Verification of Mechanism Design with Approximate Relational Refinement Types

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

Dundee, IMDEA Software Institute, Mines ParisTech, UPenn

Avril 16 2015, Rennes
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.
- *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**: include agent’s data 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*: include agent’s data vs not.

Probabilistic Reasoning
Interesting mechanisms are randomized, properties rely on:

- Expected values.
- *Distance* on distributions.
Higher-Order Approximate Relational Refinement Types!

**Types:** Functional programs, properties as types.

**Refinements:** We can build more precise types using formulas:

**Higher-Order:** We can refine over functions:

**Relational:** Types are *relations* over two runs.

**Approximate:** Primitive *relations* over two runs.

Gluing everything together is not easy!
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.
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.
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.
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 $M : A^n \rightarrow \mathcal{O}$.
- A payoff for every agent $P_i : \mathcal{O} \rightarrow R^+$.
- Probabilistic algorithms are common! Payoff becomes expected payoff.
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 $M : A^n \rightarrow \mathcal{O}$.
- A payoff for every agent $P_i : \mathcal{O} \rightarrow R^+$.
- 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.
Auctions

- Buyers (agents), \textit{bids} (actions), seller (mechanism).
- Outcome: price, goods assignation.
- An auction is \textit{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 $i$, $P_i : A^n \rightarrow R^+$, 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.
Example: Truthful Auctions for Digital Goods

- Price a good with infinite supply. (i.e: Digital goods)
Example: Truthful Auctions for Digital Goods

- Price a good with infinite supply. (i.e: Digital goods)
- Bidders and seller.
Example: Truthful Auctions for Digital Goods

- 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.
Example: Truthful Auctions for Digital Goods

- 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.
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.
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!
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.
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 $b_\downarrow = v$ vs $b_\uparrow \neq v$. The interesting cases are when the bidder gets the item in one run and doesn’t in the other:

- If $b_\uparrow$ got the item, utility is negative, thus less than 0 for the $b_\downarrow$ case (remember $b_\downarrow$ didn’t get the item).

- If $b_\downarrow$ got the item, utility will be greater or equal than 0, thus better or equal than $b_\uparrow$’s utility (0).
Verifying that the fixed price auction is truthful goes in two steps:

- We write a program that runs the auction and computes the utility of the buyer.

\[
\text{utility} : \ (\text{price} : \mathbb{R}) \rightarrow (\text{val} : \mathbb{R}) \rightarrow \{\text{bid} : \mathbb{R}\} \rightarrow \{u : \mathbb{R}\}
\]
Verifying that the fixed price auction is truthful goes in two steps:

- We write a program that runs the auction and computes the utility of the buyer.
- We encode truthfulness in the types of the utility function.

\[
\text{utility} : \ (\text{price} : \mathbb{R}) \rightarrow (\text{val} : \mathbb{R}) \rightarrow \{\text{bid} : \mathbb{R} \mid \text{bid} = \text{val}\} \rightarrow \{u : \mathbb{R} \mid u = u_{\triangleright}\}
\]
The System: Relational Refinement Types

Variables
Relational variables, \( x \in \mathcal{X}_R \); left/right instances \( x_\downarrow, x_\uparrow \in \mathcal{X}_R^\times \).

Expressions
\[
e^m ::= C \mid x \in \mathcal{X}^m \mid e \mid \lambda x. e \mid \text{case } e \text{ with } \{ \varepsilon \Rightarrow e \mid x :: x \Rightarrow e \}
\mid \text{letrec}^\uparrow f x = e \mid \text{letrec}^\downarrow f x = e
\mid e^\uparrow \mid \text{let}^\uparrow x = e \text{ in } e \mid \text{unit}_M e \mid \text{bind}_M x = e \text{ in } e
\]

Regular Types
\[
\tilde{\tau}, \tilde{\sigma}, \ldots \in \text{CoreTy} ::= \bullet \mid \mathbb{B} \mid \mathbb{N} \mid \overline{\mathbb{R}} \mid \overline{\mathbb{R}}^+ \mid L[\tilde{\tau}]
\]
\[
\tau, \sigma, \ldots \in \text{Ty} ::= \tilde{\tau} \mid M[\tau] \mid C[\tau] \mid \tau \rightarrow \sigma
\]

Relational Refinement Types
\[
T, U \in \mathcal{T} ::= \tilde{\tau} \mid M_{\varepsilon, \delta}[T] \mid C[T] \mid \Pi(x :: T). T \mid \{ x :: T \mid \phi \}
\]
\[
\phi, \psi \in \mathcal{A} ::= Q(x :: \tau). \phi \mid Q(x :: T). \phi
\mid C(\phi_1, \ldots, \phi_n) \mid e^\times = e'^\times \mid e^\times \leq e'^\times
\]
\[
C = \{ \top/0, \bot/0, \neg/1, \lor/2, \land/2, \Rightarrow/2 \} \]
Regular refinement types no enough to capture some properties.

\( k \)-sensitive function
Regular refinement types no enough to capture some properties.

$k$-sensitive function
Regular refinement types no enough to capture some properties.

$k$-sensitive function
Regular refinement types no enough to capture some properties.

\[ k \text{-sensitive function} \]

\[ \forall x_1, x_2. |f(x_1) - f(x_2)| \leq k \cdot |x_1 - x_2| \]
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?
For the property:

\[ \forall x_1, x_2. |f(x_1) - f(x_2)| \leq k \cdot |x_1 - x_2| \]
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:

$$\{ f : \mathbb{R} \rightarrow \mathbb{R} \mid \forall x :: \mathbb{R}. |f(x_{\downarrow}) - f(x_{\uparrow})| \leq k \cdot |x_{\downarrow} - x_{\uparrow}| \}$$
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 subtyping for non-relational types:

\[\Gamma \vDash e : T \implies \Gamma \vDash \phi[x/e] \implies e : \{x : T | \phi\}\]
The System: Semantics

Semantic subtyping for non-relational types:

\[
\begin{align*}
\Gamma \vdash e : T & \quad \Gamma \models \phi[x/e] \\
\vdash e : \{x : T \mid \phi\} & \quad \vdash e : T \Rightarrow e \in \llbracket T \rrbracket
\end{align*}
\]
The System: Semantics

Semantic subtyping for non-relational types:

\[
\begin{align*}
\frac{\vdash e : T \quad \Gamma \models \phi[x/e]}{\vdash e : \{x : T \mid \phi\}} \\
\vdash e : T \Rightarrow e \in \llbracket T \rrbracket
\end{align*}
\]

\[
\begin{align*}
\frac{\forall \vdash \phi(v)}{v \in \llbracket T \rrbracket \vdash \phi(v)} \\
\frac{v \in \llbracket \{x : T \mid \phi(x)\} \rrbracket}{v \in \llbracket \{x : T \mid \phi(x)\} \rrbracket}
\end{align*}
\]
The System: Semantics

Semantic subtyping for non-relational types:

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

Our types are relations over values:
The System: Semantics

Semantic subtyping for non-relational types:

\[ \vdash e : T \quad \Gamma \models \phi[x/e] \quad \vdash e : \{x : T \mid \phi\} \]

\[ \vdash e : T \Rightarrow e \in \llbracket T \rrbracket \quad \forall v \in \llbracket T \rrbracket \quad \models \phi(v) \]

\[ \forall v \in \llbracket \{x : T \mid \phi(x)\} \rrbracket \]

Our types are relations over values:

\[ (\llbracket T \rrbracket_\theta \subseteq \llbracket \llbracket T \rrbracket \times \llbracket T \rrbracket) \]

\[ (d_1, d_2) \in \llbracket \llbracket T \rrbracket \times \llbracket \llbracket T \rrbracket \]

\[ (d_1, d_2) \in (\llbracket T \rrbracket_\theta \times \llbracket T \rrbracket_\theta) \]

\[ (d_1, d_2) \in (\llbracket \llbracket x : T \mid \phi \rrbracket \rrbracket_\theta \times \llbracket \llbracket x : T \mid \phi \rrbracket \rrbracket_\theta) \]

\[ (f_1, f_2) \in \llbracket \llbracket T \rrbracket \rightarrow \llbracket U \rrbracket \rrbracket \quad \forall (d_1, d_2) \in (\llbracket \llbracket T \rrbracket_\theta \times \llbracket \llbracket T \rrbracket_\theta) \]

\[ (f_1, f_2) \in \llbracket \llbracket \Pi(x : T). U \rrbracket \rrbracket_\theta \]
SubTyping

**Sub-Reflected**

\[
\frac{g \vdash T}{g \vdash T \leq T}
\]

**Sub-Transitive**

\[
\frac{g \vdash T \leq U \quad g \vdash U \leq V}{g \vdash T \leq V}
\]

**Sub-Left**

\[
\frac{g \vdash \{x :: T \mid \phi\}}{g \vdash \{x :: T \mid \phi\} \leq T}
\]

**Sub-Right**

\[
\frac{g \vdash T \leq U \quad \|g, x :: U\| \vdash \phi \quad \forall \theta. \theta \vdash g, x :: T \Rightarrow [\phi]_\theta}{g \vdash T \leq \{x :: U \mid \phi\}}
\]

**Sub-Product**

\[
\frac{g \vdash T_2 \leq T_1 \quad g, x :: T_2 \vdash U_1 \leq U_2}{g \vdash \Pi(x :: T_1). U_1 \leq \Pi(x :: T_2). U_2}
\]
The typing judgment relates two programs to a type:

\[ G \vdash e_1 \sim e_2 :: T \]
The System: Typing

The typing judgment relates two programs to a type:

\[ \mathcal{G} \vdash e_1 \sim e_2 :: T \]

Soundness

\[ \mathcal{G} \vdash e_1 \sim e_2 :: T \Rightarrow \forall \mathcal{G} \vdash \theta, ([e_1]_\theta, [e_2]_\theta) \in ([T]_\theta) \]
The typing judgment relates two programs to a type:

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

Soundness

$$\mathcal{G} \vdash e_1 \sim e_2 :: T \Rightarrow \forall \mathcal{G} \vdash \theta, ([e_1]_{\theta}, [e_2]_{\theta}) \in (T)_{\theta}$$

Synchronicity

In most cases programs are synchronous, so we use:

$$\mathcal{G} \vdash e :: T \equiv \mathcal{G} \vdash e_{\leftarrow} \sim e_{\rightarrow} :: T$$

with $e_{\leftarrow}$, $e_{\rightarrow}$ projecting the variables in $e$. 
Base Typing Rules

\[
\begin{align*}
\text{VAR} & \quad \frac{x :: T \in \text{dom}(\mathcal{G})}{\mathcal{G} \vdash x :: T} \\
\text{ABS} & \quad \frac{\mathcal{G}, x :: T \vdash e :: U}{\mathcal{G} \vdash \lambda x. e :: \Pi(x :: T). U} \\
\text{APP} & \quad \frac{\mathcal{G} \vdash e_f :: \Pi(x :: T). U \quad \mathcal{G} \vdash e_a :: T}{\mathcal{G} \vdash e_f \ e_a :: U\{x \mapsto e_a\}}
\end{align*}
\]
Base Typing Rules

\[
\begin{align*}
\text{VAR} & \quad x :: T \in \text{dom}(G) \\
& \quad \frac{\quad G \vdash x :: T}{}
\end{align*}
\]

\[
\begin{align*}
\text{ABS} & \quad G, x :: T \vdash e :: U \\
& \quad \frac{\quad G \vdash \lambda x. e :: \Pi(x :: T) \cdot U}{}
\end{align*}
\]

\[
\begin{align*}
\text{APP} & \quad G \vdash e_f :: \Pi(x :: T) \cdot U \\
& \quad \frac{\quad G \vdash e_a :: T}{}
\end{align*}
\]

\[
\begin{align*}
G \vdash e :: L[\tilde{\tau}] & \quad \forall \theta. \theta \vdash G \Rightarrow \text{skeleton}(e_\downarrow, e_\uparrow) \\
& \quad G, \{e_\downarrow = e_\uparrow = \epsilon\} \vdash e_1 :: T
\end{align*}
\]

\[
\begin{align*}
\text{CASE} & \quad G, x :: \tilde{\tau}, y :: L[\tilde{\tau}], \{e_\downarrow = x_\downarrow :: y_\downarrow \land e_\uparrow = x_\uparrow :: y_\uparrow\} \vdash e_2 :: T \\
& \quad \frac{\quad G \vdash \text{case } e \text{ with } [\epsilon \Rightarrow e_1 \mid x :: y \Rightarrow e_2] :: T}{}
\end{align*}
\]
Typing Rules for Recursion

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

\[
\begin{align*}
    \text{LETREC} & \\
    G, f : \Pi(x :: T). U \vdash \lambda x. e : \Pi(x :: T). U \\
    G \vdash \Pi(x :: T). U \quad \text{SN-guard} \\
    \frac{}{G \vdash \text{letrec} \downarrow f \ x = e : \Pi(x :: T). U}
\end{align*}
\]
To ensure consistency at higher-types, we must embed non-terminating computations in the partiality monad:

\[
\begin{align*}
\text{LETREC} & \quad \frac{G, f :: \Pi(x :: T). U \vdash \lambda x. e :: \Pi(x :: T). U}{G \vdash \Pi(x :: T). U} \quad \text{SN-guard} \\
\text{LETRECSN} & \quad \frac{G \vdash \Pi(x :: T). U}{G \vdash \text{letrec} f x = e :: \Pi(x :: T). U}
\end{align*}
\]

\[
\begin{align*}
\text{LETREC} & \quad \frac{G \vdash \Pi(x :: T). \mathcal{C}[U]}{G, f :: \Pi(x :: T). \mathcal{C}[U] \vdash \lambda x. e :: \Pi(x :: T). \mathcal{C}[U]} \\
\text{LETRECSN} & \quad \frac{G \vdash \Pi(x :: T). \mathcal{C}[U]}{G \vdash \text{letrec} f x = e :: \Pi(x :: T). \mathcal{C}[U]}
\end{align*}
\]
Typing Rules for Recursion

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

\[
\begin{align*}
\text{LETRECSN} & \quad \frac{\Gamma, f :: \Pi(x :: T). \ U \vdash \lambda x. \ e :: \Pi(x :: T). \ U \quad \text{SN-guard}}{\Gamma \vdash \text{letrec} \downarrow f \ x = e :: \Pi(x :: T). \ U}
\end{align*}
\]

\[
\begin{align*}
\text{LETREC} & \quad \frac{\Gamma \vdash \Pi(x :: T). \ C[U]}{\Gamma, f :: \Pi(x :: T). \ C[U] \vdash \lambda x. \ e :: \Pi(x :: T). \ C[U]}
\end{align*}
\]

\[
\begin{align*}
\text{UNITC} & \quad \frac{\Gamma \vdash e :: T}{\Gamma \vdash e \uparrow :: \ C[T]}
\end{align*}
\]

\[
\begin{align*}
\text{BINDC} & \quad \frac{\Gamma \vdash e_1 :: \ C[T_1] \quad \Gamma \vdash e_2 :: \ C[T_2]}{\Gamma \vdash \text{let} \uparrow x = e_1 \ \text{in} \ e_2 :: \ C[T_2]}
\end{align*}
\]
Asynchronous Rules

\[
\begin{align*}
\text{ASYM} & \quad \frac{G \vdash e_1 \sim e_2 :: T}{G \leftrightarrow \vdash e_2 \leftrightarrow \sim e_1 \leftrightarrow :: T} \\
\text{AREDLEFT} & \quad \frac{e_1 \rightarrow e'_1 \quad G \vdash e_1 \sim e_2 :: T}{G \vdash e'_1 \sim e_2 :: T}
\end{align*}
\]
Asynchronous Rules

ASYM
\[
\frac{G \vdash e_1 \sim e_2 :: T}{G \leftrightarrow \vdash e_2 \leftrightarrow \sim e_1 \leftrightarrow :: T}
\]

AREDLEFT
\[
\frac{e_1 \rightarrow e'_1 \quad G \vdash e_1 \sim e_2 :: T}{G \vdash e'_1 \sim e_2 :: T}
\]

ACASE
\[
\frac{|G| \vdash e : L[\tilde{\tau}] \quad |G| \vdash e' : |T| \quad G, \{e_\triangleleft = \epsilon\} \vdash e_1 \sim e' :: T \quad G, x :: \tilde{\tau}, y :: L[\tilde{\tau}], \{e_\triangleleft = x_\triangleleft :: y_\triangleleft\} \vdash e_2 \sim e' :: T}{G \vdash \text{case } e \text{ with } [\epsilon \Rightarrow e_1 \mid x :: y \Rightarrow e_2] \sim e' :: T}
\]
We model the utility as a program:

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

We use asynchronous reasoning. The interesting case is:

\[ \{ b \triangleq v, b \triangleright p, b \triangleright p \} \models v - p \sim 0.0 \]

substituting \[ v - p / u \triangleq, 0.0 / u \triangleright \] we get the proof obligation:

\[ v \geq p \Rightarrow v - p \geq 0.0 \]
We model the utility as a program:

```plaintext
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 model the utility as a program:

\[
\text{let } \text{fp\_utility} (v : R) \{b :: R \mid b\triangleleft = v\} (p : R) :
\{ u :: R \mid u\triangleleft \geq u\triangledown \} = \\
\text{if } b \geq p \text{ then } v - p \\
\text{else } 0.0
\]

We use asynchronous reasoning. The interesting case is:

\[
\{b\triangleleft = v, b\triangleleft \geq p, b\triangledown < p\} \vdash v - p \sim 0.0 :: \{u :: R \mid u\triangleleft \geq u\triangledown \}
\]

substituting \([v - p/u\triangleleft, 0.0/u\triangledown]\) we get the proof obligation:
The Fixed Price Auction

We model the utility as a program:

\[
\text{let } \text{fp\_utility} (v : R) \{b :: R | b ▽ = v\} (p : R) : \{ u :: R | u ▽ ≥ u ▼ \} = \\
\text{if } b ≥ p \text{ then } v - p \\
\text{else } 0.0
\]

We use asynchronous reasoning. The interesting case is:

\[
\{b ▽ = v, b ▽ ≥ p, b ▼ < p\} ⊢ v - p ∼ 0.0 :: \{u :: R | u ▽ ≥ u ▼ \}
\]

substituting \([v - p/u ▽, 0.0/u ▼]\) we get the proof obligation:

\[
v ≥ p \Rightarrow v - p ≥ 0.0
\]
The Distribution Type

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

A first approach to lifting

\[(\mu_1, \mu_2) \in \mathcal{M}[|T|] \times \mathcal{M}[|T|]\]

\[\frac{(\mu_1, \mu_2) \in (\mathcal{M}[|T|])_\theta}{(\mu_1, \mu_2) \in (\mathcal{M}[|T|])_\theta}\]
The Distribution Type

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

A first approach to lifting

\[(d_1, d_2) \in \langle T \rangle_\theta \quad (\mu_1, \mu_2) \in \mathcal{M}[|T|] \times \mathcal{M}[|T|] \]

\[(\mu_1, \mu_2) \in \langle \mathcal{M}[T] \rangle_\theta\]
The Distribution Type

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

A first approach to lifting

\[
(\mathbf{d}_1, \mathbf{d}_2) \in (\mathcal{T})_\theta \quad (\mu_1, \mu_2) \in \mathcal{M}[|\mathcal{T}|] \times \mathcal{M}[|\mathcal{T}|]
\]

\[
(\mu_1, \mu_2) \in (\mathcal{M}[\mathcal{T}])_\theta
\]
The Distribution Type

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

A first approach to lifting

\[
(d_1, d_2) \in (|T|)_\theta \\
(\mu_1, \mu_2) \in \mathcal{M}[|T|] \times \mathcal{M}[|T|] \\
(\mu_1, \mu_2) \in (\mathcal{M}[|T|])_\theta
\]

We need to relate \((d_1, d_2)\) to \((\mu_1, \mu_2)\)! Informally, we have to respect the relation on the base type.
The Distribution Type

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

A first approach to lifting

\[(d_1, d_2) \in \langle |T| \rangle_\theta \quad (\mu_1, \mu_2) \in \mathcal{M}[|T|] \times \mathcal{M}[|T|] \quad (\mu_1, \mu_2) \in \langle \mathcal{M}[T] \rangle_\theta \]

We need to relate \((d_1, d_2)\) to \((\mu_1, \mu_2)\)! Informally, we have to respect the relation on the base type.

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

\[
\Pr_{x \leftarrow \mu_1} x \in [T] = \sum_{y \in T} \Pr_{(x, y) \leftarrow \mu} (x, y) \in \langle T \rangle_\theta
\]
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 \( \mu \in \mathcal{M}[T_1 \times T_2] \) such that for every \( H \subseteq T_1 \), we have

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

and symmetrically for \( T_2 \).

“Probability of events in \( \mu_1 \ \mu_2 \) must respect the relation”.

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

\[
\mu_1, \mu_2 \in M[|T|] \quad L((|T|)_\theta) \mu_1 \mu_2 \\
(\mu_1, \mu_2) \in (M[|T|])_\theta
\]

In particular, the type \( M[\{x :: T \mid x_\downarrow = x_\uparrow\}] \) forces equal distributions.
Examples of Lifting

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

\[
\begin{align*}
\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{align*}
\[
\begin{align*}
\mu_1(F) &= 1 & \mu(F, F) &= 1 \\
\mu_1(T) &= 0 & \mu(F, T) &= 0 \\
\mu_2(F) &= 1 & \mu(T, F) &= 0 \\
\mu_2(T) &= 0 & \mu(T, T) &= 0
\end{align*}
\]
Expectation

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

$$E_\mu f := \sum_{x \in D} (f x) \cdot (\mu x)$$
Expectation

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

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

We capture monotonicity of expectation as:

$$I := [0, 1]$$

$$IBF := \{ f :: D \rightarrow I \mid \forall d : D. f \downarrow d \geq f \uparrow d \}$$

$$E : \Pi(\mu :: M[\{ x :: D \mid x \downarrow = x \uparrow \}]). \Pi(f :: IBF). \{ e :: I \mid e \downarrow \geq e \uparrow \}$$

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.
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)
let E (mu : M[ r : α | r_\triangleleft = r_\triangleright ]) (f : α → real | ∀ x : α, f_\triangleleft x ≥ f_\triangleright x): { r :: real | r_\triangleleft ≥ r_\triangleright } = ....

let utility (v : real) (bid :: { b :: R | b_\triangleleft = v }) (otherbids : L[R]) (g, groups) : (B * L[B]) : { u :: real | u_\triangleleft ≥ u_\triangleright } = ...

let auction (n : N) (v : R) (bid :: { b :: R | b_\triangleleft = v }) (otherbids : L[R]): { u :: real | u_\triangleleft ≥ u_\triangleright } =
let grouping :: M{ r :: (B * B list) | r_\triangleleft = r_\triangleright } = ...
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 $F : T \rightarrow S$ 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 $f(t) + \text{random noise}$, scaled by $k$.
Many algorithms are DP: private database release, counters, analytics, strong connection to Mechanism Design!
Differential Privacy

Formal Definition
A probabilistic function $F : T \rightarrow S$ 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 $f(t) + \text{random noise}$, scaled by $k$. 
Differential Privacy

Formal Definition
A probabilistic function $F : T \rightarrow S$ 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 $f(t) + \text{random noise}$, scaled by $k$.

Many algorithms are DP: private database release, counters, analytics, strong connection to Mechanism Design!
We can capture DP with a refinement over the type of probability distributions using the definition of $\Delta$-distance:

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

$$\{d :: T \mid \text{Adj}(d_\downarrow, d_\uparrow)\} \rightarrow \{r :: \mathcal{M}[\mathbb{R}] \mid \Delta_\epsilon(r_\downarrow, r_\uparrow) \leq \delta\}$$

However, verification conditions involving $\Delta$ are quite hard.
Our solution: Internalize distribution distance in the types:

\[
\mu_1, \mu_2 \in \mathcal{M}[|T|] \quad \mathcal{L}_{\epsilon, \delta}(\langle |T| \rangle_\theta) \quad \mu_1 \mu_2 \quad (\mu_1, \mu_2) \in \langle \mathcal{M}_{\epsilon, \delta}[T] \rangle_\theta
\]

Lifting is extended from \( p = p_1 \) to \( p \leq p_1 \leq \exp(p) + \delta \).
Our solution: Internalize distribution distance in the types:

\[
\mu_1, \mu_2 \in \mathcal{M}[|T|] \quad \mathcal{L}_{\epsilon,\delta}(|T|) \mu_1 \mu_2 \\
(\mu_1, \mu_2) \in (\mathcal{M}_{\epsilon,\delta}[T])_\theta
\]

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

Capturing DP

The interpretation of \( \mathcal{M}_{\epsilon,\delta}[\{ r :: \mathbb{R} \mid r_< = r_> \}] \) is the set of pairs of probability distributions that are \( (\epsilon, \delta) \)-apart, capturing DP.
Our solution: Internalize distribution distance in the types:

\[
\begin{align*}
\mu_1, \mu_2 \in \mathcal{M}[|T|] & \quad \mathcal{L}_{\epsilon,\delta}(|T|) \mu_1 \mu_2 \\
(\mu_1, \mu_2) \in (\mathcal{M}_{\epsilon,\delta}[T])_\theta
\end{align*}
\]

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

Capturing DP

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

\[
f : \{ d :: T \mid \text{Adj}(d_\downarrow, d_\uparrow) \} \rightarrow \mathcal{M}_{\epsilon,\delta}[\{ r :: \mathbb{R} \mid r_\downarrow = r_\uparrow \}]
\]
Reasoning about distance is compositional:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SUB-M</strong></td>
<td>$G \vdash T \leq U $</td>
</tr>
<tr>
<td></td>
<td>$\forall \theta. \theta \vdash G, x :: T \Rightarrow [\epsilon_1 \leq \epsilon_2 \land \delta_1 \leq \delta_2]_\theta$</td>
</tr>
<tr>
<td></td>
<td>$G \vdash M_{\epsilon_1,\delta_1}[T] \leq M_{\epsilon_2,\delta_2}[U]$</td>
</tr>
<tr>
<td><strong>UNITM</strong></td>
<td>$G \vdash e :: T $</td>
</tr>
<tr>
<td></td>
<td>$G \vdash \text{unit}<em>M e :: M</em>{\epsilon,\delta}[T]$</td>
</tr>
<tr>
<td><strong>BINDM</strong></td>
<td>$G \vdash e_1 :: M_{\epsilon_1,\delta_1}[T_1]$</td>
</tr>
<tr>
<td></td>
<td>$G, x :: T_1 \vdash e_2 :: M_{\epsilon_2,\delta_2}[T_2]$</td>
</tr>
<tr>
<td></td>
<td>$G \vdash \text{bind}<em>M x = e_1 \text{ in } e_2 :: M</em>{\epsilon_1+\epsilon_2,\delta_1+\delta_2}[T_2]$</td>
</tr>
</tbody>
</table>

Bind is distance-adjusting sampling.
Recall the Laplace Mechanism:

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

$$\text{lap} : \Pi(\epsilon :: \mathbb{R}). \Pi(x :: \mathbb{R}). \mathcal{M}_{\epsilon*|x\downarrow-x\uparrow|,0}[\{r :: \mathbb{R} \mid r\downarrow = r\uparrow\}]$$

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

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

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

Note that the actual distance $\epsilon \ast |x_\triangleleft - x_\triangleright|$ 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.

```ocaml
let rec histogram {l :: L(R) | Adj xₜ xₜₜ} : M[e * d(lₜ,lₜₜ)] { r :: L(R) | rₜ = rₜₜ} =
  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:

\[
e \cdot d(xₜ :: xs, xₜₜ :: xs) \geq e \cdot (d(xₜ, xₜₜ) + d(xs, xs))
\]

which is implied by the adjacency precondition.
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< :: xs<, x> :: xs>) ≥ e * (d(x<, x>) + d(xs<, xs>))
```

which is implied by the adjacency precondition.
We verify the computation of an approximate Nash-equilibrium.

\[ (a_1, \ldots, a_n) \text{ is an } \alpha\text{-approximate Nash-equilibrium if no single agent } i \text{ can gain more than } \alpha \text{ payoff by unilateral deviation: For all agents } i \text{ and actions } a'_i: \]
\[ \mathbb{E}[P_i(a_1, \ldots, a_i', \ldots, a_N)] \geq \mathbb{E}[P_i(a_1, \ldots, a_i, \ldots, a_N)] - \alpha. \]

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.
We verify the computation of an approximate Nash-equilibrium.

$n$ agents 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 $a'_i$:

$$E[P_i(a_1, \ldots, a_i, \ldots a_N)] \geq E[P_i(a_1, \ldots, a'_i, \ldots a_N)] - \alpha.$$
We verify the computation of an approximate Nash-equilibrium.

\( n \) agents 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 \( a'_i \):

\[
E[P_i(a_1, \ldots, a_i, \ldots a_N)] \geq E[P_i(a_1, \ldots, a'_i, \ldots a_N)] - \alpha.
\]

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.
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 $i$. 
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:

```plaintext
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:

```haskell
let aggregative_utility ( ... )
    { dev :: act → act | ∀ a : act. dev ≪ a = a ) }:
    { u :: real | u ≪ u ≥ u ≫ - alpha };
```

Relate expectation to distance on the distributions:

$$E : \Pi(\mu :: M_{\epsilon,\delta}[\{ x :: l | x ≪ x ≫ + c \}]).\{ e :: l | e ≪ e ≥ e ≫ + \epsilon + c + \delta e^{-\epsilon} \}$$
Non trivial mechanism and property.

Given algorithm $A$ that takes agent’s reported types and produces an outcome, $RSM$ turns $A$ into a Bayesian Incentive compatible mechanism.

Uses Vickrey-Clarkes-Grove auction.

Program equivalence a significant challenge for our system: Use EasyCrypt.

\[
\{ \text{om} : \text{list}(T \rightarrow M T) \mid \forall j \in [n].(ot \leftarrow \mu \text{ in } \text{om}[j](ot)) = \mu \}
\]

HO proof obligations a challenge for SMT, solve 4 of them manually in Coq.
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.
<table>
<thead>
<tr>
<th>Example</th>
<th># Lines</th>
<th>Verif. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>histogram</td>
<td>25</td>
<td>2.66 s.</td>
</tr>
<tr>
<td>dummysum</td>
<td>31</td>
<td>11.95 s.</td>
</tr>
<tr>
<td>noisysum</td>
<td>55</td>
<td>3.64 s.</td>
</tr>
<tr>
<td>two-level-a</td>
<td>38</td>
<td>2.55 s.</td>
</tr>
<tr>
<td>two-level-b</td>
<td>56</td>
<td>3.94 s.</td>
</tr>
<tr>
<td>binary</td>
<td>95</td>
<td>18.56 s.</td>
</tr>
<tr>
<td>idc</td>
<td>73</td>
<td>27.60 s.</td>
</tr>
<tr>
<td>dualquery</td>
<td>128</td>
<td>27.71 s.</td>
</tr>
<tr>
<td>competitive-b</td>
<td>81</td>
<td>2.80 s.</td>
</tr>
<tr>
<td>competitive</td>
<td>75</td>
<td>4.19 s.</td>
</tr>
<tr>
<td>fixedprice</td>
<td>10</td>
<td>0.90 s.</td>
</tr>
<tr>
<td>summarization</td>
<td>471</td>
<td>238.42 s.</td>
</tr>
</tbody>
</table>

**Table: Benchmarks**
Future work and Conclusions:

Future Work:

- More examples from algorithms/security/cryptography.
- More properties: accuracy, fancier distributions.
- Extensions to the language.
Future work and Conclusions:

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

Conclusions
- Higher-Order Approximate Probabilistic Relational Refinement Types: HOARRe2
- Built-in support for approximate reasoning.
- Logic seems to capture many examples.
- Automatic verification worked reasonably well.
- SMT interaction is still a challenge.