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

Victor Sannier   Patrick Baillot

FSCD 2024
Tallinn, Estonia

CRIStAL Laboratory, Lille, France

Overview

Type systems allow us to guarantee certain properties of computer programs.

Overview

Type systems allow us to guarantee certain properties of computer programs.

In this work, we are interested in the concept of **sensitivity**, which addresses the question of how much the output changes when the input changes.

Overview

Type systems allow us to guarantee certain properties of computer programs.

In this work, we are interested in the concept of **sensitivity**, which addresses the question of how much the output changes when the input changes.

This concept has applications in various fields, including privacy protection.

1. Introduction

Function Sensitivity and Differential Privacy

Function Sensitivity

Definition

Let (X, d_X) and (Y, d_Y) be two metric spaces.
A function $f : X \to Y$ is **s-sensitive** (for $s \in [0, +\infty]$) if for all x and x' in X, we have:

$$d_Y(f(x), f(x')) \leq s \cdot d_X(x, x') .$$

Function Sensitivity

Definition

Let (X, d_X) and (Y, d_Y) be two metric spaces.
A function $f : X \to Y$ is **s-sensitive** (for $s \in [0, +\infty]$) if for all x and x' in X, we have:

$$d_Y(f(x), f(x')) \leq s \cdot d_X(x, x') .$$

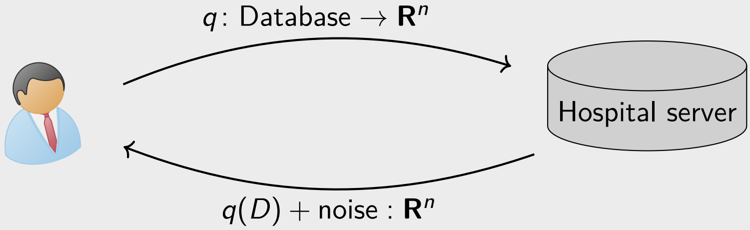
Example

Often (Y, d_Y) will be $(\mathbf{R}^n, L^1)$, that is $\mathbf{R}^n$ endowed with the following metric:

$$d_1(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} |x_i - y_i|$$

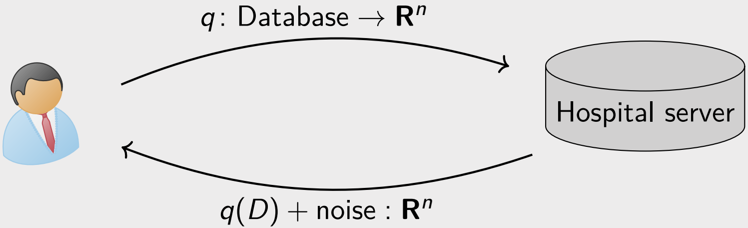
Example

How much noise to add so that the reply contains no private information?



Example

How much noise to add so that the reply contains no private information?



The more sensitive a query is, the more it depends on the presence of a single individual, and the more noise we should add to protect their privacy.

Differential Privacy [DMNS06]

Definition

A randomised algorithm q is **ϵ-differentially private**
whenever for all inputs x and x' such that $d(x, x') = 1$,
the outputs $q(x)$ and $q(x')$ are ϵ-indistinguishable.

Differential Privacy [DMNS06]

Definition

A randomised algorithm q is **ϵ-differentially private**
whenever for all inputs x and x' such that $d(x, x') = 1$,
the outputs $q(x)$ and $q(x')$ are ϵ-indistinguishable.

If we know the sensitivity of an algorithm, then we can *automatically* make it private.

Differential Privacy [DMNS06]

Definition

A randomised algorithm q is **ϵ-differentially private**
whenever for all inputs x and x' such that $d(x, x') = 1$,
the outputs $q(x)$ and $q(x')$ are ϵ-indistinguishable.

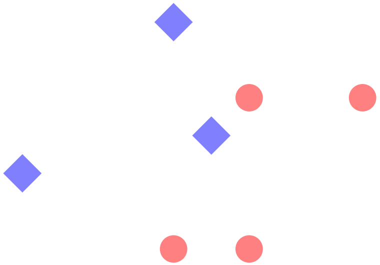
If we know the sensitivity of an algorithm, then we can *automatically* make it private.

Example theorem (Laplace mechanism)

If q is a s-sensitive query to $\mathbf{R}^n$ endowed with the L^1 metric,
then the function $q + \mathbf{Lap}_{s/\epsilon}$ is ϵ-differentially private.

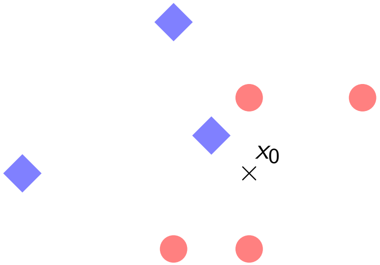
Example: Neighbour Classification

Given a database of labelled points in the plane $\mathbf{R}^2$,



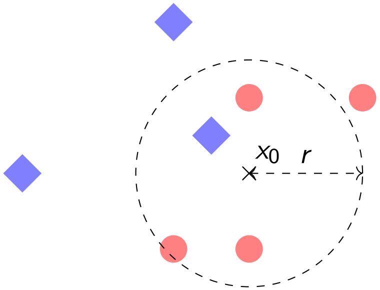
Example: Neighbour Classification

Given a database of labelled points in the plane $\mathbf{R}^2$,



Example: Neighbour Classification

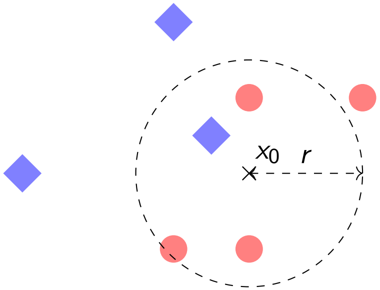
Given a database of labelled points in the plane $\mathbf{R}^2$,



we want to predict the label of a point x_0 by a majority vote weighted by the distance to its neighbours: weight: $d \mapsto 1 - 1/(1 + \mathrm{e}^{-4(d-r)})$.

Example: Neighbour Classification

Given a database of labelled points in the plane $\mathbf{R}^2$,



we want to predict the label of a point x_0 by a majority vote weighted by the distance to its neighbours: weight: $d \mapsto 1 - 1/(1 + \mathrm{e}^{-4(d-r)})$.

$$\text{weight}(0) \approx 1 \qquad \text{weight}(r) = 1/2 \qquad \lim_{d \to +\infty} \text{weight}(d) = 0$$

Scoring Function

```
let score (l : label) (db : database) : real
```

Scoring Function

```
let score (l : label) (db : database) : real  = db
```

```
let score (l : label) (db : database) : real  = db
        |> setfilter (fun r -> get_label r = l)
```

Scoring Function

```
let score (l : label) (db : database) : real  = db
        |> setfilter (fun r -> get_label r = l)
        |> setmap (fun r -> distance (0, 0) (get_pos r))
```

Scoring Function

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

Classification Algorithm

Remark

```
let predict (db : database) : label
        = argmax labels (fun l -> score l db)
```

where `argmax : a set -> (a -> real) -> a`.
It returns the best element according to some scoring function.

This implementation might leak private information.

Classification Algorithm

Remark

```
let predict (db : database) : label
      = argmax labels (fun l -> score l db)
```

where `argmax : a set -> (a -> real) -> a`.
It returns the best element according to some scoring function.

This implementation might leak private information.

But if we know the sensitivity of `score`, then we can approximatively maximise it in a private manner.

2. Linear Logic and Type Systems

Typing Judgements for Function Sensitivity

- $x : A \vdash b : B$
  means that $[\![b]\!]$ is a 1-sensitive function from $[\![A]\!]$ to $[\![B]\!]$,

Typing Judgements for Function Sensitivity

- $x : A \vdash b : B$
  means that $[\![b]\!]$ is a 1-sensitive function from $[\![A]\!]$ to $[\![B]\!]$,
- $[x : A]_s \vdash b : B$
  means that $[\![b]\!]$ is a 1-sensitive function from $!_s[\![A]\!]$ to $[\![B]\!]$,
  (where we define $!_s(X, d_X)$ as X endowed with the metric $(x, x') \mapsto s \cdot d_X(x, x')$).

Typing Judgements for Function Sensitivity

- $x : A \vdash b : B$
  means that $[\![b]\!]$ is a 1-sensitive function from $[\![A]\!]$ to $[\![B]\!]$,
- $[x : A]_s \vdash b : B$
  means that $[\![b]\!]$ is a 1-sensitive function from $!_s[\![A]\!]$ to $[\![B]\!]$,
  (where we define $!_s(X, d_X)$ as X endowed with the metric $(x, x') \mapsto s \cdot d_X(x, x')$).
  that is a s-sensitive function from $[\![A]\!]$ to $[\![B]\!]$,

Typing Judgements for Function Sensitivity

- $x : A \vdash b : B$
  means that $[\![b]\!]$ is a 1-sensitive function from $[\![A]\!]$ to $[\![B]\!]$,
- $[x : A]_s \vdash b : B$
  means that $[\![b]\!]$ is a 1-sensitive function from $!_s[\![A]\!]$ to $[\![B]\!]$,
  (where we define $!_s(X, d_X)$ as X endowed with the metric $(x, x') \mapsto s \cdot d_X(x, x')$).
  that is a s-sensitive function from $[\![A]\!]$ to $[\![B]\!]$,
- $[x_1 : A_1]_{s_1}, \ldots, [x_n : A_n]_{s_n} \vdash a : A$
  means that $[\![b]\!]$ is a 1-sensitive from $!_{s_1}[\![A_1]\!] \times \cdots \times !_{s_n}[\![A_n]\!]$ to $[\![B]\!]$,

Typing Judgements for Function Sensitivity

- $x : A \vdash b : B$
  means that $[\![b]\!]$ is a 1-sensitive function from $[\![A]\!]$ to $[\![B]\!]$,
- $[x : A]_s \vdash b : B$
  means that $[\![b]\!]$ is a 1-sensitive function from $!_s[\![A]\!]$ to $[\![B]\!]$,
  (where we define $!_s(X, d_X)$ as X endowed with the metric $(x, x') \mapsto s \cdot d_X(x, x')$).
  that is a s-sensitive function from $[\![A]\!]$ to $[\![B]\!]$,
- $[x_1 : A_1]_{s_1}, \ldots, [x_n : A_n]_{s_n} \vdash a : A$
  means that $[\![b]\!]$ is a 1-sensitive from $!_{s_1}[\![A_1]\!] \times \cdots \times !_{s_n}[\![A_n]\!]$ to $[\![B]\!]$,
  where the product is endowed with the L^1 metric.

Fuzz Typing Rules

Such judgements can be derived in a linearly typed lambda-calculus introduced by Reed and Pierce called Fuzz [RP10].

Fuzz Typing Rules

Such judgements can be derived in a linearly typed lambda-calculus introduced by Reed and Pierce called Fuzz [RP10].

$$\frac{s \geq 1}{[x : A]_s \vdash x : A}$$

Fuzz Typing Rules

Such judgements can be derived in a linearly typed lambda-calculus introduced by Reed and Pierce called Fuzz [RP10].

$$\frac{s \geq 1}{[x : A]_s \vdash x : A} \qquad \frac{\Gamma \vdash a : A \qquad \Delta \vdash b : B}{\Gamma + \Delta \vdash (a, b) : A \otimes B}$$

Fuzz Typing Rules

Such judgements can be derived in a linearly typed lambda-calculus introduced by Reed and Pierce called Fuzz [RP10].

$$\frac{s \geq 1}{[x : A]_s \vdash x : A} \qquad \frac{\Gamma \vdash a : A \qquad \Delta \vdash b : B}{\Gamma + \Delta \vdash (a, b) : A \otimes B} \qquad \frac{\Gamma, [x : A]_s \vdash b : B}{\Gamma \vdash \lambda x.b : !_s A \multimap B}$$

$[\![A \multimap B]\!] = [\![A]\!] \multimap [\![B]\!]$, that is the space of 1-sensitive functions from $[\![A]\!]$ to $[\![B]\!]$.

Fuzz Typing Rules

Such judgements can be derived in a linearly typed lambda-calculus introduced by Reed and Pierce called Fuzz [RP10].

$$\frac{s \geq 1}{[x : A]_s \vdash x : A} \qquad \frac{\Gamma \vdash a : A \qquad \Delta \vdash b : B}{\Gamma + \Delta \vdash (a, b) : A \otimes B} \qquad \frac{\Gamma, [x : A]_s \vdash b : B}{\Gamma \vdash \lambda x.b : !_s A \multimap B}$$

$[\![A \multimap B]\!] = [\![A]\!] \multimap [\![B]\!]$, that is the space of 1-sensitive functions from $[\![A]\!]$ to $[\![B]\!]$.

This way, we can derive *statically* an upper bound on the sensitivity of an algorithm written in a functional language.

Other Metrics

Question

What if we want to apply the Gaussian mechanism and know the L^2-sensitivity of an algorithm? More generally, what if we want to work on vectors with the L^p metric?

Recall. $d_p(\mathbf{x}, \mathbf{y}) = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p}$.

Bunched Fuzz Types [wABG23]

Bunched Fuzz is an extension of Fuzz with one product constructor $\otimes_p$ and one arrow constructor $\multimap_p$ for each p in $[1, +\infty]$.

$$\llbracket A \otimes_p B \rrbracket = (\llbracket A \rrbracket \times \llbracket B \rrbracket, d_p)$$
$$\llbracket A \multimap_p B \rrbracket = (\{\, \text{1-sensitive functions from } \llbracket A \rrbracket \text{ to } \llbracket B \rrbracket \,\}, d_p^*)$$

Bunched Fuzz Types [wABG23]

Bunched Fuzz is an extension of Fuzz with one product constructor $\otimes_p$ and one arrow constructor $\multimap_p$ for each p in $[1, +\infty]$.

$$[\![A \otimes_p B]\!] = ([\![A]\!] \times [\![B]\!], d_p)$$
$$[\![A \multimap_p B]\!] = (\{\, \text{1-sensitive functions from } [\![A]\!] \text{ to } [\![B]\!] \,\}, d_p^*)$$

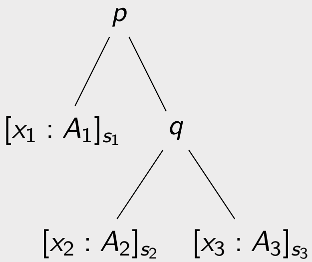
Example

Semantically, we have: $\mathrm{distance}((0,0), -) : \mathrm{Real} \otimes_2 \mathrm{Real} \multimap_2 \mathrm{Real}$.

Bunched Fuzz Contexts

Contexts are no longer lists, but trees (or *bunches*).

Example



$$[\![\Gamma]\!] = [\![A_1]\!] \otimes_p ([\![A_2]\!] \otimes_q [\![A_3]\!])$$

Bunched Fuzz Typing Rules

$$\frac{\Gamma \vdash a : A \qquad \Delta \vdash b : B}{\Gamma \,,_p \Delta \vdash (a, b) : A \otimes_p B}$$

Bunched Fuzz Typing Rules

$$\frac{\Gamma \vdash a : A \qquad \Delta \vdash b : B}{\Gamma \,,_p \Delta \vdash (a, b) : A \otimes_p B}$$

$$\frac{\Gamma([x : A]_s \,,_p [x' : A]_r) \vdash a : A}{\Gamma\Big([x : A]_{\sqrt[p]{s^p + r^p}}\Big) \vdash a[x/x'] : A}$$

Bunched Fuzz Typing Rules

$$\frac{\Gamma \vdash a : A \qquad \Delta \vdash b : B}{\Gamma \,,_p\, \Delta \vdash (a, b) : A \otimes_p B} \qquad \frac{\Gamma([x : A]_s \,,_p\, [x' : A]_r) \vdash a : A}{\Gamma\left([x : A]_{\sqrt[p]{s^p + r^p}}\right) \vdash a[x/x'] : A}$$

Example

$$\cfrac{\cfrac{[x : A]_1 \vdash x : A \qquad [x : A]_1 \vdash x : A}{[x : A]_1 \,,_2\, [x' : A]_1 \vdash (x, x') : A \otimes_2 A}}{[x : A]_{\sqrt{2}} \vdash (x, x) : A \otimes_2 A}$$

Lack of subject reduction

In Bunched Fuzz, from $\vdash a : A$ and $a \downarrow v$, we cannot always deduce $\vdash v : A$.

Intuitively, this comes from the fact that when we substitute a context Γ for a variable, the parameters in Γ alter the sensitivity analysis.

3. Contribution

The Plurimetric Fuzz Type System

Overview of Plurimetric Fuzz

We consider

- the same contexts as Fuzz, but annotated with a parameter p,
- the same types as Bunched Fuzz
  (in particular $\otimes_p$ and $\multimap_p$ for all p),
- recursive types and a form of subtyping.

Plurimetric Fuzz Typing Rules

Plurimetric Fuzz judgements have the following form:
$(p)\ [x_1 : A_1]_{s_1}, \ldots, [x_n : A_n]_{s_n} \vdash a : A.$

Plurimetric Fuzz Typing Rules

Plurimetric Fuzz judgements have the following form:
$(p)\ [x_1 : A_1]_{s_1}, \ldots, [x_n : A_n]_{s_n} \vdash a : A.$

Interpretation: $[\![(p)\ [x_1 : A_1]_{s_1}, \ldots, [x_n : A_n]_{s_n}]\!] = !_{s_1}[\![A_1]\!] \otimes_p \cdots \otimes_p !_{s_n}[\![A_n]\!]$

Plurimetric Fuzz Typing Rules

Plurimetric Fuzz judgements have the following form:
$(p)\ [x_1 : A_1]_{s_1}, \ldots, [x_n : A_n]_{s_n} \vdash a : A.$

Interpretation: $[\![(p)\ [x_1 : A_1]_{s_1}, \ldots, [x_n : A_n]_{s_n}]\!] = !_{s_1}[\![A_1]\!] \otimes_p \cdots \otimes_p !_{s_n}[\![A_n]\!]$

Example

$$\frac{(p)\ [x : C]_{s_a} \vdash a : A \qquad (p)\ [x : C]_{s_b} \vdash b : B}{(p)\ [x : C]_{\sqrt[p]{s_a^p + s_b^p}} \vdash (a, b) : A \otimes_p B}$$

Plurimetric Fuzz Typing Rules

Plurimetric Fuzz judgements have the following form:
$(p)\ [x_1 : A_1]_{s_1}, \ldots, [x_n : A_n]_{s_n} \vdash a : A$.

Interpretation: $[\![(p)\ [x_1 : A_1]_{s_1}, \ldots, [x_n : A_n]_{s_n}]\!] = !_{s_1}[\![A_1]\!] \otimes_p \cdots \otimes_p !_{s_n}[\![A_n]\!]$

Example

$$\frac{(p)\ [x : C]_{s_a} \vdash a : A \qquad (p)\ [x : C]_{s_b} \vdash b : B}{(p)\ [x : C]_{\sqrt[p]{s_a^p + s_b^p}} \vdash (a, b) : A \otimes_p B}$$

Subtyping rules allow for modifying the parameter of a context by appropriately adjusting the sensitivity of its variables.

Lemma

For all parameters p and for all metric complete partial orders (CPOs) X and Y, the spaces $X \otimes_p Y$ and $X \multimap_p Y$ are also metric CPOs.

Recursive Types

Lemma

For all parameters p and for all metric complete partial orders (CPOs) X and Y, the spaces $X \otimes_p Y$ and $X \multimap_p Y$ are also metric CPOs.

Theorem ([AGH+17])

$\mathbf{MetCPO}_{\perp}$, *the category of metric CPOs and continuous non-expansive functions, is cartesian closed and algebraically compact.*

As we can consequence we can solve the domain equations associated with the recursive types of the (deterministic fragment) of the language.

Metatheoretical Properties

Theorem (Subject Reduction)

In Plurimetric Fuzz, from $\vdash a : A$ *and* $a \downarrow v$*, we can deduce* $\vdash v : A$*.*

Theorem (Adequacy)

If $\vdash a : A$ *and* $[\![a]\!] \neq \bot$*, then there exists a value* v *such that* $a \downarrow v$*.*

... and metric preservation, etc.

4. Back to our example

Neighbour classification

Recall

We want to predict the label of a point x_0 by a majority vote weighted by the distance to its neighbours, that is we want to find the label that maximises the following function.

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

Sensitivity Analysis of the Scoring Function

We consider the following types:
$\mathsf{Point} = \mathsf{Real} \otimes_2 \mathsf{Real}$, $\mathsf{Row} = \mathsf{Label} \otimes_1 \mathsf{Point}$, $\mathsf{Database} = \mathsf{Set}(\mathsf{Row})$.

Sensitivity Analysis of the Scoring Function

We consider the following types:
Point $=$ Real $\otimes_2$ Real, Row $=$ Label $\otimes_1$ Point, Database $=$ Set(Row).

`distance (0, 0)` is 1-sensitive from Real $\otimes_2$ Real to Real.

Sensitivity Analysis of the Scoring Function

We consider the following types:
Point $=$ Real $\otimes_2$ Real, Row $=$ Label $\otimes_1$ Point, Database $=$ Set(Row).

`distance (0, 0)` is 1-sensitive from Real $\otimes_2$ Real to Real.
The set primitives `setfold`, `setmap`, `setsum`, etc. are all 1-sensitive.

Sensitivity Analysis of the Scoring Function

We consider the following types:
Point $=$ Real $\otimes_2$ Real, Row $=$ Label $\otimes_1$ Point, Database $=$ Set(Row).

`distance (0, 0)` is 1-sensitive from Real $\otimes_2$ Real to Real.
The set primitives `setfold`, `setmap`, `setsum`, etc. are all 1-sensitive.

We can derive the following judgement for the scoring function:
$\vdash$ score : Label $\otimes_1$ Database $\multimap_1$ Real.

Sensitivity Analysis of the Scoring Function

We consider the following types:
Point $=$ Real $\otimes_2$ Real, Row $=$ Label $\otimes_1$ Point, Database $=$ Set(Row).

`distance (0, 0)` is 1-sensitive from Real $\otimes_2$ Real to Real.
The set primitives `setfold`, `setmap`, `setsum`, etc. are all 1-sensitive.

We can derive the following judgement for the scoring function:
$\vdash$ score : Label $\otimes_1$ Database $\multimap_1$ Real.

Theorem

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

This implementation is 1-differentially private.

We have proven that it does not leak any private information from the database.

5. Conclusion

Contributions

We have introduced Plurimetric Fuzz:

- an extension of Fuzz to L^p metrics,
- including recursive types and a form of subtyping,
- which enjoys the subject reduction property.

Additionally, we have studied translations from and to Fuzz (see the paper for more information).

Future work

Future work might address the following problems:

- type checking and type inference
  (sensitivity constraints are not linear)
- a denotational semantics that handles both probability and recursive types.

References I

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.

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.

References II

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.

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.

Differential Privacy

Definition

Let M be a randomised algorithm from an input (X, d_X) to an output space Y.
We say that M is ϵ-differentially private whenever for all adjacent inputs x and x' (that is for all x and x' in X such that $d_X(x, x') = 1$), and for all subsets S of Y,

$$\Pr[M(x) \in S] \leq e^{\epsilon} \cdot \Pr[M(x') \in S] + \delta\,.$$

Lack of Subject Reduction

In Bunched Fuzz:

Lack of the substitution property

From $\Gamma \vdash a : A$ and $\Delta([x : A]_s) \vdash b : B$, we cannot always deduce $\Delta(s\Gamma) \nvdash b[a/x] : B$.

This arises from the fact that we obtain different sensitivity analyses depending on whether the substitution is performed before or after applying a contraction rule.

Lack of Subject Reduction

In Bunched Fuzz:

Lack of the substitution property

From $\Gamma \vdash a : A$ and $\Delta([x : A]_s) \vdash b : B$, we cannot always deduce $\Delta(s\Gamma) \nvdash b[a/x] : B$.

This arises from the fact that we obtain different sensitivity analyses depending on whether the substitution is performed before or after applying a contraction rule.

Lack of subject reduction

From $\vdash a : A$ and $a \downarrow v$, we cannot always deduce $\vdash v : A$.

Translation mappings

For all parameters p in $[1, +\infty]$, the following diagram commutes: