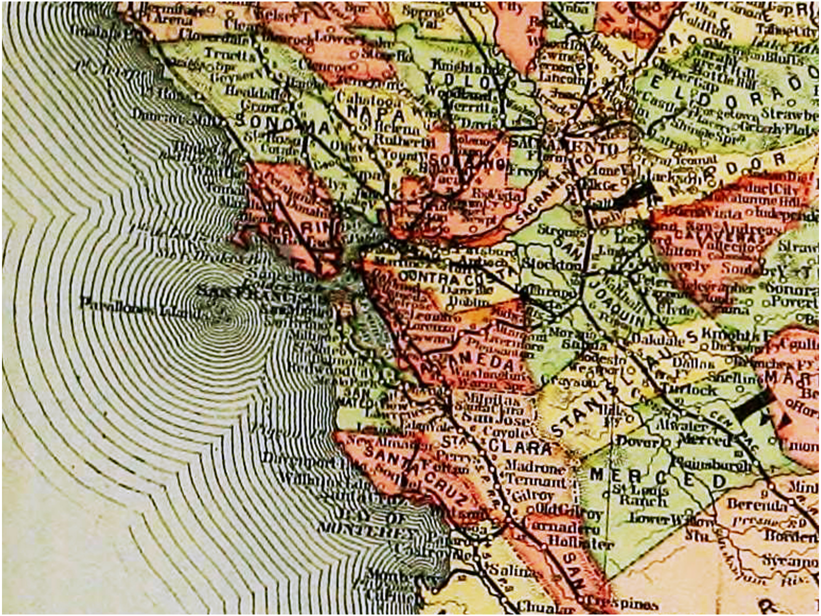
Type-checking Linear Dependent Types

Arthur Azevedo de Amorim [1 2] Marco Gaboardi [3] Emilio Jesús Gallego Arias [1]
Justin Hsu [1]

[1]University of Pennsylvania

[2]INRIA Paris-Rocquencourt

[3]University of Dundee


SONOMA
NAPA
YOLO
SACRAMENTO
SOLANO
MARIN
CONTRA COSTA
ALAMEDA
SAN JOAQUIN
STANISLAUS
MERCED
SANTA CRUZ
STA CLARA
EL DORADO
CALAVERAS
BAY OF MONTEREY
Calistoga
Woodland
Lincoln
Stockton
Danville
Dublin
Livermore
Milpitas
San Jose
Coyote
Madrone
Tennant
Gilroy
Old Gilroy
Hollister
Salinas
Tres pinos
Modesto
Turlock
Merced
Dover
Berenda
Borden
Sonora


NETFLIX


Movie Ratings
Anonymization
Internet

DEBENT IGNARI RES FERRE ET POST OPERARI
IVS LAPIDIS CARI VILIS SED DENIQ3 RARI
VNICA RES CERTA VILIS SED VBIQ3 REPERTA

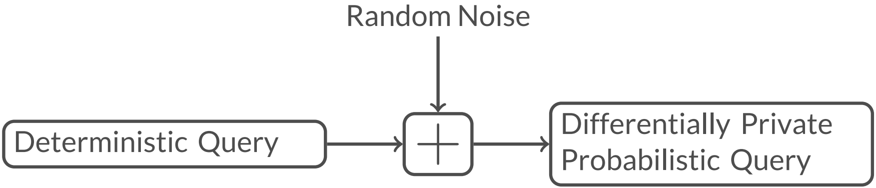
QVATVOR INSERTA NATVRIS IN NVBE REFERTA
NVLLA MINERALIS RES EST VBI PRINCIPALIS
SED TALIS QVALIS REPERITVR VBIQ3 LOCALIS

How to allow database queries and retain privacy guarantees?

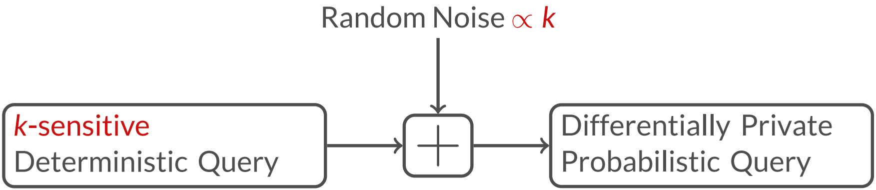
Differential Privacy

- 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, …

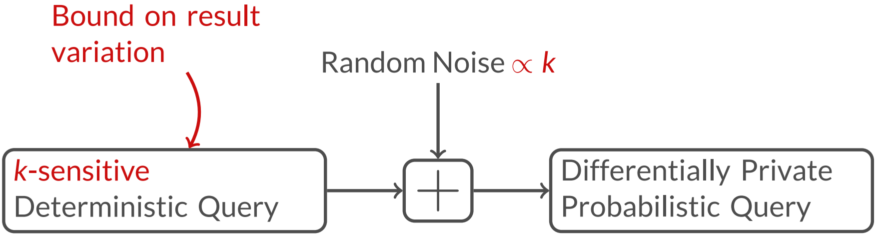
Ensuring Differential Privacy



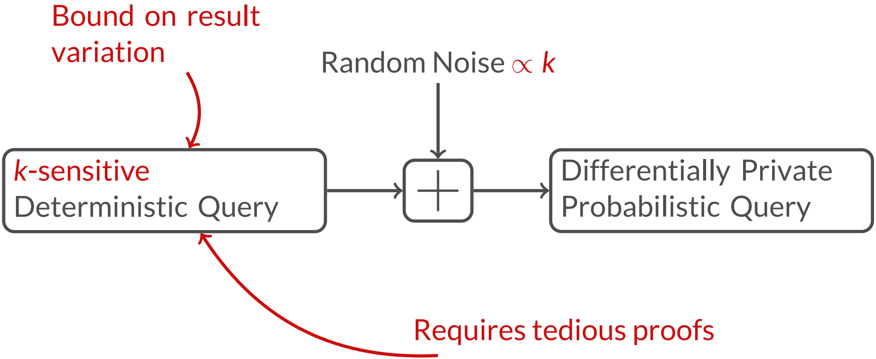
Ensuring Differential Privacy



Ensuring Differential Privacy



Ensuring Differential Privacy



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 is k-sensitive: $d(f(x), f(y)) \leq k \cdot d(x, y)$

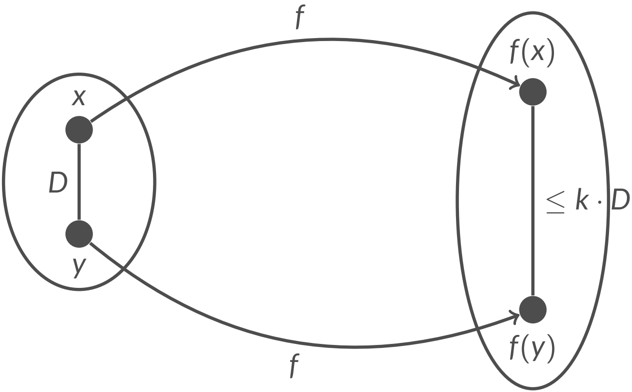
Function Sensitivity

Bound output variation based on input variation

f is *k-sensitive*: $d(f(x), f(y)) \leq k \cdot d(x, y)$

Distance functions

Function Sensitivity



DFuzz in a Nutshell

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

DFuzz in a Nutshell

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

Multivariate polynomial

DFuzz in a Nutshell

- $!_k \sigma \multimap \tau$: *k*-sensitive function (= linear)
- $list_n\ \sigma$: list of length *n* (mechanisms that depend on input size)

DFuzz in a Nutshell

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

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

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

Analyzing Example

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

k-sensitive

Analyzing Example

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

length n

Analyzing Example

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

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

Distance for lists

Analyzing Example

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

Triangle inequality

Analyzing Example

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

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

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

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

Max difference between f and g

Analyzing Example

$$
\begin{aligned}
& d(map(f, \vec{x}), map(g, \vec{y})) \\
&= \sum_{i=1}^{n} d(f(x_i), g(y_i)) \\
&\leq \sum_{i=1}^{n} [d(f(x_i), g(x_i)) + d(g(x_i), g(y_i))] \\
&\leq \sum_{i=1}^{n} [d(f, g) + k \cdot d(x_i, y_i)]
\end{aligned}
$$

Definition of sensitivity

Analyzing Example

$$
\begin{aligned}
& d(map(f, \vec{x}), map(g, \vec{y})) \\
&= \sum_{i=1}^{n} d(f(x_i), g(y_i)) \\
&\leq \sum_{i=1}^{n} [d(f(x_i), g(x_i)) + d(g(x_i), g(y_i))] \\
&\leq \sum_{i=1}^{n} [d(f, g) + k \cdot d(x_i, y_i)] \\
&= n \cdot d(f, g) + k \cdot d(\vec{x}, \vec{y})
\end{aligned}
$$

Typing Example in DFuzz

$$map : !_n(!_k\sigma \multimap \tau) \multimap !_k list_n\ \sigma \multimap list_n\ \tau$$

Typing Example in DFuzz

$$map : !_n(!_k\sigma \multimap \tau) \multimap !_k list_n\ \sigma \multimap list_n\ \tau$$

Argument sensitivities

Some Rules

$$\frac{\Gamma, x :_k \sigma \vdash e : \tau}{\Gamma \vdash \lambda x :_k \sigma . e : !_k \sigma \multimap \tau} \quad (\multimap I)$$

Some Rules

Keep track of sensitivity

$$\frac{\Gamma, x :_k \sigma \vdash e : \tau}{\Gamma \vdash \lambda x :_k \sigma . e : !_k \sigma \multimap \tau} \quad (\multimap I)$$

Some Rules

$$\frac{\Gamma, x :_k \sigma \vdash e : \tau}{\Gamma \vdash \lambda x :_k \sigma . e : !_k \sigma \multimap \tau} \quad (\multimap I)$$

Propagate sensitivity to type

Some Rules

$$\frac{\Gamma \vdash e_1 : !_k \sigma \multimap \tau \qquad \Delta \vdash e_2 : \sigma}{\Gamma + k \cdot \Delta \vdash e_1\ e_2 : \tau} \quad (\multimap E)$$

Context split, combine sensitivities

Some Rules

$$\frac{\Gamma \vdash e_1 :!_k \sigma \multimap \tau \qquad \Delta \vdash e_2 : \sigma}{\Gamma + k \cdot \Delta \vdash e_1\ e_2 : \tau} \quad (\multimap E)$$

Composition: multiply sensitivities

Some Rules

$$\frac{\begin{array}{c}\Delta \vdash e : list_n\ \sigma \\ \Gamma \vdash e_{nil} : \tau \\ \Gamma, h :_k \sigma, t :_k list_i\ \sigma \vdash e_{cons} : \tau\end{array}}{\Gamma + k \cdot \Delta \vdash case\ e\ of\ [] \to e_{nil} \mid h :: t \to e_{cons} : \tau} \quad (list\ E)$$

Some Rules

Assuming $n = 0$

$$\frac{\Delta \vdash e : list_n\ \sigma \qquad \Gamma \vdash e_{nil} : \tau \qquad \Gamma, h :_k \sigma, t :_k list_i\ \sigma \vdash e_{cons} : \tau}{\Gamma + k \cdot \Delta \vdash case\ e\ of\ [] \rightarrow e_{nil} \mid h :: t \rightarrow e_{cons} : \tau} \quad (list\ E)$$

Some Rules

Assuming $n = i + 1$

$$\frac{\Delta \vdash e : list_n\ \sigma \qquad \Gamma \vdash e_{nil} : \tau \qquad \Gamma, h :_k \sigma, t :_k list_i\ \sigma \vdash e_{cons} : \tau}{\Gamma + k \cdot \Delta \vdash case\ e\ of\ [] \rightarrow e_{nil} \mid h :: t \rightarrow e_{cons} : \tau}\ (list\ E)$$

Some Rules

$$\frac{\Delta \vdash e : list_n\ \sigma \qquad \Gamma \vdash e_{nil} : \tau \qquad \Gamma, h :_k \sigma, t :_k list_i\ \sigma \vdash e_{cons} : \tau}{\Gamma + k \cdot \Delta \vdash case\ e\ of\ [] \rightarrow e_{nil} \mid h :: t \rightarrow e_{cons} : \tau}\ (list\ E)$$

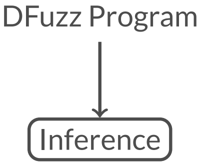
Track sensitivity on list

Type Checking and Inference

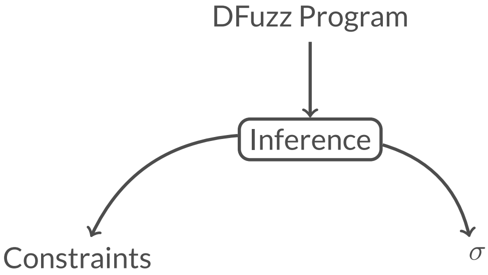
Plan

DFuzz Program

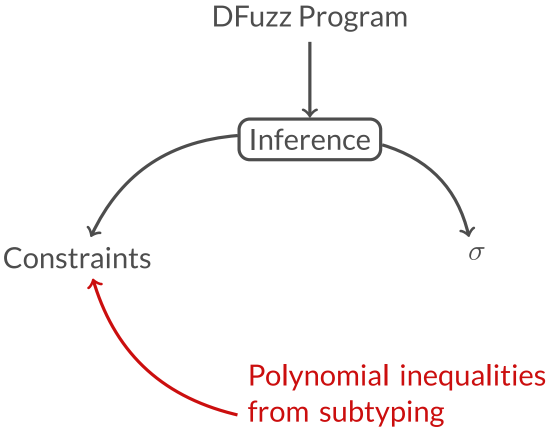
Plan



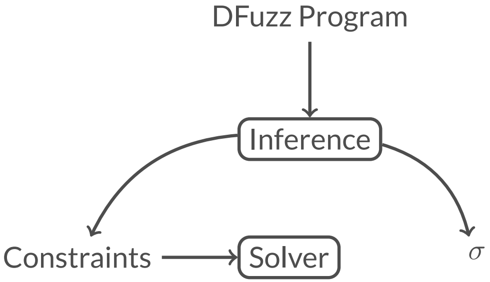
Plan



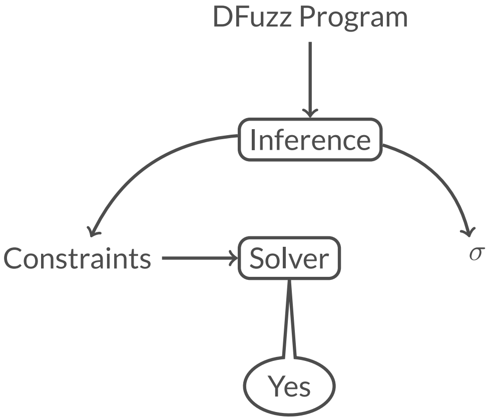
Plan


DFuzz Program
Inference
Constraints
σ
Polynomial inequalities
from subtyping

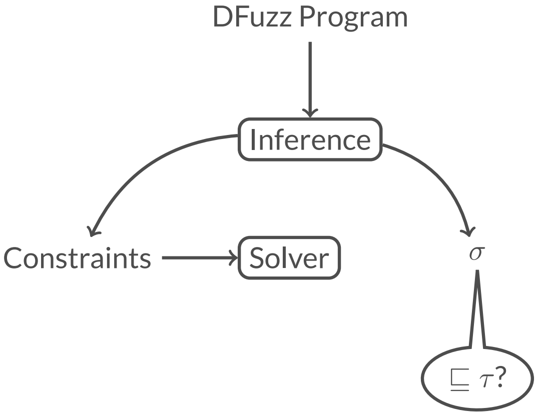
Plan



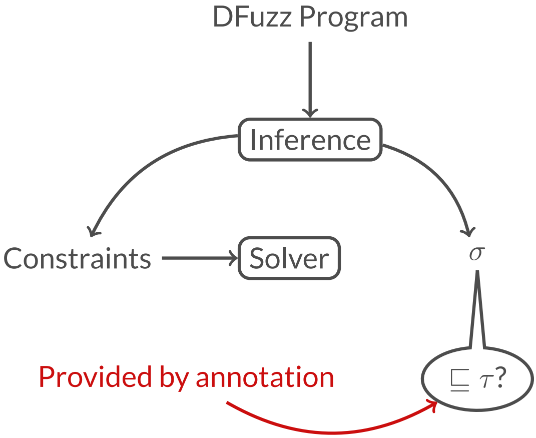
Plan



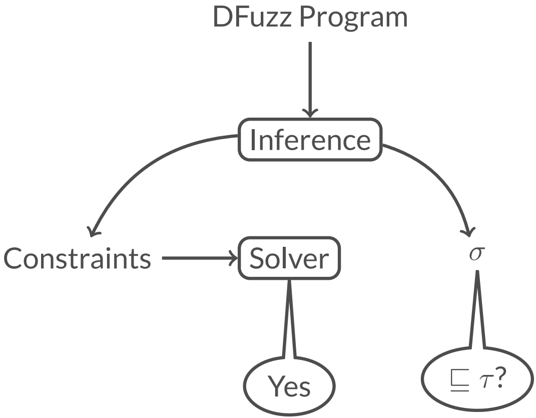
Plan


DFuzz Program
Inference
Constraints
Solver
σ
$\sqsubseteq \tau$?

Plan


DFuzz Program
Inference
Constraints
Solver
σ
$\sqsubseteq \tau$?
Provided by annotation

Plan



DFuzz Program
Inference
Constraints
Solver
σ
Yes
$\sqsubseteq \tau$?

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

$$\ldots e_1 \ldots e_2 \ldots$$

Important Points

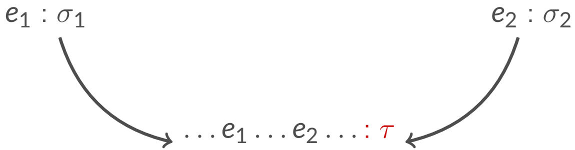
- Context splitting imposes a bottom-up strategy: start with leaves, combine sensitivities progressively

$e_1 : \sigma_1$ $\qquad\qquad$ $e_2 : \sigma_2$

$\ldots e_1 \ldots e_2 \ldots$

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 $!_{\_}(!_{k}\alpha \multimap \alpha) \multimap \alpha$

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

$$\frac{\Gamma \vdash e_1 : !_k \sigma \multimap \tau \qquad \Delta \vdash e_2 : \sigma' \qquad \sigma' \sqsubseteq \sigma}{\Gamma + k \cdot \Delta \vdash e_1\ e_2 : \tau} \quad (\multimap E)$$

Syntax-Directed Rules

Not necessarily equal

$$\frac{\Gamma \vdash e_1 : !_k\sigma \multimap \tau \qquad \Delta \vdash e_2 : \sigma' \qquad \sigma' \sqsubseteq \sigma}{\Gamma + k \cdot \Delta \vdash e_1\ e_2 : \tau}\ (\multimap E)$$

Syntax-Directed Rules

Subtype check

$$\frac{\Gamma \vdash e_1 :!_k\sigma \multimap \tau \qquad \Delta \vdash e_2 : \sigma' \qquad \sigma' \sqsubseteq \sigma}{\Gamma + k \cdot \Delta \vdash e_1\ e_2 : \tau} \quad (\multimap E)$$

Syntax-Directed Rules

$$\frac{\begin{array}{c}\Delta \vdash e : \mathit{list}_n\ \sigma \\ \Gamma \vdash e_{nil} : \tau_{nil} \\ \Gamma, h :_k \sigma, t :_k \mathit{list}_i\ \sigma \vdash e_{cons} : \tau_{cons} \\ \tau = \mathit{case}(n, \tau_{nil}, i, \tau_{cons})\end{array}}{\Gamma + k \cdot \Delta \vdash \mathit{case}\ e\ \mathit{of}\ [] \to e_{nil} \mid h :: t \to e_{cons} : \tau} \quad (\mathit{list}\ E)$$

Syntax-Directed Rules

$$\frac{\begin{array}{c}\Delta \vdash e : list_n\ \sigma \\ \Gamma \vdash e_{nil} : \tau_{nil} \\ \Gamma, h :_k \sigma, t :_k list_i\ \sigma \vdash e_{cons} : \tau_{cons} \\ \tau = case(n, \tau_{nil}, i, \tau_{cons})\end{array}}{\Gamma + k \cdot \Delta \vdash case\ e\ of\ [] \to e_{nil} \mid h :: t \to e_{cons} : \tau}\ (list\ E)$$

Sensitivity-level case lifted to types

Solver Integration

- Need to convert constraints so that standard solvers understand them
- Avoid alternating quantifiers

Solver Integration

$$k \geq case(n, k_0, i, k_s)$$

$$\downarrow$$

$$(n = 0 \Rightarrow k \geq k_0) \wedge (\forall i, n = i + 1 \Rightarrow k \geq k_s)$$

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

$$
\begin{aligned}
d_{\mathbb{R}}(x, y) &= |x - y| \\
d_{\sigma \to \tau}(f, g) &= \sup_{x \in \sigma} d_\tau(f(x), g(x)) \\
d_{list\,\sigma}(l_1, l_2) &= \begin{cases} \infty & \text{if } length(l_1) \neq length(l_2) \\ \sum_i d_\sigma(l_1[i], l_2[i]) & \text{otherwise} \end{cases} \\
d_{set\,\sigma}(s_1, s_2) &= |s_1 \setminus s_2 \cup s_2 \setminus s_1| \\
d_{\mathcal{P}(\sigma)}(\mu, \nu) &= \int_\sigma \log\left(\frac{d\mu}{d\nu}\right) d\mu
\end{aligned}
$$

More Typing Rules

$$\frac{}{\Gamma, x :_1 \sigma \vdash x : \sigma} \quad (\mathit{Var})$$

More Typing Rules

$$\frac{\Gamma, x :_\infty \sigma \vdash e : \sigma}{\infty \cdot \Gamma \vdash \mathit{fix}\ x : \sigma.e : \sigma} \quad (\mathit{Fix})$$

More Typing Rules

$$\frac{\Delta \sqsubseteq \Gamma \qquad \Gamma \vdash e : \sigma \qquad \sigma \sqsubseteq \tau}{\Delta \vdash e : \tau} \quad (\sqsubseteq)$$

Metric Preservation

Suppose

$$\begin{aligned} \vdash e :!_k\sigma \multimap \tau \\ \vdash v_1 : \sigma \\ \vdash v_2 : \sigma \\ e\ v_1 \to^* v_1' \end{aligned}$$

There exists v_2' such that $e\ v_2 \to^* v_2'$ and

$$d_\tau(v_1', v_2') \le k \cdot d_\sigma(v_1, v_2)$$