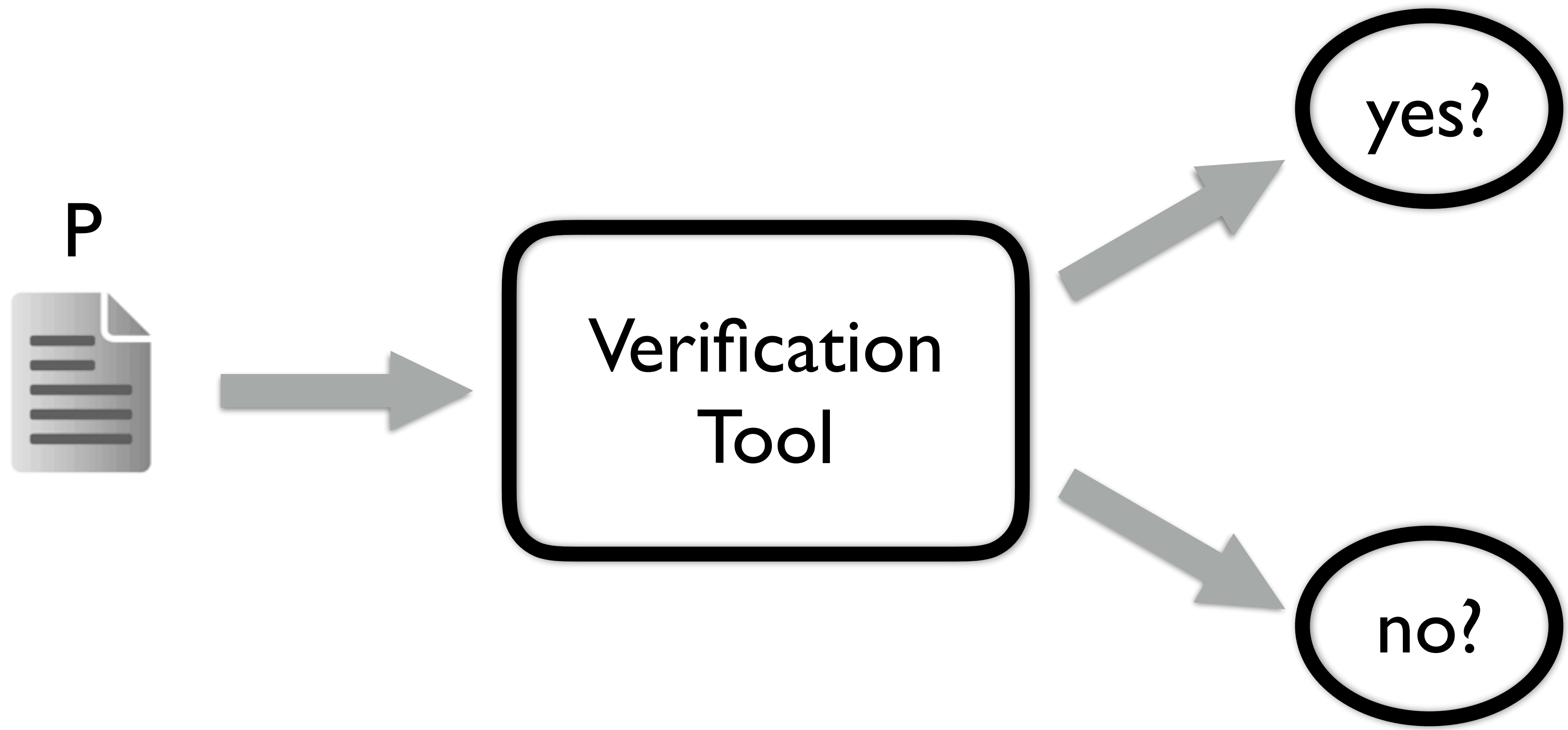
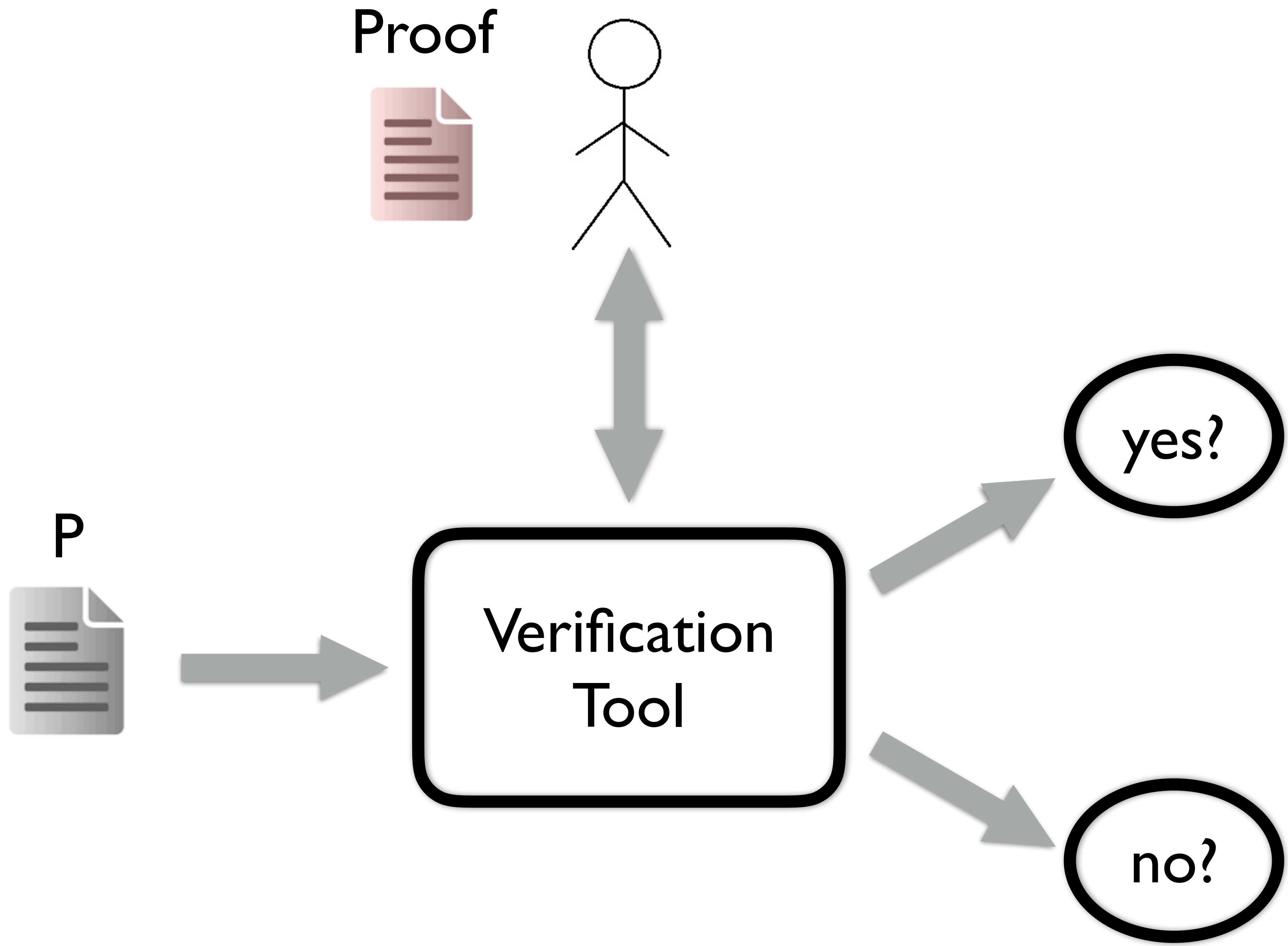
Differential Privacy and Verification

Marco Gaboardi
University at Buffalo, SUNY

Given a program P, is it differentially private?


P
Verification Tool
yes?
no?



Proof
P
Verification Tool
yes?
no?

Given a differentially private program P, does it maintain its accuracy promises?

Given a differentially private program P that maintains its accuracy promises, can we guarantee that it is also efficient?

An algorithm

Algorithm 2 DualQuery

Input: Database $D \in \mathbb{R}^{|\mathcal{X}|}$ (normalized) and linear queries $q_1, \ldots, q_k \in \{0,1\}^{|\mathcal{X}|}$.
Initialize: Let $\mathcal{Q} = \bigcup_{j=1}^{k} q_j \cup \overline{q_j}$, Q^1 uniform distribution on $\mathcal{Q}$,

$$T = \frac{16 \log |\mathcal{Q}|}{\alpha^2}, \qquad \eta = \frac{\alpha}{4}, \qquad s = \frac{48 \log \left(\frac{2|\mathcal{X}|T}{\beta} \right)}{\alpha^2}.$$

For $t = 1, \ldots, T$:
  Sample s queries $\{q_i\}$ from $\mathcal{Q}$ according to Q^t.
  Let $\overline{q} := \frac{1}{s} \sum_i q_i$.
  Find x^t with $\langle \overline{q}, x^t \rangle \geq \max_x \langle \overline{q}, x \rangle - \alpha/4$.
  Update: For each $q \in \mathcal{Q}$:
    $Q_q^{t+1} := \exp(-\eta \langle q, x^t - D \rangle) \cdot Q_q^t$.
  Normalize Q^{t+1}.
Output synthetic database $\widehat{D} := \bigcup_{t=1}^{T} x^t$.

A program

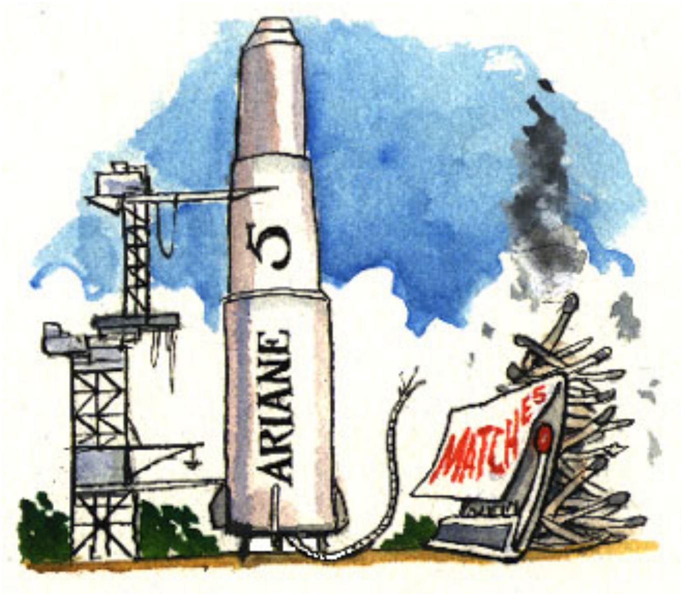
https://github.com/ejgallego/dualquery/

Some issues

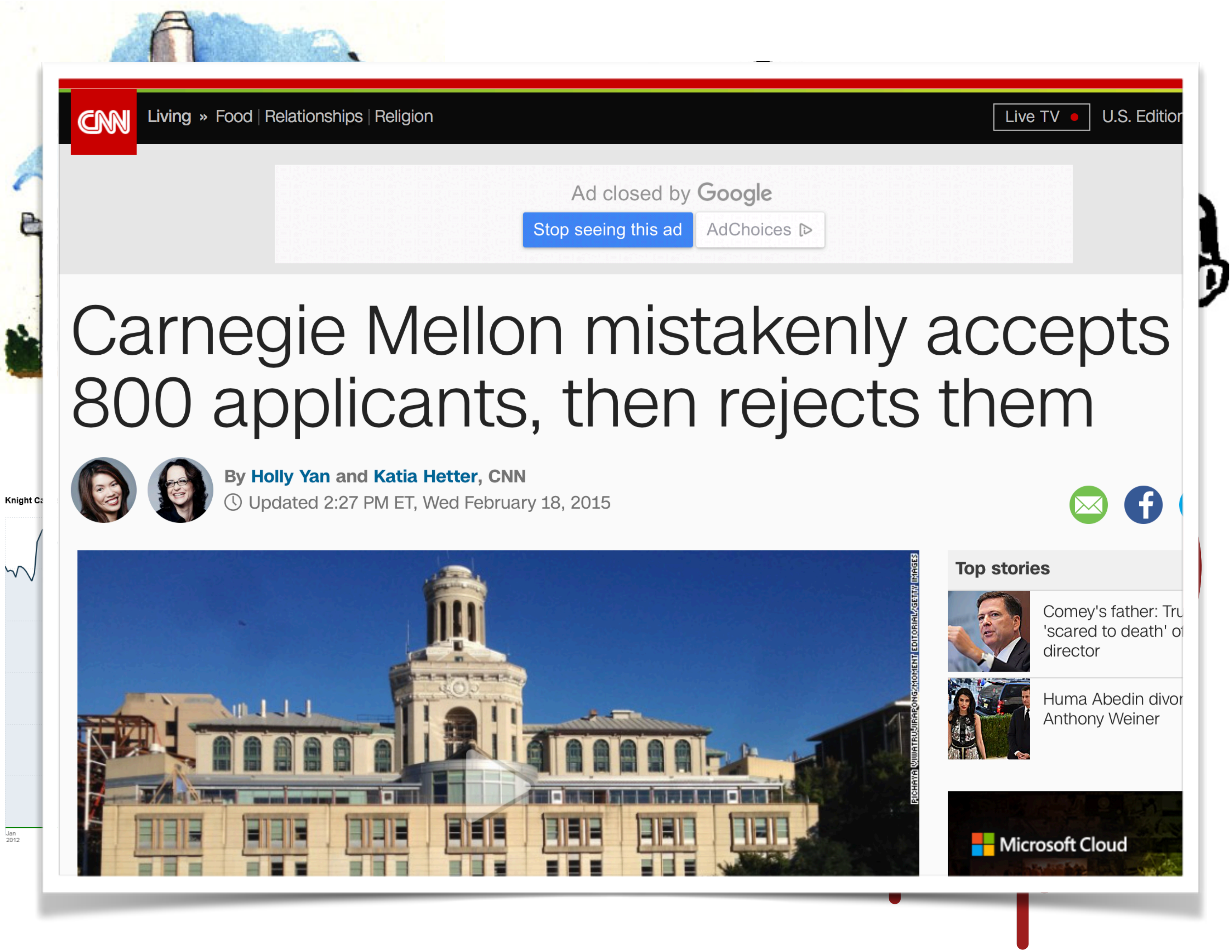
- Are the algorithms bug-free?
- Do the implementations respect their specifications?
- Is the system architecture bug-free?
- Is the code efficient?
- Do the optimization preserve privacy and accuracy?
- Is the actual machine code correct?
- Is the full stack attack-resistant?

Outline

- Few more words on program verification,
- Challenges in the verification of differential privacy,
- Few verification methods developed so far,
- Looking forward.


ARIANE
5
MATCHES


Knight Capital Group
-75%
Jan 2012
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
14
12
10
8
6
4



Living » Food | Relationships | Religion

Live TV U.S. Edition

Carnegie Mellon mistakenly accepts 800 applicants, then rejects them

By **Holly Yan** and **Katia Hetter**, CNN

Updated 2:27 PM ET, Wed February 18, 2015

PICHAYA VIWATRUJIRAPONG/MOMENT EDITORIAL/GETTY IMAGES

Top stories

Comey's father: Tru 'scared to death' of director

Huma Abedin divor Anthony Weiner

Algorithm 1 An instantiation of the SVT proposed in this paper.

Input: $D, Q, \Delta, \mathbf{T} = T_1, T_2, \cdots, c$.
$\epsilon_1 = \epsilon/2, \quad \rho = \mathsf{Lap}(\Delta/\epsilon_1)$
$\epsilon_2 = \epsilon - \epsilon_1$, count = 0
for each query $q_i \in Q$ **do**
  $\nu_i = \mathsf{Lap}(2c\Delta/\epsilon_2)$
  if $q_i(D) + \nu_i \geq T_i + \rho$ **then**
    Output $a_i = \top$
    count = count + 1, **Abort** if count $\geq c$.
  else
    Output $a_i = \perp$

Algorithm 2 SVT in Dwork and Roth 2014 [8].

Input: D, Q, Δ, T, c.
$\epsilon_1 = \epsilon/2, \quad \rho = \mathsf{Lap}(c\Delta/\epsilon_1)$
$\epsilon_2 = \epsilon - \epsilon_1$, count = 0
for each query $q_i \in Q$ **do**
  $\nu_i = \mathsf{Lap}(2c\Delta/\epsilon_1)$
  if $q_i(D) + \nu_i \geq T + \rho$ **then**
    Output $a_i = \top$, $\rho = \mathsf{Lap}(c\Delta/\epsilon_2)$
    count = count + 1, **Abort** if count $\geq c$.
  else
    Output $a_i = \perp$

Algorithm 3 SVT in Roth's 2011 Lecture Notes [15].

Input: D, Q, Δ, T, c.
$\epsilon_1 = \epsilon/2, \quad \rho = \mathsf{Lap}(\Delta/\epsilon_1)$,
$\epsilon_2 = \epsilon - \epsilon_1$, count = 0
for each query $q_i \in Q$ **do**
  $\nu_i = \mathsf{Lap}(c\Delta/\epsilon_2)$
  if $q_i(D) + \nu_i \geq T + \rho$ **then**
    Output $a_i = q_i(D) + \nu_i$
    count = count + 1, **Abort** if count $\geq c$.
  else
    Output $a_i = \perp$

Algorithm 4 SVT in Lee and Clifton 2014 [13].

Input: D, Q, Δ, T, c.
$\epsilon_1 = \epsilon/4, \quad \rho = \mathsf{Lap}(\Delta/\epsilon_1)$
$\epsilon_2 = \epsilon - \epsilon_1$, count = 0
for each query $q_i \in Q$ **do**
  $\nu_i = \mathsf{Lap}(\Delta/\epsilon_2)$
  if $q_i(D) + \nu_i \geq T + \rho$ **then**
    Output $a_i = \top$
    count = count + 1, **Abort** if count $\geq c$.
  else
    Output $a_i = \perp$

Algorithm 5 SVT in Stoddard et al. 2014 [18].

Input: D, Q, Δ, T.
$\epsilon_1 = \epsilon/2, \quad \rho = \mathsf{Lap}(\Delta/\epsilon_1)$
$\epsilon_2 = \epsilon - \epsilon_1$
for each query $q_i \in Q$ **do**
  $\nu_i = 0$
  if $q_i(D) + \nu_i \geq T + \rho$ **then**
    Output $a_i = \top$
  else
    Output $a_i = \perp$

Algorithm 6 SVT in Chen et al. 2015 [1].

Input: $D, Q, \Delta, \mathbf{T} = T_1, T_2, \cdots$.
$\epsilon_1 = \epsilon/2, \quad \rho = \mathsf{Lap}(\Delta/\epsilon_1)$
$\epsilon_2 = \epsilon - \epsilon_1$
for each query $q_i \in Q$ **do**
  $\nu_i = \mathsf{Lap}(\Delta/\epsilon_2)$
  if $q_i(D) + \nu_i \geq T_i + \rho$ **then**
    Output $a_i = \top$
  else
    Output $a_i = \perp$

	Alg. 1	**Alg. 2**	**Alg. 3**	**Alg. 4**	**Alg. 5**	**Alg. 6**
ϵ_1	$\epsilon/2$	$\epsilon/2$	$\epsilon/2$	$\epsilon/4$	$\epsilon/2$	$\epsilon/2$
Scale of threshold noise ρ	Δ/ϵ_1	$c\Delta/\epsilon_1$	Δ/ϵ_1	Δ/ϵ_1	Δ/ϵ_1	Δ/ϵ_1
Reset ρ after each output of $\top$ **(unnecessary)**		Yes				
Scale of query noise ν_i	$2c\Delta/\epsilon_2$	$2c\Delta/\epsilon_2$	$c\Delta/\epsilon_1$	Δ/ϵ_2	0	Δ/ϵ_2
Outputting $q_i + \nu_i$ instead of $\top$ **(not private)**			Yes			
Outputting unbounded $\top$'s **(not private)**					Yes	Yes
Privacy Property	ϵ-DP	ϵ-DP	∞-DP	$\left(\frac{1+6c}{4}\epsilon\right)$-DP	∞-DP	∞-DP

Some successful stories - I

- CompCert - a fully verified C compiler,
- Sel4, CertiKOS - formal verification of OS kernel
- A formal proof of the Odd order theorem,
- A formal proof of Kepler conjecture (lead by T. Hales).

Some successful stories - I

- CompCert - a fully verified C compiler,
- Sel4, CertiKOS - formal verification of OS kernel
- A formal proof of the Odd order theorem,
- A formal proof of Kepler conjecture (lead by T. Hales).

Years of work from very specialized researchers!

Some successful stories - II

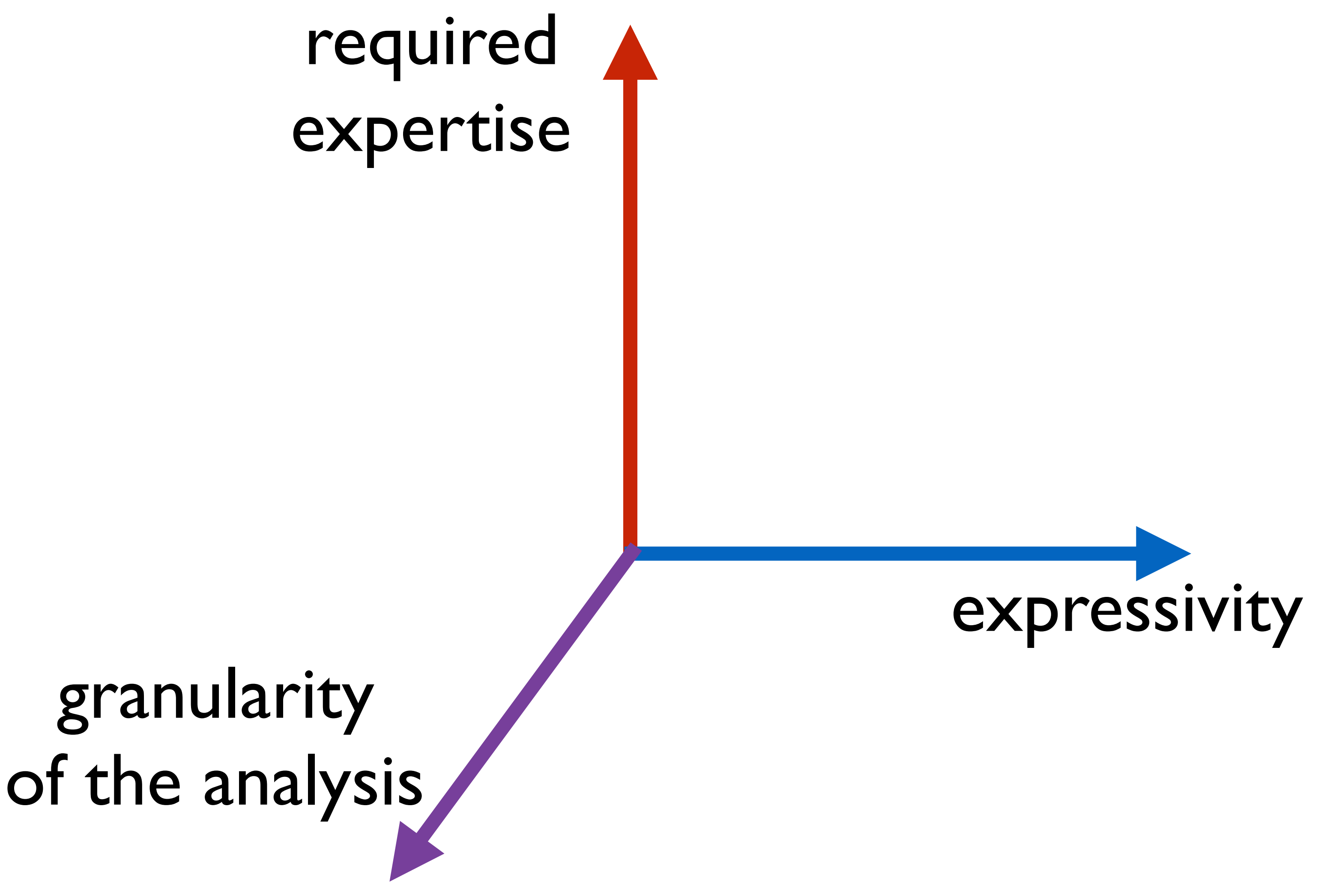
- Automated verification for Integrated Circuit Design.
- Automated verification for Floating point computations,
- Automated verification of Boeing flight control - Astree,
- Automated verification of Facebook code - Infer.

Some successful stories - II

- Automated verification for Integrated Circuit Design.
- Automated verification for Floating point computations,
- Automated verification of Boeing flight control - Astree,
- Automated verification of Facebook code - Infer.

The years of work go in the design of the techniques!

Verification trade-offs



What program verification isn't…

- Algorithm design,
- Trial and error,
- Program testing,
- System engineering,
- A certification process.

What program verification can help with…

- Designing languages for non-experts,
- Guaranteeing the correctness of algorithms,
- Guaranteeing the correctness of code,
- Designing automated techniques for guaranteeing differential privacy,
- Help providing tools for certification process.

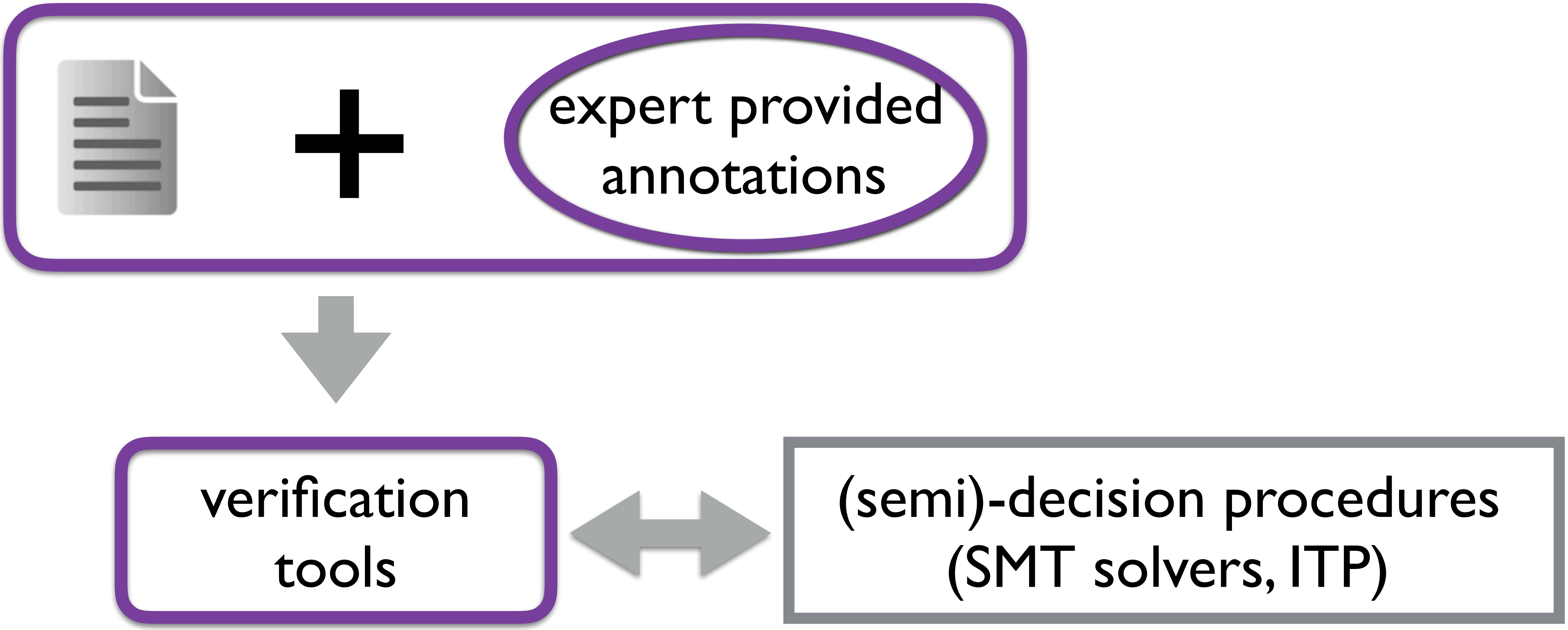
The challenges of differential privacy

Given $\varepsilon,\delta \geq 0$, a mechanism M: db $\rightarrow$ O is
(ε,δ)-differentially private iff
$\forall b_1, b_2$:db at distance one and for every $S \subseteq O$:

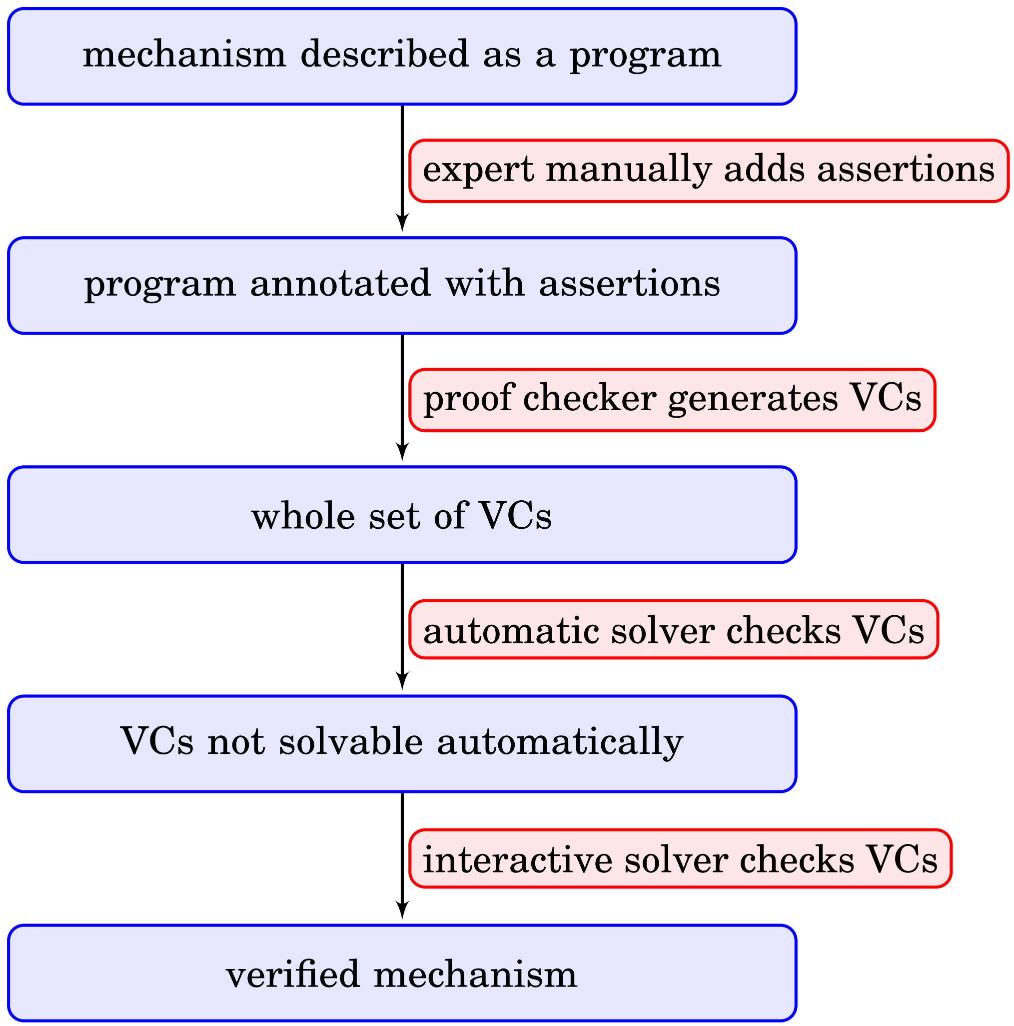
$$\Pr[M(b_1) \in S] \leq \exp(\varepsilon) \cdot \Pr[M(b_2) \in S] + \delta$$

- Relational reasoning,
- Probabilistic reasoning,
- Quantitative reasoning

A 10 thousand ft view on program verification



Work-flow



VCs = Verification Conditions

Semi-decision procedures

- Require a good decomposition of the problem,
- Handle well logical formulas, numerical formulas and their combination,
- Limited support for probabilistic reasoning (usually through decision procedures for counting).

Compositional Reasoning about the privacy budget

Sequential Composition
Let M_i be ϵ_i-differentially private $(1 \leq i \leq k)$.
Then $M(x) = (M_1(x), \ldots, M_k(x))$ is $\sum_{i=0}^{k} \epsilon_i$.

- We can reason about DP programs by monitoring the privacy budget,
- If we have basic components for privacy we can just focus on counting,
- It requires a limited reasoning about probabilities,
- This way of reasoning adapt to other compositions.

Iterated - CDF

CDF(X) = number of records with value ≤ X.

Joe	29	19144	diabets
Bob	48	19146	tumor
Jim	25	34505	flue
Alice	62	19144	diabets
Bill	39	16544	anemia
Sam	61	19144	diabets
...			

CDF(Xn)
.....
CDF(50)=4
CDF(40)=3
CDF(30)=2

List of buckets

PINQ-like languages - McSherry

```
it-CDF (raw : data) (budget : R) (buckets : list) (ε: R)
        : list
{
  var agent = new PINQAgentBudget(budget);
  var db = new PINQueryable<data>(rawdata, agent);
  foreach (var b in buckets)
     b = db.where(y => y.val ≤ b).noisyCount(ε);
     yield return b;
}
```

PINQ-like languages - McSherry

```
it-CDF (raw : data) (budget : R) (buckets : list) (ε: R)
       : list
{
  var agent = new PINQAgentBudget(budget);
  var db = new PINQueryable<data>(rawdata, agent);
  foreach (var b in buckets)
     b = db.where(y => y.val ≤ b).noisyCount(ε);
     yield return b;
}
```

agent is responsible for the budget

PINQ-like languages - McSherry

```
it-CDF (raw : data) (budget : R) (buckets : list) (ε: R)
        : list
{
  var agent = new PINQAgentBudget(budget);
  var db = new PINQueryable<data>(rawdata, agent);
  foreach (var b in buckets)
     b = db.where(y => y.val ≤ b).noisyCount(ε);
     yield return b;
}
```

raw data are accessed through a PINQueryable

PINQ-like languages - McSherry

```
it-CDF (raw : data) (budget : R) (buckets : list) (ε: R)
       : list
{
  var agent = new PINQAgentBudget(budget);
  var db = new PINQueryable<data>(rawdata, agent);
  foreach (var b in buckets)
     b = db.where(y => y.val ≤ b).noisyCount(ε);
     yield return b;
}
```

we have transformations (scaling factor)

PINQ-like languages - McSherry

```
it-CDF (raw : data) (budget : R) (buckets : list) (ε: R)
        : list
{
  var agent = new PINQAgentBudget(budget);
  var db = new PINQueryable<data>(rawdata, agent);
  foreach (var b in buckets)
     b = db.where(y => y.val ≤ b).noisyCount(ε);
     yield return b;
}
```

we have transformations (scaling factor)

aggregate operations (actual budget consumption)

Enough budget?

```
it-CDF (raw : data) (budget : R) (buckets : list) (ε: R)
        : list
{
  var agent = new PINQAgentBudget(budget);
  var db = new PINQueryable<data>(rawdata, agent);
  foreach (var b in buckets)
     b = db.where(y => y.val ≤ b).noisyCount(ε);
     yield return b;
}
```

We can check local vs global budget

Compositional reasoning about sensitivity

$$GS(f) = \max_{v \sim v'} |f(v) - f(v')|$$

- It allows to decompose the analysis/construction of a DP program,
- A metric property of the function (DMNS06),
- It requires a limited reasoning about probabilities,
- Similar worst case reasoning as basic composition.

Fuzz-like languages - Penn

```
it-CDF (b : data) (buckets : list) : list
{
  case buckets of
    |nil    => nil
    |x::xs => size (filter (fun y => y ≤ x) b)))
                :: it-CDF xs b
}
```

Fuzz-like languages - Penn

How sensitive?

```
it-CDF (b :[??] data) (buckets : list) : list
{
  case buckets of
   |nil    => nil
   |x::xs => size (filter (fun y => y ≤ x) b)))
                 :: it-CDF xs b
}
```

Fuzz-like languages - Penn

```
it-CDF (b :[??] data) (buckets : list) : list
{
  case buckets of
   |nil    => nil
   |x::xs => size (filter (fun y => y ≤ x) b))
              :: it-CDF xs b
}
```

Let's assume |- size : [1]data --o R

Fuzz-like languages - Penn

```
it-CDF (b :[??] data) (buckets : list) : list
{
  case buckets of
    |nil    => nil
    |x::xs => size (filter (fun y => y ≤ x) b)))
                 :: it-CDF xs b
}
```

Similarly, `|- filter : [∞]prop --o [1]data --o R`

Fuzz-like languages - Penn

```
it-CDF (b :[??] data) (buckets : list) : list
{
  case buckets of
   |nil   => nil
   |x::xs => size (filter (fun y => y ≤ x) b)))
                :: it-CDF xs b
}
```

(b :[0] data)

(b :[n+1] data)

Fuzz-like languages - Penn

n-sensitive!

```
it-CDF (b :[n] data) (buckets : list[n]) : list
{
  case buckets of
   |nil   => nil
   |x::xs => size (filter (fun y => y ≤ x) b)))
                :: it-CDF xs b
}
```

Fuzz-like languages - Penn

```
it-CDF (b :[ε*n] data) (buckets : list[n]) (ε:num): nlist
{
  case buckets of
   |nil   => nil
   |x::xs => Lap ε size (filter (fun y => y ≤ x) b)))
                 :: it-CDF xs b
}
```

adding Noise!

Reasoning about DP via probabilistic coupling - BGGHS

For two (sub)-distributions $\mu_1, \mu_2 \in \mathsf{Dist}(A)$ we have an **approximate coupling** $\mu_1 \, \mathcal{C}_{\epsilon,\delta}(R) \, \mu_2$ iff there exists $\mu \in \mathsf{Dist}(A \times A)$ s.t.

- $\mathsf{supp}\mu \subseteq R$
- $\pi_i \mu \leq \mu_i$
- $\max_A(\pi_i\mu - e^\epsilon \mu_i, \mu_i - e^\epsilon \pi_i \mu) \leq \delta$

- Generalize indistinguishability to other relations allowing more general relational reasoning,
- More involved reasoning about probability distances and divergences,
- Preserving the ability to use semi-decision logical and numerical procedures.

pRHL-like languages

CDF example similar to the previous ones

$$b \sim b' \Rightarrow (\underline{\mathsf{itcdf}}\, b\, l\, \epsilon)\ \mathcal{C}_{\epsilon,0}(=)\ (\underline{\mathsf{itcdf}}\, b'\, l\, \epsilon)$$

pRHL-like languages

CDF example similar to the previous ones

$$b \sim b' \Rightarrow (\underline{\mathsf{itcdf}}\, b\, l\, \epsilon)\ \mathcal{C}_{\epsilon,0}(=)\ (\underline{\mathsf{itcdf}}\, b'\, l\, \epsilon)$$

Having two copies of the program allows to compare different parts of the same program.

Why this helps?

It allows to internalize better the properties of Laplace

$$(\underline{\mathsf{Lap}}\,(1/\epsilon)\,v_1)\ \mathcal{C}_{|k+v_1-v_2|\epsilon,0}(x_1+k=x_2)\ (\underline{\mathsf{Lap}}\,(1/\epsilon)\,v_2)$$

can be used to assert symbolically several facts about probabilities.

Why this helps?

$$(\underline{\mathsf{Lap}}\,(1/\epsilon)\,v_1)\ \mathcal{C}_{|v_1 - v_2|\epsilon, 0}(x_1 = x_2)\ (\underline{\mathsf{Lap}}\,(1/\epsilon)\,v_2)$$

expresses

$$\left|\log\left(\frac{\Pr(\underline{\mathsf{Lap}}\,(1/\epsilon)\,v_1 = r)}{\Pr(\underline{\mathsf{Lap}}\,(1/\epsilon)\,v_2 = r)}\right)\right| \leq |v_1 - v_2|\epsilon$$

Why this helps?

$$|v_1 - v_2| \leq k \Rightarrow (\underline{\mathsf{Lap}}\,(1/\epsilon)\,v_1)\ \mathcal{C}_{2k\epsilon,0}(x_1 + k = x_2)\ (\underline{\mathsf{Lap}}\,(1/\epsilon)\,v_2)$$

expresses

$$|v_1 - v_2| \leq k \Rightarrow \left|\log\left(\frac{\Pr(\underline{\mathsf{Lap}}\,(1/\epsilon)\,v_1 = r + k)}{\Pr(\underline{\mathsf{Lap}}\,(1/\epsilon)\,v_2 = r)}\right)\right| \leq 2k\epsilon$$

Why this helps?

$$(\underline{\mathsf{Lap}}\,(1/\epsilon)\,v_1)\ \mathcal{C}_{0,0}(x_1 - x_2 = v_1 - v_2)\ (\underline{\mathsf{Lap}}\,(1/\epsilon)\,v_2)$$

expresses

$$\left|\log\left(\frac{\Pr(\underline{\mathsf{Lap}}\,(1/\epsilon)\,v_2 + k = r + k)}{\Pr(\underline{\mathsf{Lap}}\,(1/\epsilon)\,v_2 = r)}\right)\right| \leq 0$$

Other works

- Bisimulation based methods (Tschantz&al - Xu&al)
- Fuzz with distributed code (Eigner&Maffei)
- Satisfiability modulo counting (Friedrikson&Jha)
- Bayesian Inference (BFGGHS)
- Adaptive Fuzz (Penn)
- Accuracy bounds (BGGHS)
- Continuous models (Sato)
- Lightweight verification - injective function argument (Zhang&Kifer)
- Relational symbolic execution for R - generating DP counterexamples (Chong&Farina&Gaboardi)
- Formalizing the local model (Ebadi&Sands)
- zCDP (BGHS)

Challenges

- All of these tools are research projects and most of them are usable only by experts.

Can we use them to certify correct a library of basic mechanism?

Which non-expert we should aim for?

Other Challenges

- Are there other fundamental principles that we can use?
- How can we extend them to verify accuracy and efficiency?
- There are several works on the verification of randomness, floating points, SMC, etc. Can we combine the different approaches?
- How can we internalize more involved data models assumptions?
- From benchmarks to certification?