Type-checking Linear Dependent Types

Arthur Azevedo de Amorim\textsuperscript{1,2}  Marco Gaboardi\textsuperscript{3}  Emilio Jesús Gallego Arias\textsuperscript{1}
Justin Hsu\textsuperscript{1}

\textsuperscript{1}University of Pennsylvania
\textsuperscript{2}INRIA Paris-Rocquencourt
\textsuperscript{3}University of Dundee
Movie Ratings \rightarrow \text{Anonymization} \rightarrow \text{Internet}
How to allow database queries and retain privacy guarantees?
Differential Privacy

- Rigorous bound on “privacy loss” [Dwork, 2006]
- Informally: adding one’s data doesn’t change query results by much
- Many available algorithms
  - Statistical analyses, combinatorial optimizations, machine learning, ...
Ensuring Differential Privacy

Deterministic Query + Random Noise → Differentially Private Probabilistic Query
Ensuring Differential Privacy

\[ \text{Random Noise } \propto k \]

\[ \text{k-sensitive Deterministic Query} \rightarrow \text{+} \rightarrow \text{Differentially Private Probabilistic Query} \]
Ensuring Differential Privacy

$k$-sensitive Deterministic Query

$\alpha$ Random Noise $k$

Bound on result variation

Differentially Private Probabilistic Query
Ensuring Differential Privacy

$k$-sensitive Deterministic Query

Random Noise $\propto k$

Bound on result variation

Requires tedious proofs

Differentially Private Probabilistic Query
Types to the Rescue

- **DFuzz** [Reed&Pierce10,Gaboardi13] is a type system for function sensitivity (hence, differential privacy)

- Capable of expressing many differentially private algorithms

- Metatheory ensures differential privacy

- Type-checking algorithm: proof automation
Challenge: Checking and Inference

The DFuzz type system combines interesting features:

- Linear indexed types
- Dependent types
- Subtyping

Their interplay makes it difficult to reuse existing techniques directly
Our Contributions

- A type-checking and type-inference algorithm for a system combining linear and dependent types in the presence of subtyping

- Showing how ideas from the type-checking literature for those domains can be adapted to a type system built around a special-purpose index language
Outline

- DFuzz and function sensitivity
- Type checking and inference for DFuzz
DFuzz and Function Sensitivity
Function Sensitivity

Bound output variation based on input variation

\[ f \text{ is } k\text{-sensitive}: d(f(x), f(y)) \leq k \cdot d(x, y) \]
Function Sensitivity

Bound output variation based on input variation

\[ f \text{ is } k\text{-sensitive}: d(f(x), f(y)) \leq k \cdot d(x, y) \]
Function Sensitivity

\[ F(x) \leq k \cdot D \]

\[ F(y) \leq k \cdot D \]
DFuzz in a Nutshell

• $!_{k\sigma} \rightarrow \tau$: $k$-sensitive function (= linear)
DFuzz in a Nutshell

- $!_k \sigma \rightarrow \tau$: $k$-sensitive function (= linear)

Multivariate polynomial
DFuzz in a Nutshell

- $!_k\sigma \xrightarrow{} \tau$: $k$-sensitive function (= linear)

- $\text{list}_n \sigma$: list of length $n$ (mechanisms that depend on input size)
DFuzz in a Nutshell

- $!_k \sigma \rightarrow \tau$: $k$-sensitive function (= linear)
- $list_n \sigma$: list of length $n$ (mechanisms that depend on input size)
- $\sigma \sqsubseteq \tau$: sensitivity weakening (e.g. $(!_1 \sigma \rightarrow \tau) \sqsubseteq (!_2 \sigma \rightarrow \tau)$)
A Basic Example

Consider the standard map function

```
function map f l {
    case l of {
        | [] => []
        | x :: l' => f x :: map f l'
    }
}
```

How to bound “distance” between results of two calls?
Analyzing Example

\[ d(\text{map}(f, \vec{x}), \text{map}(g, \vec{y})) \]
Analyzing Example

\[ d(map(f, \vec{x}), map(g, \vec{y})) \]

\[ \leq n \sum_{I=1}^{n} \left[ D(F(x_I), g(y_I)) + D(g(x_I), g(y_I)) \right] \]

\[ = n \cdot D(F, g) + k \cdot D(\vec{x}, \vec{y}) \]

\( k \)-sensitive length for lists 4 triangle inequality - a x difference between \( F \) and \( g \) definition of sensitivity
Analyzing Example

\[ d(\text{map}(f, \overline{x}), \text{map}(g, \overline{y})) \leq n \sum_{I=1}^{n} \left[ d(\text{map}(f, x_I), g(y_I)) + d(g(x_I), g(y_I)) \right] \leq n \cdot D(F, g) + k \cdot D(\overline{x}, \overline{y}) \]
Analyzing Example

\[ d(\text{map}(f, \bar{x}), \text{map}(g, \bar{y})) = \sum_{i=1}^{n} d(f(x_i), g(y_i)) \]

Distance for lists

14
Analyzing Example

\[ d(map(f, \vec{x}), map(g, \vec{y})) = \sum_{i=1}^{n} d(f(x_i), g(y_i)) \leq \sum_{i=1}^{n} [d(f(x_i), g(x_i)) + d(g(x_i), g(y_i))] \]

Triangle inequality
Analyzing Example

\[ d(\text{map}(f, \bar{x}), \text{map}(g, \bar{y})) \]
\[ = \sum_{i=1}^{n} d(f(x_i), g(y_i)) \]
\[ \leq \sum_{i=1}^{n} [d(f(x_i), g(x_i)) + d(g(x_i), g(y_i))] \]
\[ \leq \sum_{i=1}^{n} [d(f, g) + k \cdot d(x_i, y_i)] \]

Max difference between \( f \) and \( g \)
Analyzing Example

\[ d(\text{map}(f, \bar{x}), \text{map}(g, \bar{y})) \]

\[ = \sum_{i=1}^{n} d(f(x_i), g(y_i)) \]

\[ \leq \sum_{i=1}^{n} [d(f(x_i), g(x_i)) + d(g(x_i), g(y_i))] \]

\[ \leq \sum_{i=1}^{n} [d(f, g) + k \cdot d(x_i, y_i)] \]

Definition of sensitivity
Analyzing Example

\[ d(\text{map}(f, \vec{x}), \text{map}(g, \vec{y})) \]

\[ = \sum_{i=1}^{n} d(f(x_i), g(y_i)) \]

\[ \leq \sum_{i=1}^{n} [d(f(x_i), g(x_i)) + d(g(x_i), g(y_i))] \]

\[ \leq \sum_{i=1}^{n} [d(f, g) + k \cdot d(x_i, y_i)] \]

\[ = n \cdot d(f, g) + k \cdot d(\vec{x}, \vec{y}) \]
Typing Example in DFuzz

\[ \text{map} : !_n(!_k\sigma \to \tau) \to !_k\text{list}_n \sigma \to \text{list}_n \tau \]
Typing Example in DFuzz

\[
\text{map} : \forall n(\forall k \sigma \rightarrow \tau) \rightarrow \forall k \text{list}_n \sigma \rightarrow \text{list}_n \tau
\]
Some Rules

\[
\frac{\Gamma, x : k \sigma \vdash e : \tau}{\Gamma \vdash \lambda x : k \sigma . e : !_k \sigma \to \tau} \quad (\to I)
\]
Some Rules

Keep track of sensitivity

\[ \frac{\Gamma, x : k \sigma \vdash e : \tau}{\Gamma \vdash \lambda x : k \sigma. e : !_{k \sigma} \rightarrow^* \tau} \quad (\rightarrow^* l) \]
Some Rules

\[
\frac{\Gamma, x : k \sigma \vdash e : \tau}{\Gamma \vdash \lambda x : k \sigma . e : !_k \sigma \rightarrow \tau} \quad (\rightarrow I)
\]

Propagate sensitivity to type
Some Rules

$\Gamma \vdash e_1 : !k\sigma \rightarrow \tau \quad \Delta \vdash e_2 : \sigma$

$\Gamma + k \cdot \Delta \vdash e_1 \ e_2 : \tau$

(→ E)

Context split, combine sensitivities
Some Rules

\[ \Gamma \vdash e_1 : !k\sigma \rightarrow \tau \quad \Delta \vdash e_2 : \sigma \]

\[ \frac{\Gamma \vdash k \cdot \Delta \vdash e_1 \; e_2 : \tau}{\Gamma \vdash e_1 \; e_2 : \tau} \quad (\rightarrow E) \]

Composition: multiply sensitivities
Some Rules

\[
\begin{align*}
\Delta & \vdash e : \text{list}_n \sigma \\
\Gamma & \vdash e_{\text{nil}} : \tau \\
\Gamma, \ h : k \sigma, \ t : k \text{list}_i \sigma & \vdash e_{\text{cons}} : \tau \\
\Gamma + k \cdot \Delta & \vdash \text{case e of } [] \rightarrow e_{\text{nil}} \mid h :: t \rightarrow e_{\text{cons}} : \tau
\end{align*}
\]
Some Rules

Assuming $n = 0$

$$\Delta \vdash e : \text{list}_n$$

$$\Gamma \vdash e_{\text{nil}} : \tau$$

$$\Gamma, h : k \sigma, t : k \text{list}_i \sigma \vdash e_{\text{cons}} : \tau$$

$$\Gamma + k \cdot \Delta \vdash \text{case } e \text{ of } \texttt{[]} \rightarrow e_{\text{nil}} \mid h :: t \rightarrow e_{\text{cons}} : \tau$$

(list E)
Some Rules

Assuming $n = i + 1$

\[
\Delta \vdash e : list_n \sigma \\
\Gamma \vdash e_{\text{nil}} : \tau \\
\Gamma, h : k \sigma, t : k \text{ list}_i \sigma \vdash e_{\text{cons}} : \tau \\
\Gamma + k \cdot \Delta \vdash \text{case } e \text{ of } [] \rightarrow e_{\text{nil}} \mid h :: t \rightarrow e_{\text{cons}} : \tau
\]

(list E)
Some Rules

\[ \Delta \vdash e : \text{list}_n \sigma \]
\[ \Gamma \vdash e_{\text{nil}} : \tau \]
\[ \Gamma, h : k \sigma, t : k \text{list}_i \sigma \vdash e_{\text{cons}} : \tau \]

\[ \Gamma + k \cdot \Delta \vdash \text{case } e \text{ of } [] \rightarrow e_{\text{nil}} | h :: t \rightarrow e_{\text{cons}} : \tau \]

(list E)

Track sensitivity on list
Type Checking and Inference
Plan

DFuzz Program
Plan

DFuzz Program

\[ \sigma \]

\[ \tau \]

\[ \vdash \]

Provided by annotation

Inference
Plan

DFuzz Program

Inference

Constraints

\( \sigma \)
Plan

DFuzz Program

Inference

Constraints

$\sigma$

Polynomial inequalities from subtyping
Plan

DFuzz Program

Inference

Constraints → Solver

σ
Plan

DFuzz Program

Inference

Constraints

Solver

Yes

$\sigma$
Plan

DFuzz Program

Inference

Constraints → Solver

σ

∈ τ?
Plan

DFuzz Program

Inference

Constraints → Solver

Provided by annotation
Plan

DFuzz Program

Inference

Constraints

Solver

Yes

$\sigma$ \n
$\subseteq \tau$?
Important Points

• Context splitting imposes a **bottom-up** strategy: start with leaves, combine sensitivities progressively
Important Points

- Context splitting imposes a **bottom-up** strategy: start with leaves, combine sensitivities progressively.

\[ e_1 \ldots e_2 \ldots \]
Important Points

- Context splitting imposes a **bottom-up** strategy: start with leaves, combine sensitivities progressively.

\[
e_1 : \sigma_1 \quad e_2 : \sigma_2
\]

\[
\ldots e_1 \ldots e_2 \ldots
\]
Important Points

- Context splitting imposes a **bottom-up** strategy: start with leaves, combine sensitivities progressively.
Important Points

- Context splitting imposes a bottom-up strategy: start with leaves, combine sensitivities progressively.

- Restrict subtyping to essential places (e.g. application).
Important Points

- Context splitting imposes a **bottom-up** strategy: start with leaves, combine sensitivities progressively
- Restrict subtyping to essential places (e.g. application)
- Assume sensitivities on higher-order types are given
Important Points

- Context splitting imposes a **bottom-up** strategy: start with leaves, combine sensitivities progressively

- Restrict subtyping to essential places (e.g. application)

- Assume sensitivities on higher-order types are given
  
  E.g. \( \!_k (\!_k \alpha \rightarrow \alpha) \rightarrow \alpha \)
Problems

- Language not rich enough to express minimal sensitivities
  - E.g. point-wise maximum of two polynomials is not a polynomial
  - Solution: enrich sensitivity language with new operators
Problems

• Language not rich enough to express minimal sensitivities
  • E.g. point-wise maximum of two polynomials is not a polynomial
  • Solution: enrich sensitivity language with new operators

cf. literature on subtyping
Problems

- Language not rich enough to express minimal sensitivities
  - E.g. point-wise maximum of two polynomials is not a polynomial
  - Solution: enrich sensitivity language with new operators

- Type checking is undecidable
  - Can encode equality of integer polynomials (Hilbert’s tenth problem)
  - Completeness relative to a decider of sensitivity inequalities
Syntax-Directed Rules

Equivalent to previous ones, but directly translatable to algorithm

**Input**  Term, argument type annotations

**Output**  Minimal sensitivities, minimal type
Syntax-Directed Rules

\[
\frac{
\Gamma \vdash e_1 : !_k \sigma \rightarrow \tau \quad \Delta \vdash e_2 : \sigma'}{
\sigma' \sqsubseteq \sigma}
\]

\[
\frac{
\sigma' \sqsubseteq \sigma}
{\Gamma + k \cdot \Delta \vdash e_1 \ e_2 : \tau}
\]  
(\rightarrow E)
Syntax-Directed Rules

\[ \Gamma \vdash e_1 : k\sigma \quad \Delta \vdash e_2 : \sigma' \]

\[ \sigma' \sqsubseteq \sigma \]

\[ \Gamma + k \cdot \Delta \vdash e_1 \ e_2 : \tau \]

\(\text{(}\circ E\text{)}\)

Not necessarily equal
Syntax-Directed Rules

\[ \Gamma \vdash e_1 : !k\sigma \xrightarrow{\circ} \tau \quad \Delta \vdash e_2 : \sigma' \]

\[ \sigma' \sqsubseteq \sigma \]

\[ \Gamma + k \cdot \Delta \vdash e_1 \ e_2 : \tau \]

(Subtype check)

\((\circ E)\)
Syntax-Directed Rules

\[
\Delta \vdash e : \text{list}_n \sigma \\
\Gamma \vdash e_{\text{nil}} : \tau_{\text{nil}} \\
\Gamma, h : k \sigma, t : k \text{list}_i \sigma \vdash e_{\text{cons}} : \tau_{\text{cons}}
\]

\[
\tau = \text{case}(n, \tau_{\text{nil}}, i, \tau_{\text{cons}})
\]

\[
\Gamma + k \cdot \Delta \vdash \text{case } e \text{ of } [] \rightarrow e_{\text{nil}} \mid h :: t \rightarrow e_{\text{cons}} : \tau
\]

\text{(list E)}
Syntax-Directed Rules

\[ \Delta \vdash e : \text{list}_n \sigma \]
\[ \Gamma \vdash e_{\text{nil}} : \tau_{\text{nil}} \]
\[ \Gamma, h : k \sigma, t : k \text{list}_i \sigma \vdash e_{\text{cons}} : \tau_{\text{cons}} \]

\[ n = \text{case}(n, \tau_{\text{nil}}, i, \tau_{\text{cons}}) \]

\[ \Gamma + k \cdot \Delta \vdash \text{case } e \text{ of } [] \rightarrow e_{\text{nil}} | h :: t \rightarrow e_{\text{cons}} : \tau \]

Sensitivity-level case lifted to types
Solver Integration

- Need to convert constraints so that standard solvers understand them
- Avoid alternating quantifiers
\[ k \geq \text{case}(n, k_0, i, k_s) \]

\[ (n = 0 \Rightarrow k \geq k_0) \land (\forall i, n = i + 1 \Rightarrow k \geq k_s) \]
Wrapping Up
Conclusion

- Type-checking system with linear and dependent types
- Standard ideas adapted to exploit application domain and index structure
- Recover minimal sensitivities by extending index language
  - As simple as possible, no need for much expressive power (cf [DalLago&Petit13])
Implementation

Available at
http://cis.upenn.edu/~emilioga/dFuzz.tar.gz

Capable of checking most of the original DFuzz examples
Future Directions

- Let-generalization for sensitivities (remove higher-order annotations)
- Identify decidable fragment of DFuzz
Questions?
Some Metric Spaces

\[ d_{\mathbb{R}}(x, y) = |x - y| \]

\[ d_{\sigma \to \tau}(f, g) = \sup_{x \in \sigma} d_{\tau}(f(x), g(x)) \]

\[ d_{\text{list } \sigma}(l_1, l_2) = \begin{cases} \infty & \text{if } \text{length}(l_1) \neq \text{length}(l_2) \\ \sum_i d_{\sigma}(l_1[i], l_2[i]) & \text{otherwise} \end{cases} \]

\[ d_{\text{set } \sigma}(s_1, s_2) = |s_1 \setminus s_2 \cup s_2 \setminus s_1| \]

\[ d_{\mathcal{P}(\sigma)}(\mu, \nu) = \int_{\sigma} \log \left( \frac{d\mu}{d\nu} \right) d\mu \]
More Typing Rules

\[ \Gamma, x : 1 \sigma \vdash x : \sigma \]  
(Var)
More Typing Rules

\[
\Gamma, x : \sigma \vdash e : \sigma \\
\infty \cdot \Gamma \vdash \text{fix } x : \sigma.e : \sigma
\]

(Fix)
More Typing Rules

\[
\Delta \subseteq \Gamma \quad \Gamma \vdash e : \sigma \quad \sigma \subseteq \tau \\
\Delta \vdash e : \tau 
\]  
\((\subseteq)\)
Suppose

\[ \vdash e : !_k \sigma \rightarrow \tau \]
\[ \vdash v_1 : \sigma \]
\[ \vdash v_2 : \sigma \]
\[ e \ v_1 \rightarrow^* \ v'_1 \]

There exists \( v'_2 \) such that \( e \ v_2 \rightarrow^* \ v'_2 \) and

\[ d_{\tau}(v'_1, v'_2) \leq k \cdot d_{\sigma}(v_1, v_2) \]