# Higher-Order Approximate Relational Refinement Types for Mechanism Design and Differential Privacy

Gilles Barthe, Marco Gaboardi, Emilio Jesús Gallego Arias, Justin Hsu, Aaron Roth, Pierre-Yves Strub

UPenn-Mines ParisTech, IMDEA Software Institute, Dundee

Gallium Seminar, Nov 17th, 2014

# Motivation

## Software Verification

- Reason *formally* about programs and their behavior.
- Increase trust in software, help programmers/designers.
- Has important practical and economical utility.
- Expressiveness? Automation?

# Motivation

## Software Verification

- Reason *formally* about programs and their behavior.
- Increase trust in software, help programmers/designers.
- Has important practical and economical utility.
- Expressiveness? Automation?

## Today:

- Verification of probabilistic programs.
- *Mechanisms*: inputs controlled by strategic agents.
- *Truthfulness:* An agent gets best utility when telling the truth.
- *Privacy:* An agent's information leak is bounded.

# The Main Challenges

## Relational Reasoning

Properties of interest are relational, that is, defined over *two runs* of the *same program*:

- *Truthfulness*: agent telling the truth vs not.
- *Privacy*: run including the agent vs not.

# The Main Challenges

## Relational Reasoning

Properties of interest are relational, that is, defined over *two runs* of the *same program*:

- *Truthfulness*: agent telling the truth vs not.
- *Privacy*: run including the agent vs not.

## Probabilistic Reasoning

Interesting algorithms are randomized, properties rely on:

- Expected values.
- *Distance* on distributions.

# Our Approach:

## Related/Precursor Work:

- Relational logics.
- $F^*$, $RF^*$.
- CertiCrypt/CertiPriv.
- Fuzz/DFuzz.

# Our Approach:

## Related/Precursor Work:

- Relational logics.
- F*, RF*.
- CertiCrypt/CertiPriv.
- Fuzz/DFuzz.

## Our Contributions

- Extended type system:
  - Support for Higher-Order refinements.
  - Embedding of logical relations! DFuzz soundness proof.
  - Probabilistic approximate types.
- New application domain and examples.
- Prototype implementation.

# The System: Relational Refinement Types

### Variables

Relational variables, $x \in \mathcal{X}_{\mathcal{R}}$; left/right instances $x_{\triangleleft}, x_{\triangleright} \in \mathcal{X}_{\mathcal{R}}{}^{\bowtie}$.

### Expressions

$$
\begin{array}{rcl}
e^m & ::= & \mathsf{C} \mid x \in \mathcal{X}^m \mid e\ e \mid \lambda x.\, e \mid \mathtt{case}\ e\ \mathtt{with}\ [\epsilon \Rightarrow e \mid x :: x \Rightarrow e] \\
& \mid & \mathtt{letrec}^{\uparrow}\ f\ x = e \mid \mathtt{letrec}^{\downarrow}\ f\ x = e \\
& \mid & e_{\uparrow} \mid \mathtt{let}_{\uparrow}\ x = e\ \mathtt{in}\ e \mid \mathtt{unit}_{\mathrm{M}}\ e \mid \mathtt{bind}_{\mathrm{M}}\ x = e\ \mathtt{in}\ e
\end{array}
$$

### Regular Types

$$
\begin{array}{rcl}
\widetilde{\tau}, \widetilde{\sigma}, \ldots \in \mathbf{CoreTy} & ::= & \bullet \mid \mathbb{B} \mid \mathbb{N} \mid \overline{\mathbb{R}} \mid \overline{\mathbb{R}}^{+} \mid L[\widetilde{\tau}] \\
\tau, \sigma, \ldots \in \mathbf{Ty} & ::= & \widetilde{\tau} \mid \mathfrak{M}[\tau] \mid \mathfrak{C}[\tau] \mid \tau \to \sigma
\end{array}
$$

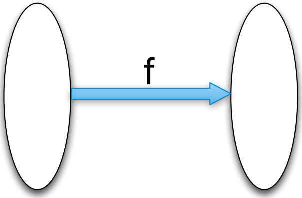
### Relational Refinement Types

$$
\begin{array}{rcl}
T, U \in \mathcal{T} & ::= & \widetilde{\tau} \mid \mathfrak{M}_{\epsilon,\delta}[T] \mid \mathfrak{C}[T] \mid \Pi(x :: T).\, T \mid \{x :: T \mid \phi\} \\
\phi, \psi \in \mathcal{A} & ::= & \mathcal{Q}\ (x : \tau).\, \phi \mid \mathcal{Q}\ (x :: T).\, \phi \\
& \mid & \mathcal{C}(\phi_1, \ldots, \phi_n) \mid e^{\bowtie} = e^{\bowtie} \mid e^{\bowtie} \leq e^{\bowtie} \\
\mathcal{C} & = & \{\top/0, \bot/0, \neg/1, \vee/2, \wedge/2, \Rightarrow/2\}
\end{array}
$$

# Relational Refinement Types: Example

Regular refinement types no enough to capture some properties.

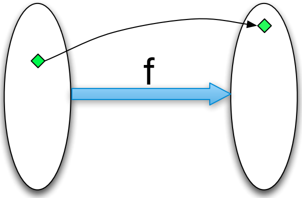
*k*-sensitive function



# Relational Refinement Types: Example

Regular refinement types no enough to capture some properties.

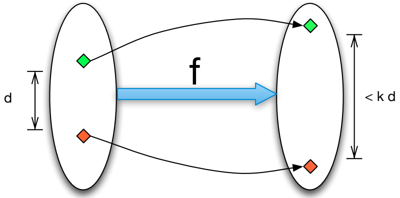
*k*-sensitive function



# Relational Refinement Types: Example

Regular refinement types no enough to capture some properties.

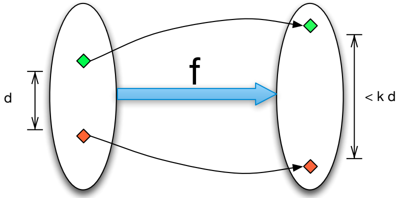
*k*-sensitive function



# Relational Refinement Types: Example

Regular refinement types no enough to capture some properties.

*k*-sensitive function

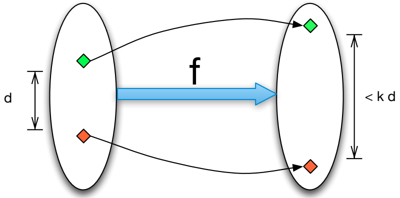



$$\forall x_1, x_2. |f(x_1) - f(x_2)| \leq k \cdot |x_1 - x_2|$$

# Relational Refinement Types: Example

Regular refinement types no enough to capture some properties.

*k*-sensitive function




$$\forall x_1, x_2. |f(x_1) - f(x_2)| \leq k \cdot |x_1 - x_2|$$

What should the type for $f$ be?

# Relational Refinement Types: Example

For the property:

$$\forall x_1, x_2 . |f(x_1) - f(x_2)| \leq k \cdot |x_1 - x_2|$$

# Relational Refinement Types: Example

For the property:

$$\forall x_1, x_2. |f(x_1) - f(x_2)| \leq k \cdot |x_1 - x_2|$$

we can do a refinement at a higher type:

$$\{f : \mathbb{R} \to \mathbb{R} \mid \forall x :: \mathbb{R}. |f(x_\triangleleft) - f(x_\triangleright)| \leq k \cdot |x_\triangleleft - x_\triangleright|\}$$

# Relational Refinement Types: Example

For the property:

$$\forall x_1, x_2. |f(x_1) - f(x_2)| \leq k \cdot |x_1 - x_2|$$

we can do a refinement at a higher type:

$$\{f : \mathbb{R} \to \mathbb{R} \mid \forall x :: \mathbb{R}. |f(x_\triangleleft) - f(x_\triangleright)| \leq k \cdot |x_\triangleleft - x_\triangleright|\}$$

or we can refer to two copies of the input:

$$f : \Pi(x :: \mathbb{R}). \{r :: \mathbb{R} \mid k \cdot |r_\triangleleft - r_\triangleright| \leq |x_\triangleleft - x_\triangleright|\}$$

Both types are equivalent in our system, but the pre/post style more convenient for reasoning.

# The System: Semantics

Semantic subytping for non-relational types:

$$\frac{\vdash e : T \qquad \Gamma \models \phi[x/e]}{\vdash e : \{x : T \mid \phi\}}$$

# The System: Semantics

Semantic subytping for non-relational types:

$$\frac{\vdash e : T \qquad \Gamma \models \phi[x/e]}{\vdash e : \{x : T \mid \phi\}} \qquad \vdash e : T \Rightarrow e \in [\![T]\!]$$

# The System: Semantics

Semantic subytping for non-relational types:

$$\frac{\vdash e : T \qquad \Gamma \models \phi[x/e]}{\vdash e : \{x : T \mid \phi\}} \quad \vdash e : T \Rightarrow e \in [\![T]\!] \quad \frac{v \in [\![T]\!] \qquad \models \phi(v)}{v \in [\![\{x : T \mid \phi(x)\}]\!]}$$

# The System: Semantics

Semantic subytping for non-relational types:

$$\frac{\vdash e : T \qquad \Gamma \models \phi[x/e]}{\vdash e : \{x : T \mid \phi\}} \qquad \vdash e : T \Rightarrow e \in [\![T]\!] \qquad \frac{v \in [\![T]\!] \qquad \models \phi(v)}{v \in [\![\{x : T \mid \phi(x)\}]\!]}$$

Semantic subtyping for HO relational types:

$$(\!|T|\!)_\theta \subseteq [\![|T|]\!] \times [\![|T|]\!]$$

$$\frac{(d_1, d_2) \in [\![\tau]\!] \times [\![\tau]\!]}{(d_1, d_2) \in (\!|\tau|\!)_\theta} \qquad \frac{(d_1, d_2) \in (\!|T|\!)_\theta \qquad [\![\phi]\!]_{\theta\left\{\substack{x_\triangleleft \mapsto d_1 \\ x_\triangleright \mapsto d_2}\right\}}}{(d_1, d_2) \in (\!|\{x :: T \mid \phi\}|\!)_\theta}$$

$$\frac{(f_1, f_2) \in [\![|T| \to |U|]\!] \quad \forall (d_1, d_2) \in (\!|T|\!)_\theta . (f_1(d_1), f_2(d_2)) \in (\!|U|\!)_{\theta\left\{\substack{x_\triangleleft \mapsto d_1 \\ x_\triangleright \mapsto d_2}\right\}}}{(f_1, f_2) \in (\!|\Pi(x :: T). U|\!)_\theta}$$

# SubTyping

$$\textsc{Sub-Refl}\ \frac{\mathcal{G} \vdash T}{\mathcal{G} \vdash T \preceq T} \qquad \textsc{Sub-Trans}\ \frac{\mathcal{G} \vdash T \preceq U \quad \mathcal{G} \vdash U \preceq V}{\mathcal{G} \vdash T \preceq V}$$

$$\textsc{Sub-Left}\ \frac{\mathcal{G} \vdash \{x :: T \mid \phi\}}{\mathcal{G} \vdash \{x :: T \mid \phi\} \preceq T}$$

$$\textsc{Sub-Right}\ \frac{\begin{array}{c}\mathcal{G} \vdash T \preceq U \\ \|\mathcal{G}, x :: U\| \vdash \phi \qquad \forall\theta.\, \theta \vdash \mathcal{G}, x :: T \Rightarrow [\![\phi]\!]_\theta\end{array}}{\mathcal{G} \vdash T \preceq \{x :: U \mid \phi\}}$$

$$\textsc{Sub-Prod}\ \frac{\mathcal{G} \vdash T_2 \preceq T_1 \qquad \mathcal{G}, x :: T_2 \vdash U_1 \preceq U_2}{\mathcal{G} \vdash \Pi(x :: T_1).\, U_1 \preceq \Pi(x :: T_2).\, U_2}$$

# The System: Typing

The typing judgment relates two programs to a type:

$$\mathcal{G} \vdash e_1 \sim e_2 :: T$$

# The System: Typing

The typing judgment relates two programs to a type:

$$\mathcal{G} \vdash e_1 \sim e_2 :: T$$

## Soundness

$$\mathcal{G} \vdash e_1 \sim e_2 :: T \Rightarrow \forall \mathcal{G} \vdash \theta, (\llbracket e_1 \rrbracket_\theta, \llbracket e_2 \rrbracket_\theta) \in (\!| T |\!)_\theta$$

# The System: Typing

The typing judgment relates two programs to a type:

$$\mathcal{G} \vdash e_1 \sim e_2 :: T$$

## Soundness

$$\mathcal{G} \vdash e_1 \sim e_2 :: T \Rightarrow \forall \mathcal{G} \vdash \theta, (\llbracket e_1 \rrbracket_\theta, \llbracket e_2 \rrbracket_\theta) \in (\!| T |\!)_\theta$$

## Synchronicity

In most cases programs are synchronous, so we use:

$$\mathcal{G} \vdash e :: T \equiv \mathcal{G} \vdash e_\triangleleft \sim e_\triangleright :: T$$

with $e_\triangleleft, e_\triangleright$ projecting the variables in $e$.

# Base Typing Rules

$$\textsc{Var}\ \frac{x :: T \in \mathrm{dom}(\mathcal{G})}{\mathcal{G} \vdash x :: T} \qquad \textsc{Abs}\ \frac{\mathcal{G}, x :: T \vdash e :: U}{\mathcal{G} \vdash \lambda x.\, e :: \Pi(x :: T).\, U}$$

$$\textsc{App}\ \frac{\mathcal{G} \vdash e_f :: \Pi(x :: T).\, U \qquad \mathcal{G} \vdash e_a :: T}{\mathcal{G} \vdash e_f\ e_a :: U\{x \mapsto e_a\}}$$

# Base Typing Rules

$$\textsc{Var}\ \frac{x :: T \in \mathrm{dom}(\mathcal{G})}{\mathcal{G} \vdash x :: T} \qquad \textsc{Abs}\ \frac{\mathcal{G}, x :: T \vdash e :: U}{\mathcal{G} \vdash \lambda x.\, e :: \Pi(x :: T).\, U}$$

$$\textsc{App}\ \frac{\mathcal{G} \vdash e_f :: \Pi(x :: T).\, U \qquad \mathcal{G} \vdash e_a :: T}{\mathcal{G} \vdash e_f\ e_a :: U\{x \mapsto e_a\}}$$

$$\textsc{Case}\ \frac{\begin{array}{c}\mathcal{G} \vdash e :: L[\widetilde{\tau}] \qquad \forall \theta.\, \theta \vdash \mathcal{G} \Rightarrow \mathrm{skeleton}(e_\triangleleft, e_\triangleright) \\ \mathcal{G}, \{e_\triangleleft = e_\triangleright = \epsilon\} \vdash e_1 :: T \\ \mathcal{G}, x :: \widetilde{\tau}, y :: L[\widetilde{\tau}], \{e_\triangleleft = x_\triangleleft :: y_\triangleleft \wedge e_\triangleright = x_\triangleright :: y_\triangleright\} \vdash e_2 :: T\end{array}}{\mathcal{G} \vdash \texttt{case}\ e\ \texttt{with}\ [\epsilon \Rightarrow e_1 \mid x :: y \Rightarrow e_2] :: T}$$

# Typing Rules for Recursion

To ensure consistency at higher-types, we must embed non-terminating computations in the partiality monad:

$$\textsc{LetRecSN}\ \frac{\mathcal{G}, f :: \Pi(x :: T).\, U \vdash \lambda x.\, e :: \Pi(x :: T).\, U \qquad \mathcal{G} \vdash \Pi(x :: T).\, U \qquad \mathcal{SN}\text{-guard}}{\mathcal{G} \vdash \mathtt{letrec}^{\downarrow}\ f\ x = e :: \Pi(x :: T).\, U}$$

# Typing Rules for Recursion

To ensure consistency at higher-types, we must embed non-terminating computations in the partiality monad:

$$\textsc{LetRecSN}\ \frac{\mathcal{G}, f :: \Pi(x :: T).\, U \vdash \lambda x.\, e :: \Pi(x :: T).\, U \qquad \mathcal{G} \vdash \Pi(x :: T).\, U \qquad \mathcal{SN}\text{-guard}}{\mathcal{G} \vdash \texttt{letrec}^{\downarrow}\ f\ x = e :: \Pi(x :: T).\, U}$$

$$\textsc{LetRec}\ \frac{\mathcal{G} \vdash \Pi(x :: T).\, \mathfrak{C}[U] \qquad \mathcal{G}, f :: \Pi(x :: T).\, \mathfrak{C}[U] \vdash \lambda x.\, e :: \Pi(x :: T).\, \mathfrak{C}[U]}{\mathcal{G} \vdash \texttt{letrec}\ f\ x = e :: \Pi(x :: T).\, \mathfrak{C}[U]}$$

# Typing Rules for Recursion

To ensure consistency at higher-types, we must embed non-terminating computations in the partiality monad:

$$\textsc{LetRecSN}\ \frac{\mathcal{G}, f :: \Pi(x :: T).\, U \vdash \lambda x.\, e :: \Pi(x :: T).\, U \qquad \mathcal{G} \vdash \Pi(x :: T).\, U \qquad \mathcal{SN}\text{-guard}}{\mathcal{G} \vdash \mathtt{letrec}^{\downarrow}\ f\ x = e :: \Pi(x :: T).\, U}$$

$$\textsc{LetRec}\ \frac{\mathcal{G} \vdash \Pi(x :: T).\, \mathfrak{C}[U] \qquad \mathcal{G}, f :: \Pi(x :: T).\, \mathfrak{C}[U] \vdash \lambda x.\, e :: \Pi(x :: T).\, \mathfrak{C}[U]}{\mathcal{G} \vdash \mathtt{letrec}\ f\ x = e :: \Pi(x :: T).\, \mathfrak{C}[U]}$$

$$\textsc{UnitC}\ \frac{\mathcal{G} \vdash e :: T}{\mathcal{G} \vdash e_{\uparrow} :: \mathfrak{C}[T]} \qquad \textsc{BindC}\ \frac{\mathcal{G} \vdash e_1 :: \mathfrak{C}[T_1] \qquad \mathcal{G} \vdash \mathfrak{C}[T_2] \qquad \mathcal{G}, x :: T_1 \vdash e_2 :: \mathfrak{C}[T_2]}{\mathcal{G} \vdash \mathtt{let}_{\uparrow}\ x = e_1\ \mathtt{in}\ e_2 :: \mathfrak{C}[T_2]}$$

# Asynchronous Rules

$$\text{ASYM}\ \frac{\mathcal{G} \vdash e_1 \sim e_2 :: T}{\mathcal{G}^{\leftrightarrow} \vdash {e_2}^{\leftrightarrow} \sim {e_1}^{\leftrightarrow} :: T^{\leftrightarrow}}$$

$$\text{AREDLEFT}\ \frac{e_1 \to e_1' \qquad \mathcal{G} \vdash e_1 \sim e_2 :: T}{\mathcal{G} \vdash e_1' \sim e_2 :: T}$$

# Asynchronous Rules

$$\text{ASym}\ \frac{\mathcal{G} \vdash e_1 \sim e_2 :: T}{\mathcal{G}^{\leftrightarrow} \vdash {e_2}^{\leftrightarrow} \sim {e_1}^{\leftrightarrow} :: T^{\leftrightarrow}}$$

$$\text{ARedLeft}\ \frac{e_1 \to e_1' \qquad \mathcal{G} \vdash e_1 \sim e_2 :: T}{\mathcal{G} \vdash e_1' \sim e_2 :: T}$$

$$\text{ACase}\ \frac{\begin{array}{c} |\mathcal{G}| \vdash e : L[\widetilde{\tau}] \qquad |\mathcal{G}| \vdash e' : |T| \\ \mathcal{G}, \{e_\triangleleft = \epsilon\} \vdash e_1 \sim e' :: T \\ \mathcal{G}, x :: \widetilde{\tau}, y :: L[\widetilde{\tau}], \{e_\triangleleft = x_\triangleleft :: y_\triangleleft\} \vdash e_2 \sim e' :: T \end{array}}{\mathcal{G} \vdash \mathtt{case}\ e\ \mathtt{with}\ [\epsilon \Rightarrow e_1 \mid x :: y \Rightarrow e_2] \sim e' :: T}$$

# More on Mechanism Design

*Mechanism design* is the study of algorithm design where the inputs to the algorithm are controlled by strategic agents, who must be *incentivized* to faithfully report them.

# More on Mechanism Design

*Mechanism design* is the study of algorithm design where the inputs to the algorithm are controlled by strategic agents, who must be *incentivized* to faithfully report them.

Formally

- $n$ agents, with type for actions $A_i$, $i \in \{1, \ldots, n\}$.
- A mechanism $M : A^n \to \mathcal{O}$.
- A payoff for every agent $P_i : \mathcal{O} \to R^+$.
- **Probabilistic algorithms are common!**
  Payoff becomes *expected payoff*.

# More on Mechanism Design

*Mechanism design* is the study of algorithm design where the inputs to the algorithm are controlled by strategic agents, who must be *incentivized* to faithfully report them.

## Formally

- $n$ agents, with type for actions $A_i$, $i \in \{1, \ldots, n\}$.
- A mechanism $M : A^n \to \mathcal{O}$.
- A payoff for every agent $P_i : \mathcal{O} \to R^+$.
- **Probabilistic algorithms are common!**
  Payoff becomes *expected payoff*.

## Verification

Incentives are not enough, *the agents need to believe them*.
Verification is an attractive way to convince them.

# Mechanism Examples

## Auctions

- Buyers (agents), *bids* (actions), seller (mechanism).
- Outcome: price, goods assignation.
- An auction is *truthful* if the buyer gets maximal payoff when she reports her true valuation.

# Mechanism Examples

## Auctions

- Buyers (agents), *bids* (actions), seller (mechanism).
- Outcome: price, goods assignation.
- An auction is *truthful* if the buyer gets maximal payoff when she reports her true valuation.

## Nash Equilibrium Computation

- $n$ players, action type $A$.
- Payoff for $i$, $P_i : A^n \to R^+$, depends on others actions.
- The mechanism suggests an *action profile* $(a_1, \ldots, a_n)$.
- If all the other players follow the suggestion, player $i$ gets the best payoff by following too.

# Digital Goods Auctions

- Price a good with infinite supply. (i.e: Digital goods)

# Digital Goods Auctions

- Price a good with infinite supply. (i.e: Digital goods)
- Bidders and seller.

# Digital Goods Auctions

- Price a good with infinite supply. (i.e: Digital goods)
- Bidders and seller.
- Bidders have a secret *true* value for the item $v_i$, and make a public *bid* $b_i$ before the price is known.

# Digital Goods Auctions

- Price a good with infinite supply. (i.e: Digital goods)
- Bidders and seller.
- Bidders have a secret *true* value for the item $v_i$, and make a public *bid* $b_i$ before the price is known.
- The seller knows the bids, but not the real values. Sets the price $p$ after the bids.

# Digital Goods Auctions

- Price a good with infinite supply. (i.e: Digital goods)
- Bidders and seller.
- Bidders have a secret *true* value for the item $v_i$, and make a public *bid* $b_i$ before the price is known.
- The seller knows the bids, but not the real values. Sets the price $p$ after the bids.
- If $b_i \geq p$, the bidder $i$ gets the item, with utility $v_i - p$. Otherwise she doesn't get it, and utility is 0.

# Digital Goods Auctions

- Price a good with infinite supply. (i.e: Digital goods)
- Bidders and seller.
- Bidders have a secret *true* value for the item $v_i$, and make a public *bid* $b_i$ before the price is known.
- The seller knows the bids, but not the real values. Sets the price $p$ after the bids.
- If $b_i \geq p$, the bidder $i$ gets the item, with utility $v_i - p$. Otherwise she doesn't get it, and utility is 0.

The auction is truthful if buyers have optimal utility when they reports the true value $v_i$ as their bids $b_i$.
In general, an auction cannot be truthful if it depends on the bidder's price!

# The Fixed Price Auction

## Fixed Price Auctions

The simplest truthful auction is the *fixed price auction*. The seller will set $p$ independently of the bid $b$ for a seller with true value $v$. If $b \geq p$, then utility $v - p$, else 0. Note the bad revenue properties.

# The Fixed Price Auction

## Fixed Price Auctions

The simplest truthful auction is the *fixed price auction*. The seller will set $p$ independently of the bid $b$ for a seller with true value $v$. If $b \geq p$, then utility $v - p$, else 0. Note the bad revenue properties.

## Informal proof of truthfulness

The price $p$ is fixed, we compare $b_{\triangleleft} = v$ vs $b_{\triangleright} \neq v$. The interesting cases are when the bidder gets the item in one run and doesn't in the other:

- If $b_{\triangleright}$ got the item, utility is negative, thus less than 0 for the $b_{\triangleleft}$ case (remember $b_{\triangleleft}$ didn't get the item).
- If $b_{\triangleleft}$ got the item, utility will be greater or equal than 0, thus better or equal than $b_{\triangleright}$'s utility (0).

# The Fixed Price Auction

We model the utility as a program:

```
let fp_utility (v : R) {b :: R      ◁      } (p : R)
                : { u :: R    ◁        ▷ } =
   if b >= p then v - p
              else 0.0
```

# The Fixed Price Auction

We model the utility as a program:

```
let fp_utility (v : R) {b :: R | b◁ = v} (p : R)
               : { u :: R | u◁ >= u▷ } =
  if b >= p then v - p
            else 0.0
```

# The Fixed Price Auction

We model the utility as a program:

```
let fp_utility (v : R) {b :: R | b◁ = v} (p : R)
                : { u :: R | u◁ >= u▷ } =
  if b >= p then v - p
             else 0.0
```

We use asynchronous reasoning. The interesting case is:

$$\{b_\triangleleft = v, b_\triangleleft \geq p, b_\triangleright < p\} \vdash v - p \sim 0.0 :: \{u :: \mathbb{R} \mid u_\triangleleft \geq u_\triangleright\}$$

substituting $[v - p/u_\triangleleft, 0.0/u_\triangleright]$ we get the proof obligation:

# The Fixed Price Auction

We model the utility as a program:

```
let fp_utility (v : R) {b :: R | b◁ = v} (p : R)
                : { u :: R | u◁ >= u▷ } =
  if b >= p then v - p
            else 0.0
```

We use asynchronous reasoning. The interesting case is:

$$\{b_\triangleleft = v, b_\triangleleft \geq p, b_\triangleright < p\} \vdash v - p \sim 0.0 :: \{u :: \mathbb{R} \mid u_\triangleleft \geq u_\triangleright\}$$

substituting $[v - p/u_\triangleleft, 0.0/u_\triangleright]$ we get the proof obligation:

$$v \geq p \Rightarrow v - p \geq 0.0$$

# The Distribution Type

We didn't specify the semantics of relational distribution types.

A first approach to lifting

$$\frac{(\mu_1, \mu_2) \in \mathfrak{M}[|T|] \times \mathfrak{M}[|T|]}{(\mu_1, \mu_2) \in (\!|\mathfrak{M}[T]|\!)_\theta}$$

# The Distribution Type

We didn't specify the semantics of relational distribution types.

## A first approach to lifting

$$\frac{(d_1, d_2) \in (\!| T |\!)_\theta \qquad (\mu_1, \mu_2) \in \mathfrak{M}[| T |] \times \mathfrak{M}[| T |]}{(\mu_1, \mu_2) \in (\!| \mathfrak{M}[T] |\!)_\theta}$$

# The Distribution Type

We didn't specify the semantics of relational distribution types.

## A first approach to lifting

$$\frac{?? \qquad (d_1, d_2) \in (\!|T|\!)_\theta \qquad (\mu_1, \mu_2) \in \mathfrak{M}[|T|] \times \mathfrak{M}[|T|]}{(\mu_1, \mu_2) \in (\!|\mathfrak{M}[T]|\!)_\theta}$$

# The Distribution Type

We didn't specify the semantics of relational distribution types.

A first approach to lifting

$$\frac{?? \qquad (d_1, d_2) \in (\!| T |\!)_\theta \qquad (\mu_1, \mu_2) \in \mathfrak{M}[|T|] \times \mathfrak{M}[|T|]}{(\mu_1, \mu_2) \in (\!| \mathfrak{M}[T] |\!)_\theta}$$

We need to relate $(d_1, d_2)$ to $(\mu_1, \mu_2)$!

# The Distribution Type

We didn't specify the semantics of relational distribution types.

A first approach to lifting

$$\frac{?? \qquad (d_1, d_2) \in (\!|T|\!)_\theta \qquad (\mu_1, \mu_2) \in \mathfrak{M}[|T|] \times \mathfrak{M}[|T|]}{(\mu_1, \mu_2) \in (\!|\mathfrak{M}[T]|\!)_\theta}$$

We need to relate $(d_1, d_2)$ to $(\mu_1, \mu_2)$!

Solution: define a lifting of the relation $(\!|T|\!)_\theta$ through a witness distribution $\mu = \mathfrak{M}[|T| \times |T|]$, such that:

$$\Pr_{x \leftarrow \mu_1} x \in [\![T]\!] = \sum_{y \in T} \Pr_{(x,y) \leftarrow \mu} (x, y) \in (\!|T|\!)_\theta$$

# Lifting

More formally, for a relation $\Phi : T_1 \times T_2$, the predicate $\mathcal{L}(\Phi)\ \mu_1\ \mu_2$ holds iff there exists a distribution $\mu \in \mathfrak{M}[T_1 \times T_2]$ such that for every $H \subseteq T_1$, we have

$$\Pr_{x \leftarrow \mu_1}[H(x)] = \sum_{y \in T_2} \Pr_{(x,y) \leftarrow \mu}[H(x) \wedge \Phi(x, y)]$$

and symmetrically for $T_2$.
"Probability of events in $\mu_1\ \mu_2$ must respect the relation".

# Examples of Lifting

As an example, for $\Phi \equiv \{(F,F),(F,T),(T,T)\}$ we have liftings:

$$
\begin{aligned}
\mu_1(F) &= 2/3 & \mu(F,F) &= 1/3\\
\mu_1(T) &= 1/3 & \mu(F,T) &= 1/3\\
\mu_2(F) &= 1/3 & \mu(T,F) &= 0\\
\mu_2(T) &= 2/3 & \mu(T,T) &= 1/3
\end{aligned}
$$

$$
\begin{aligned}
\mu_1(F) &= 1 & \mu(F,F) &= 1\\
\mu_1(T) &= 0 & \mu(F,T) &= 0\\
\mu_2(F) &= 1 & \mu(T,F) &= 0\\
\mu_2(T) &= 0 & \mu(T,T) &= 0
\end{aligned}
$$

# Semantics of the Distribution Type

We can now interpret the relational distribution type as all the distributions satisfying the lifting:

$$\frac{\mu_1, \mu_2 \in \mathfrak{M}[|T|] \qquad \mathcal{L}(\llparenthesis T \rrparenthesis_\theta)\ \mu_1\ \mu_2}{(\mu_1, \mu_2) \in \llparenthesis \mathfrak{M}[T] \rrparenthesis_\theta}$$

In particular, the type $\mathfrak{M}[\{x :: T \mid x_\triangleleft = x_\triangleright\}]$ forces equal distributions.

# Higher-Order Refinements and Probability

## Expectation

Expectation of a function $f$ over $\mu$ is:

$$\mathsf{E}\ \mu\ f := \sum_{x \in D} (f\ x) \cdot (\mu\ x)$$

# Higher-Order Refinements and Probability

## Expectation

Expectation of a function $f$ over $\mu$ is:

$$\mathsf{E}\ \mu\ f := \sum_{x \in D} (f\ x) \cdot (\mu\ x)$$

We capture monotonicity of expectation as:

$$\begin{gathered} I := [0, 1] \\ IBF := \{f :: D \to I \mid \forall d : D.\ f_{\triangleleft}\ d \geq f_{\triangleright}\ d\} \\ \mathsf{E} : \Pi(\mu :: \mathfrak{M}[\{x :: D \mid x_{\triangleleft} = x_{\triangleright}\}]).\, \Pi(f :: IBF).\, \{e :: I \mid e_{\triangleleft} \geq e_{\triangleright}\} \end{gathered}$$

Sound as a primitive; other types are possible.

# Randomized Auctions

- Using the probabilistic primitives, we can now define and verify randomized auctions, which have much better revenue properties than the fixed price one.
- The price a bidder gets won't still depend on her bid, however:
- we *randomly* split the bidders in two groups, $g_a, g_b$, we compute the revenue-maximizing price for each group, $p_a, p_b$, and sell to $g_a$ using $p_b$ and conversely.
- This auction is truthful on the *expected* utility.

Universal truthfulness:
A bidder will be never able to gain from lying, even knowing the random coins of the mechanism.

# The Competitive Auction

```
let utility (v            :  real)
            (bid          :: { b :: R | b◁ = v })
            (otherbids    :  L[R])
            (g, groups)   :  (B ∗ L[B])
    : { u :: real | u◁ >= u▷ } =
 match split g bid others otherbids with
 | (g1, g2) →
   if g then fixedprice v bid (prices g2)
        else fixedprice v bid (prices g1)

let auction (n : N) (v : R)
            (bid         :: { b  :: R | b◁ = v })
            (otherbids   :  L[R])
    : { u :: real | u◁ >= u▷ } =
 let grouping :: M{ r :: (B ∗ B list) | r◁ = r▷} =
     mlet  mycoin = flip    in
     mlet  coins  = flipN n in
     munit (mycoin, coins)
 in E grouping (utility v bid otherbids)
```

# The Competitive Auction

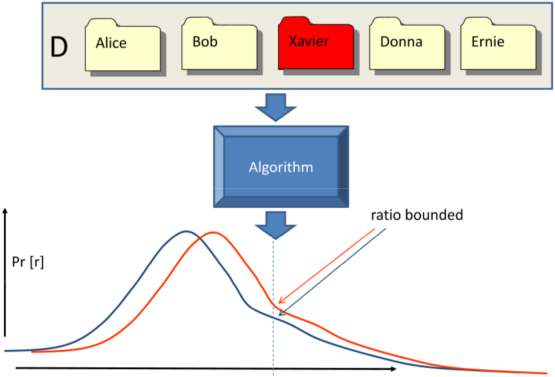
```
let E (mu : M[ r : α | r◁ = r▷ ])
      (f : α → real | ∀ x : α, f◁ x >= f▷ x)
    : { r :: real | r◁ >= r▷ } = ....

let utility (v          :  real)
            (bid        :: { b :: R | b◁ = v })
            (otherbids  :  L[R])
            (g, groups) :  (B ∗ L[B])
    : { u :: real | u◁ >= u▷ } = ...

let auction (n : N) (v : R)
            (bid       :: { b  :: R | b◁ = v })
            (otherbids :  L[R])
    : { u :: real | u◁ >= u▷ } =
 let grouping :: M{ r :: (B ∗ B list) | r◁ = r▷} = ...
 in E grouping (utility v bid otherbids)
```

# Differential Privacy

Contribution of a single individual to the output of a mechanism cannot be effectively distinguished by an attacker under worst-case assumptions.



# Differential Privacy

## Formal Definition

A probabilistic function $F : T \to S$ is $(\epsilon, \delta)$-*Differentially Private* if for all pairs of *adjacent* $t_1, t_2 \in T$ and for every $E \subseteq S$:

$$\Pr_{x \leftarrow F\ t_1}[x \in E] \leq \exp(\epsilon) \Pr_{x \leftarrow F\ t_2}[x \in E] + \delta$$

# Differential Privacy

## Formal Definition

A probabilistic function $F : T \to S$ is $(\epsilon, \delta)$-*Differentially Private* if for all pairs of *adjacent* $t_1, t_2 \in T$ and for every $E \subseteq S$:

$$\Pr_{x \leftarrow F\ t_1}[x \in E] \leq \exp(\epsilon) \Pr_{x \leftarrow F\ t_2}[x \in E] + \delta$$

## Example: The Laplace Mechanism:

- Compute the *sensitivity* $k$ of $f$.
- For input $t$, release $f(t)$ + *random noise*, scaled by $k$.

# Differential Privacy

## Formal Definition

A probabilistic function $F : T \to S$ is $(\epsilon, \delta)$-*Differentially Private* if for all pairs of *adjacent* $t_1, t_2 \in T$ and for every $E \subseteq S$:

$$\Pr_{x \leftarrow F\ t_1}[x \in E] \leq \exp(\epsilon) \Pr_{x \leftarrow F\ t_2}[x \in E] + \delta$$

## Example: The Laplace Mechanism:

- Compute the *sensitivity* $k$ of $f$.
- For input $t$, release $f(t)$ + *random noise*, scaled by $k$.

Many algorithms are DP: private database release, counters, analytics, **strong connection to Mechanism Design!**

# Approximately Reasoning over Distributions

We can capture DP with a refinement over the type of probability distributions using the definition of Δ-distance:

$$\Delta_\epsilon(\mu_1, \mu_2) = \max_{E \subseteq U} \left( \Pr_{x \leftarrow \mu_2}[x \in E] - \exp(\epsilon) \Pr_{x \leftarrow \mu_1}[x \in E] \right)$$

# Approximately Reasoning over Distributions

We can capture DP with a refinement over the type of probability distributions using the definition of $\Delta$-distance:

$$\Delta_\epsilon(\mu_1, \mu_2) = \max_{E \subseteq U} \left( \Pr_{x \leftarrow \mu_2}[x \in E] - \exp(\epsilon) \Pr_{x \leftarrow \mu_1}[x \in E] \right)$$

Then, $f$ is $(\epsilon, \delta)$ differentially private if it has type:

$$\{d :: T \mid \mathsf{Adj}(d_\triangleleft, d_\triangleright)\} \to \{r :: \mathfrak{M}[\mathbb{R}] \mid \Delta_\epsilon(r_\triangleleft, r_\triangleright) \leq \delta\}$$

However, verification conditions involving $\Delta$ are quite hard.

# The Relational Distribution Type

Our solution: Internalize distribution distance in the types:

$$\frac{\mu_1, \mu_2 \in \mathfrak{M}[|T|] \qquad \mathcal{L}_{\epsilon,\delta}(\llparenthesis T \rrparenthesis_\theta)\ \mu_1\ \mu_2}{(\mu_1, \mu_2) \in \llparenthesis \mathfrak{M}_{\epsilon,\delta}[T] \rrparenthesis_\theta}$$

Lifting is extended from $p = p_1$ to $p \leq p_1 \leq exp(p) + \delta$.

# The Relational Distribution Type

Our solution: Internalize distribution distance in the types:

$$\frac{\mu_1, \mu_2 \in \mathfrak{M}[|T|] \qquad \mathcal{L}_{\epsilon,\delta}((\!|T|\!)_\theta)\ \mu_1\ \mu_2}{(\mu_1, \mu_2) \in (\!|\mathfrak{M}_{\epsilon,\delta}[T]|\!)_\theta}$$

Lifting is extended from $p = p_1$ to $p \leq p_1 \leq exp(p) + \delta$.

## Capturing DP

The interpretation of $\mathfrak{M}_{\epsilon,\delta}[\{r :: \mathbb{R} \mid r_\triangleleft = r_\triangleright\}]$ is the set of pairs of probability distributions that are $(\epsilon, \delta)$-apart, capturing DP.

# The Relational Distribution Type

Our solution: Internalize distribution distance in the types:

$$\frac{\mu_1, \mu_2 \in \mathfrak{M}[|T|] \qquad \mathcal{L}_{\epsilon,\delta}(\llparenthesis T \rrparenthesis_\theta)\ \mu_1\ \mu_2}{(\mu_1, \mu_2) \in \llparenthesis \mathfrak{M}_{\epsilon,\delta}[T] \rrparenthesis_\theta}$$

Lifting is extended from $p = p_1$ to $p \leq p_1 \leq exp(p) + \delta$.

## Capturing DP

The interpretation of $\mathfrak{M}_{\epsilon,\delta}[\{r :: \mathbb{R} \mid r_\triangleleft = r_\triangleright\}]$ is the set of pairs of probability distributions that are $(\epsilon, \delta)$-apart, capturing DP.
DP algorithms are typed as:

$$f : \{d :: T \mid \mathrm{Adj}(d_\triangleleft, d_\triangleright)\} \to \mathfrak{M}_{\epsilon,\delta}[\{r :: \mathbb{R} \mid r_\triangleleft = r_\triangleright\}]$$

# The Probability Polymonad

Reasoning about distance is compositional:

$$\textsc{Sub-M}\ \frac{\mathcal{G} \vdash T \preceq U \qquad \forall \theta.\, \theta \vdash \mathcal{G}, x :: T \Rightarrow [\![\epsilon_1 \leq \epsilon_2 \wedge \delta_1 \leq \delta_2]\!]_\theta}{\mathcal{G} \vdash \mathfrak{M}_{\epsilon_1,\delta_1}[T] \preceq \mathfrak{M}_{\epsilon_2,\delta_2}[U]}$$

$$\textsc{UnitM}\ \frac{\mathcal{G} \vdash e :: T}{\mathcal{G} \vdash \mathtt{unit}_{\mathrm{M}}\ e :: \mathfrak{M}_{\epsilon,\delta}[T]}$$

$$\textsc{BindM}\ \frac{\mathcal{G} \vdash e_1 :: \mathfrak{M}_{\epsilon_1,\delta_1}[T_1] \qquad \mathcal{G}, x :: T_1 \vdash e_2 :: \mathfrak{M}_{\epsilon_2,\delta_2}[T_2]}{\mathcal{G} \vdash \mathtt{bind}_{\mathrm{M}}\ x = e_1\ \mathtt{in}\ e_2 :: \mathfrak{M}_{\epsilon_1+\epsilon_2,\delta_1+\delta_2}[T_2]}$$

Bind is distance-adjusting sampling.

# Type for the Laplace Mechanism

Recall the Laplace Mechanism:

For a $k$-sensitive $f$, $f$ plus $k/\epsilon$-scaled Laplacian noise is DP. This is captured by the type:

# Type for the Laplace Mechanism

Recall the Laplace Mechanism:

For a $k$-sensitive $f$, $f$ plus $k/\epsilon$-scaled Laplacian noise is DP. This is captured by the type:

$$\mathsf{lap} : \Pi(\epsilon :: \mathbb{R}).\, \Pi(x :: \mathbb{R}).\, \mathfrak{M}_{\epsilon * |x_\triangleleft - x_\triangleright|, 0}[\{r :: \mathbb{R} \mid r_\triangleleft = r_\triangleright\}]$$

Note that the actual distance $\epsilon * |x_\triangleleft - x_\triangleright|$ depends on the distance of the inputs. This is a better alternative than using a precondition on $x$.

# Type for the Laplace Mechanism

Recall the Laplace Mechanism:

For a $k$-sensitive $f$, $f$ plus $k/\epsilon$-scaled Laplacian noise is DP. This is captured by the type:

$$\mathsf{lap} : \Pi(\epsilon :: \mathbb{R}).\, \Pi(x :: \mathbb{R}).\, \mathfrak{M}_{\epsilon * |x_{\triangleleft} - x_{\triangleright}|, 0}[\{r :: \mathbb{R} \mid r_{\triangleleft} = r_{\triangleright}\}]$$

Note that the actual distance $\epsilon * |x_{\triangleleft} - x_{\triangleright}|$ depends on the distance of the inputs. This is a better alternative than using a precondition on $x$.
Using the bind rule, we can sample from laplace and assume the sampled value equal in both runs.

# Example: Private Histogram

We add noise to an histogram to make it private.

```
let rec histogram {l :: L(R) | Adj x◁ x▷) }
    : M[e ∗ d(l◁,l▷)] { r :: L(R) | r◁ = r▷ } =
 match l with
 | []      → unit []
 | x :: xs →
   mlet y  = lap eps x     in
   mlet ys = histogram xs in
   munit (y :: ys)
```

# Example: Private Histogram

We add noise to an histogram to make it private.

```
let rec histogram {l :: L(R) | Adj x◁ x▷) }
    : M[e ∗ d(l◁,l▷)] { r :: L(R) | r◁ = r▷ } =
  match l with
  | []      → unit []
  | x :: xs →
    mlet y  = lap eps x     in
    mlet ys = histogram xs in
    munit (y :: ys)
```

The main proof obligation is:

$$e * d(x_{\triangleleft} :: xs_{\triangleleft}, x_{\triangleright} :: xs_{\triangleright}) \geq e * (d(x_{\triangleleft}, x_{\triangleright}) + d(xs_{\triangleleft}, xs_{\triangleright}))$$

which is implied by the adjacency precondition.

# Combining MD and DP: Aggregative Games

- We verify the computation of an approximate Nash-equilibrium.
- $n$ agents can choose over a space of actions $a_i \in A$.

# Combining MD and DP: Aggregative Games

- We verify the computation of an approximate Nash-equilibrium.
- $n$ agents can choose over a space of actions $a_i \in A$.
- $(a_1, \ldots, a_n)$ is an $\alpha$-approximate Nash-equilibrium if no single agent $i$ can gain more than $\alpha$ payoff by *unilateral* deviation: For all agents $i$ and actions $a_i'$:

$$\mathrm{E}[P_i(a_1, \ldots, a_i, \ldots a_N)] \geq \mathrm{E}[P_i(a_1, \ldots, a_i', \ldots a_N)] - \alpha.$$

# Combining MD and DP: Aggregative Games

- We verify the computation of an approximate Nash-equilibrium.
- $n$ agents can choose over a space of actions $a_i \in A$.
- $(a_1, \ldots, a_n)$ is an $\alpha$-approximate Nash-equilibrium if no single agent $i$ can gain more than $\alpha$ payoff by *unilateral* deviation: For all agents $i$ and actions $a_i'$:

$$\mathrm{E}[P_i(a_1, \ldots, a_i, \ldots a_N)] \geq \mathrm{E}[P_i(a_1, \ldots, a_i', \ldots a_N)] - \alpha.$$

- Assumption: Payoff for $i$ depends only on $a_i$ plus a *signal*, a positive (bounded) real number depending on the aggregated actions of all players.

# Combining MD and DP: Aggregative Games

- The key: use differential privacy to compute the equilibria.
- Mediator: The mechanism suggests the equilibria action $a_i$.
- We prove that the player gets optimal utility if she does $a_i$.
- We reason over a deviation function $dev_i$ for player $i$.

# Combining MD and DP: Aggregative Games

- The key: use differential privacy to compute the equilibria.
- Mediator: The mechanism suggests the equilibria action $a_i$.
- We prove that the player gets optimal utility if she does $a_i$.
- We reason over a deviation function $dev_i$ for player $i$.

In types:

```
let aggregative_utility ( ... )
    { dev :: act → act | ∀ a : act. dev◁ a = a) }
  : { u :: real | u◁ >= u▷ - alpha }
```

# Combining MD and DP: Aggregative Games

- The key: use differential privacy to compute the equilibria.
- Mediator: The mechanism suggests the equilibria action $a_i$.
- We prove that the player gets optimal utility if she does $a_i$.
- We reason over a deviation function $dev_i$ for player $i$.

In types:

```
let aggregative_utility ( ... )
    { dev :: act → act | ∀ a : act. dev◁ a = a) }
  : { u :: real | u◁ >= u▷ - alpha }
```

Relate expectation to distance on the distributions:

$$\mathsf{E} : \Pi(\mu :: \mathfrak{M}_{\epsilon,\delta}[\{x :: I \mid x_{\triangleleft} \leq x_{\triangleright} + c\}]).\, \{e :: I \mid e_{\triangleleft} \leq e_{\triangleright} + \epsilon + c + \delta e^{-\epsilon}\}$$

# The Implementation

- Hybrid SMT/Bidirectional type checking.
- Why3 as the SMT backend, multiple solvers required.
- Verification using top-level annotations (+2 cuts).
- Top-level types act as the specification.
- Support for debug of type-checking failures important.

# Benchmarks

| Example | # Lines | Verif. time |
|---|---|---|
| histogram | 25 | 2.66 s. |
| dummysum | 31 | 11.95 s. |
| noisysum | 55 | 3.64 s. |
| two-level-a | 38 | 2.55 s. |
| two-level-b | 56 | 3.94 s. |
| binary | 95 | 18.56 s. |
| idc | 73 | 27.60 s. |
| dualquery | 128 | 27.71 s. |
| competitive-b | 81 | 2.80 s. |
| competitive | 75 | 4.19 s. |
| fixedprice | 10 | 0.90 s. |
| summarization | 471 | 238.42 s. |

Table : Benchmarks

# Future work and Conclusions:

## Future Work:

- More examples from the algorithms community.
- More examples from the security/cryptography domain.
- More properties: accuracy, fancier distributions.
- Extensions to the language.

# Future work and Conclusions:

## Future Work:

- More examples from the algorithms community.
- More examples from the security/cryptography domain.
- More properties: accuracy, fancier distributions.
- Extensions to the language.

## Conclusions

- Higher-Order Approximate Probabilistic Relational Refinement Types: HOARe2
- Built-in support for approximate reasoning.
- Logic seems to capture many examples.
- Automatic verification worked reasonably well.
- SMT interaction is still a challenge.

# Thank you

Questions?

# More on the Aggregative Example:

## Expected Payoff for the deviating agent

```
let expay br∗ dev∗ br dev =
  E (mlet   sums = mkSums k br∗ br       in
     let    s•   = search k br∗ br sums in
     let    a∗   = dev∗ (br∗ s•)         in
     let    a    = dev  (br  s•)         in
     let    p∗   = pay∗ a∗ (sign a∗ a)   in
     munit p∗) (λx. x)
```

$s^{\bullet}$ is close to the true signal on the strategy profile.

# More on the Aggregative Example:

## Type for *expay*

$$\begin{array}{rl} & \{\mathtt{br*}\ \ :: \mathbb{R} \to A \mid \forall s, a.\ \mathtt{pay*}\ (\mathtt{br*}_{\triangleleft}\ s)\ s \geq \mathtt{pay*}\ a\ s\} \\ \to & \{\mathtt{dev*} :: A \to A \mid \forall x.\ \mathtt{dev*}_{\triangleleft}\ x = x\} \\ \to & \{\mathtt{br}\ \ :: \mathbb{R} \to A \mid \mathtt{br}_{\triangleleft} = \mathtt{br}_{\triangleright}\} \\ \to & \{\mathtt{dev}\ \ :: A \to A \mid \forall a.\ \mathtt{dev}_{\triangleleft}\ a = \mathtt{dev}_{\triangleright}\ a = a\} \\ \to & \{u\ \ :: \overline{\mathbb{R}}^{+} \mid u_{\triangleleft} \geq u_{\triangleright} - \alpha\}. \end{array}$$

## Extended type for Laplace

**`lap`** with a refinement type capturing *accuracy*:

$$\Pi(x :: \mathbb{R}).\, \mathfrak{M}_{\epsilon|x_{\triangleleft} - x_{\triangleright}|, \beta}[\{u :: \mathbb{R} \mid u_{\triangleleft} = u_{\triangleright} \wedge |x_{\triangleleft} - u_{\triangleleft}| < T\}]$$