An Abstraction Quantifying Information Flow over Probabilistic Semantics

Chunyan Mu [1] David Clark [2]

Department of Computer Science
King's College London
The Strand, London, UK

Abstract

In a batch program, information about confidential inputs may flow to insecure outputs. The size of this leakage, considered as a Shannon measure, may be automatically and exactly calculated via probabilistic semantics as we have shown in our earlier work. This approach works well for small programs with small state spaces. As the scale increases the calculation suffers from a form of state space explosion and the time complexity grows. In this paper we scale up the programs and state spaces that can be handled albeit at the cost of replacing an exact result with an upper bound. To do this we introduce abstraction on the state space via interval-based partitions, adapting an abstract interpretation framework introduced by Monniaux. The user can define the partitions and the more coarse the partitions, the more coarse the resulting upper bound. In this paper we summarise our previous contribution, define the abstract interpretation, show its soundness, and prove that the result of an abstract computation is always an upper bound on the true leakage, i.e. is a safe estimate. Finally we illustrate the approach by means of some examples.

Keywords: Security, Abstraction, Information Flow, Measurement, Language

1 Introduction

Information-flow security enforces limits on the use of information propagation in programs. The goal of information flow security is to ensure that the information propagates throughout the execution environment without security violations so that no secure information is leaked to public outputs. The traditional theory based reasoning and analysis of software systems are not concerned with security measurement. Quantitative information flow (QIF) proposes to determine *how much* information flows from *confidential* inputs to *public* outputs. It is concerned with measuring the *amount* of information flow caused by interference between variables by using information theoretic quantities. Quantitative analysis therefore relaxes the well known non-interference [14] property by introducing a new policy: the program is secure if the amount of information flow from confidential inputs to public outputs is not too big.

[1] Email: chunyan.mu@kcl.ac.uk
[2] Email: david.j.clark@kcl.ac.uk

In [26], we presented an automatic analyzer for measuring information flow within software systems. We considered the mutual information between a high security variable at the beginning of a batch program and a low one at the end of the program, conditioned on the initial values of low, as the measure of how much of the initial secret information is leaked by executing the program [4]. We incorporated the leakage computation into Kozen's probabilistic semantics to build an automatic analysis tool. This provided a more precise analysis. It took as input a probability distribution on the initial store and calculated a probability distribution on the final store when the program sometimes terminates. In fact it calculates a probability distribution at each program point. These can then be used to calculate the exact leakage. Specifically, while-loops were handled by applying the more general definition of *entropy of generalized distributions* [27] and related properties in order to provide a more precise analysis when incorporating elapsed observed time into the analysis. The drawback of this approach is that it is lacking in abstraction techniques, and time complexity can become large in certain circumstances. One of the time complexity issues is from mutual information computation [26]. Another one is introduced by applying concrete semantics to while loop, which can be improved by applying the abstraction considered in this paper.

We introduce here an approximation method based on measurable partitions and Monniaux's abstract probabilistic semantics [25]. We define a basic abstract domain using interval-based partitions with weights to abstract the measure space. We then define abstract semantic functions which transform the abstract state space in the abstract domain. The definition of leakage upper bound due to such semantics is presented. We introduce *uniformalization* to provide upper bounds on the leakage computation in the framework of information theory. The security property we consider is an a priori bound on leakage, that is, the program is regarded as sufficiently secure if the leakage is less than the a priori bound. Since the actual leakage in the program is required to be less than or equal to the a priori bound, the analysis is safe if we find an upper bound on the leakage [4].

The rest of the paper is organized as follows. Section 2 explains the concrete semantics and leakage computation used in our earlier work. In Section 3, we present a method to approximate our concrete leakage analysis and prove some properties of this. Finally, we present related work and draw conclusions in Section 4 and 5.

2 Leakage Analysis via Probabilistic Semantics

We have developed an automatic leakage analysis system in [26] without abstraction. This previous work is summarized here for the sake of clarity.

2.1 Measure space and programs

Assume that the tuple of program variables $\vec{X}$ range over the state space Ω. The denotational semantics of a command is a mapping from the set X of possible environments before a command into the set X' of possible environments after the command. These spaces updated by semantic transformation functions, can be used to calculate leakage at each program point. Some useful definitions from [30] are reviewed as follows. A *measure space* is a triple $(\Omega, \mathcal{B}, \mu)$, where Ω is a set, $\mathcal{B}$ is a

σ-algebra of subsets of Ω, μ is a nonnegative, countably finite additive set function on $\mathcal{B}$. A *σ-algebra* is a set of subsets of a set X that contains $\emptyset$, and is stable under countable union and complementation. A set X with a *σ-algebra* σ_X defined on it is called a *measurable space* and the elements of the *σ-algebra* are the measurable subsets. If X and X' are measurable spaces, $f : X \to X'$ is a *measurable function* if for all W measurable in X', $f^{-1}(W)$ is measurable in X. A *positive measure* is a function μ defined on a *σ-algebra* σ_X, which is countable additive, *i.e.* if taking $(E_n)_{n\in N}$ a disjoint collection of elements of σ_X, then $\mu(\bigcup_{n=0}^{\infty} E_n) = \sum_{n=0}^{\infty} \mu(E_n)$. A *probability measure* is a positive measure of total weight 1. A *sub-probability measure* has total weight less than or equal to 1.

2.2 The language

The language we considered is standard, presented in Table 1.

$c \in \texttt{Cmd}$ $\quad \texttt{x} \in \texttt{Var}$ $\quad \texttt{e} \in \texttt{Exp}$ $\quad \texttt{b} \in \texttt{BExp}$ $\quad \texttt{n} \in \texttt{Num}$
$c ::= \texttt{skip} \mid \texttt{x} := \texttt{e} \mid \texttt{c}_1; \texttt{c}_2 \mid \texttt{if b then c}_1 \texttt{ else c}_2 \mid \texttt{while b do c}$
$e ::= x \mid n \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \mid e_1 / e_2$
$b ::= \neg b \mid e_1 < e_2 \mid e_1 \leq e_2 \mid e_1 = e_2$

Table 1
The language

2.3 Probabilistic semantics and leakage computation

In [26], we used Kozen's probabilistic semantics for a while language [17] to calculate the probability distribution on the final state from knowledge of the initial state. We then used this to calculate the quantity of leakage from any initial variable to any final variable.

Information theory introduced the definition of *entropy*, $\mathcal{H}$, to measure the average uncertainty in random variables. Shannon's measures were based on a logarithmic measure of the unexpectedness of a probabilistic event (random variable). The unexpectedness of an event which occurred with some non-zero probability p was $\log_2 \frac{1}{p}$. Hence the total information carried by a set of *events* was computed as the weighted sum of their unexpectedness: $\mathcal{H} = \sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i}$. Assume we have two types of input variables: H(confidential) and L(public), and the inputs are equipped with probability distributions, so the inputs can be viewed as a joint random variable (H, L). The semantic function for programs maps the state of inputs to the state of outputs. We present the basic leakage definition due to Clark, Hunt and Malacaria [4] for programs as follows.

Definition 2.1 (Leakage) *Let H be a random variable in high security inputs, L be one in low security inputs, and let L' be a random variable in the output observation, the secure information flow (or interference) is defined by $\mathcal{I}(L'; H|L)$,* i.e. *the conditional mutual information between the output and the high input given knowledge of the low output. Note that for deterministic programs, we have*

$\mathcal{I}(L'; H|L) = \mathcal{H}(L'|L)$, i.e. *interference between the uncertainty in the public output given knowledge of the low input.*

The denotational semantics for measure space transformations are in the following forms:

$$Val \triangleq \langle \Omega, \mathcal{B}, \mu \rangle \qquad \Sigma \triangleq \vec{X} \to Val$$
$$C[\![\cdot]\!] : \texttt{Cmd} \to (\Sigma \to \Sigma) \qquad E[\![\cdot]\!] : \texttt{Exp} \to (\Sigma \to \texttt{Val})$$
$$B[\![\cdot]\!] : \texttt{BExp} \to (\Sigma \to \Sigma)$$

Consider the program as a state-transformation machine, and assume that the tuple of program variables $\vec{X}$ range over the state space Ω. The program variables $\vec{X}$ satisfy some joint distribution μ on program inputs: $\mu(\vec{X})$ assigned to each $\vec{X} \in \Omega$; $\mu(\vec{X}) \geq 0$; $\sum_{\vec{X} \in \Omega} \mu(\vec{X}) = 1$. Let $[\![\cdot]\!]$ denote any measure space transformer given by a semantic function, precisely, $[\![\cdot]\!]$ means $[\![\texttt{CB}]\!]$ where $\texttt{CB} \in \texttt{Cmd} \cup \texttt{BExp}$, *i.e.* $[\![\cdot]\!] : \Sigma \to \Sigma'$. In Kozen's probabilistic semantics, a program transformation function $[\![\cdot]\!]$ maps distributions μ on $\vec{X}$ to distributions $\mu' = [\![\cdot]\!](\mu)$ on $\vec{X}$. Let X ranges over Ω in Σ, X' ranges over Ω in Σ', μ is in Σ, and μ' is in Σ', *i.e.* X and X' denote the measure space over $\vec{X}$ before and after $[\![\cdot]\!]$. Consider the general form of an action of one elementary operator:

- $X \to (X' = [\![\cdot]\!](X))$
- $\mu \to \mu'$
- $\mu'(X') = \sum_{X \in \Omega} \mu(X) \delta_{X',\, [\![\cdot]\!](X)}$ where, $\begin{cases} \delta_{a,b} = 1 \ \textit{iff}\ a = b \\ \delta_{a,b} = 0 \ \textit{iff}\ a \neq b \end{cases}$

We concentrate on the distributions and present the space transformation functions offered by semantic functions by using Lambda Calculus and the notation of inverse function following [25] in Table 2.

$[\![x := e]\!](\mu)$	$\triangleq$	$\lambda W.\mu([\![x := e]\!]^{-1}(W))$
$[\![c_1; c_2]\!](\mu)$	$\triangleq$	$[\![c_2]\!] \circ [\![c_1]\!](\mu)$
$[\![\texttt{if b then c}_1\ \texttt{else c}_2]\!](\mu)$	$\triangleq$	$[\![c_1]\!] \circ [\![b]\!](\mu) \ + \ [\![c_2]\!] \circ [\![\neg b]\!](\mu)$
$[\![\texttt{while b do c}]\!](\mu)$	$\triangleq$	$[\![\neg b]\!](\lim_{n \to \infty}(\lambda \mu'.\mu + [\![c]\!] \circ [\![b]\!](\mu'))^n)(\lambda X.\bot))$
		where, $[\![B]\!](\mu) = \lambda X.\mu(X \cap B)$

Table 2
Probabilistic Denotational Semantics of Programs

Due to the measurability of the semantic functions, for all measurable $W \in X'$, $[\![x := e]\!](W)$ is measurable in X. The function $[\![B]\!]$ for boolean test B defines the set of environments matched by the condition B: $[\![B]\!](\mu) = \lambda X.\mu(X \cap B)$, which causes the space to split apart, $X \cap B$ denotes the part of the space X which makes boolean test B to be true. *Conditional statement* is executed on the conditional probability distributions for either the *true* branch or *false* branch: $[\![c_1]\!] \circ [\![b]\!](\mu) + [\![c_2]\!] \circ [\![\neg b]\!](\mu)$. In the *while loop*, the measure space with distribution μ goes around the loop, and

at each iteration, the part that makes test b *false* breaks off and exits the loop, while the rest of the space goes around again. The output distribution $[\![\texttt{while b do c}]\!](\mu)$ is thus the sum of all the partitions that finally find their way out. Note that these partitions are part of the space when the loop partially terminates, which implies the outputs are partially observable and hence produce an incomplete distribution. The incomplete distribution incorporates the non-termination part (the rest of the space) to be a complete distribution finally. For the case that the loop is always non-terminating, $\perp$ is returned and leakage is 0.

In what follows we present some of concrete semantic operations with leakage analysis.

- **Assignment** *Assignment* updates the state such that the measure space of assigned variable x is updated to become the measure space for expression e. For example, if there is no low input, the secure information leaked to low security variable x after command $[\![x := e]\!]$ is give by $\mathcal{L}_{[\![x:=e]\!]} = \mathcal{H}(\mu_e)$. The secure information contained in e is considered as the entropy of its distribution μ_e.
- **Sequential composition command** The distribution transformation function for the *sequential composition operator* is obtained via functional composition: $[\![c_1; c_2]\!](\mu) = [\![c_2]\!] \circ [\![c_1]\!](\mu)$.
- **Conditionals** Conditional tests cause a partition of the measure space into values that make the boolean b true and those that make it false: let $[\![b]\!]_s$ denote the standard state transformer of boolean test, $\mathcal{P}_0 \subseteq X, \forall x \in \mathcal{P}_0, [\![b]\!]_s x = \texttt{tt}$, *i.e.* k makes test b be true; $\mathcal{P}_1 \subseteq X, \forall x \in \mathcal{P}_1, [\![b]\!]_s x = \texttt{ff}$, *i.e.* k makes test b be false, and $\mathcal{P}_0 \cup \mathcal{P}_1 = X \wedge \mathcal{P}_0 \cap \mathcal{P}_1 = \emptyset$. Let $\mu_b(\texttt{tt}) = \sum_{\texttt{x} \in \mathcal{P}_0} \mu(\texttt{x})$ denote the probability that b is true, and $\mu_b(\texttt{ff}) = \sum_{\texttt{x} \in \mathcal{P}_1} \mu(\texttt{x})$ denote the probability that b is false under the current space. Let $\mathcal{P}_0 = \{p_0 = \mu_b(\texttt{tt})\}$, $\mathcal{P}_1 = \{\texttt{p}_1 = \mu_\texttt{b}(\texttt{ff})\}$ denote the partitions due to the test b, and $\mathcal{Q}_0^l = \{q_{00}, \ldots, q_{0m}\}$, $\mathcal{Q}_1^l = \{q_{10}, \ldots, q_{1n}\}$ denote the normalized distribution of the low security variable l inside the partitions. The semantic function is given by $[\![\texttt{if b then c}_1 \texttt{ else c}_2]\!](\mu) = [\![\texttt{c}_1]\!] \circ [\![\texttt{b}]\!](\mu) + [\![\texttt{c}_2]\!] \circ [\![\neg\texttt{b}]\!](\mu)$. The leakage into l due to the `if` statement can be computed by:

$$\mathcal{L}_{[\![\texttt{if}]\!]} = \widetilde{\mathcal{H}}(\mathcal{P}_0 \cup \mathcal{P}_1) + \widetilde{\mathcal{H}}(\mathcal{Q}_0^l \cup \mathcal{Q}_1^l \mid \mathcal{P}_0 \cup \mathcal{P}_1)$$

where $\widetilde{\mathcal{H}}$ denotes the entropy of generalised distributions [26].

Example 2.2 To show how the method works, let us consider a simple program [26].

```
if (h==0) then l=0 else l=1;
```

Assume h is a 32-bit *high* security variable with possible values $0, \ldots, s^k - 1$ under uniform distribution, l is a *low* security variable. The boolean test splits the original space and we get $\mathcal{P}_0 = \{\frac{1}{2^{32}}\}$, $\mathcal{P}_1 = \{1 - \frac{1}{2^{32}}\}$, and $\mathcal{Q}_0^l = \{\mu_l(0)\} = \{\frac{1}{2^{32}}\}$, $\mathcal{Q}_1^l = \{\mu_l(1)\} = \{1 - \frac{1}{2^{32}}\}$. The resulting set of distribution transformation is obtained as:

$$[\![\texttt{if}]\!] \left(\langle \texttt{h}, \texttt{l} \rangle \mapsto \begin{bmatrix} \langle 0, \perp \rangle & \to 1/2^{32} \\ \ldots & \ldots \\ \langle 2^{32} - 1, \perp \rangle & \to 1/2^{32} \end{bmatrix} \right) = \left(l \mapsto \begin{bmatrix} 0 \to 1/2^{32} \\ 1 \to 1 - 1/2^{32} \end{bmatrix} \right)$$

The distribution of low security variable l after the execution of the program is as follows: $l \mapsto \begin{bmatrix} 0 \to 1/2^{32} \\ 1 \to 1 - 1/2^{32} \end{bmatrix}$, and the information leakage due to this example can be computed by:

$$\begin{aligned}\mathcal{H}_{[\![\texttt{if}]\!]} &= \widetilde{\mathcal{H}}(\mathcal{P}_0 \cup \mathcal{P}_1) + \widetilde{\mathcal{H}}(\mathcal{Q}_0^l \cup \mathcal{Q}_1^l | \mathcal{P}_0 \cup \mathcal{P}_1) \\ &= \widetilde{\mathcal{H}}(\{\frac{1}{2^{32}}\} \cup \{1 - \frac{1}{2^{32}}\}) + 0 = 7.8 \times 10^{-9}\end{aligned}$$

It is clear that this example just releases a little bit information to the public. The leakage analysis result also agrees with our intuition: the possibility of $h = 0$ is quite low and the uncertainty of h under condition $h \neq 0$ is still big, *i.e.* only small information released to the public.

- **While Loop** In the while loop, the distribution goes around the loop, at each round, the part that makes test b *false* breaks off and exits the loop, and the rest goes around again. Command $[\![\texttt{while b do c}]\!]$ can be rewritten as $[\![\texttt{if } \neg\texttt{b then skip else } \{\texttt{c; while b do c}\}]\!]$. The operator for loops can be described in a recursive way, we have:

$$\begin{cases} [\![\texttt{while b do c}]\!]^0(\mu) = \mu \\ [\![\texttt{while b do c}]\!]^{\texttt{n}}(\mu) = [\![\neg\texttt{b}]\!](\mu) + [\![\texttt{c}]\!] \circ [\![\texttt{b}]\!]([\![\texttt{while b do c}]\!]^{\texttt{n}-1}(\mu)) \end{cases}$$

For the always terminating loops, the output distribution is the sum of all the sub-distributions that fail the conditional test on each iteration and find their way out of the loop untill $[\![b]\!](\mu) = \emptyset$. The loop is considered as partially/completely non-terminated when no new partitions are produced but the test is still satisfied with respect to the partial/whole space. Consider a terminating loop as a *sub-measure* transformer which builds a set of accumulated *incomplete* distributions, *i.e.* due to the k^{th} iteration, $\mathcal{P}([\![\texttt{while b do c}]\!]) = \bigcup_{0 \leq \texttt{i} \leq \texttt{k}} \mathcal{P}_{\texttt{i}}([\![\texttt{while b do c}]\!])$, where $k \leq n$, and n is the maximum number of the partitions produced by the terminating loops. Let

$$\mathcal{P}_i = \{p_i\} = \{\mu(e^i)\},\ \textit{where}\ e^i = \begin{cases} b^0 = \texttt{ff}, & i = 0 \\ b^0 = \texttt{tt} \wedge \texttt{b}^1 = \texttt{ff}, & i = 1 \\ b^0 = \texttt{tt}, \ldots, \texttt{b}^{\texttt{i}-1} = \texttt{tt} \wedge \texttt{b}^{\texttt{i}} = \texttt{ff}, & i > 1 \end{cases}$$

where e^i is the event that the loop test b is true until the i^{th} iteration, b^i denotes the value of the boolean test b at the i^{th} iteration. Consider the union of the decompositions

$$\mathcal{P} = (\mathcal{P}_0 \cup \mathcal{P}_1 \cup \ldots \cup \mathcal{P}_k)_{0 \leq k \leq n} = (\{p_0\} \cup \{p_1\} \cup \ldots \cup \{p_k\})_{0 \leq k \leq n}$$

the events $\mathcal{P}_0, \ldots, \mathcal{P}_k$ build the out-going partitions of the states for a while loop. The leakage computation of the loop given by addition of the entropy of the union of the boolean test for each iteration and the sum of the entropy of the loop body for each weighted sub-probability measures by applying the definition of entropy of partitions [29,27]:

$$\mathcal{L}_{\texttt{while}}(k) = \widetilde{\mathcal{H}}(\mathcal{P}_0 \cup \ldots \cup \mathcal{P}_k) + \widetilde{\mathcal{H}}(\mathcal{Q}_0^L \cup \ldots \cup \mathcal{Q}_k^L | \mathcal{P}_0 \cup \ldots \cup \mathcal{P}_k)$$

$$= \frac{\sum_{i=0}^{n}(p_i \log_2 \frac{1}{p_i})}{\sum_{i=0}^{n} p_i} + \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} q_{ij} \log_2 \frac{q_{ij}}{p_i}}{\sum_{i=0}^{n}\sum_{j=0}^{m} q_{ij}}$$

where $\mathcal{P}_i = \{p_i\}$, thus $\widetilde{\mathcal{H}}(\mathcal{P}_i) = \log_2 \frac{1}{p_i}$. The leakage computation formula works for terminating, partially terminating, and non-terminating loops. Further details and examples refer to [26].

3 Abstraction on Information Flow Measurement

To address the problem of tractability when the size and number of variables get large, we present a method to help us approximate our concrete analysis for leakage. Firstly, we define an abstraction on the measure space. The abstraction technique considered is partitioning of the concrete measure space into blocks. Each block is described by intervals and has a coefficient which is the upper bound on measures. Secondly, abstract semantic operations are applied on these partitions. The abstract transformation behavior of the distributions is described by the abstract transition function $[\![\cdot]\!]^\sharp$ on the abstract objects based on Monniaux's [25] abstract probabilistic semantics. To guarantee security, the analysis is safe if we find an upper bound on the leakage. Our analysis is required to over estimate the leakage and never under estimate it. Therefore, we introduce *uniformalization* to estimate the abstract spaces to provide safe bounds on the entropy computation. The leakage computation is obtained by estimating the abstract space to maximise the entropy and thus give safe (*i.e.* upper) bounds on the entropy.

3.1 Measurable Partitions and Abstract Domain

In the first step, we aim to provide an abstraction on the measure space transformation which provides a basis for the leakage computation. The basic principle of abstraction here is to collapse sets of concrete states into single abstract states such that concrete states are simulated by abstract ones. We give our basic abstract domain expressing the above idea formally, and take Monniaux's [25] framework of abstract probabilistic semantics to construct abstract operations on such abstract objects.

Our abstract domain is based on partitions of the measure space. Any collection of non-empty disjoint sets that cover measure space X is said to be a *partition* of X. A *partition* of space X is defined as a family $\xi = \{E_i | i \in I\}$ of nonempty disjoint subsets of X, whose union is X: $\bigcup_{E_i \in \xi} E_i = X$. The sets E_i are the *blocks* of ξ. This can be equivalently considered as a surjection $\phi : X \to X/\xi$, where the points of the space X/ξ correspond to the sets $\{E_i\}$ of the partition. An element of ξ is the inverse image $\phi^{-1}(e)$ of a point $e \in X/\xi$. The quotient *σ-algebra* on X/ξ is defined as the pushforward under ϕ of the *σ-algebra* on X, *i.e.* a subset $M \subset X/\xi$ is measurable if $\phi^{-1}(M) \subset X$ is measurable. ϕ is called a measurable partition function. Suppose we have a measure μ on X and a measurable partition ξ. The quotient measure μ' on X/ξ is defined as the pushforward of μ under the natural projection, *i.e.* for a subset $M \subset X/\xi$, define $\mu'(M) = \mu(\phi^{-1}(M'))$.

Partitions of sets can be viewed as random variables: if X is a finite set and $\xi = \{E_1, \ldots, E_n\}$ is a partition of X, $W(X)$ denotes the weight of X, then $p =$

$(\frac{W(E_1)}{W(X)}, \ldots, \frac{W(E_n)}{W(X)})$ is a discrete probability distribution. The Shannon entropy of p equals to the Shannon entropy of ξ according to Rokhlin [29]. This relation allows the transfer of certain probabilistic and information-theoretical notions to partitions of sets, where we can take advantage of the partial order between partitions.

Definition 3.1 *The partial order relation on $\Xi(X)$ is defined by $\xi \leq \xi'$ for $\xi, \xi' \in \Xi(X)$ if $\mathcal{H}(\xi) \leq \mathcal{H}(\xi')$. It is easy to get that if every block of ξ is included in a block of ξ' then $\xi \geq \xi'$.*

The largest element of this lattice is the partition $\alpha_X = \{\{x\}|x \in X\}$; the least one is the partition $\omega_X = \{X\}$. The set of partitions of X is denoted by $\Xi(X)$. A partial order relation on $(\Xi(X), \leq)$ is actually a bounded lattice.

Definition 3.2 (Abstract domain) *Let $I_v = [a_v, b_v]$, where v in $\vec{X}$. Then let $[E_i] \stackrel{\text{def}}{=} \{I_v | v \text{ in } \vec{X}\}$ be the interval-based partition of X. The abstract domain is defined as: $X^\sharp \stackrel{\text{def}}{=} \{\langle \alpha_i, [E_i] \rangle\}_{0<i\leq n}$, where n is the number of the partitions. A component of the abstract domain $X^\sharp$ is defined as a pair $\langle \alpha_i, [E_i] \rangle$, where α_i denotes the weight on $[E_i]$, $[E_i]$ is a partition on concrete space X and described by a set of intervals on possible values of each variable v in the vector of program variables $\vec{X}$: $\exists$ intervals $\{I_v = [a_v, b_v]_{v \in \vec{X}}\}$ on partition E_i, where $a_v = \texttt{min}\{Val_v\}$, $b_v = \texttt{max}\{Val_v\}$, and Val_v denotes the possible values of v inside E_i.*

Intuitively, a single partition is constructed by choosing an interval on possible values for each variable. The partitions can be arbitrary, but certain human intervention can optimise the analysis. All different strategies of partitioning on the space can produce safe leakage bounds, but with different precision with regard to the resulting upper bound. The abstract space can be viewed as $X\langle \alpha, \gamma \rangle X^\sharp$, where X denotes the concrete space, $X^\sharp$ denotes the abstract space, α denotes the abstraction function, and γ denotes the concretisation function as usual.

Definition 3.3 (Abstraction funtion) *The abstraction function α is a mapping from concrete space X to the sets of interval-based partitions $X^\sharp$: $X \xrightarrow{w} [X/\xi]$ with weights w over the interval-based partitions $[X/\xi]$, where $[X/\xi] = \{\langle \alpha_i, [E_i] \rangle\}_{0<i\leq n}$ (assume n is the number of the partitions).*

The measurability of this collection of partitions follows from their disjointness. However, non-injective semantics $[\![\cdot]\!]$ can produce non-disjoint sets in some cases. For the case of non-disjoint partitions created by semantic function $[\![\cdot]\!] : X \to X$, we extend the space to eliminate the collisions. The method we considered is putting an index i on the overlapped part to distinguish them, where i is from the index of the partition E_i in which it is located. Since the initial partitions $\{E_i | i \in I\}$ before semantic functions are disjoint, there exist measures μ_i such that $\sum \mu_i = \mu(\bigcup E_i)$, and for all i, μ_i is concentrate on E_i and $\mu_i \leq \alpha_i$. It is clear that such scenario will never underestimate the leakage computation due to the definition of leakage upper bound in Definition 3.12 (see Proposition 3.15).

The concretisation of such abstract objects is thus the set of all measures matching the above conditions. Consider a sub-partition η on each element of the final abstract space $X^\sharp$ to concretize it.

Definition 3.4 (Concretisation function) *The concretisation function γ is thus*

a mapping: $X^\sharp \to \bigcup\{x | x \in [E_i/\eta]\}$*, where* E_i *is the blocks of the abstract object* $X^\sharp$*,* η *is a sub-partition on each block under uniform distribution.*

The sub-partition η will be further discussed in section 3.4 for the purpose of safe leakage computation.

Proposition 3.5 $X\langle\alpha,\gamma\rangle X^\sharp$ *is a Galois connection.*

Proof. For complete lattices X and $X^\sharp$, to prove the pair $\langle\alpha,\gamma\rangle$ form a Galois connection, we need to prove that for all $C \in X$ and $A \in X^\sharp$, $C \sqsubseteq_X \gamma\circ\alpha(C)$ and $\alpha\circ\gamma(A) \sqsubseteq_{X^\sharp} A$.

The concrete space X with measures $(q_1,\ldots,q_m)$ can be viewed as a composition of two partitions denoted by: $\xi\eta_0$, where $\xi = \{E_1,\ldots,E_n\}$ with measures $\mu_1,\ldots,\mu_n$ is the same partition as the abstraction $X^\sharp$ given by α. Sub-partition η_0 is the partition on ξ which recovers the abstract space $X^\sharp$ to be the original concrete space, *i.e.* assume N_k is the number of elements of E_k (where $1 \le k \le n$ and therefore $m = \sum_{k=1}^{n} N_k$), according to the definition of entropy of partitions [29,27], for all $C \in X$, we have

$$(1)\quad \mathcal{H}(C) = \mathcal{H}(\xi\eta_0) = \mathcal{H}(\xi) + \mathcal{H}(\eta_0|\xi) = \mathcal{H}(\mu_1,\ldots,\mu_n) + \sum_{i=1}^{n} \mu_i \mathcal{H}(\frac{p_{i1}}{\mu_i},\ldots,\frac{p_{iN_i}}{\mu_i})$$

where $\frac{p_{11}}{\mu_1} = q_1,\ \ldots,\ \frac{p_{nN_n}}{\mu_n} = q_m$ is the measures over X, and $m = \sum_{k=1}^{n} N_k$.

The abstract space $X^\sharp$ can be viewed as a partition ξ over X given by abstraction function α. The concretisation on X^{sharp} is given by sub-partition η (uniformalization) on each block. Therefore $\gamma\circ\alpha$ can be considered as the composition of partitions $\xi\eta$, *i.e.* for all $A \in X^\sharp$, we have

$$(2)\quad \mathcal{H}(\gamma(\alpha(A))) = \mathcal{H}(\xi\eta) = \mathcal{H}(\xi) + \mathcal{H}(\eta|\xi) = \mathcal{H}(\mu_1,\ldots,\mu_n) + \sum_{i=1}^{n} \mu_i \mathcal{H}(\frac{1}{N_i},\ldots,\frac{1}{N_i})$$

since uniform distribution maximise the entropy, we have

$$(3)\qquad H(\frac{1}{N_i},\ldots,\frac{1}{N_i}) \ge \mathcal{H}(\frac{p_{i1}}{\mu_i},\ldots,\frac{p_{iN_i}}{\mu_i}) \qquad forall\, 0 < i \le n$$

By (1)(2)(3) it is easy to get that $\mathcal{H}(C) \le \mathcal{H}(\gamma(\alpha(C)))$, and similarly $\mathcal{H}(\alpha(\gamma(A))) \le \mathcal{H}(A)$. Therefore, $C \sqsubseteq_X \gamma\circ\alpha(C)$ and $\alpha\circ\gamma(A) \sqsubseteq_{X^\sharp} A$. □

3.2 Abstract Transformation

We have presented an abstract domain to approximate the concrete measure spaces. We need abstract semantic functions to produce the transformations of such abstract spaces. The leakage upper bound U_v on variable v will be discussed in Section 3.4. We just concentrate on the abstract space transformation here. Assume $[\![\cdot]\!]^\sharp$ is the abstract semantic function, which drives the abstract spaces to transform. For the purpose of leakage computation, the abstract transformations are required to keep the measurability and properties of entropy computation. We concentrate on the abstract semantic functions over measurable partitions. It is easy to extend to abstract transformation over interval-based measurable partitions defined in our abstract domain.

Proposition 3.6 *If $\phi : X \to X/\xi$ is a measurable partition function on the measurable space X and $[\![\cdot]\!]$ is the measurable space transformation function on X given by semantic functions, then $[\![\cdot]\!]^\sharp$ (the abstract analogue of $[\![\cdot]\!]$) is a measurable function on $X^\sharp = \phi(X)$.*

Proof. To prove $[\![\cdot]\!]^\sharp$ is measurable, we need to prove that for all $M \subset X/\xi$, $[\![\cdot]\!]^{\sharp-1}(M)$ is measurable.

$$[\![\cdot]\!]^{\sharp-1}(M) = \{x|\phi([\![\cdot]\!](x)) \in M\} = \{x|[\![\cdot]\!](x) \in \phi^{-1}(M)\}$$

i.e. $[\![\cdot]\!]^{\sharp-1}(M)$ is the set of all those elements of X which are mapped into M by $[\![\cdot]\!]$.

Since ϕ is measurable, $\phi^{-1}(M)$ is a subset of elements of X, the measurability of the set

$$[\![\cdot]\!]^{\sharp-1}(M) = \{x|[\![\cdot]\!](x) \in \phi^{-1}(M)\} = [\![\cdot]\!]^{-1}(\phi^{-1}(M))$$

follows from the measurability of $[\![\cdot]\!]$.

□

3.2.1 Arithmetic Expressions

In concrete analysis, we denote the *probability function* of variable v as μ_v. Let $\mu_v(c)$ be the probability that the value of v is c, the domain of μ_v will be the set of all possible values that v could be. The computation function $e(\mu)$ of an arithmetic expression e defines the measure space that the arithmetic expression can evaluate to in a given set of environments. Specifically, consider the joint distribution over the discrete random variable X, Y, by the definition of conditional probability, the distribution function for arithmetic expression $e(x, y)$ is given by:

$$\mu_{\mathtt{e(x,y)}}(z) = \sum_{\mathtt{domain_y}} \mu_{\mathtt{x}}(e^{-1}(z, y))\mu_{\mathtt{y}}(y)$$

where z stands for a possible value of $e(x, y)$ in the current environment. For instance, the concrete probability density function for *addition* is: $\mu_{x+y}(z) = \sum_{\mathtt{domain_y}} \mu_{\mathtt{x}}(z - y)\mu_{\mathtt{y}}(y) = \sum_{\mathtt{domain_x}} \mu_{\mathtt{y}}(z - x)\mu_{\mathtt{x}}(x)$.

Now consider the abstract arithmetic operation $e(x, y)^\sharp = z^\sharp$. We define the abstract arithmetic operation $\langle \alpha_i, \{[a_v, b_v]_{v\in\vec{X}}\}_i\rangle \to \langle \alpha'_i, \{[a'_v, b'_v]_{v\in\vec{X}}\}_i\rangle$ as: $z^\sharp = e(x, y)^\sharp(\langle \alpha_i, \{[a_v, b_v]_{v\in\vec{X}}\}_i\rangle)$, and for all $i \in N$, we take,

- $\alpha'_i = \alpha_i$
- $$\begin{cases} \textit{if } v = z \quad a'_v = \mathtt{min}\{\mathtt{e(z_x, z_y)}|\mathtt{a_x} \le \mathtt{z_x} \le \mathtt{b_x}, \mathtt{a_y} \le \mathtt{z_y} \le \mathtt{b_y}\} \\ \qquad\qquad\; b'_v = \mathtt{max}\{\mathtt{e(z_x, z_y)}|\mathtt{a_x} \le \mathtt{z_x} \le \mathtt{b_x}, \mathtt{a_y} \le \mathtt{z_y} \le \mathtt{b_y}\} \\ \textit{otherwise } a'_v = a_v,\ b'_v = b_v \end{cases}$$

The arithmetic expression returns the probable values on the abstract lattice with the current context: the weight of each abstract element is unchanged, *i.e.* $\alpha'_i = \alpha_i$; the interval of variable v is updated by the intervals on probable values of the expression e if v suppose to be the assigned variable.

Example 3.7 Consider expression $[\![x+2y]\!]$, assume the initial distribution for $\langle x, y\rangle$

is $\mu_{\langle x,y\rangle}$ (the distribution function on a set M is viewed as a mapping: $M \to [0,1]$):

$$\mu_{\langle x,y\rangle} \mapsto \left(\begin{array}{llll} \langle 0,0\rangle \to 0.1, & \langle 1,1\rangle \to 0.1, & \langle 2,1\rangle \to 0.1 & E_1 \\ - \\ \langle 3,2\rangle \to 0.2, & \langle 4,2\rangle \to 0.2, & \langle 5,2\rangle \to 0.1 & \\ \langle 6,3\rangle \to 0.1, & \langle 7,3\rangle \to 0.1 & & E_2 \end{array}\right)$$

- We take the partition: $\begin{cases} E_1\langle [0,2]_x, [0,1]_y\rangle : \alpha_1 = 0.3 \\ E_2\langle [3,7]_x, [2,3]_y\rangle : \alpha_2 = 0.7 \end{cases}$
- Apply the arithmetic expression $\mu_{[\![x+2y]\!]}$, we get: $\begin{cases} E_1\langle [0,4]_{[\![x+2y]\!]}\rangle : \alpha_1 = 0.3 \\ E_2\langle [5,10]_{[\![x+2y]\!]}\rangle : \alpha_2 = 0.7 \end{cases}$

3.2.2 Assignment

Assignment operation $[\![x := e]\!]^\sharp$ updates the state such that the abstract domain of assigned variable x is mapped to the domain of expression e. The upper bounds on the weight of partitions remain unchanged: $\alpha_i' = \alpha_i$, and the interval of assigned variable x is updated by the interval of the expression e: $a_x = \texttt{min}\{\mathit{Val}_e\}$, $b_x = \texttt{max}\{\mathit{Val}_e\}$.

3.2.3 Sequential command

The sequential operator can be obtained by composition: $[\![c_1; c_2]\!]^\sharp(X^\sharp) = [\![c_2]\!]^\sharp \circ [\![c_1]\!]^\sharp(X^\sharp)$. Assume $X^{\sharp\prime} = [\![c_1]\!](X^\sharp)$, and $X^{\sharp\prime\prime} = [\![c_1]\!](X^{\sharp\prime})$, then $X^{\sharp\prime\prime}$ is the new state space by executing the sequential command over $X^\sharp$: $X^\sharp \xrightarrow{[\![c_1;c_2]\!]^\sharp} X^{\sharp\prime\prime}$.

3.2.4 Conditional

The tests split the abstract space into two parts according to the result of boolean test over the space. The weight on each part is the probability of the part of the space that makes the boolean test be true or false. The if statement returns the sum of the two parts by executing the statements under true and false branches, therefore the intervals on variables are updated but the weight on each abstract block remains unchanged. Let $+^\sharp$ be an abstraction of the sum operation on measures, the abstract semantic function of tests are given by:

$$[\![\texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2]\!]^\sharp(X^\sharp) \triangleq [\![c_1]\!]^\sharp \circ [\![b]\!]^\sharp(X^\sharp) \ +^\sharp \ [\![c_2]\!]^\sharp \circ [\![\neg b]\!]^\sharp(X^\sharp)$$

We define the abstract operation $(X^\sharp, X^{T\sharp}, X^{F\sharp}) \to X'^\sharp$, where $X^{T\sharp}$ denotes the abstract object under *true* branch: $X^{T\sharp}\langle \alpha_i^T, \{[a_v^T, b_v^T]_{v\in\vec{X}}\}_i\rangle = [\![c_1]\!]^\sharp \circ [\![b]\!]^\sharp(X^\sharp)$, and $X^{F\sharp}$ denotes the abstract object under *false* branch: $X^{F\sharp}\langle \alpha_i^F, \{[a_v^F, b_v^F]_{v\in\vec{X}}\}_i\rangle = [\![c_2]\!]^\sharp \circ [\![\neg b]\!]^\sharp(X^\sharp)$. We have, $\forall v \in \vec{X}$:

$$(\langle \alpha_i, \{[a_v, b_v]\}_i\rangle, \langle \alpha_i^T, \{[a_v^T, b_v^T]\}_i\rangle, \langle \alpha_i^F, \{[a_v^F, b_v^F]\}_i\rangle) \to \langle \alpha_i', \{[a_v', b_v']\}_i\rangle$$

for all $v \in \vec{X}$, $i \in N$, we take:

- $\alpha_i' = \alpha_i^T + \alpha_i^F = \alpha_i$;

- $a'_v = \texttt{min}(a_v^T, a_v^F)$, and $b'_v = \texttt{max}(b_v^T, b_v^F)$ on each interval-based partition $[E_i]$.

Example 3.8 Consider program

```
if(x==0) then y:=0 else y:=1;
```

Assume x is a 3-bit high security variable with distribution

$$\begin{pmatrix} 0 \to 0.1 & 1 \to 0.1 & 2 \to 0.1 & 3 \to 0.1 \\ 4 \to 0.2 & 5 \to 0.2 & 6 \to 0.1 & 7 \to 0.1 \end{pmatrix}$$

y is a 3-bit low security variable under any distribution. Let us take partition as: $\begin{cases} E_1\langle [0,3]_x, [0,7]_y\rangle : \alpha_1 = 0.4 \\ E_2\langle [4,7]_x, [0,7]_y\rangle : \alpha_2 = 0.6 \end{cases}$. After applying the abstract operation for *if statement*, we have: $\begin{cases} E'_1\langle [0,3]_x, [0,1]_y\rangle : \alpha'_1 = 0.4 \\ E'_2\langle [4,7]_x, [1,1]_y\rangle : \alpha'_2 = 0.6 \end{cases}$.

3.2.5 Loops

The concrete semantics of loops are given as infinite sums of all the partitions which have found their way out in a recursive way (see Section 2.3). In order to consider the approximation of the fixpoint of some monotone operators on the concrete lattices and study the abstract operations, we rewrite the semantic function of loops as $[\![\texttt{while b do c}]\!](\texttt{X}) = [\![\neg\texttt{b}]\!](\lim_{\texttt{n}\to\infty} \texttt{X}_\texttt{n})$ following [25], where X_n is defined recursively: $X_0 = \lambda X.0$, and $X_{n+1} = X_n + [\![c]\!] \circ [\![b]\!](X_n)$. We shall deal with it using fixpoints on the abstract lattice $(X^\sharp, \sqsubseteq)$. Let us take $X^\sharp$ as the abstraction of measure space X, assume $\texttt{lfp}^\sharp : (\texttt{X}^\sharp \overset{\texttt{monotone}}{\longrightarrow} \texttt{X}^\sharp) \to \texttt{X}^\sharp$ is an approximation of the least fixpoint, *i.e.* if $[\![\cdot]\!]^\sharp : X^\sharp \to X^\sharp$ is monotonic then we have the approximation relation $[\![\cdot]\!]^\sharp(\texttt{lfp}^\sharp(\texttt{f})) \sqsubseteq \texttt{lfp}^\sharp(\texttt{f})$. Let $+^\sharp$ be an abstraction of the sum operation on measures, $\bot^\sharp$ be an abstraction of the null measure, the abstract function is given by: $[\![\texttt{while b do c}]\!]^\sharp(\texttt{X}^\sharp) \triangleq [\![\neg\texttt{b}]\!]^\sharp(\texttt{lfp}^\sharp\ \chi^\sharp \mapsto \texttt{X}^\sharp \sqcup [\![\texttt{c}]\!]^\sharp([\![\texttt{b}]\!]^\sharp(\chi^\sharp)))$. We define the abstract operation: $(\langle\alpha_i, \{[a_v, b_v]_{v\in\vec{X}}\}_i\rangle, \langle\alpha'_i, \{[a'_v, b'_v]_{v\in\vec{X}}\}_i\rangle) \to \langle\alpha''_i, \{[a''_v, b''_v]_{v\in\vec{X}}\}_i\rangle$ as the least upper bound: $\langle\alpha_i, \{[a_v, b_v]_{v\in\vec{X}}\}_i\rangle \sqcup^\sharp \langle\alpha'_i, \{[a'_v, b'_v]_{v\in\vec{X}}\}_i\rangle$, and $\forall i \in N$, we take:

- $\alpha''_i = \texttt{max}(\alpha_\texttt{i}, \alpha'_\texttt{i})$
- $a''_v = \texttt{min}(\texttt{a}_\texttt{v}, \texttt{a}'_\texttt{v})$, and $b''_v = \texttt{max}(\texttt{b}_\texttt{v}, \texttt{b}'_\texttt{v})$, for all $v \in \vec{X}$ on $[E_i]$

Furthermore, to figure out the approximation of least fixpoint operation $\texttt{lfp}^\sharp$, we need an operation to be applied using widening operators ∇ to guarantee termination of the increasing sequences, and the feasibility of the fixed point computation. By widening, the abstract function $(X, X') \to X''$, *i.e.* $(\langle\alpha_i, \{[a_v, b_v]_{v\in\vec{X}}\}_i\rangle, \langle\alpha'_i, \{[a'_v, b'_v]_{v\in\vec{X}}\}_i\rangle) \to \langle\alpha''_i, \{[a''_v, b''_v]_{v\in\vec{X}}\}_i\rangle$ can be noted as $\langle\alpha_i, \{[a_v, b_v]_{v\in\vec{X}}\}_i\rangle \nabla \langle\alpha'_i, \{[a'_v, b'_v]_{v\in\vec{X}}\}\rangle$ as standard. By induction, the upward iteration sequence is defined as $X^{\sharp\prime}_{n+1} = X^{\sharp\prime}_n \nabla X^\sharp_n$ as usual, which is required to be eventually stationary for every ascending chain elements and limited by a sound upper approximation of $\texttt{lfp}^\sharp$ for any sequence $(X^\sharp_n)_{n\in N}$. Since our abstract domain is based on partitions of the measure space with coefficients of intervals of variable

values, *i.e.* interval-based approximation, we choose the ordinary widening operator for intervals on the integers ∇_I [8]: $x \nabla_I y = \infty$, if $x < y$, otherwise $x \nabla_I y = y$. The intuition behind applying this widening operator is that if a sequence of coefficients I_v keeps ascending, an upper approximation of it is obtained by using $+\infty$. For all $i \in N$, we take:

- $\alpha_i'' = \mathtt{max}(\alpha_\mathtt{i}, \alpha_\mathtt{i}')$
- Intervals $I_v = [a_v, b_v]_{v \in \vec{X}}$ on $[E_i]$ are given as standard:

$$\begin{aligned} \bot \nabla_I [a_v, b_v] &= [a_v, b_v] \\ [a_v, b_v] \nabla_I \bot &= [a_v, b_v] \\ [a_v, b_v] \nabla_I [a_v', b_v'] &= \textit{if } a_v' < a_v \textit{ then } -\infty \textit{ else } a_v \\ &\quad \textit{if } b_v < b_v' \textit{ then } +\infty \textit{ else } b_v \end{aligned}$$

Weight on each abstract elements is the maximum one between before and after the current iteration. By applying the widening operation of intervals ∇_I, we enforce the convergence and induce termination for intervals on each abstract blocks.

3.3 Soundness of Abstract Functions

It is clear that, for a Galois connection, $X \langle \alpha, \gamma \rangle X^\sharp$, and semantic functions $[\![\cdot]\!] : X \to X$, $[\![\cdot]\!]^\sharp : X^\sharp \to X^\sharp$, $X^\sharp$ is a sound approximation of X *iff*

$$(\alpha \circ X^\sharp)(S) \sqsubseteq_{X^\sharp} (X^\sharp \circ \alpha)(S), \textit{ for all } S \in X$$

iff

$$([\![\cdot]\!] \circ \gamma)(A) \sqsubseteq_X (\gamma \circ [\![\cdot]\!]^\sharp)(A), \textit{ for all } A \in X^\sharp$$

Furthermore, given the Galois connection, $X \langle \alpha, \gamma \rangle X^\sharp$, and operation $[\![\cdot]\!] : X \to X$, the most precise $[\![\cdot]\!]^\sharp_{best} : X^\sharp \to X^\sharp$, which is sound with respect to $[\![\cdot]\!]$ is $[\![\cdot]\!]^\sharp_{best} = \alpha \circ [\![\cdot]\!] \circ \gamma$.

Lemma 3.9 (Soundness of composed functions) *Assume function $f_1^\sharp$ is a sound abstraction of function f_1, and $f_2^\sharp$ is a sound abstraction of f_2, then the abstract composed function $f_1^\sharp \circ f_2^\sharp$ is sound with respect to concrete composed function $f_1 \circ f_2$.*

$$\alpha \circ f_1 \sqsubseteq f_1^\sharp \circ \alpha \ \wedge \ \alpha \circ f_2 \sqsubseteq f_2^\sharp \circ \alpha \Longrightarrow \alpha \circ (f_2 \circ f_1) \sqsubseteq (f_2^\sharp \circ f_1^\sharp) \circ \alpha$$

Proof.

$$\begin{aligned} & f_1^\sharp \circ f_2^\sharp \circ \alpha \sqsupseteq (\alpha \circ f_1^* \circ \gamma) \circ (\alpha \circ f_2^* \circ \gamma) \circ \alpha \quad (\textit{because } \alpha \circ f^* \circ \gamma \sqsubseteq f^\sharp) \\ & = \alpha \circ f_1^* \circ (\gamma \circ \alpha) \circ f_2^* \circ (\gamma \circ \alpha) \sqsupseteq \alpha \circ f_1^* \circ f_2^* \ (\textit{because } \gamma \circ \alpha \sqsubseteq Id_c) \end{aligned}$$

□

Theorem 3.10 (Soundness of abstract functions) *$[\![\cdot]\!]^\sharp$ is sound with respect to concrete semantic functions $[\![\cdot]\!]$.*

Proof. To show the soundness of the abstraction is equivalent to prove that $[\![\cdot]\!]^\sharp$ is sound with respect to $[\![\cdot]\!]$ *iff*

$$[\![\cdot]\!]^\sharp_{best}(A) = \alpha \circ [\![\cdot]\!] \circ \gamma(A) \sqsubseteq_{X^\sharp} [\![\cdot]\!]^\sharp(A), \textit{for all } A \in X^\sharp$$

- **Assignment:** $[\![x := e]\!]^\sharp$ is sound with respect to $[\![x := e]\!]$ iff $[\![x := e]\!]^\sharp_{best} = \alpha \circ [\![x := e]\!] \circ \gamma \sqsubseteq_{X^\sharp \to X^\sharp} [\![x := e]\!]^\sharp$. Assume we concentrate on interested variable v with interval $[a_i, b_i]$, let $\mathcal{M}^\sharp_e = e^\sharp(X^\sharp) = \{\langle \alpha_i, [a_i, b_i]\rangle\}_{i\in N}$, we have $\gamma(\mathcal{M}^\sharp_e) = \bigcup_{i\in N}\{\alpha_i/N_i, s_i | a_i \le s_i \le b_i, N_i = b_i - a_i + 1\}$, then we have:

$$\begin{aligned}
&[\![x := e]\!]^\sharp_{best}(x, \mathcal{M}^\sharp_e)\ (X^\sharp)\\
=&\ \alpha \circ [\![x := e]\!](x, \gamma(\mathcal{M}^\sharp_e))\ (X^\sharp)\\
=&\ \alpha([\![x := e]\!](x, \bigcup_{i\in N}\{\langle \alpha_i/N_i, s_i\rangle | a_i \le s_i \le b_i, N_i = b_i - a_i + 1\})\ (X^\sharp)\\
=&\ X^\sharp(\bigcup_{i\in N}\{\langle \alpha_i, E_i(x \mapsto [a_i, b_i])\rangle\})\\
\sqsubseteq&\ [\![x := e]\!]^\sharp(x, \mathcal{M}^\sharp_e)\ (X^\sharp) \quad (\textit{by Proposition 3.5})
\end{aligned}$$

- **Compositional operator** The *soundness proof* of sequential abstract function is straightforward by Lemma 3.9.
- **Conditional** To prove soundness of the abstract function for *if statement*, we need to show that $\alpha \circ [\![\texttt{if b then c}_1 \texttt{ else c}_2]\!] \sqsubseteq [\![\texttt{if b then c}_1 \texttt{ else c}_2]\!]^\sharp \circ \alpha$, i.e., for all $S \in X$:

$$\alpha([\![\texttt{if b then c}_1 \texttt{ else c}_2]\!)\texttt{S} \sqsubseteq [\![\texttt{if b then c}_1 \texttt{ else c}_2]\!]^\sharp(\alpha(\texttt{S}))$$

  By Proposition 3.5, it is clear that,

$$\alpha([\![b]\!])S = \bigcup_{i\in N}\{\langle \alpha^T_i, \{[a^T_v, b^T_v]_{v\in\vec{X}}\}\rangle\} \sqsubseteq [\![b]\!]^\sharp(\alpha(S)) \circ \alpha$$

$$\alpha([\![\neg b]\!])S = \bigcup_{i\in N}\{\langle \alpha^F_i, \{[a^F_v, b^F_v]_{v\in\vec{X}}\}\rangle\} \sqsubseteq [\![\neg b]\!]^\sharp(\alpha(S)) \circ \alpha$$

  By Lemma 3.9 and IH, it is easy to get that,

$$[\![c_1]\!] \circ [\![b]\!] + [\![c_2]\!] \circ [\![\neg b]\!] \sqsubseteq [\![c_1]\!]^\sharp \circ [\![b]\!]^\sharp + [\![c_2]\!]^\sharp \circ [\![\neg b]\!]^\sharp$$

- **Loop** To prove soundness of $[\![\texttt{while b do c}]\!]^\sharp$ with respect to $[\![\texttt{while b do c}]\!]$, we need to show $\alpha \circ [\![\texttt{while b do c}]\!] \sqsubseteq [\![\texttt{while b do c}]\!]^\sharp \circ \alpha$, *i.e.* for all $S \in X$

$$\alpha([\![\texttt{while b do c}]\!](\texttt{S})) \sqsubseteq [\![\texttt{while b do c}]\!]^\sharp(\alpha(\texttt{S}))$$

  Let $F = \lambda\chi.X \cup [\![c]\!]([\![b]\!]\chi)$, and $F^\sharp = \lambda\chi.X^\sharp \sqcup [\![c]\!]^\sharp([\![b]\!]^\sharp\chi)$ and thus

$$\texttt{lfp F} = \texttt{lfp}\ \lim_{\texttt{n}\to\infty}(\lambda\chi.\texttt{X} \cup [\![\texttt{c}]\!]([\![\texttt{b}]\!]\chi))^{\texttt{n}}$$

$$\texttt{lfp F}^\sharp = \texttt{lfp}\ \lim_{\texttt{n}\to\infty}(\lambda\chi.\texttt{X}^\sharp \sqcup [\![\texttt{c}]\!]^\sharp([\![\texttt{b}]\!]^\sharp\chi))^{\texttt{n}}$$

  Starting with $\bot$, we apply the function F over and over again to build up a sequence of partial functions F_0, F_1, ..., F_n, similar to $F^\sharp$. By IH, we have:

(i) The base case is obvious when the function $F^\sharp$ is applied 0 time:

$$\alpha \circ \emptyset \sqsubseteq \bot \circ \alpha \implies \alpha \circ (\lambda\chi.X \cup [\![c]\!]([\![b]\!]\chi))^0 \sqsubseteq (\lambda\chi.X^\sharp \sqcup [\![c]\!]^\sharp([\![b]\!]^\sharp\chi))^0 \circ \alpha$$

(ii) IH: Assume $\forall\ n < k,\ \alpha \circ F_n \sqsubseteq F^\sharp_n \circ \alpha$
  then, $\alpha \circ F_{k-1} \sqsubseteq F^\sharp_{k-1} \circ \alpha$ by IH $(k-1 < k)$
  and $\alpha \circ F_1 \sqsubseteq F^\sharp_1 \circ \alpha$ by IH $(1 < k)$
  by Lemma 3.9 and definition of F_n and $F^\sharp_n$, we have

$$\alpha \circ (F_{k-1} \circ F_1) \sqsubseteq (F^\sharp_{k-1} \circ F^\sharp_1) \circ \alpha \quad \textit{iff} \quad \alpha \circ F_k \sqsubseteq F^\sharp_k \circ \alpha$$

Therefore, soundness of abstract function $F^\sharp$ is obtained as:

$$\alpha \circ (\lambda\chi.X \cup [\![c]\!]([\![b]\!]\chi))^{n+1} \sqsubseteq (\lambda\chi.X^\sharp \sqcup [\![c]\!]^\sharp([\![b]\!]^\sharp\chi))^{n+1} \circ \alpha$$

By Lemma 3.9, we have

$$\alpha \circ [\![\neg b]\!](\texttt{lfp}\ \lambda\chi.\mathtt{X} \cup [\![\mathtt{c}]\!]([\![\mathtt{b}]\!]\chi)) \sqsubseteq [\![\neg \mathtt{b}]\!]^\sharp(\texttt{lfp}\ \lambda\chi.\mathtt{X}^\sharp \sqcup [\![\mathtt{c}]\!]^\sharp([\![\mathtt{b}]\!]^\sharp\chi)) \circ \alpha$$

i.e. $\alpha([\![\texttt{while b c}]\!](\mathtt{S})) \sqsubseteq [\![\texttt{while b c}]\!]^\sharp(\alpha(\mathtt{S}))$. This completes the proof of the soundness of abstract semantic function for loops.

□

3.4 Entropy of Measurable Partition and Leakage Computation

Sections 3.1 and 3.2 have described an abstraction on the measure space and the abstract semantic operations. In order to measure the information flow, we also need to consider an approximation on the leakage computation. Let ξ be our measurable partition of space X and $E_1, E_2, \ldots, E_n$ be elements of ξ of positive measure (where n is the number of partitions), and μ_i such that $\sum_{i=1}^n \mu_i = \mu(\bigcup E_i)$ and μ_i is concentrated on E_i. The entropy of ξ is given by Rokhlin [29]: if $\mu(X - \bigcup E_i) = 0$, $\mathcal{H}(\xi) = -\sum_{i=1}^n \mu_i \log_2 \mu_i$; if $\mu(X - \bigcup E_i) > 0$, $\mathcal{H}(\xi) = +\infty$. $\mathcal{H}(\xi)$ is called the entropy of ξ.

We have introduced that if $\xi, \xi' \in \Xi(X)$ and $\xi \leq \xi'$, then $\mathcal{H}(\xi) \leq \mathcal{H}(\xi')$. However, the partition entropy may underestimate the leakage computation. To get a conservative leakage analysis, we need a re-approximation on the final abstract space aimed to provide a safe (upper) bound on the leakage computation at the end. The basic method here is that we do a *subpartition* η on each element of the final abstract objects obtained by the abstract operations to maximise entropy. The subpartition we considered is performing a *uniformalization* over each element of the final partition $\xi' = [\![\cdot]\!]\xi$ according to the fact that uniform distribution maximises entropy. We call a block *uniform* if all its states inside have the same probability, *i.e.* all possible values of the distribution within that block are uniform.

Definition 3.11 (Uniformalization) *Uniformalization is defined as a transformation of each block of space of a variable into a space with uniform distribution on each block,* i.e. *each partition is sub-partitioned under uniform distribution due to the values of an interested (low) variable.*

Based on the definition of uniformalization on measurable partitions, next consider the leakage upper bound U_v due to our abstract domain. Let us consider a subpartition η under uniform distribution on the final partition ξ' in which any element $E'_i \in \xi'$ comprises of N_i sub-elements with equal probability (where $[\![\cdot]\!]\xi = \xi'$, and ξ is the initial partition at the begin of the program), and let us attach to this partition the finite discrete uniform probability distribution whose corresponding entropy will be $\log_2(N_i)$, where N_i denotes the size of the final abstract element E'_i due to the low security variable considered. We consider the entropy of the composition of partitions ξ' and η denoted by $\mathcal{H}(\xi'\eta)$, which is defined as the sum of the well-defined entropy $\mathcal{H}(\xi')$ and the mean conditional entropy of η with respect to ξ': $\mathcal{H}(\eta|\xi')$ [29], *i.e.*

$$\mathcal{H}(\xi'\eta) = \mathcal{H}(\xi') + \mathcal{H}(\eta|\xi')$$

The conditional entropy $\mathcal{H}(\eta|\xi')$ is a non-negative measurable function on the factor space X/ξ, which is defined by [29]: $\mathcal{H}(\eta|\xi') = \sum_{i=1}^{n} \mu_i \mathcal{H}(\eta)$, where μ_i denotes the measure of each partition given by ξ'. The leakage upper bound is therefore defined as the entropy on the compositional partitions $\xi'\eta$. It is clear that the entropy of a measurable sub-partition η on partition of ξ' into N_i sets is less than or equal to $\log_2(N_i)$ if and only if every element of the sub-partition has measure $1/N_i$. This meets the intuition of our method: computation time is saved by doing abstract transformations on abstract partitions ξ of the measure space, the safety of the leakage computation is guaranteed by considering mean conditional entropy of uniform partition η with respect to the final abstract partition ξ' returned by the execution of the program.

Definition 3.12 (Leakage upper bound) *The leakage upper bound is given by the entropy of paritition $\xi'\eta$, according to the definition of entropy on partitions and definitions of ξ' and η, we have:*

$$\begin{aligned} U_v &= \mathcal{H}(\xi'\eta) = \mathcal{H}(\xi') + \mathcal{H}(\eta|\xi') \\ &= \mathcal{H}(\mu_1, \ldots, \mu_n) + \sum_{i=1}^{n} \mu_i \mathcal{H}(\frac{\mu_i/N_i}{\mu_i}, \ldots, \frac{\mu_i/N_i}{\mu_i}) \\ &= \mathcal{H}(\mu_1, \ldots, \mu_n) + \sum_{i=1}^{n} \mu_i \mathcal{H}(\frac{1}{N_i}, \ldots, \frac{1}{N_i}) \\ &= \mathcal{H}(\mu_1, \ldots, \mu_n) + \sum_{i=1}^{n} \mu_i \log_2(N_i) \end{aligned}$$

where N_i is the size of the partition E_i, i.e. *$N_i = (b_v - a_v + 1)$ for partition $E_i \in \xi'$.*

Example 3.13 Consider $[\![\mathtt{l} := \mathtt{h} + \mathtt{1}]\!]$ as an example to show some intuition of the method. Assume the initial distribution of h is $\begin{pmatrix} 1 \to 0.3 & 2 \to 0.4 \\ 3 \to 0.1 & 4 \to 0.1 \\ 5 \to 0.1 & \end{pmatrix}$, and the distribution of l is $\begin{pmatrix} 0 \to 0.5 \\ 1 \to 0.5 \end{pmatrix}$. Consider the partitions ξ_1 on the joint distribution: $\begin{cases} E_1\langle [1,1]_h, [0,1]_l \rangle : \alpha_1 = 0.3 \\ E_2\langle [2,2]_h, [0,1]_l \rangle : \alpha_2 = 0.4 \\ E_3\langle [3,5]_h, [0,1]_l \rangle : \alpha_3 = 0.3 \end{cases}$. The abstract semantic functions transform the above abstract objects into ξ_1': $\begin{cases} E_1'\langle [1,1]_h, [2,2]_l \rangle : \alpha_1' = 0.3 \\ E_2'\langle [2,2]_h, [3,3]_l \rangle : \alpha_2' = 0.4 \\ E_3'\langle [3,5]_h, [4,6]_l \rangle : \alpha_3' = 0.3 \end{cases}$. In the next step,

we perform uniformalization on ξ_1', and concentrate on the low security variable l:

$$\xi_1' \begin{pmatrix} [2,2]_l \to 0.3 \\ [3,3]_l \to 0.4 \\ [4,6]_l \to 0.3 \end{pmatrix} \overset{\texttt{Uniformalization}}{\Longrightarrow} \xi_1'\eta_1 \begin{pmatrix} 2 \to 0.3/1 \\ 3 \to 0.4/1 \\ 4 \to 0.3/3 \\ 5 \to 0.3/3 \\ 6 \to 0.3/3 \end{pmatrix}$$

Finally we do the leakage computation due to such spaces based on the leakage definition: the leakage upper bound $U_l = \mathcal{H}(0.3, 0.4, 0.3) + 0.3 * \log_2 3 = 2.0464$.

In order to show that the strategy of partitioning affect the precision of leakage computation, let us consider another strategy way of partitions ξ_2: $\begin{cases} E_1\langle [1,3]_h, [0,1]_l\rangle : \alpha_1 = 0.8 \\ E_2\langle [4,5]_h, [0,1]_l\rangle : \alpha_2 = 0.2 \end{cases}$. The abstract semantic functions transform the above abstract objects into ξ_2': $\begin{cases} E_1\langle [1,3]_h, [2,4]_l\rangle : \alpha_1 = 0.8 \\ E_2\langle [4,5]_h, [5,6]_l\rangle : \alpha_2 = 0.2 \end{cases}$. After doing uniformalization, we have:

$$\xi_2' \begin{pmatrix} [2,4]_l \to 0.8 \\ [5,6]_l \to 0.2 \end{pmatrix} \overset{\texttt{Uniformalization}}{\Longrightarrow} \xi_2'\eta_2 \begin{pmatrix} 2 \to 0.8/3 \\ 3 \to 0.8/3 \\ 4 \to 0.8/3 \\ 5 \to 0.2/2 \\ 6 \to 0.2/2 \end{pmatrix}$$

and the leakage upper bound of variable l is given by:

$$U_l = \mathcal{H}(0.8, 0.2) + 0.8 \times \log_2 3 + 0.2 \times \log_2 2 = 2.1899$$

Furthermore, it is easy to get that the exact leakage due to this simple example is $\mathcal{L} = \mathcal{H}(0.3, 0.4, 0.1, 0.1, 0.1) = 2.0464$. This simple example suggests that uniformalization on abstract objects preserves the safety of leakage computations, and the strategy of partition may affect the precision of the computation: the first strategy of partition in this example is much better than the second one since the first one is closer to the exact leakage. But both of them provide safe leakage computation, *i.e.* never underestimate the leakage analysis. The precision of partitioning strategy depends on the initial distribution of high security inputs and also the programs, but not in general. One can image that the analyzer can produce more precise result if the partitions produced by the partitioning strategy and abstract operations in the abstract lattice is closer to the concrete partitions produced by the programs (especially conditional tests and loops) in the concrete lattice.

Example 3.14 Consider a loop program

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

assume h is 3-bit high security variable with distribution $\begin{pmatrix} 0 \to 0.1,\ 1 \to 0.1 \\ 2 \to 0.1,\ 3 \to 0.1 \\ 4 \to 0.2,\ 5 \to 0.2 \\ 6 \to 0.1,\ 7 \to 0.1 \end{pmatrix}$, l is 3-bit low security variable with an initial value of 0. Let us take partition ξ as: $\begin{cases} E_1\langle [0,3]_h, [0,0]_l\rangle : \alpha_1 = 0.4 \\ E_2\langle [4,7]_h, [0,0]_l\rangle : \alpha_2 = 0.6 \end{cases}$. According to the above abstract operation, we can get that,

- $\alpha_1' = 0.4,\ [a_h, b_h] = [0,3], [a_l, b_l] = [0,3]$
- $\alpha_2' = 0.6,\ [a_h, b_h] = [4,7], [a_l, b_l] = [0,7]$

Finally we compute the entropy of y by leakage definition after uniformalization: leakage upper bound $U_l = \mathcal{H}(0.4, 0.6) + 0.4 * \log_2 4 + 0.6 * \log_2 8 = 3.57$. That is of course not as good as the exact computation:

$$\mathcal{L} = \mathcal{H}(0.1, 0.1, 0.1, 0.1, 0.2, 0.2, 0.1, 0.1) = 2.92$$

but still offers a reasonable upper bound. This example also shows our abstraction techniques improves the time consuming problem of leakage analysis. Note that the concrete analysis needs 8 time-unit(iteration) to get the result while the abstract analysis needs 2 units.

The measurement produced by our technique should be an upper bound on the actual information flow, so that our technique can overestimate the amount of information leaked, but can never underestimate it. To address this concern, Proposition 3.15 is presented to show the soundness of our approximation for abstract leakage analysis formally.

Proposition 3.15 *Assume the final abstract space obtained by the semantic functions $[\![\cdot]\!]$ be a finite measurable partition ξ. After performing uniformalization sub-partition η on these output abstract objects ξ, we get safe leakage upper bound by computing the entropy of $\xi\eta$.*

Proof. Consider a distribution μ over a set of events E containing N elements and let $\xi = \{E_1, \ldots, E_n\}$ be the partitions of this set, η be a sub-partition on ξ. Assume N_k is the number of elements of E_k, we have $N = \sum_{k=1}^{n} N_k$, and let $\mu(E_k)$ be the weight of the partition $E_k (k = 1, \ldots, n)$. Let the elements of E be labeled from 1 to N, *i.e.* $E = \{e_1, \ldots, e_N\}$ and the elements of $E_k (k = 1, \ldots, n)$ be labeled from 1 to N_k. According to the definition of entropy of partitions [29,28], the entropy of such space over E can be computed by:

$$\mathcal{H}(\xi\eta) = \mathcal{H}(\xi) + \mathcal{H}(\eta|\xi) = \mathcal{H}(\mu_1, \ldots, \mu_n) + \sum_{i=1}^{n} \mu_i \mathcal{H}(\frac{\mu_{i1}}{\mu_i}, \ldots, \frac{\mu_{iN_i}}{\mu_i})$$

Now consider that the η is under uniform distribution over each element of partition ξ, since the number of elements in partition E_k is N_k, the entropy of the space after uniformalization on the partitions can be computed by:

$$\bar{\mathcal{H}}(\xi\eta) = \mathcal{H}(\mu_1, \ldots, \mu_n) + \sum_{i=1}^{n} \mu_i \mathcal{H}(\frac{1}{N_i}, \ldots, \frac{1}{N_i})$$

Since uniform distribution maximum entropy, it is clear that

$$\sum_{i=1}^{n} \mathcal{H}(\frac{1}{N_i}, \ldots, \frac{1}{N_i}) \geq \sum_{i=1}^{n} \mathcal{H}(\frac{\mu_{i1}}{\mu_i}, \ldots, \frac{\mu_{iN_i}}{\mu_i})$$

Therefore we get $\bar{\mathcal{H}}(\xi\eta) \geq \mathcal{H}(\xi\eta)$, *i.e.* we show that uniformalization on the partitions keeps the entropy computation safe. □

We have already introduced a method to provide an approximation analysis of our automatic leakage analysis system presented at [26]. The method produces safe leakage bounds on variables to address the scalability problem of concrete leakage analysis.

4 Related Work

Quantitative information flow has recently become an active research topic in the computer security community. The precursor for this work was that of Denning in the early 1980's. Denning [10] presented that the data manipulated by a program can be typed with security levels, which naturally assumes the structure of a partially ordered set. Moreover, this partially ordered set is a lattice under certain conditions [9]. However, he did not suggest how to automate the analysis. In 1987, Millen [24] first built a formal correspondence between non-interference and mutual information, and established a connection between Shannon's information theory and state-machine models of information flow in computer systems. Later related work is that of McLean and Gray and McIver and Morgan in 1990's. McLean presented a very general Flow Model in [23], and also gave a probabilistic definition of security with respect to flows of a system based on this flow model. The main weakness of this model is that it is too strong to distinguish between statistical correlation of values and casual relationships between high and low object. It is also difficult to be applied to real systems. Gray presented a less general and more detailed elaboration of McLean's flow model in [15], making an explicit connection with information theory through his definition of the channel capacity of flows between confidential and non-confidential variables. Webber [32] defined a property of n-limited security, which took flow-security and specifically prevented downgrading unauthorized information flows. Wittbold and Johnson [33] gave an analysis of certain combinatorial theories of computer security from information-theoretic perspective and introduced non-deducibility on strategies due to feedback. Gavin Lowe [19] measured information flow in CSP by counting refusals. Aldini and Di Pierro [1] introduced a method of quantifying information flow on a probabilistic process system, The method is based on process similarity relation with regard to an approximation of weak bisimulation of CCS. Backes [2] gave a definition for measuring the quantity of information flow within interactive settings by measuring the distance between different behaviours of high user from low user's views. McIver and Morgan [22] devised a new information theoretic definition of information flow and channel capacity. They added demonic non-determinism as well as probabilis-

tic choice to *while* program thus deriving a non-probabilistic characterization of the security property for a simple imperative language. There are some other attempts in the 2000s: Di Pierro, Hankin and Wiklicky [11,12,13] gave a definition of probabilistic measures on flows in a probabilistic concurrent constraint system where the interference came via probabilistic operators. Clarkson et al. [7] suggested a probabilistic beliefs-based approach to non-interference. This work might not works for most of situations. Different attackers have different beliefs, therefore the worst case is required. Boreale [3] studied the quantitative models of information leakage in the process calculi. Clark, Hunt, and Malacaria [4,5,6] presented a more complete reasonable quantitative analysis for a particular program in imperative languages but the bounds for loops are over pessimistic. Malacaria [20] gave a more precise quantitative analysis of loop construct but it is hard to automate. McCamant and Ernst [21] investigated techniques for quantifying information flow revealed by complex programs by building flow graphs and considering the weight of the maximum flow over it. Köpf and Basin [16] developed a quantitative model for assessing a system's vulnerability to adaptive side-channel attacks.

5 Conclusions and Future Works

We aimed to develop automatic and high quality leakage analysis tools for programs. In [26], we devised a method that implements Kozen's [18] concrete probabilistic semantics, and applied this semantics to calculate leakage. In this paper, we present an approach to provide an approximation on such leakage analysis.

We are currently integrating the abstraction techniques into our implementation of concrete leakage analysis tool. We also expect to improve our analysis based on the experimental results of the influence of certain parameters over the quality of approximation. We propose to incorporate more parameters to obtain a better approximation result, especially due to the time complexity issues introduced by the calculation of mutual information. Another possible plan is to produce an abstract backward analysis for quantitative information flow. The idea is that we start from a description of an output event, and compute back to the description of the input domains describing their distribution of making the behavior happen. This allows the effective computation of upper bounds on the probability of outcomes of the program. Furthermore, it is necessary to attack the problem of computation expense of mutual information. We consider to develop a chopped information flow graph for representing information flow of program by capturing and modeling the information flow of data tokens on data slice. Smith [31] introduced a new foundation based on a concept of vulnerability recently, which measures uncertainty by applying Rényi's min-entropy rather than Shannon entropy. Our method is based on probability distribution, we may also consider to adapt our method to different information theoretic measures as Smith's [31] definition.

Acknowledgement

This research is supported by the EPSRC grant EP/C009967/1 Quantitative Information Flow.

References

[1] Alessandro Aldini, Alessandra Di Pierro, and Ra Di Pierro. A quantitative approach to noninterference for probabilistic systems, 2004.

[2] Michael Backes. Quantifying probabilistic information flow in computational reactive systems. In *Proceedings of 10th European Symposium on Research in Computer Security (ESORICS)*, volume 3679 of *Lecture Notes in Computer Science*, pages 336–354. Springer, September 2005.

[3] Michele Boreale. Quantifying information leakage in process calculi. In *ICALP (2)*, pages 119–131, 2006.

[4] David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantitative analysis of the leakage of confidential data. In *Electronic Notes in Theoretical Computer Science*, volume 59, Elsevier, 2002.

[5] David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantitative information flow, relations and polymorphic types. *J. Log. and Comput.*, 15(2):181–199, 2005.

[6] David Clark, Sebastian Hunt, and Pasquale Malacaria. A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security*, 15:321–371, 2007.

[7] Michael R. Clarkson, Andrew C. Myers, and Fred B. Schneider. Quantifying information flow with beliefs. *Journal of Computer Security*, 2007.

[8] Patrick Cousot and Rahida Cousot. Abstract interpretation and application to logic programs. *J. Log. Program.*, 13(2-3):103–179, 1992.

[9] Dorothy Elizabeth Robling Denning. A lattice model of secure informatin flow. *Communications of the ACM*, 19(5):236–243, May 1976.

[10] Dorothy Elizabeth Robling Denning. *Cryptography and Data Security*. Addison-Wesley, 1982.

[11] Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky. Approximate non-interference. In *CSFW*, pages 3–17, 2002.

[12] Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky. Approximate non-interference. *Journal of Computer Security*, 12(1):37–82, 2004.

[13] Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky. Quantitative static analysis of distributed systems. *J. Funct. Program.*, 15(5):703–749, 2005.

[14] Joseph Goguen and Jose Meseguer. Security policies and security models. In *IEEE Symposium on Security and privacy*, pages 11–20. IEEE Computer Society Press, 1982.

[15] James W. III Gray. Toward a mathematical foundation for informatin flow security. In *IEEE Security and Privacy*, pages 21–35, Oakland, California, 1991.

[16] Boris Köpf and David Basin. An information-theoretic model for adaptive side-channel attacks. In *CCS '07: Proceedings of the 14th ACM Conference on Computer and Communications Security*, pages 286–296, New York, NY, USA, November 2007. ACM SIGSAC, ACM Press.

[17] Dexter Kozen. Semantics of probabilistic programs. In *20th Annual Symposium on Foundations of Computer Science*, pages 101–114, Long Beach, California, USA, October 1979.

[18] Dexter Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 22(3):328–350, 1981.

[19] Gavin Lowe. Defining information flow quantity. *Journal of Computer Security*, 12(3-4):619–653, 2004.

[20] Pasquale Malacaria. Assessing security threats of looping constructs. In *Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 225–235, Nice, France, 2007. ACM Press.

[21] Stephen McCamant and Michael D. Ernst. Quantitative information flow as network flow capacity. In *Proceedings of the ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation*, Tucson, AZ, USA, June 9–11, 2008.

[22] Annabelle McIver and Carroll Morgan. A probabilistic approach to information hiding. In *Programming methodology*, pages 441–460, New York, NY, USA, 2003. Springer-Verlag New York.

[23] John McLean. Security models and information flow. In *Proceeding of the 1990 IEEE Symposium on Security and Privacy*, Oakland, California, May 1990.

[24] Jonathan Millen. Covert channel capacity. In *Proceeding 1987 IEEE Symposium on Resarch in Security and Privacy*. IEEE Computer Society Press, 1987.

[25] David Monniaux. Abstract interpretation of probabilistic semantics. In *SAS'00: Proceedings of the 7th International Symposium on Static Analysis*, pages 322–339, London, UK, 2000. Springer-Verlag.

[26] Chunyan Mu and David Clark. Quantitative analysis of secure information flow via probabilistic semantics. In *ARES: The International Dependability Conference*, Fukuoka, Japan, 2009. IEEE Computer Society, to appear.

[27] Alfred Renyi. On measures of entropy and information. In *Proceedings of the 4th Berkeley Symposium on Mathematics, Statistics and Probability*, pages 547–561, 1961.

[28] Alfred Renyi. *Probability theory*. North-Holland Publishing Company, Amsterdam, 1970.

[29] Vladimir Abramovich Rokhlin. Lectures on the entropy theory of measure-preserving transformations. *Russian Mathematical Surveys*, 22(5):1–52, 1965.

[30] Walter Rudin. *Real and Complex Analysis*. McGraw-Hill, 1966.

[31] Geoffrey Smith. On the foundations of quantitative information flow. In *FOSSACS*, pages 288–302, 2009.

[32] Douglas G. Weber. Quantitative hook-up security for covert channel analysis. In *CSFW 1988*, Franconia, NH, 1988. IEEE.

[33] J. Todd Wittbold and Dale M. Johnson. Information flow in nondeterministic systems. In *IEEE Symposium on Security and Privacy*, pages 144–161, 1990.