we introduce a predicate WT_τ on $\overline{\tau}$, whose extension should be the elements, which are typable with type τ in Fuzz$_0$. In fact, it suffices to take $\mathrm{WT}_\tau(x) := d_\tau(x,x) \leq 0$. We will define the metric predicates in such a way that a real-valued function $f : \mathbb{R} \to \mathbb{R}$ that is not 1-sensitive will not satisfy the predicate $\mathrm{WT}_{\mathbb{R} \multimap \mathbb{R}}(f)$.

Definition 3.3. *For each Fuzz$_0$ type τ, we add a basic predicate d_τ on $\overline{\tau}$ defined by:*

1. $d_{\tau_1 \otimes \tau_2}(\langle x_1, y_1\rangle, \langle x_2, y_2\rangle) \leq u \Leftrightarrow \exists u_1, u_2 : \mathbb{R}.d_{\tau_1}(x_1,y_1) \leq u_1 \wedge d_{\tau_2}(x_2,y_2) \leq u_2 \wedge u_1 + u_2 = u$,
2. $d_{\tau_1 \,\&\, \tau_2}(\langle x_1, x_2\rangle, \langle y_1, y_2\rangle) \leq u \Leftrightarrow d_{\tau_1}(x_1,y_1) \leq u \wedge d_{\tau_2}(x_2,y_2) \leq u$,
3. $d_{!_s\tau}(x,y) \leq s \cdot u \Leftrightarrow d_\tau(x,y) \leq u$
4.

$$d_{\tau_1 \multimap \tau_2}(x,y) \leq u \Leftrightarrow (\forall z : \overline{\tau_1}.\, \mathrm{WT}_{\tau_1}(z) \Rightarrow d_{\tau_2}(xz,yz) \leq u) \wedge \qquad (\multimap 1)$$

$$(\forall z_1, z_2 : \overline{\tau_1}.d_{\tau_1}(z_1,z_2) \leq v \Rightarrow d_{\tau_2}(xz_1,xz_2) \leq v \wedge d_{\tau_2}(yz_1,yz_2) \leq v) \qquad (\multimap 2)$$

The first conjunct ($\multimap 1$) of the definition of $d_{\tau_1 \multimap \tau_2}$ is the definition of the metric, while the second one ($\multimap 2$) ensures that both x and y are 1-sensitive functions. In particular, the distance predicate $d_{\tau_1 \multimap \tau_2}$ will *not* be defined on terms that do not represent 1-sensitive programs.

We only introduced the axioms of the metric predicates for basic types (in Def. 3.2). It is possible to *prove* the same properties for the other metric predicates defined in Def. 3.3.

To illustrate the importance of condition ($\multimap 2$), consider the following Fuzz$_0$ type $!_2(\mathbb{R} \multimap \mathbb{R}) \multimap (\mathbb{R} \multimap \mathbb{R})$ and a function $g(f) := f \circ f$, which takes a real valued function and composes it with itself. The result $g(f)$ is 1-sensitive if g is allowed to take as inputs only 1-sensitive functions, that is, g has type $!_2(\mathbb{R} \multimap \mathbb{R}) \multimap (\mathbb{R} \multimap \mathbb{R})$ in Fuzz$_0$. The result $g(f)$ is *not* 1-sensitive however, if we allow 2-sensitive functions as inputs, that is g is not typable in Fuzz$_0$ with type $!_2(!_2\mathbb{R} \multimap \mathbb{R}) \multimap (\mathbb{R} \multimap \mathbb{R})$. This is captured in RHOL by the fact that the predicate $d_{!_2(!_2\mathbb{R} \multimap \mathbb{R}) \multimap (\mathbb{R} \multimap \mathbb{R})}(g, g) \leq 0$ does not hold, because property ($\multimap 2$) is not satisfied.

4 The translation of Fuzz$_0$ derivations into RHOL

We can now encode judgments. The Fuzz$_0$ judgement $x_1 :_{r_1} \tau_1, \ldots x_n :_{r_n} \tau_n \vdash e : \tau$ is translated in RHOL as:

$$a_1 : \overline{\tau_1}, \ldots a_n : \overline{\tau_n}, b_1 : \overline{\tau_1}, \ldots b_n : \overline{\tau_n}, u_1 : \mathbb{R}, \ldots u_n : \mathbb{R} \mid$$
$$d_{\tau_1}(a_1, b_1) \leq u_1, \ldots d_{\tau_n}(a_n, b_n) \leq u_n \vdash \overline{e}[a_i/x_i] : \overline{\tau} \sim \overline{e}[b_i/x_i] : \overline{\tau}\{d_\tau(v_1, v_2) \leq \sum_{i=1}^{n} r_i u_i\} \quad (1)$$

The intuition of this RHOL judgement is to express the (*) property of Sect. 2, by bounding the distance $d_\tau(v_1, v_2)$ between the two results by means of the distances $d_{\tau_i}(a_i, b_i)$ and sensitivities r_i.

Our main result is the following theorem.

Theorem 1. *Whenever $\Gamma \vdash e : \tau$ is a derivable judgment in Fuzz$_0$, then the judgment (1) above is derivable in RHOL.*

The rules that pose the biggest problems are linear arrow introduction and elimination, as well as tensor elimination. In particular, condition ($\multimap 2$) is crucial in the translation of the linear arrow elimination rule.

5 Perspectives

The next step will be to extend our translation to the probabilistic part of Fuzz. In this way we would be able to translate Fuzz statements of ϵ-differential privacy into RHOL. For that, we will need to extend RHOL with probabilities, for which there are several possible choices. We could follow the approach of Aguirre [Agu20] of using a coupling-based logic. However, this approach might not be expressive enough for our needs. Other possible approaches include assertion-based logics [BEG$^+$18], quantitative logics [ABDG25] and separation logic [HAG$^+$26].

References

[ABDG25] Martin Avanzini, Gilles Barthe, Davide Davoli, and Benjamin Grégoire. A quantitative probabilistic relational hoare logic. *Proc. ACM Program. Lang.*, 9(POPL):1167–1195, 2025.

[Agu20] Alejandro Aguirre. *Relational logics for higher-order effectful programs*. PhD thesis, Technical University of Madrid, Spain, 2020.

[BEG+18] Gilles Barthe, Thomas Espitau, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. An assertion-based program logic for probabilistic programs. In *ESOP 2018, Proceedings*, LNCS, pages 117–144. Springer, 2018.

[BKOB12] Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. Probabilistic relational reasoning for differential privacy. In *Proceedings of POPL 2012*, pages 97–110. ACM, 2012.

[dAGH+17] Arthur Azevedo de Amorim, Marco Gaboardi, Justin Hsu, Shin-ya Katsumata, and Ikram Cherigui. A semantic account of metric preservation. In *POPL 2017*. ACM, 2017.

[DR14] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[HAG+26] Philipp G. Haselwarter, Alejandro Aguirre, Simon Oddershede Gregersen, Kwing Hei Li, Joseph Tassarotti, and Lars Birkedal. Modular verification of differential privacy in probabilistic higher-order separation logic. In *Proceedings of PLDI 2026, to appear*. ACM, 2026.

[RP10] Jason Reed and Benjamin C. Pierce. Distance makes the types grow stronger. In *ICFP'10: Proceedings of the 15th ACM SIGPLAN international conference on Functional programming*, pages 157–168, 2010.

[SB24] Victor Sannier and Patrick Baillot. A linear type system for Lp-metric sensitivity analysis. In *FSCD 2024*, pages 12–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.

[SB26] Victor Sannier and Patrick Baillot. Dependent coeffects for local sensitivity analysis. *Proceedings of the ACM on Programming Languages*, 10(POPL):806–832, 2026.

[ZRH+19] Hengchu Zhang, Edo Roth, Andreas Haeberlen, Benjamin C. Pierce, and Aaron Roth. Fuzzi: a three-level logic for differential privacy. *Proc. ACM Program. Lang.*, 3(ICFP):93:1–93:28, 2019.