Proceedings of the 1st Workshop on Quantitative Analysis of Software (QA'09)

Sumit Gulwani, Ed.
Sanjit A. Seshia, Ed.

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2009-93
http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-93.html

June 15, 2009

Acknowledgement

For each of the three contributed refereed papers included in these proceedings, the copyright to the paper is held by the respective authors.

Proceedings of QA'09:
Workshop on Quantitative Analysis of Software

Co-located with the 21st International Conference on Computer-Aided Verification (CAV), 2009

Editors: Sumit Gulwani (Microsoft Research) and Sanjit A. Seshia (UC Berkeley)

June 15, 2009

This technical report serves as an informal proceedings for the 1st Workshop on Quantitative Analysis of Software (QA'09), held as a satellite workshop of the 2009 Interational Conference on Computer-Aided Verification (CAV), to be held in Grenoble, France.

The workshop program comprises a morning keynote talk, a session of refereed contributed papers, an afternoon session of invited talks and a final session of open discussion on the workshop topic. These proceedings include a brief statement of the goals of the workshop, the workshop program, the abstracts for the invited talks, and full-length refereed papers.

Motivation and Goals of the Workshop

Formal verification of software has mostly been concerned with Boolean properties of code, such as, are assertions satisfied, are all buffer accesses within bounds, does it always terminate, is there any undesirable information flow, etc. However, it is often desirable to ask more quantitative questions about software, such as, what is the expected number of bugs in the software and what is the mean-time between failures (to faciliate decisions about software releases), how much resources (e.g., time, memory, power) does it consume (for performance analysis, and to provide guarantees for embedded, real-time systems), how much information does it leak or how well is it obfuscated (for security related issues).

This workshop aims to explore novel techniques for quantitative analysis of software. It is particularly focussed on code-level analysis rather than analysis purely of models of software or systems. Papers on all techniques have been welcomed, including static, dynamic, and probabilistic analyses. The aim of this workshop is bring together researchers from different areas (programming languages, software engineering, embedded systems, performance analysis, computer security, formal verification, randomized/approximation algorithms, etc.) who are interested in any quantitative aspect of software, thereby providing a platform to investigate if there are common techniques that could be applied to a range of quantitative analyses.

The scope of the workshop, includes, but is not restricted to, the following topics:

- Performance Analysis
- Reliability Evaluation
- Resource Bound Analysis
- Execution Time Analysis

- Quantitative Information Flow
- Probabilistic Analysis
- Software Quality Metrics
- SAT and SMT engines for quantitative analysis, e.g., model counting techniques

Workshop Program

8:45 – 9:00	Welcome and Introduction
9:00 – 10:00	Keynote Talk *Thomas A. Henzinger (EPFL and IST Austria)* From Boolean to Quantitative System Specifications
10:00 – 10:30	Break I
10:30 – 12:00	Contributed Papers
10:30 – 11:00	*Francesco Logozzo (Microsoft Research), Corneliu Popeea (MPI-SWS), and Vincent Laviron (ENS)* Towards a Quantitative Estimation of Abstract Interpretations
11:00 – 11:30	*Jonathan Heusser and Pasquale Malacaria (Queen Mary, Univ. of London)* Quantifying Loop Leakage using a Lattice of Partitions
11:30 – 12:00	*Michael Carl Tschantz (CMU) and Aditya V. Nori (Microsoft Research)* Measuring the Loss of Privacy from Statistics
12:00 – 2:00	Lunch
2:00 – 3:30	Invited Talks
2:00 – 2:30	*Elvira Albert (Complutense University of Madrid)* Upper Bounds on Memory Usage for Garbage-Collected Languages
2:30 – 3:00	*Marta Kwiatkowska (Oxford University)* Software Verification for Ubiquitous Computing
3:00 – 3:30	*Raimund Kirner (TU Vienna)* Timing Analysis of Real-Time Software
3:30 – 4:00	Break II
4:00 – 5:00	Open Discussion

Abstracts of Invited Talks

From Boolean to Quantitative System Specifications
Thomas A. Henzinger (EPFL and IST Austria)

The boolean view holds that given a model and a specification, the specification is either true or false in the model. State transition models have been extended to accommodate certain numerical quantities, such as time stamps and probabilities. However, the boolean view of specifications is still prevalent, for example, in checking whether a timed model satisfies a formula of a real-time logic, or whether a stochastic process satisfies a formula of a probabilistic logic. We advocate the fully quantitative view, where a specification can be satisfied to different degrees, that is, the value of a given specification in a given model is a real number rather than a boolean. We survey some initial results in this direction and present some potential applications, not only in the timed and probabilistic domains, but also for specifying and analyzing the resource use, cost, and reliability of a system.

Upper Bounds on Memory Usage for Garbage-Collected Languages
Elvira Albert (Complutense University of Madrid)

The *peak memory usage* of a program is the maximum size of the *live* data on the memory during the execution of the program, i.e., the minimum amount of memory needed to run the program without exhausting the memory. It is well-known that garbage collection makes the problem of predicting the memory required to run a program difficult. We will describe a *live heap space* analysis for garbage-collected languages which infers accurate upper bounds on the peak heap usage of a program's execution that are not restricted to any complexity class, i.e., we can infer exponential, logarithmic, polynomial, etc., bounds. Our analysis is developed for an (sequential) object-oriented bytecode language with a *scoped-memory* manager that reclaims unreachable memory when methods return. The practicality of our approach is experimentally evaluated on COSTA, a COSt and Termination Analyzer for Java bytecode.

(Joint work with Samir Genaim, Miguel Gómez-Zamalloa, Puri Arenas, Germán Puebla and Damiano Zanardini).

Software verification for ubiquitous computing
Marta Kwiatkowska (Oxford University)

Ubiquitous computing systems are now widespread in e.g. intelligent buildings, environmental monitoring, healthcare monitoring and automotive software. Also called 'everyware' by Adam Greenfield, they continuously interact with the environment through sensors and actuators. In view of the characteristic environmental uncertainty and resource limitations, quantitative techniques are needed to reason about their behaviour. This talk will focus on how software verification technology, specifically model checkers for languages such as C/NesC typically used to program 'everyware' controllers, can be extended to provide automated analysis for pertinent aspects of ubiquitous computing systems: context, communication failure,

resource constraints and performance.

Timing Analysis of Real-Time Software
Raimund Kirner (TU Vienna)

The analysis of the worst-case execution time (WCET) requires detailed knowledge of the program behavior. In practice it is still not possible to obtain all needed information automatically. Within this talk we present the current state of the art of WCET analysis and point to the main challenges to be solved. Further, we show the potential of complementary approaches to static WCET analysis, ranging from richer timing information to adequate methods for soft real-time systems.

Towards a Quantitative Estimation of Abstract Interpretations (Extended abstract)

Francesco Logozzo[1] Corneliu Popeea[2] Vincent Laviron[3]

[1] Microsoft Research, Redmond, WA (USA)
logozzo@microsoft.com
[2] Max Planck Institute for Software Systems, Saarbrücken (Germany)
cpopeea@mpi-sws.mpg.de
[3] École Normale Supérieure, 45, rue d'Ulm, Paris (France)
Vincent.Laviron@ens.fr

Abstract. We aim to extend the notion of distance of sets to partially ordered sets (posets). We discuss several possible definitions, and we propose a relaxed definition of distance between elements of a domain. We apply it in the abstract interpretation theory, and we show in some preliminary examples how it seems well suited to formally quantify the relative loss of precision induced by abstract domains.

1 Introduction

Abstract interpretation is a theory of semantic program approximation based on domain theory. It precisely capture the *qualitative* relative loss of precision induced by static analyses: the more abstract the domain the less the information it captures about program executions. However, the theory does not provide a *quantitative* estimation of the precision loss induces by the abstraction. In this paper we report some preliminary thoughts and results on providing a quantitative evaluation of the errors induced by abstractions.

We are interested in defining metric on the elements of a domain. Roughly, a metric allows to measure the distance between the elements of a given set. When applied to domains (*i.e.* sets whose elements are related by some order), we would like to have a distance $d(\cdot, \cdot)$ which is somehow *compatible* with the underlying order $\sqsubseteq$. For instance, if $x \sqsubseteq y \sqsubseteq z$, then one expects that $d(x, y) \leq d(x, z)$.

The classical definition of distance of measure theory do not work well with domain theory. Intuitively, this is because in a metric space, one wants to compare *any* two elements, and it is hard to define a generic distance which is also aware, *e.g.*, that some elements are not comparable. In this paper we aim at relaxing the classical notion of distance on sets, and to conjugate it with the underlying order on abstract elements. We argue that some natural extensions are not satisfactory for our purposes, and we introduce a pseudo-metric which seems to be satisfactory on some examples.

2 Background: Distance in unordered sets

The minimal requirements that one expect for a distance of two elements x and y of an *unordered* set are: (i) that the distance between x and y is always not negative (negative distances make no sense) and it is equal to zero iff $x = y$; (ii) the distance of x from y is the same as the distance of y from x; and (iii) the distance of x from z is minimal, in that the *indirect* distance from x to some y and that from y to z is always larger than the *direct* distance.

Definition 1 (Distance). *Let S be a set. We say that $d \in [S \times S \longrightarrow \mathbf{R}]$ is a distance for s if it satisfies the following axioms:*

$d(x,y) \geq 0$ *(non-negativity)*
$d(x,y) = 0 \Leftrightarrow x = y$ *(iff-identity)*
$d(x,y) = d(y,x)$ *(symmetry)*
$d(x,z) \leq d(x,y) + d(y,z)$ *(triangle inequality)*

We recall below the definition of the Hausdorff distance which is commonly used to measure the distance of sets by taking the maximum distance of a set to the nearest point in the other set. Hausdorff distance will be useful in the following (Sect. 5.2), as it may provide a way to measure the distance of convex polyhedra.

Lemma 1 (Hausdorff distance). *Let X and Y be two subsets of a metric space that is endowed with a distance d. Then the following defines a distance*

$$d_H(X,Y) = \sup_{x \in X} \{ \inf_{y \in Y} \{ d(x,y) \} \}.$$

We will also use two variations of d_H, the minimum Hausdorff distance d_{Hmin} and the maximum Hausdorff distance d_{Hmax}:

$$d_{Hmin}(X,Y) = \inf_{x \in X} \{ \inf_{y \in Y} \{ d(x,y) \} \} \qquad d_{Hmax}(X,Y) = \sup_{x \in X} \{ \sup_{y \in Y} \{ d(x,y) \} \}$$

3 Distance on a partial order

3.1 Pseudo distance

As briefly discussed in the introduction, in general the classical definition of distance for unordered sets does not take advantage of the partial order between elements of a domain. The goal of this section is to define a pseudo-distance between elements of a domain. Let us consider properties in Def.1 one by one, and consider how we can extend them to match the distance in a domain.

It seems obvious that the distance between two elements of a domain should always be non-negative, so we leave *(non-negativity)*.

Requiring that the distance between two elements is zero if and only if the two elements are the same seems to be a too strong requirement. For instance, in abstract interpretations it is often the case that two distinct abstract elements a_1 and a_2 represent the same concrete object. In particular, useful static analyses relies on the fact that the abstract domain is simply a pre-ordered set whose order relation $\sqsubseteq$ does not enjoy the the anti-symmetric property. Formally, it can be the case that both $a_1 \sqsubseteq a_2$ and $a_2 \sqsubseteq a_1$ hold but $a_1 = a_2$ does not hold. In that case, we'd like to have the freedom to define a distance function such that $d(a_1, a_2) = 0$ even if $a_1 \neq a_2$ but $\gamma(a_1) = \gamma(a_2)$. The axiom *(iff-identity)* does not allow us such a definition. A turn-around is to change the abstract domain, and quotient it with respect to the concretization function. However, such a turn-around goes against our goal which is the definition of a notion of distance on *arbitrary* domains.

A first variation of the *(iff-identity)* would be to relax it, by replacing the equality with the $\sqsubseteq$ operator. The consequent axiom

$d(x,y) = 0 \Leftrightarrow x \sqsubseteq y$ *(iff-identity')*

it is not affected by the problem above. However, the axiom *(iff-identity')* is not useful since it implies that all elements in an ascending chain have distance 0, so that if $x \sqsubseteq y \sqsubseteq z$ then $d(x,y) = d(y,z) = d(x,z) = 0$. Even worse, when applied to the semantics of a program, usually defined as a fixpoint, *(iff-identity')* implies that all the semantics have distance zero from the bottom: $d(\bot, \sqcup_{n<+\infty} f^n(\bot)) = 0$.

Our choice is to simply relax the double implication of *(iff-identity)*, and to require that each element has distance 0 from itself, but allowing a zero-distance also for some distinct elements (Def. 1 with *(if-identity)* is called a pseudo-metric [5]):

$d(x, y) = 0 \Leftarrow x = y$ *(if-identity)*

The rationale behind is that: (i) a distance function gives a quantitative evaluation on how far are two elements in a domain; and (ii) in some cases we want the freedom to say that the distance of two *distinct* elements is negligible (for instance when they represent the same information up to some abstraction).

In an unordered set, the triangle inequality states that the distance between *any* two points is the shortest path between those. It turns out that requiring the triangle property to hold for arbitrary elements of a domain, without considering the order relation is too restrictive. We decided to use a weaker axiom, which requires the triangle inequality to hold *only* for comparable elements:

if $x \sqsubseteq z \sqsubseteq y$ then $d(x, y) \leq d(x, z) + d(z, y)$ *(weak triangle inequality)*

The axiom formalizes the intuition that the distance between elements of a chain should be compatible with the order: If y is not an immediate successor of x, then the path between x and y should not be shorter than any path passing through some element in between x and y.

The next definition sums up what said so far, and it introduces the notion of pseudo-distance compatible with a domain D. When D is clear from the context, we will simply say pseudo-distance.

Definition 2 (Pseudo-distance D-compatible). *Let $\langle D, \sqsubseteq \rangle$ be a domain ordered according to the relation $\sqsubseteq$. Let $\delta \in [D \times D \to \mathbf{R} \cup \{+\infty\}]$. We say that δ is a pseudo-distance D-compatible iff it satisfies the following axioms:*

$\delta(x, y) \geq 0$	*(non-negativity)*
$x = y \Rightarrow \delta(x, y) = 0$	*(if-identity)*
$\delta(x, y) = \delta(y, x)$	*(symmetry)*
$x \sqsubseteq z \sqsubseteq y \Rightarrow \delta(x, z) \leq \delta(x, y) + \delta(y, z)$	*(weak triangle inequality)*

It is also worth noting that, unlike the classical definition, we allow the distance between two elements to be $+\infty$.

3.2 Some simple properties of pseudo distances

The easiest example of a pseudo-distance is the zero function:

Lemma 1 (Zero) *The function $\delta^0(x, y) = 0$ is a pseudo-distance.*

Lemma 2 (Additivity) *Let δ_1, δ_2 be pseudo-distances. Then δ^Σ defined as $\delta^\Sigma(x, y) = \delta_1(x, y) + \delta_2(x, y)$ is pseudo-distance.*

It is worth noting that: (i) $\delta^0(x, y)$ is not a distance in the sense of Def. 1; and (ii) as a consequence of the lemmas above, the set of all pseudo-distances over a domain D form an additive monoid (unlike classical distances).

Lemma 3 (Multiplication by a scalar) *Let δ by a pseudo-distance and $k \in \mathbf{R} \cup \{+\infty\}$. Then δ^* defined as $\delta^*(x, y) = k \cdot \delta(x, y)$ is a pseudo-distance.*

We call an operator $\sqcup$ a *gathering* operator if it satisfies $x \sqsubseteq x \sqcup y$, $y \sqsubseteq x \sqcup y$, and $x \sqsubseteq y \Rightarrow x \sqcup y = y$. A gathering operator is a weaker notion of least upper bound operator, and it is useful in static analyses as *e.g.* [4, 6].

Lemma 4 *Let D a domain endowed with a gathering operator $\sqcup$. If $x \sqsubseteq y$, then $\delta(x \sqcup y, y) = 0$ and $\delta(x \sqcup y, x) = \delta(x, y)$.*

A similar result can be proven for an *intersection* operator $\sqcap$ satisfying $x \sqcap y \sqsubseteq x$, $x \sqcap y \sqsubseteq y$, and $x \sqsubseteq y \Rightarrow x \sqcap y = x$:

Lemma 5 *Let D a domain endowed with an intersection operator $\sqcap$. If $x \sqsubseteq y$, then $\delta(x \sqcap y, y) = \delta(x, y)$ and $\delta(x \sqcap y, x) = 0$.*

4 Some pseudo-distances

4.1 Structure-based

A first thought for defining a more interesting pseudo-distance on a domain is to count the number of intermediate elements between two elements:

Definition 1 (Path length (plen)) *The path length for two elements $x \sqsubseteq y$ of a domain D is*

$$\mathrm{plen}(x, y) = \min\{n \mid \{x_0, x_1, \ldots x_n\} \in \wp(D), x_0 = x, x_n = y, \forall 0 \leq i < n . x_i \sqsubseteq x_{i+1}\}.$$

If x and y are not comparable, we let $\mathrm{plen}(x, y) = +\infty$.

The function plen is not a pseudo-distance as it does not satisfy *(symmetry)*. Fixing it requires little work:

Lemma 6 (δ_{plen}) *The function $\delta_{\mathrm{plen}} \in [D \times D \to \mathbf{R} \cup \{+\infty\}]$ defined as*

$$\delta_{\mathrm{plen}}(x, y) = x \sqsubseteq y ? \mathrm{plen}(x, y) : (y \sqsubseteq x ? \mathrm{plen}(x, y) : +\infty)$$

is a pseudo-distance.

The pseudo-distance δ_{plen} is not very interesting, as it relates only elements that belong to the *same* chain. One can think to refine it by taking the average distance between two elements and their least upper bound (or the result of the gathering if the least upper bound is not defined):

Lemma 7 ($\delta^{\sqcup}_{\mathrm{plen}}$) *The function $\delta^{\sqcup}_{\mathrm{plen}} \in [D \times D \to \mathbf{R} \cup \{+\infty\}]$ defined as*

$$\delta^{\sqcup}_{\mathrm{plen}}(x, y) = 1/2 \cdot (\delta_{\mathrm{plen}}(x, x \sqcup y) + \delta_{\mathrm{plen}}(y, x \sqcup y))$$

is a pseudo-distance.

It is immediate to check that if x and y are comparable, then $\delta^{\sqcup}_{\mathrm{plen}}(x, y) = \delta_{\mathrm{plen}}(x, y)$.

Example 1. Let x_0, y_0, x_1, y_1 be elements of a domain such that $\delta_{\mathrm{plen}}(x_0, x_0 \sqcup y_0) = \delta_{\mathrm{plen}}(y_0, x_0 \sqcup y_0) = 50$, $\delta_{\mathrm{plen}}(x_1, x_1 \sqcup y_1) = 1$ and $\delta_{\mathrm{plen}}(y_1, x_1 \sqcup y_1) = 99$. Then, $\delta^{\sqcup}_{\mathrm{plen}}(x_0, y_0) = \delta^{\sqcup}_{\mathrm{plen}}(x_1, y_1) = 50$. □

In the example above, from the view of an abstract interpretation one would have expected that $\delta(x_0, y_0) \neq \delta(x_1, y_1)$, because when approximating x_1 and y_1 with $x_1 \sqcup y_1$ one *may* perform a small or a large error, but when approximating x_0 and y_0 with $x_0 \sqcup y_0$ one *always* performs a medium error. A way to overcome this drawback is to consider the minimal distance to the gathering (instead of the average distance):

Lemma 8 ($\delta^{\sqcup,m}_{\mathrm{plen}}$) *The function $\delta^{\sqcup,m}_{\mathrm{plen}} \in [D \times D \to \mathbf{R} \cup \{+\infty\}]$ defined as*

$$\delta^{\sqcup,m}_{\mathrm{plen}}(x, y) = \min(\delta_{\mathrm{plen}}(x, x \sqcup y), \delta_{\mathrm{plen}}(y, x \sqcup y))$$

is a pseudo-distance.

The drawback of distances based on path length is that they do not work well for infinite height lattices.

4.2 Affinity

An alternative pseudo-distance considers the *affinity* between abstract elements. The affinity distance was originally used for bounded powerset construction of Polyhedra [11]. The intuition behind is to have an percentage estimation of the simple constraints that are preserved at the gathering. Before stating it formally, we need some auxiliary definitions. Let us assume that seq is a function which given an abstract element x returns a minimal and *finite* set of terms equivalents to x. For instance, in the domain of convex Polyhedra $seq(\{-u \leq 0, u+v \leq 2, 2 \cdot u + 2 \cdot v \leq 4\}) = \{-u \leq 0, u+v \leq 2\}$. The function seq can be defined for most abstract domains, but not for all (think of $seq(\{u = f(u)\})$, as it may appear in a domain for abstract unification). We let $|\cdot|$ denote set cardinality, *eg*, $|\{-u \leq 0, u+v \leq 2\}| = 2$.

Lemma 9 (Affinity pseudo-distance) *The affinity distance of two elements x and y wrt the operator $\sqcup$ gives rise to the affinity distance defined as:*

$$\delta^{\sqcup}_{aff}(x,y) = 1 - \frac{|seq(x \sqcup y)|}{|seq(x) \cup seq(y)|}$$

Proof. The proof that the affinity satisfies the *(weak triangle inequality)* property is quite tricky. We report it in the appendix. □

It is immediate to observe that when $x \sqsubseteq y$, then $\delta^{\sqcup}_{aff}(x,y) = 1 - \frac{|seq(y)|}{|seq(x) \cup seq(y)|}$. One possibility to lift the affinity distance from a base to a powerset domain is simply to ignore the powerset structure of the domain. However, we did not yet proved it to satisfy Def. 2, so we leave it as an hypothesis:

Hypothesis 1 (Affinity for powerset domain) *Given $X = \{x_1, \ldots x_n\}$ and $Y = \{y_1, \ldots y_m\}$ elements of a powerset domain their affinity distance is:*

$$\delta^{\sqcup}_{aff}(X,Y) = 1 - \frac{|seq(X \sqcup Y)|}{|(\cup_i seq(x_i)) \cup (\cup_j seq(y_j))|}$$

4.3 Examples

The notion of pseudo-distance on a domain is useful to *quantify* the relative precision of different inferred invariants.

4.4 Nullness Domains

Let us consider the code in Fig. 1 to be analyzed with four different abstract domains: NN (nullness), TNN (type+nullness), DNN (disjunctive nullness) and DTNN (disjunctive type+nullness).It is easy to prove that: NN is the less precise domain, DTNN is the most precise, and TNN and DNN stand between the two but they are not comparable. The notion of distance of abstract elements allow us to give a *quantitative* comparison of the result of TNN and DNN.

Let us consider the method *m* written in C#-like syntax (the expression `a as B` casts `a` to `B` if `a` is a subtype of `B`, otherwise it returns `null`). The abstract states at the exit point of *m* using different abstract domains are in Fig. 2. The domain TNN is in general more precise than NN, but in the example it does not provide a more precise abstract state. The domain DNN is not comparable with TNN, and in the example it infers a more precise abstract state. Using the affinity distance we can give a quantitative formal

characterization of that (with an abuse of notation we confuse the abstract element with the name of abstract domain used):

$$\begin{aligned}
\delta^{\sqcup}_{aff}(\text{NN}, \text{TNN}) &= 1 - \tfrac{1}{1} = 0 \\
\delta^{\sqcup}_{aff}(\text{NN}, \text{DNN}) &= 1 - \tfrac{1}{3} = \tfrac{2}{3} \\
\delta^{\sqcup}_{aff}(\text{DNN}, \text{DTNN}) &= 1 - \tfrac{3}{7} = \tfrac{4}{7} \\
\delta^{\sqcup}_{aff}(\text{NN}, \text{DTNN}) &= 1 - \tfrac{1}{7} = \tfrac{6}{7}
\end{aligned}$$

The distance between NN and TNN is zero, meaning that no gain of information is obtained in the example when refining the NN just with types. The refinement with explicit disjunction (also known as trace partitioning [8]) produces an improvement of the 66%. The refinement of DNN with types improves the result by a further 57%. Overall using disjunction and types one obtains a result which is 85% more precise than the abstract domain of NN alone.

```
m(A a, out A x) {
  requires a != null;
  B b = a as B;
  if (b != null)
    x = new B(b);
  else
    x = null;
}
```

Fig. 1. An example for nullness analysis

Abstract Domain	Result
NN :	$\langle \mathtt{a} \to \mathcal{NN}, \mathtt{x} \to \top \rangle$
DNN :	$\langle \mathtt{a} \to \mathcal{NN}, \mathtt{x} \to \mathcal{NN} \rangle \vee \langle \mathtt{a} \to \mathcal{NN}, \mathtt{x} \to \mathcal{N} \rangle$
TNN :	$\langle \mathtt{a} \to \mathcal{NN}, \mathtt{x} \to \top \rangle$
DTNN :	$\langle \langle \mathtt{a} \to \mathcal{NN}, \mathtt{x} \to \mathcal{NN} \rangle, typeof(\mathtt{a}) <: \mathtt{B}, typeof(\mathtt{x}) == \mathtt{B} \rangle \vee$ $\langle \langle \mathtt{a} \to \mathcal{NN}, \mathtt{x} \to \mathcal{N} \rangle, typeof(\mathtt{a}) == \mathtt{A}, typeof(\mathtt{x}) == \mathtt{A} \rangle$

Fig. 2. The different results of the analysis using four different abstract domains. $\mathcal{NN}$ denotes that the reference is not null, $\mathcal{N}$ that it is definitely null.

4.5 McCarthy function

```
int MC(int n) {
  int t1, t2, r;
  if (n>100)
     r = n-10;
  else {
     t1 = n+11;
     t2 = MC(t1);
     r  = MC(t2);}
  return r;
}
```

Fig. 3. The McCarthy function

Abstract Domain	Result
Sign :	$\{0 < \mathtt{r}\}$
Intervals :	$\{91 \leq \mathtt{r}\}$
Octagons :	$\{91 \leq \mathtt{r}, \mathtt{n} - 10 \leq \mathtt{r}\}$

Fig. 4. The different inferred postconditions for the McCarthy function using different numerical abstract domains.

Pseudo-distances apply also when the underlying abstract domain has infinite height. Let us consider the McCarthy function (recalled in Fig. 3). We can analyze it with three different abstract domains: Signs, Intervals and Octagons. The inferred invariants are summarized in Fig. 4.5. The distances between those are given below.

$$\begin{aligned}
\delta^{\sqcup}_{aff}(\text{Signs}, \text{Intervals}) &= 1 - \tfrac{1}{2} = \tfrac{1}{2} \\
\delta^{\sqcup}_{aff}(\text{Intervals}, \text{Octagons}) &= 1 - \tfrac{1}{2} = \tfrac{1}{2} \\
\delta^{\sqcup}_{aff}(\text{Signs}, \text{Octagons}) &= 1 - \tfrac{1}{3} = \tfrac{2}{3}
\end{aligned}$$

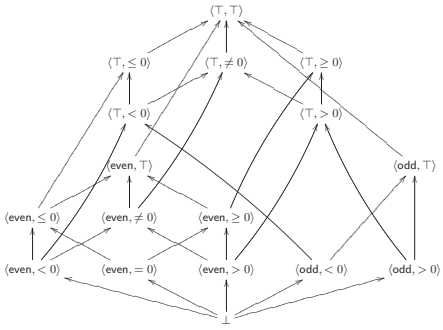


Fig. 5. The lattice Parity $\otimes$ Signs

Using Octagons, one obtains a postcondition which is 66% more precise than using Signs.

We can also consider to lift Octagons to disjunction of Octagons (DOctagons). In this case the inferred invariant is [10]:

$$\{101 \leq \mathtt{n}, \mathtt{n} - 10 \geq \mathtt{r}, \mathtt{n} - 10 \leq \mathtt{r}\} \vee \{\mathtt{n} < 101, \mathtt{r} \geq 91, \mathtt{r} \leq 91\}$$

and one can prove that the $\delta^{\sqcup}_{\mathit{aff}}(\text{Octagons}, \text{DOctagons}) = 1 - 2/6 = 2/3$.

5 Towards measuring the precision of Abstract Interpretations

The notion of distance is useful to formally quantify the error induced by using abstract elements. Given a concrete domain D and an abstract domains A, in the following we suppose that: (i) D and A are complete lattices; and (ii) they are related by a Galois connection $\langle \alpha, \gamma \rangle$.

5.1 Measuring abstract elements and domains

Definition 2 (δ-Error) *We define the error of approximating a concrete element c in A according to pseudo-distance δ as $\epsilon_\delta(c) = \delta(c, \gamma(\alpha(c))$.*

It is immediate to observe that as in a Galois connection $c \sqsubseteq \gamma(\alpha(c))$, then $\epsilon_{\delta^{\sqcup,m}_{\text{plen}}}$ boils down to $\epsilon_{\delta_{\text{plen}}}$.

Example 2. Let us consider the concrete domain to be the reduced product of Parity and Signs (Fig. 5) and the abstract domain to be Signs. Then $\gamma(\alpha(\langle \mathsf{even}, > 0\rangle)) = \gamma(> 0) = \langle \top, > 0\rangle$ implies that $\epsilon_{\delta_{\text{plen}}}(c) = \delta_{\text{plen}}(\langle \mathsf{even}, > 0\rangle, \langle \top, > 0\rangle) = 1$. If, on the other hand we chose Parity as abstract domain, then $\gamma(\alpha(\langle \mathsf{even}, > 0\rangle)) = \gamma(\mathsf{even}) = \langle \mathsf{even}, \top\rangle$ which implies that $\epsilon(c)_{\delta_{\text{plen}}} = \delta_{\text{plen}}(\langle \mathsf{even}, > 0\rangle, \langle \mathsf{even}, \top\rangle) = 2$, as $\langle \mathsf{even}, > 0\rangle \sqsubseteq \langle \mathsf{even}, \geq 0\rangle \sqsubseteq \langle \mathsf{even}, \top\rangle$ in Parity $\otimes$ Signs. $\square$

The example above suggests that one may be able to lift the Def. 2 to abstract domains, so that one can measure the loss of information induced by using an abstract domain.

Definition 3 (Error) *Let C be an abstract domain with a* finite *set of elements, and let A denote the abstraction of C. Then the average error of using A for C according to pseudo-distance δ is*

$$\epsilon_A = \frac{1}{|C|} \cdot \sum_{c \in C} \epsilon_\delta(c).$$

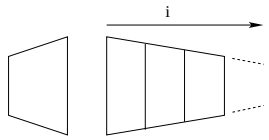


Fig. 6. The element x and the sequence of elements $\{y_1, .., y_i, ..\}$: $d_{Hmax}(x, y_i)$ grows arbitrary large with i, while $\epsilon_{\sqcup}(x, y_i)$ remains constant for all i.

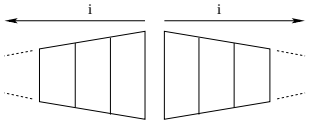


Fig. 7. Two sequences of elements $\{x_1, .., x_i, ..\}$ and $\{y_1, .., y_i, ..\}$: $d_{Hmin}(x_i, y_i)$ remains constant, while $\epsilon_{\sqcup}(x_i, y_i)$ grows arbitrary large with i.

Example 3. Let us consider two abstractions of Parity $\otimes$ Signs: Parity and SimpleSigns. Parity has four elements: $\bot$, $\top$, even and odd. The average error of using Parity as abstract domain is $18/17$. SimpleSigns has five elements: $\bot$, $\top$, < 0, > 0 and $\neq 0$. The average error of using SimpleSigns is $15/17$. □

As a consequence of the example, it turns out that even if from the point of the relative precision Parity and SimpleSigns are not comparable, on average, one may expect to have a smaller error when it uses SimpleSigns. It is worth noting that the quantitative errors we obtained are more relevant than those obtained using simple cardinality arguments. In fact $|5/17 - 4/17| = 1/17 < |18/17 - 15/17| = 3/17$.

5.2 Measuring operators

It is known that performing operations in the abstract may introduce a loss of precision. We can lift the previous results to formally evaluate the error induced by an abstract operator.

To estimate the error induced by the use of an abstract operator, we consider the average of the errors induced by applying the operator to each pair of abstract elements:

Definition 4 (op-Error) *Let op be the abstract counterpart for a concrete operator op_c. Then the average error of op with respect to δ is:*

$$\epsilon_{op} = \frac{1}{|A|^2} \sum_{a_1, a_2 \in A} \delta(\gamma(a_1 \ op \ a_2), \gamma(a_1) \ op_c \ \gamma(a_2)).$$

When $\epsilon_{op} = 0$ we say that op is δ-complete. Intuitively, a δ-complete operator does not introduce any error wrt the distance δ. A complete operator is one such that $\forall a_1, a_2.\ \gamma(a_1 \ op \ a_2) = \gamma(a_1) \ op_c \ \gamma(a_2)$. An immediate consequence of Def. 4 is that if op is a complete operator, then op is δ-complete for each δ.

The next logical step is to apply the definitions of this section to the most critical operator in a static analysis, that is the join. Our first approach was to use the Hausdorff distance, but it did not worked as one can have (i) $d_{Hmax}(x, y)$ arbitrary large, when err is constant (Fig. 6); or (ii) $d_{Hmin}(x, y)$ can be constant, when err can be arbitrary large (Fig. 7). Finding a good distance to evaluate the precision loss induced by the join that works for infinite abstract domains is still an open question for us.

6 Related Work

van Breugel [12] exploits the structure of a metric space to define the operational and the denotational semantics of a while language and he uses it to relate the two semantics, and to prove the existence of the fixpoints. His approach is a way different dual to ours, as we start from the domain structure, and we build a distance on the top of it.

Di Pierro and Wiklicky [2] propose a notion of probabilistic abstract interpretation, and they uses it to measure the measure the incompleteness of the abstract domain. With respect to our work, they change the

underlying framework (from standard abstract interpretation to linear spaces). An interesting future direction is to deepen the relation between our approach and theirs.

Distances and metric spaces have been object of wide investigation in other fields of computer science as machine learning or computer graphics. De Raedt and Ramon [1] propose to derive a distance from a partial order. They assume the existence of a weight function for the elements of the partial order, which is not clear how it works in the abstract interpretation setting, where abstract elements may approximate infinite elements. For instance we can use the affinity distance to also measure the distance between open convex polyhedra. Markov and Marinchev [7] define a semi-distance for Horn clauses. Eiter and Mannila [3] propose several distance measures for finite sets of points. Our affinity distance works also when the sets are infinite.

Monniaux [9] applies abstract interpretation-based techniques to bound the worst-case probability for some properties of interest. The affinity distance was originally used in [11] (where it was named "planar affinity measure") to construct a powerset extension of the polyhedron base domain. In general, such a powerset extension can be either expensive (the number of elements is exponential when compared to the base domain) or imprecise (when the number of disjuncts is syntactically bounded). In this context, the affinity distance was used to identify pairs of elements that are likely to be joined (using the least upper bound operator) with a small precision loss.

7 Conclusions

We presented the preliminary results on our investigations to *quantify* the loss of precision in static analyses. We show how the classical notion of distance on metric spaces is too strict, and we proposed a weaker notion, the pseudo-distance. We defined some pseudo-distances and we apply them to measure the relative precision of invariants inferred with (possibly non-comparable) abstract domains. We lifted the notion of pseudo-distance to the elements of the abstract domains (so to estimate the relative precision loss) and to operators on abstract domain. There are still some open issues, both technical and conceptual. For instance it is not clear if the affinity distance lifted to powerset is a pseudo-metric and we aim at extending the distance on abstract domain to cope with *infinite* abstract domains, which are often of more interest for static analyses.

References

1. L. De Raedt and J. Ramon. Deriving distance metrics from generality relations. *Pattern Recognition Letters*, 30(3):187–191, 2009.
2. A. Di Pierro and H. Wiklicky. Measuring the precision of abstract interpretations. In *LOPSTR (LNCS 2042: Selected Papers)*, pages 147–164, 2000.
3. T. Eiter and H. Mannila. Distance measures for point sets and their computation. *Acta Inf.*, 34(2):109–133, 1997.
4. J. Feret. The arithmetic-geometric progression abstract domain. In *VMCAI*, pages 42–58, 2005.
5. P. Giannopoulos and R. C. Veltkamp. A pseudo-metric for weighted point sets. In *Proceedings of the European Conference on Computer Vision*, 2002.
6. V. Laviron and F. Logozzo. Subpolyhedra: A (more) scalable approach to infer linear inequalities. In *VMCAI*, pages 229–244, 2009.
7. Z. Markov and I. Marinchev. Coverage-based semi-distance between horn clauses. In *AIMSA*, pages 331–339, 2000.
8. L. Mauborgne and X. Rival. Trace Partitioning in Abstract Interpretation Based Static Analyzers. In *ESOP*, 2005.
9. D. Monniaux. Abstract interpretation of programs as markov decision processes. In *SAS*, pages 237–254, 2003.
10. C. Popeea. *Disjunctive Invariants for Modular Static Analysis*. PhD thesis, School of Computing, National University of Singapore, 2008.
11. C. Popeea and W.N. Chin. Inferring disjunctive postconditions. In *ASIAN CS Conference*, 2006.
12. F. van Breugel. An introduction to metric semantics: operational and denotational models for programming and specification languages. *Theor. Comput. Sci.*, 258(1-2):1–98, 2001.

A Proof of Lemma 9

Proof: It is trivial to show that $\delta^{\sqcup}_{aff}$ satisfies the *non-negativity*, *identity* and *symmetry* properties. We then need to prove that $\delta^{\sqcup}_{aff}$ satisfies also the *weak triangle inequality*: $\delta^{\sqcup}_{aff}(x,y) \leq \delta^{\sqcup}_{aff}(x,z) + \delta^{\sqcup}_{aff}(z,y)$. Using the hypothesis $x \sqsubseteq z \sqsubseteq y$, the inequality reduces to:

$$1 - \frac{|seq(y)|}{|seq(x) \cup seq(y)|} \leq 1 - \frac{|seq(z)|}{|seq(x) \cup seq(z)|} + 1 - \frac{|seq(y)|}{|seq(y) \cup seq(z)|}$$

Subsequently:

$$\frac{|seq(x) \cup seq(y)| - |seq(y)|}{|seq(x) \cup seq(y)|} \leq \frac{|seq(x) \cup seq(z)| - |seq(z)|}{|seq(x) \cup seq(z)|} + \frac{|seq(y) \cup seq(z)| - |seq(y)|}{|seq(y) \cup seq(z)|}$$

We use P_0 to P_7 to represent cardinalities of sets where the subscript indicates the membership of edges to x,y,z:

	z	y	x	
P_1	0	0	1	$\|seq(x) \backslash seq(y) \backslash seq(z)\|$
P_2	0	1	0	$\|seq(y) \backslash seq(x) \backslash seq(z)\|$
P_3	0	1	1	$\|seq(x) \cap seq(y) \backslash seq(z)\|$
P_4	1	0	0	$\|seq(z) \backslash seq(x) \backslash seq(y)\|$
p_5	1	0	1	$\|seq(x) \cap seq(z) \backslash seq(y)\|$
P_6	1	1	0	$\|seq(y) \cap seq(z) \backslash seq(x)\|$
P_7	1	1	1	$\|seq(x) \cap seq(y) \cap seq(z)\|$

From $x \sqsubseteq z \sqsubseteq y$, we obtain that $seq(x) \cap seq(y) \subseteq seq(z)$. Subsequently, we have that $P_3 = 0$. We use the notation $N = \Sigma P_i$. The inequality to prove can then be simplified to:

$$\frac{P_1 + P_5}{N - P_4} \leq \frac{P_1}{N - P_2} + \frac{P_4 + P_5}{N - P_1}$$

This inequality can be proven as follows:

$$(P_1 + P_5)(N - P_2)(N - P_1) \leq P_1(N - P_4)(N - P_1) + (P_4 + P_5)(N - P_4)(N - P_2)$$

$$N^2P_1 + N^2P_5 + P_1^2P_2 + P_1P_2P_5 + NP_1^2 + NP_1P_4 + NP_2P_4 + NP_2P_5 + NP_4^2 + NP_4P_5 \leq$$
$$NP_1^2 + NP_1P_2 + NP_1P_5 + NP_2P_5 + N^2P_1 + P_1^2P_4 + N^2P_4 + N^2P_5 + P_2P_4^2 + P_2P_4P_5$$

$$P_1^2P_2 + P_1P_2P_5 + NP_1P_4 + NP_2P_4 + NP_4^2 + NP_4P_5 \leq$$
$$NP_1P_2 + NP_1P_5 + P_1^2P_4 + N^2P_4 + P_2P_4^2 + P_2P_4P_5$$

Since $NP_1P_4 + NP_2P_4 + NP_4^2 + NP_4P_5 \leq N^2P_4$, the inequality reduces to:
$P_1^2P_2 + P_1P_2P_5 \leq NP_1P_2 + NP_1P_5 + P_1^2P_4 + P_2P_4^2 + P_2P_4P_5$

Since $P_1^2P_2 \leq NP_1P_2$ and $P_1P_2P_5 \leq NP_1P_5$, the inequality is proven and thus$\delta^{\sqcup}_{aff}$ satisfies the weak triangle inequality. □

Quantifying Loop Leakage using a Lattice of Partitions

Jonathan Heusser Pasquale Malacaria

Department of Computer Science
Queen Mary, University of London
{jonathan,pm}@dcs.qmul.ac.uk

Abstract. We investigate the relationship between Landauer and Redmond Lattice of Information (LoI) and recent work in quantitative information flow. Elements of LoI can be seen as partitions where their blocks are observably indistinguishable states. We show how leakage of looping programs can be naturally described using a sequence of partitions that is a chain in this lattice. This interpretation is significant in light of recent work aimed at automatically quantify leakage of programs by computing their interpretation in LoI.
We also show that the measure (in a lattice theoretical sense) of this chain is Shannon's entropy and that it coincides with previous information theoretical formulas characterizing leakage of loops.

1 Introduction

Landauer and Redmond [13] showed how information over a system can be represented as a lattice. The idea is that a particular view (or observation) over the system can be associated with a partition whose blocks are the set of states that that view or observation cannot distinguish.

Observations can be ordered by refinement: $x \leq y$ if and only if y makes more distinctions than x. It turns out that this structure is a complete lattice: the Lattice of Information.

The interaction between equivalence relations, partitions and quantitative information flow has been shown in [6], the idea being that the basic bricks of a quantitative analysis, i.e. random variables, seen as maps from states to real numbers, can be identified with their kernel, i.e. a partition of states.

In this work, we develop these ideas further. In particular we show how the most challenging quantitative analysis of commands, i.e. loops, can be simplified and conceptually clarified by the use of algebraic tools. Leakage of looping programs turns out to be describable by chains in this lattice. The $n-$th element in the chain corresponds to the leakage of the program up to the $n-$th iteration. This correspondence between iterations and partitions may have relevance for implementation, in light of recent work by Backes, Köpf and Rybalchenko [1] who use model-checking techniques like counter-example guided refinement to compute the interpretation of programs as partitions in the Lattice of Information.

For example, our approach makes it simple and clear what it means to refine a loop up the $n-$th iteration and how to provide a safe bound for the remaining iterations.

Hence, we argue that this algebraic interpretation improves order, elegance and abstraction of quantitative reasoning techniques of looping programs.

1.1 Related Work

On the information flow side, Landauer and Redmond's paper describes the lattice of information [13] which builds the basis of our paper and proved to be a good model for information flow in a series of work in related fields [5, 8, 23]. Yatsuka Nakamura described already in 1970 the lattice-theoretic basis of the information theoretical notion of entropy [20].

Quantitative Information Flow can be traced back to the work by Denning [10], Millen [19] and McLean [18]. An early attempt to combine quantitative ideas in the framework of LoI was by Weber [23] where a

partition blocks counting technique is used in the context of state machines. There have been a number of recent works relevant to the present paper e.g. [5, 6, 14, 8, 1].

2 Lattice of Information

Landauer and Redmond [13] showed how information can be represented as a lattice. Let Σ be the set of all states in a system. They described two ways as how elements of Σ can be seen as a lattice. First, as the set of equivalence relations on the set Σ where the equivalence classes express the set of states whose information is indistinguishable. The second way is where Σ is the domain of some function f to some other set X, where X describes the information extracted for some state.

Let us define the set $\mathcal{I}(\Sigma)$ which stands for the set of all possible equivalence relations on the set Σ. The ordering of $\mathcal{I}(\Sigma)$ is now defined as

$$\approx \sqsubseteq \sim \leftrightarrow \ \forall \sigma_1, \sigma_2 \ (\sigma_1 \sim \sigma_2 \Rightarrow \sigma_1 \approx \sigma_2) \tag{1}$$

where $\approx, \sim \in \mathcal{I}$ and $\sigma_1, \sigma_2 \in \Sigma$. Furthermore, the join $\sqcup$ and meet $\sqcap$ lattice operations stand for the intersection of relations and the transitive closure union of relations respectively. Thus, higher elements in the lattice can distinguish more while lower elements in the lattice can distinguish less states. It easily follows from (1) that $\mathcal{I}(\Sigma)$ is a complete lattice.

In this paper we will assume this lattice to be finite; this is motivated by considering information storable in programs variables: such information is $\leq 2^k$ where k is the number of bits of the secret variable.

We give a typical example of how these equivalence relations can be used in an information flow setting. Let us assume the set of states Σ consists of a tuple $\langle l, h \rangle$ where l is a low variable and h is a confidential variable. One possible observer can be described by the equivalence relation

$$\langle l_1, h_1 \rangle \approx \langle l_2, h_2 \rangle \leftrightarrow l_1 = l_2$$

That is the observer can only distinguish two states whenever they agree on the low variable part. Clearly, a more powerful attacker is the one who can distinguish any two states from one another, or

$$\langle l_1, h_1 \rangle \sim \langle l_2, h_2 \rangle \leftrightarrow l_1 = l_2 \wedge h_1 = h_2$$

The $\sim$-observer gains more information than the $\approx$-observer by comparing states, therefore $\approx \sqsubseteq \sim$.

Finally a useful equivalent description of the Lattice of Information is as the lattice of partitions of set of states. This is justified by considering the trivial bijection between partitions and equivalence relations.

2.1 Lattice of Random Variables

A random variable can be seen as map $X : D \to \mathcal{R}(X)$, where D is a finite set with a probability distribution and $\mathcal{R}(X)$, a measurable set, is the range of X. For each element $d \in D$, the probability of it is denoted $p(d)$. For every element $x \in \mathcal{R}(X)$ we write $p(x)$ to mean the probability that X takes on the value x, i.e. $p(x) \stackrel{def}{=} \sum_{d \in X^{-1}(x)} p(d)$. In other words, what we observe by $X = x$ is that the input to X in D belongs to the set $X^{-1}(x)$. From that perspective, X partitions the space D into sets which are indistinguishable to an observer who sees the value that X takes on. This can be stated relationally by taking the kernel of X which defines the following equivalence relation $\ker(X)$:

$$d \ \ker(X) \ d' \text{ iff } X(d) = X(d') \tag{2}$$

The entropy of a random variable X is denoted $\mathcal{H}(X)$, defined as follows

$$H(X) = -\sum_x p(x) \log p(x)$$

As seen from the definition of $p(x)$, the entropy of X only depends on its set of inverse images $X^{-1}(x)$. Thus, if two random variables X and Y have the same inverse images they will necessarily have the same entropy. More formally, we write $X \simeq Y$ whenever the following holds

$$X \simeq Y \text{ iff } \{X^{-1}(x) : x \in \mathcal{R}(X)\} = \{Y^{-1}(y) : y \in \mathcal{R}(Y)\}$$

and thus if $X \simeq Y$ then $\mathcal{H}(X) = \mathcal{H}(Y)$.

This shows that each element of the lattice $\mathcal{I}(\Sigma)$ can be seen as a random variable. We can hence identify LoI with a lattice of random variables ordered by (1).

Our lattice has a top element $\approx_\top$ which is the identity relation, distinguishing all states from one another. If the lattice is built from two random variables X and Y then top will be the joint random variable (X, Y). The bottom element $\approx_\bot$ relates every state to every other and represents the least information of the system.

The join and meet operations on this lattice are the same as for the lattice of information: the join $\sqcup$ is the intersection of relations, making the equivalence classes finer; the meet $\sqcap$ conversely is the transitive closure of the union of two relations. Notice that the $\sqcup$ of two random variables is the classic notion of *joint* random variable, i.e. $X \sqcup Y = (X, Y)$.

Notice that in general, $\mathcal{I}(\Sigma)$ is not distributive.

2.2 Join Semivaluation

A join semivaluation [2] on $\mathcal{I}(\Sigma)$ is a real valued map $\nu : \mathcal{I}(\Sigma) \to \mathbb{R}$, that satisfies the following properties:

$$\nu(X \sqcap Y) + \nu(X \sqcup Y) \leq \nu(X) + \nu(Y) \tag{3}$$

$$X \sqsubseteq Y \text{ implies } \nu(X) \leq \nu(Y) \tag{4}$$

for every element X and Y in a lattice $\mathcal{I}(\Sigma)$ [20]. The property (4) is order-preserving: a higher element in the lattice has a larger valuation than elements below itself. The first property (3) is a weakened inclusion-exclusion principle.

Proposition 1. *The map*

$$\nu_\sqcup(X \sqcup Y) = H(X, Y) \tag{5}$$

is a join semivaluation.

3 Information Leakage - Algebraic View

3.1 Leakage of Deterministic Programs

Programs are naturally interpreted as equivalence relations on states [14] and hence as random variables in the Lattice of Information. The random variable associated to a program P is the equivalence relation on the states defined by:

$$\sigma \simeq \sigma' \iff P(\sigma) =_{\texttt{obs}} P(\sigma')$$

in this paper we will consider $=_{\texttt{obs}}$ to be the relation "to have the same output". However, a more general view of observational equivalences in this context has been presented in [15].

Example. A simple example demonstrates these relationships

```
if h=0 then access else deny
```

We associate a random variable for the confidential program variable h, ranging over $\{0, \ldots, 3\}$. The output random variable O associated to the program contains the two outcomes $\{\text{access}, \text{deny}\}$, mapping all inputs to these two outcome classes. The equivalence relation (i.e. partition) associated to the above program is hence

$$O = \{\underbrace{\{0\}}_{\text{access}} \underbrace{\{1,2,3\}}_{\text{deny}}\}$$

The outcome *access* is observed for O whenever h is 0; all other values of h are captured by *deny*.

Compare the above program with the following one

```
if h=0 then access
else { if h=1 then maybe else deny }
```

This program releases more information about the secret than the previous one because of the additional output `maybe`. Formally this is reflected by its associated random variable

$$O' = \{\underbrace{\{0\}}_{\text{access}} \underbrace{\{1\}}_{\text{maybe}} \underbrace{\{2,3\}}_{\text{deny}}\}$$

clearly, it is the case that $O \sqsubseteq O'$.

Based on this interpretation of programs Clark, Hunt and Malacaria [5] defined the *leakage* of a program O depending on a secret input h and public input l by

$$\iota(O; h|l) = I(O; h|l)$$

i.e the conditional mutual information between the program and the secret given the low input.

If the program is deterministic, i.e. $O = f(h, l)$ then we have

$$\begin{aligned}\iota(f(h,l), h|l) &= \nu(f(h,l)|l) - \nu(f(h,l)|h,l) \\ &= H(O|l) - 0 = H(O|l)\end{aligned}$$

For normal programs, i.e. deterministic programs depending only on the high input, $H(O|l) = H(O)$ reduces the leakage to the semivaluation of its output random variable.

We can relate order in LoI and amount of leakage by the following result

Proposition 2. *Let P, P' be two normal programs. Then $P \sqsubseteq P'$ in LoI iff for all probability distributions on states in LoI, $H(P) \leq H(P')$.*

Proof. The $(\Rightarrow)$ direction follows from the definition of semivaluation. For the other direction suppose $P \not\sqsubseteq P'$; this means we can find a block in P' that is not a subset of any block in P, for example we have $\{\{a\}, \{b, c\}\}$ in P and $\{\{a, b, c\}\}$ in P'. Choose then a distribution 0 everywhere apart from a, b, c: for such a distribution we then have $H(P) > H(P')$

3.2 Loops

The idea is to interpret looping programs in the lattice of information as least upper bounds of increasing sequences; for some loops (those with collisions) this is not immediately true: we will show however that all loops can be interpreted as the meet of the l.u.b. of an increasing sequence and a point in the lattice representing the collisions.

We also show that the semivaluation of this interpretation of loops is the information theoretical formula presented in [14]. The main result of the cited work is summarised in the next paragraph, to support understanding.

Leakage of Loops Let us consider a loop of the form `while e M`, where `e` is representing the Boolean guard and `M` is making up the random variables associated with the commands in the body of the loop. Such a loop can be seen as a family of functions $f_0 \oplus \cdots \oplus f_n \oplus \ldots$, where f_i is the function corresponding to the loop ending in exactly i iterations. For a deterministic program this family has disjoint domains because given an input there exists a unique i such that the loop terminates in i iterations. However, in the case of *collisions*, the codomains of all $(f_i)_{0\leq i}$ are not necessarily disjoint. Two or more functions f_i and f_j could share a common point in their images. We can define a loop as *collision free* if all iterations generate different output observations, i.e.

$$\forall \sigma, \sigma', i, j \neq i, \;\; f_i(\sigma) \neq f_j(\sigma')$$

The domain of f_i is given by all states σ such that $f_j(\sigma)(e)$ is true from $0 \leq j \leq i$ and always false from $i+1$ on. We denote this set of states also the *event* $e^{<i>}$; this event can be seen as the evaluation of the guard e. The body `M` is just a random variable as described in [14], representing the commands in the body. We denote $\mathtt{M}^i$ as the ith iteration of the body `M`.

Now, the entropy of such a loop is given by the entropy of the probabilities of the events $e^{<0>}, \ldots, e^{<n>}$ plus the entropy of $\mathtt{M}^i$ given the knowledge of $e^{<i>}$. This is a consequence of the *partition property* in [14].

Let's denote by $W(e, M)$ the leakage of a loop `while e M`

Theorem 1.

(i) If the loop is collision-free then $W(e, M)$ is the limit (for $n \to \infty$) of the formula $W(e, M)_n$ where

$$W(e,M)_n = \underbrace{H(\mu(e^{<0>}), \cdots, \mu(e^{<n>}))}_{guard} + \underbrace{\sum_{1\leq i\leq n} \mu(e^{<i>}) H(M^i | e^{<i>})}_{body}$$

(ii) If the loop is not collision-free, the leakage is $W'(e, M)$ defined as

$$W'(e,M) = W(e,M) - \sum_{\sigma \in C} [\sigma] H(\frac{\tau_1^\sigma}{[\sigma]}, \ldots, \frac{\tau_n^\sigma}{[\sigma]})$$

where C is the set of collisions of f, $[\sigma]$ is the (probability of the) inverse images of the collision σ under f, the probability of its elements intersected with f_i are denoted as $\tau_1^\sigma, \ldots, \tau_n^\sigma$ and $W(e, M)$ is the leakage of the disambiguation *of* `while e M`

Example. Consider the loop `l=0; while(l < h) l++` with 3 bit variables. There are only two events possible: the one where the loop exits, i.e. where `l >= h` and the one where `h` is already 0 and no iterations take place. This is represented by the events $e^{<n>}$ and $e^{<0>}$. Notice that the body can't leak anything because there are no confidential variables referenced in it. Thus, the leakage of the guard and therefore the whole loop is, assuming uniform input distribution of `h`, $H(\mu(e^{<0>}), \cdots, \mu(e^{<n>})) = H(\frac{1}{8}, \cdots, \frac{1}{8}) = \log_2(8) = 3$. As was expected, all 3 bits of the secret `h` leaked into `l` in this program.

Algebraic Interpretation of collision free loops Given a loop W, let W_n be the program W up to the n-th iteration. The random variable associated to W_n is hence a partition where only the outputs of W up to the $n-$th iteration are distinguished. Hence, W_{n+1} will refine W_n by introducing additional blocks.

As a simple example of a collision free program consider the "linear search" program P below

```
l=0;
while (l<h) do
    l=l+1;
```

We get the following corresponding family of partitions P_n:

$$P_n = \{\{0\}, \{1\} \ldots, \{n-1\}, \{x |\ x \geq n\}\}$$

The following proposition establishes the relation between collision free loops and the chain W_n being increasing:

Proposition 3. *For all n, $W_n \sqsubseteq W_{n+1}$ iff the loop W is collision-free.*

Proof. The direction $\Rightarrow$ follows immediately from the definition of collision. For the $\Leftarrow$ suppose $W_n \not\sqsubseteq W_{n+1}$, then at least a block in W_{n+1} is not a refinement of a block in W_n, e.g. $\{\{a\}, \{b, c\}\}$ in W_n and $\{\{a, b, c\}\}$ in W_{n+1} and by definition of W_n either $\{\{a\}$ or $\{b, c\}\}$ (w.l.g. we can say is $\{a\}$) corresponds to an output o after $\leq n$ iterations. Then $\{\{a, b, c\}\}$ in W_{n+1} corresponds to a collision, namely the collision which send a, b, c to the same output o in a different number of iterations.

Also note that in this case the chain $W_0 \sqsubseteq W_1 \sqsubseteq \ldots$ satisfies the ascending chain condition. There exists an integer n such that $W_m = W_n$ for all $m > n$, because W_{i+1} destructively refines ("splits") a finite block of W_i into smaller equivalence classes.

Proposition 4. *The random variable W of a collision-free loop is the Kleene fixpoint $\sqcup_{n\geq 0} W_n$ of the chain $(W_n)_{n\geq 0}$.*

Proof. The result follows from Proposition 3 and the fact that the number of states is finite.

In our "linear search" example above, the inverse image of f_i produces the block $\{i\}$ in partition P_n. Since we have shown that entropy *is* a semivaluation it follows that $W(e, M)_n$ is the semivaluation of the partition P_n above. The least upper bound of that partition is the partition where every state is in a singleton class, i.e. distinguishable; given k-bit variables

$$W = \sqcup_{n\geq 0} W_n = \{\{0\}, \{1\} \ldots, \{2^k - 1\}\}$$

which is the same as the partition produced by $f_0 \oplus \cdots \oplus f_{2^k-1}$, thus $\nu_\sqcup(W) = \lim_{n\to\infty} W(e, M)_n$. This is a true in general as stated in the following:

Theorem 2. *Given a collision-free loop `while e M`, the leakage $\lim_{n\to\infty} W(e, M)_n$ as shown in Theorem 1 (i) is equal to the semivaluation $\nu_\sqcup(\sqcup_{n\geq 0} W_n)$.*

Proof. This follows from Proposition 4.

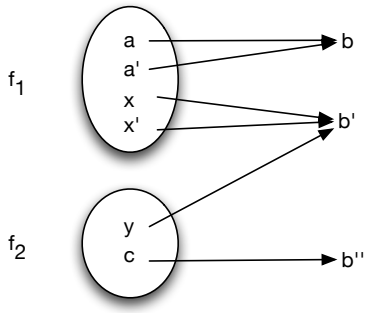


Fig. 1. Two iterations with one collision at b'

3.3 Loops with Collisions

Let us look at the colliding program shown in Figure 1. It consists of two iterations, represented by functions f_1 and f_2.

The exact partition for this program is

$$P = \{\{a, a'\}, \{x, x', y\}, \{c\}\}$$

The chain of partitions associated to the program is the following:

$$W_1 = \{\{a, a'\}, \{x, x'\}, \{y, c\}\}$$
$$W_2 = \{\{a, a'\}, \{x, x', y\}, \{c\}\}$$

We see that W_2 extends the block containing x, x' with y because all three of them have the same image b'. This reflects the idea of collisions, namely that two (or more) elements of the codomain of two different iteration functions, here f_1 and f_2 coincide. The result is that their inverse images are indistinguishable from one another and therefore end up being in the same block, here $\{x, x', y\}$. Then, W_2 is equal to P. However, because W_2 extends a block in W_1 this is not an ascending chain anymore; actually by choosing a distribution assigning probability 0 to c, we can see that $\nu_{\sqcup}(W_1) > \nu_{\sqcup}(W_2)$ and therefore Theorem 2 is false in case of collisions.

This conflict can be solved by applying the same technique as in [14]; i.e. by extending the codomain of the responsible functions by new elements until all collisions are resolved. In the case of this example, for $f_2 : X \to Y$ we extend Y to Y' with one new element to avoid the collision. This will result in a new function f_2' with $f_2'^{-1}(Y')$ producing a new distinguishable block $\{y\}$; now input y can be kept apart from inputs x, x' and the collision is resolved. The new chain is now

$$W_1' = W_1,\ W_2' = \{\{a, a'\}, \{x, x'\}, \{y\}, \{c\}\}$$

W_2' will turn the W_n sequence of partitions into an ascending chain again. Since W_2' has a larger semivaluation than W_2 some elements have to be subtracted to achieve the correct semivaluation (entropy). In [14] this problems is solved by creating a set of collisions and subtracting the weighted sum of the entropies of these collisions from the newly created, collision-free entropy. For this example, assuming uniform distribution Theorem 1(ii) produces the computation

$$H(W_2') - \frac{3}{6} H(\frac{2/6}{3/6}, \frac{1/6}{3/6}) = 1.4591 \tag{6}$$

Algebraically, this is handled by creating a *collision partition* C whose blocks are sets of colliding points; for this example we have

$$C = \{\{a\}, \{a'\}, \{x, x', y\}, \{c\}\}$$

It is easy to see that to recover W_2 from W_2' and C the meet operation can be applied

$$\nu_{\sqcup}(W_2) = \nu_{\sqcup}(W_2' \sqcap C)$$

notice now that we recover the same result as in equation (6) by this algebraic mean, i.e.

$$\nu_{\sqcup}(W_2' \sqcap C) = \nu_{\sqcup}(W_2) = H(\frac{2}{6}, \frac{3}{6}, \frac{1}{6}) = 1.4591$$

To generalize this construction, first define the *disambiguation* of a sequence of partitions $(W_i)_{i\geq 0}$ of a loop W by $(W_i')_{i\geq 0}$ where:

$$W_i' = \sqcup_{j\leq i} W_j$$

It is clear that $(W_i')_{i\geq 0}$ is an increasing chain and it is easy to see that it achieve the same chain defined above by adding new distinguishable blocks.

Define now the *collision equivalence* of a loop W by $\sigma \simeq_C \sigma'$ iff σ, σ' generate the same output from different iterations, formally

$$\sigma \simeq_C \sigma' \iff \exists i \neq j \;\; f_i(\sigma) = f_j(\sigma')$$

We denote by C the partition generated by taking the reflexive transitive closure of $\simeq_C$. We are now ready to relate the leakage of arbitrary loops with semivaluations on LoI.

Theorem 3. *The leakage of an arbitrary loop $W(e, M)$ as of Theorem 1(ii) is equivalent to semivaluating the meet of the least upper bound of its increasing chain W_n' and its collision partition C, i.e.*

$$W(e, M) = \nu_{\sqcup}(\sqcup_{n\geq 0} W_n' \sqcap C)$$

Proof. Notice first that increasing chains with a maximal element in a lattice do distribute, i.e.:

$$(\sqcup_{n\geq 0} x_n) \sqcap y = \sqcup_{n\geq 0}(x_n \sqcap y)$$

Assuming distributivity the argument is then easy to show:

$$(\sqcup_{n\geq 0} W_n' \sqcap C) = \sqcup_{n\geq 0}(W_n \sqcap C)$$

Notice now that $(W_n \sqcap C)_{n\geq 0}$ is a chain cofinal to the sequence $(W_n)_{n\geq 0}$ and so we can conclude that $\sqcup_{n\geq 0}(W_n \sqcap C)$ is the partition whose semivalutation corresponds to $W(e, M)$.

4 Applicability of LoI for Automation of Quantifying Information Flow

Tools to automatically quantify precise or approximative information flows are gaining a lot of momentum [17] [16] [1] [11]. We argue that while the material introduced in this paper is more of theoretical nature, there is still a surprisingly large applicability of these algebraic foundations. In this section we review a few recent developments and touch upon future work.

Most recently, Backes, Köpf, and Rybalchenko described their push-button verification tool *DisQuant* [1] which automatically calculates the LoI element given a program. It does so by iteratively generating a logical formula by counterexample-guided refinement using Model Checking and SAT solving. Once the partition has been exhaustively refined it enumerates its equivalence classes using constraint solving.

The authors of this paper also developed a dynamic analysis to precisely quantify information flows of loops [11]. The limitating factor of that tool is the size of the secret. If a user does not want to exhaustively run the program an all secrets an upper bound is calculated, using the formula from Theorem 1. If the leakage up to now (i.e. the current iteration) is described by L':

$$L' = H(m_1, \ldots, m_s, q) + \sum_{i=1}^{s} m_i V_i$$

where L' signifies the lower bound, then the upper bound is just $\mathtt{min}(k, L + q(k - L'))$; a conservative upper bound based on the principle of maximum entropy. Where L

$$L = H(p\, m_1, \ldots, p\, m_s, \overbrace{\frac{q}{t-s}, \ldots, \frac{q}{t-s}}^{t-s}) + \sum_{j=1}^{s} m_i V_i$$

is the leakage where the remaining probability $q = 1 - p$, and $p = \sum_{1 \leq i \leq s} m_i$ is distributed uniformly over the remaining $t - s$ events, where t is the loop bound.

In our new algebraic framework, this upper bound can be represented as widening operator which terminates in one iteration: At step n in the iteration sequence $(W_n)_{n \geq 0}$ the remaining, unhandled inputs get distributed in singleton blocks in the widened partition.

Future work will include work on quantitative declassification policies and enforcement systems using the metric induced by the semivaluation on LoI. Also, a very interesting direction for an application of this theory are exploiting recent advantages in SAT solvers, specifically in model counting.

5 Conclusions

Information theory, and in general probability theory, is based on underlying lattice structures. We investigated how this lattice-theoretic perspective fits in our recent framework of quantifying information flow [5, 14]. We showed that we can use join semivaluations on the lattice of information for quantitative information flow analysis of programs.

References

1. Michael Backes and Boris Köpf and Andrey Rybalchenko: Automatic Discovery and Quantification of Information Leaks. Proc. 30th IEEE Symposium on Security and Privacy (S& P '09), to appear
2. Birkhoff, G., Lattice theory. Amer. Math. Soc. Colloq. Publ. 25 (1948).
3. Cachin, C.: Entropy Measures and Unconditional Security in Cryptography. PhD thesis, Swiss Federal Institute of Technology (1997)
4. Han Chen, Pasquale Malacaria: Quantifying Maximal Loss of Anonymity in Protocols. In Proceedings ACM Symposium on Information, Computer and Communication Security 2009.
5. David Clark, Sebastian Hunt, Pasquale Malacaria: A static analysis for quantifying information flow in a simple imperative language. Journal of Computer Security, Volume 15, Number 3 / 2007.
6. David Clark, Sebastian Hunt, and Pasquale Malacaria: Quantitative information flow, relations and polymorphic types. Journal of Logic and Computation, Special Issue on Lambda-calculus, type theory and natural language, 18(2):181-199, 2005.

7. Cortesi, A: Widening Operators for Abstract Interpretation. In Proceedings of the 2008 Sixth IEEE international Conference on Software Engineering and Formal Methods.
8. Boris Köpf and David Basin: An information-theoretic model for adaptive side-channel attacks. CCS '07: Proceedings of the 14th ACM conference on Computer and communications security, 2007, 286-296
9. T.Cover, J. Thomas. Elements of Information Theory. Wiley
10. Denning, Dorothy E. A lattice model of secure information flow. Commun. ACM, 19, 5, 1976, 236–243 ACM, New York, NY, USA
11. Jonathan Heusser, and Pasquale Malacaria: Measuring Insecurity of Programs, Draft, Queen Mary University of London 2007. http://www.dcs.qmul.ac.uk/˜jonathan/publications.html
12. J A Goguen, J Meseguer: Security policies and security models In Proceedings of the 1982 IEEE Computer Society Symposium on Security and Privacy
13. Landauer, J., and Redmond, T.: A Lattice of Information. In Proc. of the IEEE Computer Security Foundations Workshop. IEEE Computer Society Press, 1993.
14. Pasquale Malacaria: Assessing security threats of looping constructs. Proc. ACM Symposium on Principles of Programming Language, 2007.
15. Pasquale Malacaria, Han Chen: Lagrange Multipliers and Maximum Information Leakage in Different Observational Models. ACM SIGPLAN Third Workshop on Programming Languages and Analysis for Security. June, 2008.
16. Stephen McCamant and Michael D. Ernst: Quantitative information flow as network flow capacity. PLDI 2008, Proceedings of the ACM SIGPLAN 2008, Conference on Programming Language Design and Implementation, Tucson, AZ, USA, 2008
17. Stephen McCamant and Michael D. Ernst: A simulation-based proof technique for dynamic information flow. PLAS 2007: ACM SIGPLAN Workshop on Programming Languages and Analysis for Security, San Diego, CA, USA, 2007
18. John Mclean: Security Models and Information Flow. In Proc. IEEE Symposium on Security and Privacy, 1990, 180–187 IEEE Computer Society Press
19. Jonathan K. Millen: Covert Channel Capacity. IEEE Symposium on Security and Privacy, 0, 1987, 1540-7993, 60 IEEE Computer Society, Los Alamitos, CA, USA
20. Y. Nakamura. Entropy and Semivaluations on Semilattices. Kodai Math. Sem. Rep 22 (1970), 443 468
21. Dan Simovici,: Metric-Entropy Pairs on Lattices. Journal of Universal Computer Science, vol. 13, no. 11 (2007), 1767-1778
22. Geoffrey Smith: On the Foundations of Quantitative Information Flow. In Proc. FOSSACS 2009: Twelfth International Conference on Foundations of Software Science and Computation Structures LNCS 5504, pp. 288-302, York, UK, March 2009
23. D.G. Weber: Quantitative Hook-Up Security for Covert Channel Analysis In Proc. IEEE Computer Security Foundations Workshop 1988.

Measuring the Loss of Privacy from Statistics

Michael Carl Tschantz*
Computer Science Department
Carnegie Mellon University
mtschant@cs.cmu.edu

Aditya V. Nori
Rigorous Software Engineering
Microsoft Research India
adityan@microsoft.com

Abstract

We present a specialization of quantitative information flow to programs that compute statistics. We provide an approach for estimating the information flows present in such programs based on Monte Carlo simulation and argue that it is more accurate than previous approaches in this domain.

1 Introduction

Organizations often collect sensitive information about survey respondents. To protect the privacy of the respondents, they only publish aggregate statistics about the responses rather than the responses themselves. These statistics are designed to provide information about the responses as a whole without providing a detailed view of any one response. However, under some circumstances, these statistics may reveal sensitive information about a particular respondent. We would like to quantify how much information about a single respondent can be learned from a given statistic.

For example, a trivially unsafe program might just report the responses themselves including the name of the person who provided each response. Likewise, a trivially safe program might always report "access denied" providing no information.

For a less trivial example, consider a program that takes two non-negative integer salaries and returns their sum:

```
return (salary1 + salary2)
```

Such a program provides an upper bound on each respondent's salary since neither can be greater than twice the average. Furthermore, if the sum is zero, the sum also provides the exact salary of each respondent. If, on the other hand, the sum is one, then two possibilities for each respondent's salary remain: zero and one. As the sum goes up, the number of possibilities goes up. Thus, unlike the trivial cases above in which the program could be analyzed independently of the response it produces, in this case, the value of the produced statistic influences the amount of privacy maintained.

Our goal is to provide an automated method for determining the amount of information that flows through a program that computes a statistic. We further desire that our analysis is accurate enough to provide reasonable results for common statistics. For example, Clark et al. provide an analysis for measuring the mutual information flow from sensitive inputs to public outputs [1]. While their approach produces results accurate enough for their problem domain, confidentiality, it is not accurate enough for use on statistics.

*This work was primarily done while the first author was an intern at Microsoft Research India.

For example, it cannot distinguish between a program that simply lists the responses and a program that provides the sum of all the responses.

To meet this goal, we use Monte Carlo simulation. This simple approach has many advantages. By treating the program as a black box, it can work on any program written in any language and is fully automated. We need not create any models or ensure that the program obeys a typing discipline. Running the actual implementation rather than analyzing a specification of what a statistic should calculate catches the effects of bugs. Despite not having a soundness guarantee, with enough samples, our approach will approach the exact values, whereas sound analyses often provide very loose bounds.

Rather than simply provide one number that measures this loss, we provide both the probability distribution over the sensitive attribute for a respondent before and after learning the value of the statistic. From these distributions, many measures of information flow (privacy loss) used in other works can be easily calculated including mutual information [1] and the change in distribution accuracy [2].

First, we present an formal model of programs that produce statistics from a list of survey responses. Second, we formalize the problem and discuss a related problem that is more practical in many settings. Third, we discuss our analysis. Fourth, evaluate our analysis on simple statistics. Lastly, we discuss related work and conclude. While an intuitive understanding of probability suffices for understanding this work, the appendix formalizes our models with measure theory.

2 Model

We model a program that computes a statistic as a function f that accepts as input a finite list of survey responses and produces as output the value of the statistic. Let each survey response be an element of the countable set $\mathscr{X}$ and let the value of the statistic range over the countable set $\mathscr{Y}$. Thus, the program is treated as a function f from $\mathscr{X}^*$ to $\mathscr{Y}$. While the restriction to countable sets $\mathscr{X}$ and $\mathscr{Y}$ might seem unnatural given a survey of continuous values such as weights or heights, this is not a limitation in practice since respondents only ever provide this information to a fixed accuracy (such as to the nearest kilogram for weight). Also, we can model probabilistic programs by having f accept a second argument that determines the probabilistic choices. In our implementation, it's irrelevant since it treats programs as black boxes.

The program operates in an environment from which its input comes. Let the set Ω represent the set of possible worlds and P be a probability measure over these worlds. The survey is conducted and program ran in one of these worlds $\underline{\omega}$, the actual world.

Let $\mathbf{X}$ be a random variable from Ω to $\mathscr{X}^*$ that provides the inputs to the program. This models the process of conducting the survey, which provides the program some information about the actual world $\underline{\omega}$. We use $\underline{\mathbf{x}} = \langle \underline{x}_1, \ldots, \underline{x}_n \rangle$ to denote the actual survey responses provided to the program: that is, $\underline{\mathbf{x}} = \mathbf{X}(\underline{\omega})$.

The program f computes the value of a statistic of the provided survey responses. This defines a new random variable $Y = f \circ \mathbf{X}$ from Ω to $\mathscr{Y}$. We use $\underline{y}$ to denote the actual value of the statistic: $\underline{y} = f(\underline{\mathbf{x}}) = f(\mathbf{X}(\underline{\omega}))$.

For example, X_i could be random variable that relates the weight of the ith surveyed person. That is, $X_i(\omega)$ represents the weight of the ith surveyed person in the possible world ω. Since $\underline{\omega}$ is the actual world, $\underline{x}_i = X_i(\underline{\omega})$ is the actual weight of the ith surveyed person. The program f could accept a list of such weights and compute their mean. Then Y would be a random variable that provides the mean of the respondents given a possible world with the actual mean being $\underline{y} = f(\underline{x}_1, \ldots, \underline{x}_n)$.

We model an adversary as attempting to determine the value taken by some random variable Z where Z ranges over $\mathscr{Z}$. That is, the adversary, would like to determine $\underline{z} = Z(\underline{\omega})$. For example, Z might be *The weight of Bob* or *Bob has AIDS*. The surveyor must determine for which random variables Z the adversary

should not be able to determine the value taken. These random variables will vary from survey to survey depending on the information collected by the survey and privacy expectations of the respondents.

The adversary has some prior beliefs about $\underline{z}$. We assume that the adversary knows what worlds are possible, how the survey was conducted, and what statistic was computed (that is, he knows Ω, $\mathbf{X}$, and f). However, we assume that the adversary does not know the actual world $\underline{\omega}$ or the actual responses $\underline{x} = \mathbf{X}(\underline{\omega})$. Rather than knowing the actual probability measure P, which is impossible to know exactly in many realistic environments, the adversary has beliefs about the world represented as a probability measure Q.

3 Problem Formalization

Before formalizing the problem, we provide some notation. Given a random variable Z and probability measure Q, we use Q_Z to denote the distribution D over $\mathscr{Z}$ such that $D(z) = Q[Z = z]$ for all z in $\mathscr{Z}$. Similarly, $(Q|Y = y)_Z$ represents the distribution D such that $D(z) = Q[Z = z|Y = y]$ for $y \in \mathscr{Y}$ such that $Q[Y = y] \neq 0$.

Our goal is to provide an analysis that computes a comparison of the adversary's knowledge before and after seeing the statistic $\underline{y}$. That is, a comparison of the distribution of Q_Z and the distribution $(Q|Y = \underline{y})_Z$. Since many such comparisons exist, our analysis will provide both Q_Z and $(Q|Y = \underline{y})_Z$ and allow the analysis user to perform any selected comparison upon them.

While a comparison of Q_Z and $(Q|Y = \underline{y})_Z$ is ideal, it seems unreasonable that the surveyor would know the adversary's prior beliefs Q. Furthermore, the surveyor cannot do a worse case analysis over all possible values for Q_Z since it could be arbitrarily bad as an adversary could be arbitrarily ignorant before seeing the program output. Thus, we must make some assumptions about the adversary to produce a problem that the surveyor can practically solve given reasonably accessible information.

First, we assume that the adversary bases his prior distribution Q_Z on the actual probability measure P. That is, we assume that Q_Z is P_Z. This assumption, as pointed out by Clarkson et al. [2], is made implicitly by most works on quantitative information flow (e.g., the work of Clark et al. [1]). This first assumption might appear to not help us since we have traded one unknown, Q, for another unknown, P. However, unlike Q_Z, the surveyor can estimate P_Z using the next three assumptions.

Second, we assume that Z is determined by $\mathbf{X}$. That is, we assume that the surveyor can decompose Z using some function g such that $Z = g \circ \mathbf{X}$. For example, if Z is the response of the first respondent, then g is a function that returns the first response from the sequence of actual responses $\mathbf{X}(\underline{\omega})$. This assumption is reasonable since such random variables are the most vulnerable to attack. (If Z is not completely determined by $\mathbf{X}$, then the surveyor would have to also provide an estimation of the other factors that determine Z. It would still be possible to use our approach, but we wish to avoid this complication.)

Third, we assume that the adversary knows the number of responses in the actual responses $\underline{\mathbf{x}} = \langle \underline{x}_1, \ldots, \underline{x}_n \rangle = \mathbf{X}(\underline{\omega})$. That is, he knows n. Since most surveys publish the number of responses examined, this assumption is not too limiting. Fixing n, we can treat $\mathbf{X}$ as consisting of n random variables X_1 to X_n with each X_i producing one response x_i.

Fourth, we assume that X_1 to X_n are independent and identically distributed. Statistically accurate surveys will meet this assumption by design. Under this assumption, x_1 to x_n are n samples from a single distribution P_X. Given the n samples $\mathbf{x}$, the surveyor can approximate P_X. Let $\hat{P}_X$ be one such approximation selected by surveyor. This estimates $P_{\mathbf{X}}$ as $\hat{P}_{X^n}$ (i.e., the distribution resulting from n independent and identically distributed copies of X).

These assumptions combine to allow the surveyor to estimate Q_Z as $\hat{P}_{g \circ X^n}$. The problem then becomes to compute a comparison of $\hat{P}_{g \circ X^n}$ and $(\hat{P}|Y = \underline{y})_{g \circ X^n}$ from the following inputs:

- the program f where $Y = f \circ \mathbf{X}$,
- the actual value of the responses $\underline{\mathbf{x}} = \mathbf{X}(\underline{\omega})$,
- a function g where adversary is attempting to learn $Z(\underline{\omega}) = g(\mathbf{X}(\underline{\omega}))$, and
- an approximation $\hat{P}_X$ of the distribution P_X that generated the responses and determines Z.

Note the problem depends not just on the statistic f, but also on the actual value of the statistic, the information that the adversary would like to learn, and the estimation of the distribution P_X. This requires that the survivor solve this problem each time the statistic is to be applied to different responses or with a different adversary. However, as argued in the introduction, the amount of information flow is sensitive to these changes.

4 Analysis

We now present a simple analysis for providing an approximate answer to the practical version of the problem above. We also discuss our implementation of this analysis.

We use Monte Carlo simulation to estimate $(\hat{P}|Y = y)_Z$ as follows. We repetitively use $\hat{P}_X$ to generate a sample $\mathbf{x}'$ from $\hat{P}_{X^n}$, we run f on $\mathbf{x}'$ to produce y', and we run g on $\mathbf{x}'$ to produce z'. By keeping track of the value z' takes on each time y' is equal to $\underline{y}$, we can construct estimations of $\hat{P}_Z$ and $(\hat{P}|Y = y)_Z$ in the usual way: we estimate $\hat{P}_Z(z)$ as the number of samples that result in $Z = z$ divided by the number of samples and we estimate $(\hat{P}|Y = y)_Z(z)$ as the number of samples that resulted in both $Z = z$ and $Y = y$ divided by the number of samples that resulted in $Y = y$.

An advantage of this method is it works for any f and g that are functions. (The method also works for randomized functions provided that the surveyor can model their sources of randomness.) The method runs on large, complex programs even without source code.

Since constructing $(\hat{P}|Y = y)_Z$ takes memory linear in $\mathscr{Z}$ (not counting any memory used by f or g), this approach will not work for large $\mathscr{Z}$. However, one may choose to focus on a subset of $\mathscr{Z}$ that indicate sensitive outcomes to reduce memory usage to the size of this subset. For example, one might focus only on $\underline{z}$, the actual value that Z takes on, and calculate $\hat{P}(Z = \underline{z}|Y = y)$ for comparison to $\hat{P}(Z = \underline{z})$.

Several factors can slow down gaining an accurate estimation. If f or g is a time intensive computation, our dynamic analysis will be slow. A large size of $\mathscr{X}$ or n, or a low value for $\hat{P}(Y = \underline{y})$ can each result in needing a large number of samples for constructing an accurate estimation of $(\hat{P}|Y = y)_Z$. While surveys that ask for exact answers can have a large $\mathscr{X}$, many only ask multiple choice questions yielding a more manageable $\mathscr{X}$.

In general a large n can be problematic, but in the following special case, we can optimize our analysis to not depend upon n. Some statistics strips sensitive information (such as name) from each X_i and lists the sanitized form. Such statistics f have the form $f([X_1, X_2, \ldots, X_n]) = [f'(X_1), f'(X_2), \ldots, f'(X_n)]$ for some function f'. If Z is independent of all X_i except one of them, say X_i, then

$$\begin{aligned}\hat{P}(Z = z|Y = y) &= \hat{P}[Z = z|f([X_1, \ldots, X_i, \ldots, X_n]) = [y_1, \ldots, y_i, \ldots, y_n]] \\ &= \hat{P}[Z = z|f'(X_1) = y_1, \ldots, f'(X_i) = y_i, \ldots, f'(X_n) = y_n] \\ &= \hat{P}[Z = z|f'(X_i) = y_i]\end{aligned}$$

where the last equality follows from Z being independent of all X_j other than X_i. Thus, we can ignore all X_j other than X_i. This greatly speeds up the approximation.

5 Evaluation

To evaluate our approach, we fix a method of comparing $\hat{P}_Z$ and $(\hat{P}|Y = \underline{y})_Z$. The method we choose uses *entropy*, an information theoretic measure of the amount of uncertainty associated with a distribution. The entropy of the distribution $\hat{P}_Z$ is

$$\mathcal{H}(\hat{P}_Z) = -\sum_{z \in \mathscr{Z}} \hat{P}[Z = z] \log_2 \hat{P}[Z = z]$$

and the entropy of the distribution $(\hat{P}|Y = \underline{y})_Z$ is

$$\mathcal{H}((\hat{P}|Y = \underline{y})_Z) = -\sum_{z \in \mathscr{Z}} \hat{P}[Z = z|Y = y] \log_2 \hat{P}[Z = z|Y = y]$$

(One usually speaks of the entropy of a random variable with the underlying probability measure P being understood. Since we are dealing with two probability measures, $\hat{P}$ and $\hat{P}|Y = y$, we choose to make them explicit.)

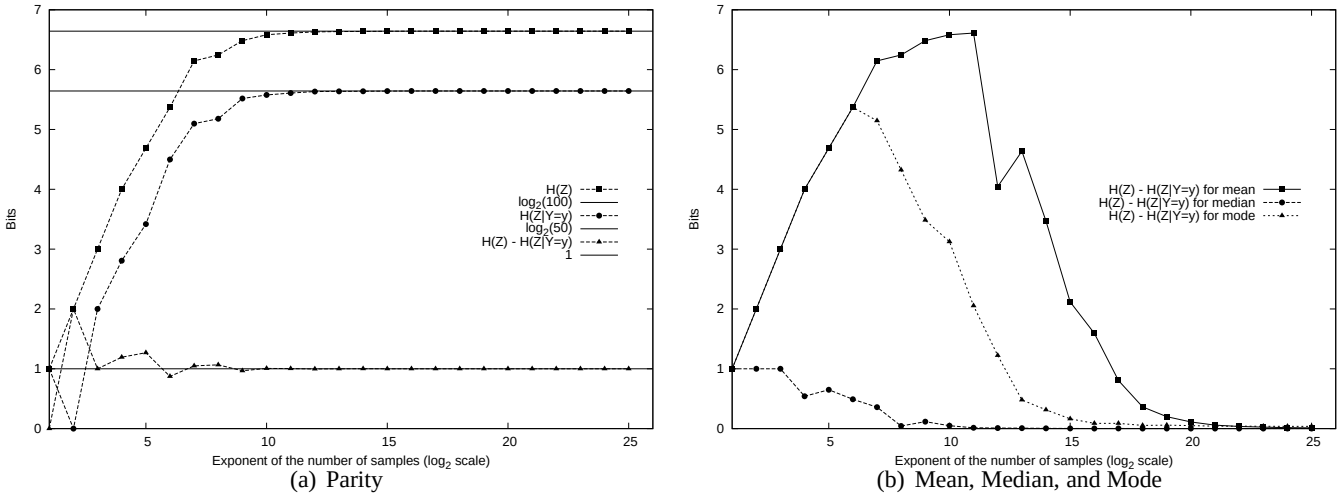
The comparison of the distributions $\hat{P}_Z$ and $(\hat{P}|Y = \underline{y})_Z$ we use is the difference of their entropies: $\mathcal{H}(\hat{P}_Z) - \mathcal{H}((\hat{P}|Y = \underline{y})_Z)$. Clark et al. [1] argues that this difference measures the amount of information that flows from $Y = \underline{y}$ to the adversary about Z since it is the decrease in the uncertainty of Z after learning that Y is equal to $\underline{y}$. Indeed, this difference is related to mutual information, an information theoretic measure of how much information one random variable provides about another. Ignoring that $Y = \underline{y}$ is a condition and not a random variable, $\mathcal{H}(\hat{P}_Z) - \mathcal{H}((\hat{P}|Y = \underline{y})_Z)$ may be seen as providing the mutual information $\mathcal{I}(Z; Y = \underline{y})$ between Z and $Y = y$ for a deterministic program.

Using entropy, we computed the difference between $\hat{P}_Z$ and $(\hat{P}|Y = \underline{y})_Z$ for various statistics. In all cases we used the uniform distribution over 0 to 99 for each X_i. We selected the uniform distribution since by having a high variance, we expected it to be a challenging distribution for the analysis in the sense of requiring a large number of samples. For Z, we used the value of the first input X_1.

The first statistic we consider is the parity of X_1. This is not a particularly interesting statistic, but we can exactly calculate $\mathcal{H}(\hat{P}_Z)$ to be $\log_2(100)$ and $\mathcal{H}((\hat{P}|Y = \underline{y})_Z)$ to be $\log_2(50)$ allowing us to see the accuracy of our analysis. To study convergence and show that our analysis can provide accurate estimations, we show the estimations produced using various numbers of samples in Figure 1(a). The y-axis shows the estimated values for the entropies and mutual information while the x-axis shows the number of samples performed, which ranges from 2^1 to 2^{25}. This table shows that the estimations of the values of $\mathcal{H}(\hat{P}_Z)$ and $\mathcal{H}((\hat{P}|Y = \underline{y})_Z)$ approach their real values as the number of samples increases. Thus, the estimation of $\mathcal{H}(\hat{P}_Z) - \mathcal{H}((\hat{P}|Y = \underline{y})_Z)$ approaches its real value as well. By 2^{25} samples, the mutual information is less then 0.0000003 bits away from the exact value of 1.

Note that the estimations of $\mathcal{H}(\hat{P}_Z)$ and $\mathcal{H}((\hat{P}|Y = \underline{y})_Z)$ tend to approach from below. Indeed, our estimator is a biased one. While others have created less unbiased estimators ([5] provides a recent overview), we simply opt to use more samples instead.

The results for more realistic statistics (mean, median, and mode) are shown in Figure 1(b). Note that the value of the estimations for all three statistics stabilized by 2^{23} samples. The raise and fall of the estimations is due to both the estimations of $\mathcal{H}(\hat{P}_Z)$ and $\mathcal{H}((\hat{P}|Y = \underline{y})_Z)$ approaching their real values from below with $\mathcal{H}(\hat{P}_Z)$ approaching it's real value more quickly than $\mathcal{H}((\hat{P}|Y = \underline{y})_Z)$. This creates a period where $\mathcal{H}(\hat{P}_Z)$ is a reasonable estimation and $\mathcal{H}((\hat{P}|Y = \underline{y})_Z)$ is a radical underestimation resulting in $\mathcal{H}(\hat{P}_Z) - \mathcal{H}((\hat{P}|Y = \underline{y})_Z)$ being a radical overestimation.



(a) Parity (b) Mean, Median, and Mode

Figure 1: Estimations for Various Statistics

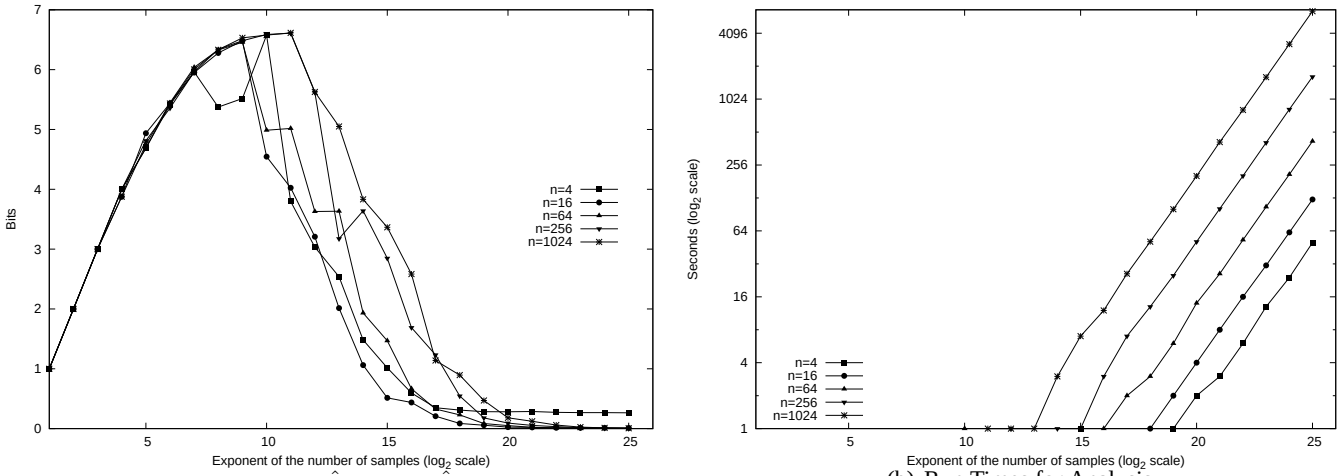
Statistic	$\mathcal{H}(\hat{P}_Z) - \mathcal{H}((\hat{P}\|Y = \underline{y})_Z)$	Run Time (secs)
Parity	0.999999797131	639
Mean	0.0125233560025	684
Median	0.00205987477602	1498
Mode	0.0376910036281	2444

Table 1: Summery of Analysis Results for Four Statistics

Table 1 summarizes the estimations for 2^{25} samples and shows the amount of time taken to compute these results for running on a 3.2 GHz, 64-bit processor. Note that the estimations of $\mathcal{H}(\hat{P}_Z) - \mathcal{H}((\hat{P}|Y = \underline{y})_Z)$ for the mean, median, and mode are all lower than for parity. This conforms our suspicion that aggregate statistics tend to reveal little about their respondents. The time for estimating these values grow linearly with the number of samples as expected. The slowest was mode, which took 41 minutes for 2^{25} samples. However, an estimation that differs by less than 0.021 bits (0.32%) is available in under a minute using 2^{19} samples.

To explore how the number samples n affects the value of $\mathcal{H}(\hat{P}_Z) - \mathcal{H}((\hat{P}|Y = \underline{y})_Z)$ and the rate of convergence to it, Figure 2(a) shows the estimations of $\mathcal{H}(\hat{P}_Z) - \mathcal{H}((\hat{P}|Y = \underline{y})_Z)$ for the mean for varying sizes n. Using more respondents decreased the difference between $\mathcal{H}(\hat{P}_Z)$ and $\mathcal{H}((\hat{P}|Y = \underline{y})_Z)$. However, it increased the number of samples needed for convergence since convergence requires seeing many samples such that $Y = \underline{y}$, which becomes a less common event as n increases. Furthermore, it increased the amount of time needed to compute the value of the statistic keeping the number of samples constant since calculating the mean over more respondents takes longer. In the worse case, the mean over 1024 respondents, it took 109 minutes for 2^{25} samples with convergence still not reached. Figure 2(b) plots these run times.

Our implementation may be downloaded from `http://www.cs.cmu.edu/~mtschant/mcqif/`



(a) Estimations of $\mathcal{H}(\hat{P}_Z) - \mathcal{H}((\hat{P}|Y = \underline{y})_Z)$

(b) Run Times for Analysis

Figure 2: Results for Mean with Various Numbers of Respondents

6 Related Work

Quantitative Information Flow. Much work has been done on information flow analysis. We will only discuss those works that deal with quantifying the flow of information. These works concern themselves with either confidentiality or integrity. In both cases, the tool user partitions the inputs and outputs of the program into high-level and low-level classes. Quantitative information flow for confidentiality measures how much the high-level inputs affect the low-level outputs. Quantitative information flow for integrity, on the other hand, measures how much the low-level inputs affect the high-level outputs. The two problems are dual and an analysis for one will apply to the other. Since our work fits under the confidentiality problem, we will discuss all related works from this angle even if they were created with integrity in mind.

The work of Clark, Hunt, and Malacaria presents a formal model of programs for quantifying information flows and a static analysis that provides lower and upper bounds on the amount of information that flows [1]. They measure information flow as the mutual information between the high-level inputs and low-level outputs given that the adversary has control over the low-level inputs. That is, they measure $\mathcal{I}(L^{\mathsf{out}}; H^{\mathsf{in}}|L^{\mathsf{in}})$ where L^{out} is a random variable representing the low-level outputs, H^{in} is one representing the high-level inputs, and L^{in} is one representing low-level inputs. Unlike our work that measures the information flow in a program given a particular input, their analysis provides upper and lower bounds on the size of the information flow in a given program regardless of the actual inputs or the distributions that generate them. Since the upper bound holds for all input distributions, it is an upper bound on the channel capacity of the program.

Their analysis, if implemented, could be used for our problem by treating the inputs $\mathbf{X}$ as H^{in}, using Y as L^{out}, and assuming that $Z = \mathbf{X}$. (L^{in} is unused since we do not allow the adversary to control any inputs to the statistic.) However, their analysis produces bounds that are too loose for our purposes. For example, no matter how many independent and identically distributed samples goes into a mean, their analysis will state that all the information about the first sample is provided as output despite the fact that it would be hidden amongst other samples.

McCamant and Ernst provide a dynamic analysis for quantitative information flow using the mutual information formalization [6]. Their analysis provides an upper bound on the flow of information of a single

path of execution in a program. Their analysis converts a path of execution into a flow network. They then find the max cut of the network to bound the information flow. Unlike us, they provide a sound upper bound for that path of execution instead of an estimate. However, like the work of Clark et al., their analysis does not account for information hiding in the calculations like a sum making the bound too loose to be useful for our purposes.

Newsome and Song also provide a dynamic analysis for quantitative information flow using the mutual information formalization [7]. Their analysis converts a single path of execution into a logical formula that characterizes the path. Each solution to this formula corresponds to a value that the output Y can take on while taking that path of execution. If all such solutions are found, this provides the channel capacity between $\mathbf{X}$ and Y provided only the analyzed path of execution is ever used. In practice, a theorem prover can rarely find all such solutions, and thus, their analysis only provides a lower bound on the channel capacity. Whether or not this bound is tight enough for our uses depends on the theorem prover and the formula.

Clarkson, Myers, and Schneider object to the mutual information formulation of quantitative information flow [2]. Instead they proposed a formulation using the beliefs of the adversary. However, such a formulation is often not practical since the surveyor often will not know the adversary's beliefs. After adjusting their definitions for our uses, information flow is defined to be $\mathcal{D}(Q_Z \to \dot{\underline{z}}) - \mathcal{D}((Q|Y = \underline{y})_Z \to \dot{\underline{z}})$ where Q is the adversary's beliefs, $\underline{z} = Z(\omega)$ is the actual value of the random variable the adversary is attempting to learn, $\dot{\underline{z}}$ is a distribution over $\mathscr{Z}$ that assigns 1 to $\underline{z}$ and 0 to every other element of $\mathscr{Z}$, and $\mathcal{D}(Q_Z \to P_Z)$ is the relative entropy:

$$\mathcal{D}(Q_Z \to P_Z) = \sum_{z \in \mathscr{Z}} P_Z(z) \log \frac{P_Z(z)}{Q_Z(z)}$$

For deterministic programs, they prove that $\mathcal{D}(Q_Z \to \dot{z}) - \mathcal{D}((Q|Y = y)_Z \to \dot{z})$ reduces to $-\log Q_Y(y)$. We can calculate this given an approximation of Q_Y directly. We could also calculate this using our sampling approach given an approximation of Q or Q_X.

Preserving Privacy. Statistical disclosure limitation attempts to preserve privacy despite releasing statistics. (For an overview see [4].) Most of the methods used in this line of work are specialized for a single class of statistics. Most often this is the class of frequency tables, tables that record the number of respondents with various combinations of attributes. Tables of magnitudes and sanitized individual responses (microdata) are also considered. While our approach is more efficient for some statistics than others, it can work on any statistic provided it is calculated by a computer.

Other works in statistical disclosure limitation use Monte Carlo simulation for purposes other than ours. For example, Slavković uses it construct an estimation of probability distributions over outputs ($\hat{P}_X$ in our notation) [8].

Differential privacy is a formalization of what it means for a statistic to maintain the privacy of the respondents about which it is calculated [3]. It requires that the output that the program produces is probably no different from the output it would have produced if one respondent were dropped from or added to the survey. In particular, for a statistic f to have ϵ-differential privacy, it must be the case that for all sets D_1 and D_2 of responses that differ on at most one response and all subsets S of the range of f

$$\Pr[f(D_1) \in S] \leq e^{\epsilon} \Pr[f(D_2) \in S]$$

This ensures that the probability of the statistic's output falling in some set S changes only by a factor e^{ϵ} as a single respondent's information is either added or removed from the survey. Intuitively, if the statistic probably looks the same regardless of if a person is surveyed or not, an adversary cannot learn much information

about the person. While we could consider ϵ to be measure of information flow, it does not lend itself to the analysis of many standard statistics since they do not have ϵ-differential privacy for any value of ϵ. For example, the mean of respondent incomes would not satisfy ϵ-differential privacy for any ϵ since it would surely change by at least a small amount with a respondent removed. (A version of the mean statistic that adds random noise to the result could be constructed to satisfy ϵ-differential privacy for an ϵ that depends upon the amount of noise added.)

7 Conclusions and Future Work

We have provided an analysis for determining the amount of information that an adversary learns from a statistic given various assumptions. Future work could ease these assumptions. However, this work and all works on quantitative information flow must make some assumption about the adversary. In most works, including our own, they assume that the adversary's beliefs Q are in line with the actual word P and that adversary has no additional background knowledge. Clarkson et al. instead assume they can model the adversary. Both of these assumptions are troubling. This suggests that methods that do not depend on the adversary, such as differential privacy [3], might provide a better solution to protecting privacy. However, it considers every standard statistic (mean, median, mode, etc.) equally and completely unprivate.

Other future work could combine our method with static approaches for information flow such as the work of Clark et al. [1]. Such a hybrid approach, if possible, might scale to systems too large or slow for our Monte Carlo approach while using our approach to closely examine key components of the program.

References

[1] CLARK, D., HUNT, S., AND MALACARIA, P. A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security 15* (2007), 321–371.

[2] CLARKSON, M. R., MYERS, A. C., AND SCHNEIDER, F. B. Belief in information flow. In *CSFW '05: Proceedings of the 18th IEEE workshop on Computer Security Foundations* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 31–45.

[3] DWORK, C. Differential privacy. In *33rd International Colloquium on Automata, Languages and Programming (ICALP 2006)* (2006), vol. 2, pp. 1–12.

[4] FEDERAL COMMITTEE ON STATISTICAL METHODOLOGY. Statistical disclosure limitation methodology. Statistical Policy Working Paper 22, 2005.

[5] KENNEL, M. B., SHLENS, J., ABARBANEL, H. D. I., AND CHICHILNISKY, E. J. Estimating entropy rates with bayesian confidence intervals. *Neural Computation 17*, 7 (jul 2005), 1531–1576.

[6] MCCAMANT, S., AND ERNST, M. D. A simulation-based proof technique for dynamic information flow. In *PLAS '07: Proceedings of the 2007 workshop on Programming languages and analysis for security* (New York, NY, USA, 2007), ACM, pp. 41–46.

[7] NEWSOME, J., AND SONG, D. Influence: A quantitative approach for data integrity. Tech. Rep. CMU-CyLab-08-005, CyLab, Carnegie Mellon University, 2008.

[8] SLAVKOVIĆ, A. B. *Statistical Disclosure Limitation Beyond the Margins: Characterization of Joint Distributions for Contingency Tables*. PhD thesis, Carnegie Mellon University, 2004.

A The Model More Formally

In this section, we provide definitions that are more formal then the ones found in Sections 2 and 3.

Formally, we model the environment from which program inputs come as a probability space $\langle \Omega, \mathcal{F}, P \rangle$ with the sample space Ω, events $\mathcal{F}$, and probability measure P that models this environment.

Let $\mathscr{X}^*$ be the set of inputs that the modeled program can consume. We assume that $\mathscr{X}$ is countable, implying that $\mathscr{X}^*$ is countable. This ensures that $\langle \mathscr{X}^*, 2^{\mathscr{X}^*} \rangle$ a measurable space. The random variable $\mathbf{X}$, which models program inputs, is from the probability space $\langle \Omega, \mathcal{F}, P \rangle$ to the measurable space $\langle \mathscr{X}^*, 2^{\mathscr{X}^*} \rangle$.

Let $\mathscr{Y}$ be the set of outputs that the modeled program can produce. We assume that $\mathscr{Y}$ is countable, and thus, $\langle \mathscr{Y}, 2^{\mathscr{Y}} \rangle$ is a measurable space. Let $f : \mathscr{X}^* \to \mathscr{Y}$ be a function that models the program. Let Y be $f \circ \mathbf{X}$, which models the output of the program. Y is from the probability space $\langle \Omega, \mathcal{F}, P \rangle$ to the measurable space $\langle \mathscr{Y}, 2^{\mathscr{Y}} \rangle$. Y is a well-defined random variable since for any $S \in 2^{\mathscr{Y}}$, $f^{-1}(S)$ must be in $2^{\mathscr{X}^*}$ and the state space of $\mathbf{X}$ is $\langle \mathscr{X}^*, 2^{\mathscr{X}^*} \rangle$ ensuring that $\mathbf{X}^{-1}(f^{-1}(S)) = Y^{-1}(S)$ is in $\mathcal{F}$.

We model an adversary as attempting to determine the value took on by some random variable Z from $\langle \Omega, \mathcal{F}, P \rangle$ to some measurable space $\langle \mathscr{Z}, 2^{\mathscr{Z}} \rangle$, again, assuming that $\mathscr{Z}$ is countable.

We model the adversary's beliefs about the world as a probability measure Q on $\langle \Omega, \mathcal{F} \rangle$.

Given a random variable X from $\langle \Omega, \mathcal{F}, P \rangle$ to $\langle \mathscr{X}, \Sigma \rangle$, the *distribution* P_X is the pushforward measure of P by X. That is, $P_X(E) = P(X^{-1}(E))$ for $E \in \Sigma$.

Given a probability space $\langle \Omega, \mathcal{F}, P \rangle$ and random variable X from $\langle \Omega, \mathcal{F}, P \rangle$ to $\langle \mathscr{X}, \Sigma \rangle$, we write $P|Y = y$ for the probability measure such that $(P|Y = y)(E) = P(E \cap Y^{-1}(\{y\}))/P(Y^{-1}(\{y\}))$. Note that $\langle \Omega, \mathcal{F}, P|Y = y \rangle$ is a probability space with the same random variables as $\langle \Omega, \mathcal{F}, P \rangle$.

Thus, given probability space $\langle \Omega, \mathcal{F}, P \rangle$, random variable Y from $\langle \Omega, \mathcal{F}, P \rangle$ to $\langle \mathscr{Y}, \Sigma_Y \rangle$, and random variable Z from $\langle \Omega, \mathcal{F}, P \rangle$ to $\langle \mathscr{Z}, \Sigma_Z \rangle$, $(P|Y = y)_Z$ is the distribution D such that $D(z) = P(Z^{-1}(\{z\}) \cap Y^{-1}(\{y\}))/P(Y^{-1}(\{y\}))$ for $y \in \mathscr{Y}$ such that $P(Y^{-1}(\{y\})) \neq 0$.