

Quantified Interference: Information Theory and Information Flow (Extended Abstract)

David Clark¹ and Sebastian Hunt² and Pasquale Malacaria³

¹ Department of Computer Science, Kings College, London. david@dcs.kcl.ac.uk

² Department of Computing, City University, London. seb@soi.city.ac.uk

³ Department of Computer Science, Queen Mary, London. pm@dcs.qmul.ac.uk

Abstract. The paper investigates which of Shannon's measures (entropy, conditional entropy, mutual information) is the right one for the task of quantifying information flow in a programming language. We examine earlier relevant contributions from Denning, McLean and Gray and we propose and motivate a specific quantitative definition of information flow. We prove results relating equivalence relations, interference of program variables, independence of random variables and the flow of confidential information. Finally, we show how, in our setting, Shannon's Perfect Secrecy theorem provides a sufficient condition to determine whether a program leaks confidential information.

1 Introduction

Our present program is to develop and implement program analyses measuring either the amount or the rate of flow of information between variables as a result of the execution of code.

In its most general conception this could be interference between any kind of objects in a computational context. However, we initially conceived this ambition as a more modest one: that of weakening the restrictions that non-interference places on the construction of programs designed to maintain access control policies. It is well known, by bankers and systems administrators among others, that acceptable access control can be achieved in spite of allowing opportunities to guess PINs and passwords. Furthermore, the non-interference security community is aware of limitations of non-interference [1]. Since real programs often do leak confidential information [2] it seems sensible to try to measure that leakage as best we can. To date, we have shown that the idea is feasible [3], and have described an implementable analysis for a while language which handles loops, branching, equality tests and arithmetic expressions [4]. We note that we have so far dealt with deterministic languages although our approach is not necessarily restricted to them.

Apart from access control security, we believe that there will be a use for quantifying interference in a broader range of applications, some of which are discussed in our conclusions.

In this paper we set out a theoretical framework intended to form a basis for analysis (automated or otherwise) of programs to estimate the degree of interference between variables.

We critically review three significant past contributions by Denning [5], McLean [6], and Gray [7], all of which have a bearing on our own approach. Aspects of our method of calculation were pre-figured by Denning [5] (further on in the paper we reveal a flaw in Denning’s flow condition and show how our definition corrects it), our system model is essentially McLean’s [6], and the quantity we estimate in our analyses was first defined by Gray [7] in the context of a channel capacity theorem. Although not included in the review per se we should also mention the contribution of Jonathan Millen [8] which points to the relevance of Shannon’s use of finite state systems in the analysis of channel capacity.

Some differences between the achievements of these approaches and our own (which has built on theirs) are summarised in figure 1 (CHM denotes our approach). Note that we are referring in the figure only to program analyses based on information theory. Denning does indeed describe an automatic analysis of flows [9] but this not quantitative and is not based on information theory. In this

	Denning	McLean	Gray	CHM
Probabilistic Systems	N	Y	Y	Y
Analysis of Programs	N	N	N	Y
Quantitative results	Y	N	N	Y

Fig. 1. Comparison of approaches

paper we consider a probabilistic semantic setting and we use the phrase ‘deterministic’ to mean programs with no ‘internal’ probabilistic behaviours (outputs are determined uniquely by inputs). We do not consider CSP-style non-determinism (see [10], for example).

We give two quantitative (information theoretic) definitions of information flow and show that one is a specialisation of the other, appropriate for deterministic languages; we establish that, for deterministic languages, it is safe to estimate information flow into a collection of variables by summing the flows into the individual variables. Our main aim following this is to formalise the relationships between our quantitative definition of information flow for deterministic languages and the more widely studied (non quantitative) notion of non-interference. This is achieved (in part) by establishing a correspondence between random variables and equivalence relations.

Once the correspondence between random variables and relations is established, we show how to express quantitative flow in a relational form, creating the possibility of calculating upper bounds on flows through a program based

on compositional analysis of relational properties of its parts. Finally we demonstrate a strong connection with Shannon’s perfect secrecy theorem [11].

The remainder of this introduction discusses related work and summarises basic definitions and notation for information theory and random variables. The review of Denning’s, McLean’s and Gray’s work is given in section 2 and we discuss our own definition of leakage and its relationship to the older work in section 3. The relationships between our definitions and non-interference are explored in section 4. Section 5 concludes.

1.1 Related work

Information theory is not the only quantitative approach to information flow.

Contemporary with our own work has been that of Di Pierro, Hankin and Wiklicky. Their interest has been to measure interference in the context of a probabilistic concurrent constraint setting where the interference comes via probabilistic operators. In [12] they derive a quantitative measure of the similarity between agents written in a probabilistic concurrent constraint language. This can be interpreted as a measure of how difficult a spy (agent) would find it to distinguish between the two agents using probabilistic covert channels, with a measure of 0 meaning the two agents were indistinguishable. Their approach does not deal with information in an information theoretic sense although the implicit assumption in example 4 in that paper is that the probability distribution of the value space is uniform. However, all the quantitative approaches to information flow considered in the current paper are based on information theory.

1.2 Information and conditional information

We use Shannon’s information theory to quantify the amount of information a program may leak and the way in which this depends on the distribution of inputs. Shannon’s measures are based on a logarithmic measure of the unexpectedness, or surprise, inherent in a probabilistic event. An event which occurs with some non-zero probability p is regarded as having a ‘surprisal value’ of $\log \frac{1}{p}$. Intuitively, surprise is inversely proportional to likelihood. The base for log may be chosen freely but it is conventional to use base 2 (the rationale for using a logarithmic measure is given in [11]). The total information carried by a set of n events is then taken as the weighted sum of their surprisal values:

$$\mathcal{H} = \sum_{i=1}^n p_i \log \frac{1}{p_i} \tag{1}$$

(if $p_i = 0$ then $p_i \log \frac{1}{p_i}$ is defined to be 0). This quantity is variously known as the *self-information* or *entropy* of the set of events.

In what follows we generally use the language of random variables rather than talk directly about distributions. For our purposes, a random variable is

a total function $X : D \rightarrow R$, where D and R are finite sets and D is equipped with a probability distribution. (We note that the term *random variable* is more conventionally reserved for maps into the reals, with maps of our form being known as *discrete random elements*.) We adopt the following conventions for random variables:

1. if X is a random variable we let x range over the set of values which X may take; if necessary, we denote this set explicitly by $R(X)$; the domain of X is denoted $D(X)$
2. we write $p(x)$ to mean the probability that X takes the value x ; where any confusion might otherwise arise, we write this more verbosely as $P(X = x)$
3. for a vector of (possibly dependent) random variables (X_1, \dots, X_n) , we write $p(x_1, \dots, x_n)$ for the joint probability that the X_i simultaneously take the values x_i ; equivalently, we may view the vector as a single random variable X with range $R(X_1) \times \dots \times R(X_n)$
4. when summing over the range of a random variable, we write $\sum_x f(x)$ to mean $\sum_{x \in R(X)} f(x)$; again, we use the more verbose form where necessary to avoid confusion

The entropy of a random variable X is denoted $\mathcal{H}(X)$ and is defined, in accordance with (1), as:

$$\mathcal{H}(X) = \sum_x p(x) \log \frac{1}{p(x)} \quad (2)$$

Note that, for the purposes of calculating entropy, the only relevant characteristic of X is its set of non-empty inverse images, since $p(x)$ is just the sum in D of the probabilities of the elements of $X^{-1}(x)$. We return to this point in section 4.1.

A basic definition which will be used is that of *conditional entropy* $\mathcal{H}(X|Y)$ measuring the uncertainty in X given knowledge of Y . It is defined as:

$$\mathcal{H}(X|Y) = \mathcal{H}(X, Y) - \mathcal{H}(Y)$$

Information theory provides a more general way of measuring the extent to which information may be shared between two sets of observations. Given two random variables X and Y , the mutual information between X and Y , written $\mathcal{I}(X; Y)$ is defined as follows:

$$\mathcal{I}(X; Y) = \mathcal{H}(X) + \mathcal{H}(Y) - \mathcal{H}(X, Y)$$

$\mathcal{I}(X; Y)$ is symmetric in X and Y .

This quantity is a direct measure of the amount of information carried by X which can be learned by observing Y (or vice versa). As with entropy, there are conditional versions of mutual information. The mutual information between X and Y given knowledge of Z , written $\mathcal{I}(X; Y|Z)$, may be defined as

$$\mathcal{I}(X; Y|Z) = \mathcal{H}(X|Z) + \mathcal{H}(Y|Z) - \mathcal{H}(X, Y|Z)$$

2 A review of significant contributions

2.1 Denning’s approach

In [5], Denning suggests a definition of information flow for programs, based on information theory. Given two program variables x, y in a program P and two states s, s' in the execution of P , Denning suggests that there is a flow of information from x at state s to y at state s' if uncertainty about the value of x at s given knowledge of y at s' is less than uncertainty about the value of x at s given knowledge of y at s . Using information theory, Denning translates existence of a flow thus defined into the following condition:

$$\mathcal{H}(x_s|y_{s'}) < \mathcal{H}(x_s|y_s) \tag{3}$$

Where there *is* a flow of information, Denning proposes the obvious corresponding measure of its quantity:

$$\mathcal{H}(x_s|y_s) - \mathcal{H}(x_s|y_{s'}) \tag{4}$$

i.e. the difference in uncertainty between the two situations. Denning’s work does not suggest how any analysis of leakage based on these ideas could be automated.

In section 3.1, after introducing our own definition of leakage, we argue that Denning’s definition is flawed and propose a correction, resulting in a definition equivalent to our own.

2.2 McLean’s Approach

According to McLean [6], the most stringent approach to information flow is Sutherland’s Non-deducibility model [13]. This model requires High and Low objects to be effectively independent. Non-deducibility, also called compartmentalisation, may be helpful to logicians to reason about independent subsystems of a system, however it doesn’t capture the notion of non-interference as intended in a security context.

McLean argues that an analysis of the notion of secure information flow, as opposed to compartmentalisation, requires the introduction of *time* into the model. When this is done, only certain classes of dependency between Low and High are considered security violations.

McLean’s model is highly abstract. System behaviours are modelled by the sequences of values taken by the ‘High and Low objects’ of the system. His Flow Model states that a system is secure if $p(L_t|(H_s, L_s)) = p(L_t|L_s)$, where L_t describes the values taken by the Low system objects at time t while L_s and H_s are the sequences of values taken by the Low and High objects, respectively, at times preceding t . The condition is illustrated in figure 2.

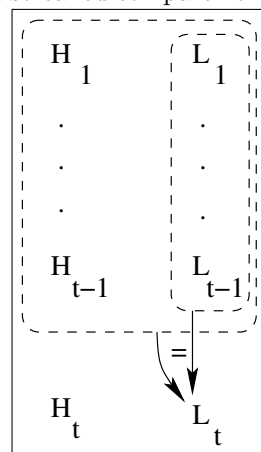


Fig. 2: McLean’s Flow Model

McLean’s Flow Model provides the right security model for a system with memory. However his work is qualitative and there is not enough machinery to implement an analysis based on it.

2.3 Gray’s approach

Gray’s system and information flow security model [7] can be seen as a less general and more detailed elaboration of McLean’s flow model.

Gray makes an explicit connection with information theory: he shows that, if the flow security condition holds, the channel capacity of the channel from High to Low is zero. His measure of the flow of information at each point in time uses conditional mutual information to measure the flow for a given history of inputs and outputs. Using McLean’s notation, this is:

$$\mathcal{I}(H_s; L_t \mid L_s) \tag{5}$$

where H_s, L_s, L_t are as given earlier. We will return to this definition when we present our own definition of leakage since, as stated in a previous paper [3], our own definition, although developed independently, can be seen as a special case of this.

3 Our approach

In our work we consider a programming language with a simple operational semantics and with the aim of analysing confidentiality properties based purely on the input-output behaviour which this semantics defines. The guarantees provided by our analysis are correspondingly limited. In particular, our analysis addresses the question of how much an attacker may learn (about confidential information) by observing the input-output behaviour of a program, but does not tell us anything about how much can be learned from its running-time. We also neglect to deal with any complications which may arise due to termination behaviour, by assuming that all programs terminate on all inputs.

The language contains just the following control constructs: assignment, **while**-statements, **if**-statements, sequential composition. The left hand sides of assignments are variable identifiers, the right hand sides are integer or boolean expressions; **while** loops and **if**-statements involve boolean expressions in the standard way. We do not fully specify the language of expressions but we make the assumption that all expressions define total functions on stores. The language is deterministic and so, for each program P , the semantics induces (assuming termination) a total function $\llbracket P \rrbracket : \Sigma \rightarrow \Sigma$, where Σ is the domain of stores. A store $\sigma \in \Sigma$ is just a finite map from variable names to k -bit integers (integers n in the range $-2^{k-1} \leq n < 2^{k-1}$) and booleans.

Given a program P , a *program point* is either the special node ω (the *exit point*), or any occurrence in P of an assignment statement, **if**-statement or **while**-statement. We call the top-most program point ι (the *entry point*). The

operational semantics is standard and defines a transition relation \rightarrow on configurations (n, σ) , where n is a program point and σ is a store.

The events of interest for us are observations of the values of variables before and after the execution of (part of) a program. Suppose that the inputs to a program take a range of values according to some probability distribution. In this case we may use a random variable to describe the values taken (initially) by a program variable, or set of program variables. In particular, we are interested in how much of the information carried by the High inputs to a program can be learned by observation of the Low outputs, assuming that the Low inputs are known. Since our language is deterministic, any variation in the outputs is a result of variation in the inputs. Once we account for knowledge of the program's Low inputs, therefore, the only possible source of surprise in an output is interference from the High inputs. Given a program variable (or set of program variables) X , let X^l and X^ω be, respectively, the corresponding random variables on entry to and exit from the program. We take as a measure of the amount of leakage into X due to the program:

$$\mathcal{L}(X) = \mathcal{H}(X^\omega | L^l) \quad (6)$$

(recall that L is the set of Low variables, thus L^l is the random variable describing the distribution of the program's non-confidential inputs).

Alternatively, we could argue that a natural definition of the leakage into X is the amount of information shared between the final value of X and the initial value of H , given that the initial value of L was already known:

$$\mathcal{L}'(X) = \mathcal{I}(H^l; X^\omega | L^l) \quad (7)$$

This definition is essentially the one used by Gray (presented as equation (5) in section 2.3 above) specialised to a simpler semantic setting: there are only two points in time, the input time and the output time; there are no outputs at input time and no inputs at output time.

In the current semantic setting, where a program defines a function from inputs to outputs, (6) and (7) are actually equivalent:

Proposition 1. *Let X, Y, Z be random variables such that, $Z = f(X, Y)$, where f is any function. Then $\mathcal{H}(Z|Y) = \mathcal{I}(X; Z|Y)$.*

Corollary 1. *\mathcal{L} and \mathcal{L}' are equivalent for a deterministic language (let $X = H^l$, $Y = L^l$, then $Z = L^\omega$ is a function of (X, Y)).*

The advantage of (7) as a definition of leakage is that it is appropriate even for a language with an inherently probabilistic semantics, whereas (6) is not. Suppose, for example, that our While language was extended with a 'fair coin' as one of its constructs, allowing us to write: $X := \text{coin}$; Clearly, this program does not leak any information into X , since the final value of X is independent of the program's initial state. This is confirmed using (7), which gives a leakage of $\mathcal{I}(H^l; X^\omega | L^l) = 0$ (one of the basic identities of information theory, since H^l

and X^ω are independent even given L^ι . By contrast, applying (6) would give a leakage of $\mathcal{H}(X^\omega|L^\iota) = \mathcal{H}(X^\omega) = \frac{1}{2} \log 2 + \frac{1}{2} \log 2 = 1$.

Another striking difference between the current ‘functional’ setting and the more general probabilistic setting, is that in the functional case it is safe (conservative) to calculate a program’s overall input-output leakage by considering its outputs separately:

Proposition 2. *Let X be a vector of random variables X_1, \dots, X_n in a non-probabilistic language. Then $\mathcal{L}(X) \leq \mathcal{L}(X_1) + \dots + \mathcal{L}(X_n)$*

This does *not* hold for \mathcal{L}' in the setting of a probabilistic language.

3.1 Comparison with Denning

In this section we identify a flaw in Denning’s definition, propose a correction, and show that the modified definition coincides with our own.

Suppose that, in state s , $y = \text{abs}(x)$ where x takes any integer value in the range $-16, 15$, with uniform distribution. Then consider the following two programs:

- (A) $\{s\} \text{if } (x = 0) \text{ then } y = 1 \text{ else } y = 2\{s'\}$
- (B) $\{s\} \text{if } (x < 0) \text{ then } y = 1 \text{ else } y = 2\{s''\}$

Calculating the quantities involved in (4) we find: $\mathcal{H}(x_s|y_s) = 1$ and $\mathcal{H}(x_s|y_{s'}) \simeq 4.8$ and $\mathcal{H}(x_s|y_{s''}) = 4$. Thus, according to (4), there is no flow from x to y in either case, since the uncertainty in x given y actually increases in both cases. Now it is implicit in Denning’s definition that the quantity of flow depends, in part, on what is observed both before and after the program runs. The first term in (4), $\mathcal{H}(x_s|y_s)$, represents the initial uncertainty in x given that y is observed, whereas the second term is intended to represent the final uncertainty in x , again given that y is observed. The problem lies with the second term: it accounts for the final observation of y but (effectively) assumes that the initial observation has been forgotten. This is a safe assumption only if we know that the observer has no memory. In general, however, we must assume that, in the end, the observer knows *both* the initial *and* final values of y . Modifying (4) in line with this assumption, we obtain:

$$\mathcal{H}(x_s|y_s) - \mathcal{H}(x_s|y_{s'}, y_s) \tag{8}$$

Note that $\mathcal{H}(X|Y, Z) \leq \mathcal{H}(X|Y)$, for any random variables X, Y, Z , and thus (4) \leq (8) will always hold.

Applying (8) to programs (A) and (B), we calculate: $\mathcal{H}(x_s|y_{s'}, y_s) = 1$ and $\mathcal{H}(x_s|y_{s''}, y_s) = 0$. So, using (8), we still find no flow for program (A) but for program (B) we have a flow of 1 bit. The crucial difference between (A) and (B) is that (A) reveals nothing *new* about x (knowing $\text{abs}(x)$ we already know if $x = 0$) whereas (B) reveals the one thing we didn’t know, namely the sign of x .

Applying (8) to the case of a program with inputs H^ι, L^ι and outputs H^ω, L^ω , we obtain:

$$\mathcal{H}(H^\iota|L^\iota) - \mathcal{H}(H^\iota|L^\omega, L^\iota) \tag{9}$$

Proposition 3. $\mathcal{L}'(L) = (9)$

4 Non-interference

Leakage of confidential information is a particular case of information flow where the source of information is a High variable and the target a Low variable. In general when there is information flow between two variables they are said to *interfere*. The absence of interference is known, reasonably enough, as *non-interference*. One attraction of non-interference is its relative simplicity, since it is a binary property which can be defined without any explicit recourse to information theory [14]. Roughly speaking, a deterministic program is said to satisfy non-interference if its Low outputs depend only on its Low inputs (hence *not* on its High inputs).

More formally, given two While program variables X, Y considered, respectively, at the entry p and exit p' of a statement we say that X and Y *interfere* if there exists an assignment of values for all remaining program variables and two values $v \neq v'$ such that Y takes different values at p' depending on whether $X = v$ or $X = v'$ at p . The m -ary definition of interference is a generalisation: X_1, \dots, X_m at p interfere with Y at p' if the state-transformer denoted by the statement is non-constant on the X_1, \dots, X_m component of the store.

For languages with a straightforward functional semantics (such as the While language we consider in this paper), a generalised definition of non-interference can be formalised by modelling different categories of observation (eg High, Low) by equivalence relations. Given relations R and S , a function f is said to *map R into S* , written $f : R \Rightarrow S$, iff $\forall x, x'. x R x' \Rightarrow f(x) S f(x')$. The notion (logical relations [15]) is quite general though, in what follows, R and S will always be equivalence relations.

Observations of a program store restricted to a particular set of variables X give rise to an equivalence relation in a natural way: $\sigma =_X \sigma'$ iff $\forall x \in X. \sigma(x) = \sigma'(x)$. Thus non-interference for a program denoting state-transformer f can be formalised simply as $f : =_L \Rightarrow =_L$ (Low-equivalence is mapped into Low-equivalence). More generally, let a statement with entry point p and exit p' denote state transformer f . Then the set of program variables $X = \{X_1, \dots, X_m\}$ at p interferes with program variable Y at p' if $f : =_{\bar{X}} \not\Rightarrow =_Y$, where \bar{X} is the *complement* of X , ie, all those program variables *not* in X .

In section 4.1 we show how our quantitative approach to leakage can also be expressed in this relational form. In section 4.2 we go on to explore in more detail the relationship between non-interference and information theory.

4.1 Equivalence relations and random variables

Though superficially rather different, equivalence relations and random variables are actually intimately related ways of modelling observations. As remarked in section 1.2, random variables with identical sets of non-empty inverse images necessarily have the same entropy. If two random variables X and Y are similar in this respect, we write $X \simeq Y$; formally:

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

Intuitively, the reason $X \simeq Y$ implies $\mathcal{H}(X) = \mathcal{H}(Y)$ is that, in this case, X and Y capture exactly the same *observation*, modulo some possible encoding differences: when we observe $X = x$, what we ‘really’ discover is just that the outcome (in $D(X)$) belongs to $X^{-1}(x)$. Thus X effectively partitions the outcomes into sets whose elements are indistinguishable by an observer who only sees the value of X . And, of course, we can state this relationally. The *relation for X* is the equivalence relation $\text{rel}(X)$ on $D(X)$, defined thus:

$$d \text{ rel}(X) d' \text{ iff } X(d) = X(d')$$

The non-empty inverse images of X are the equivalence classes of $\text{rel}(X)$, hence:

Lemma 1. $X \simeq Y$ iff $\text{rel}(X) = \text{rel}(Y)$

Conversely, any equivalence relation R on a set D induces an obvious map from D onto its quotient by R . Writing $[D]_R$ for the equivalence classes of D wrt R , the *variable for R* is the random variable $\text{var}(R) : D \rightarrow [D]_R$ defined simply as

$$\text{var}(R)(d) = [d]_R$$

Clearly, $\text{rel}(\text{var}(R)) = R$. If D is equipped with a distribution, $\text{var}(R)$ may be viewed as a random variable.

These correspondences mean that, for the purposes of defining entropy and mutual information (and their conditional versions) we can use equivalence relations and random variables interchangeably. From now on we will freely apply the notation of entropy and mutual information to equivalence relations as well as random variables. For example, $\mathcal{H}(R)$ should be understood as meaning, $\mathcal{H}(\text{var}(R))$, and so on. Since equivalence relations are naturally combined by intersection and compared by inclusion, we may ask what this means for the corresponding random variables:

Lemma 2. *Let R_1, R_2, R_3 be equivalence relations on a set D equipped with a probability distribution. Then*

1. $\text{var}(R_1 \cap R_2) = \text{var}(R_1), \text{var}(R_2)$
2. $R_1 \subseteq R_2 \Rightarrow \mathcal{H}(R_1) \geq \mathcal{H}(R_2)$
3. $R_1 \subseteq R_2 \Rightarrow \mathcal{H}(R_1|R_3) \geq \mathcal{H}(R_2|R_3)$

Return now to the relational definition of non-interference above. Once we fix on a notion of observation (equivalence relation) S on the outputs of a function $f : A \rightarrow B$, it is easily seen that there is a *coarsest possible* notion of observation on the inputs with respect to which f satisfies non-interference. We denote this relation $f^{-1}(S)$, defined thus:

$$a f^{-1}(S) b \text{ iff } f(a) S f(b)$$

The following lemma confirms that this is indeed the coarsest equivalence relation with the property we seek:

Lemma 3. *Let $f : A \rightarrow B$ with R and S equivalence relations on A and B , respectively. Then $f : R \Rightarrow S$ iff $R \subseteq f^{-1}(S)$.*

Since S is an equivalence on the range of f , not its domain, $\text{var}(S)$ cannot be viewed as a random variable just because the domain of f is equipped with a distribution. Rather, the random variable corresponding to ‘ S -observation’ of the program’s output is $\text{var}(S) \circ f$. For example, if S is $=_L$ then $\text{var}(S) \circ f$ is just L^ω (up to \simeq). It is a straightforward consequence of the above correspondences that random variable $\text{var}(S) \circ f$ captures essentially the same notion of observation as the relation $f^{-1}(S)$:

Lemma 4. $\text{rel}(\text{var}(S) \circ f) = f^{-1}(S)$ and $\text{var}(f^{-1}(S)) \simeq \text{var}(S) \circ f$.

We can now express quantitative leakage in relational form:

Theorem 1. *Let R, S be equivalence relations such that $f : R \Rightarrow S$. Then $\mathcal{H}(R) \geq \mathcal{H}(\text{var}(S) \circ f)$. If T is any other equivalence relation on the domain of f then $\mathcal{H}(R|T) \geq \mathcal{H}(\text{var}(S) \circ f|T)$.*

Proof. By lemmas 2, 3 and 4.

Corollary 2. *Let X be a program variable and let R be any equivalence relation such that $f : R \Rightarrow =_X$. Then $\mathcal{L}(X) \leq \mathcal{H}(R|=_L)$.*

4.2 Perfect Secrecy and non-interference

In this section we formally explore the relationship between non-interference of programming variables and independence of random variables⁴.

We begin with a well known result by Shannon, the *Perfect Secrecy theorem*:

Theorem 2. *Let X, Y, Z be random variables such that $\mathcal{I}(X; Y) = 0$ and $\mathcal{H}(X|Y, Z) = 0$. Then $\mathcal{H}(Z) \geq \mathcal{H}(X)$.*

The result is usually interpreted in cryptography terms as saying (X is the plain text, Y the cypher text and Z the key in a secret key cryptosystem) that if the plain text and the cypher are independent ($\mathcal{I}(X; Y) = 0$) and the plain text can be recovered by key and cypher ($\mathcal{H}(X|Y, Z) = 0$) then the key space has to have at least as much uncertainty as the plain text.

For a programming language (X is the output, Y the High input and Z Low input of the program) the result says that if the output of a deterministic program (determinism is the clause $\mathcal{H}(X|Y, Z) = 0$) has strictly more information than the Low input (clause $\mathcal{H}(Z) < \mathcal{H}(X)$) then the output and the High input are not independent (in the sense of random variables).

For the While language we can think of independence of these random variables as a probabilistic notion of non-interference for the corresponding program variables. In what follows we establish the correspondence between this notion of non-interference and the one defined relationally above.

⁴ Two random variables X, Y are independent iff for all x, y ,
 $p(x = X, y = Y) = p(x = X)p(y = Y)$.

The concepts of independence (of random variables) and non-interference are not straightforwardly related, as the following example shows:

Example: Consider the exclusive boolean or $X = Y \text{ XOR } Z$ (true when exactly one of the arguments is true) with Y, Z independent random variables uniformly distributed over the booleans; we have:

$$\mathcal{I}(Y; X) = \mathcal{H}(Y) + \mathcal{H}(X) - \mathcal{H}(Y, X) = 1 + 1 - 2 = 0$$

So although there is a clear interference between X and Y , this is not shown in $\mathcal{I}(Y; X)$. However if we take Z into account then interference will show: $\mathcal{I}(Y; X|Z) = \mathcal{H}(Y|Z) + \mathcal{H}(X|Z) - \mathcal{H}(Y, X|Z)$ and by applying standard information theory equalities, we find that the rhs is $\mathcal{H}(Y) - \mathcal{H}(Y|Z, X) = 1 - 0 = 1$.

The problem arises here because we have not taken account of the context (provided by Z) in which Y and X occur. The correct probabilistic characterisation of non-interference for programs with multiple variables is via *conditional* independence:

Proposition 4. *Let Y, X_1, \dots, X_n be random variables representing program variables with $Y = f(X_1, \dots, X_n)$. Assume a probability distribution such that⁵, for all (x_1, \dots, x_n) , $p(X_1 = x_1, \dots, X_n = x_n) \neq 0$. Then Y, X_1, \dots, X_i are non-interfering iff $\mathcal{I}(Y; X_1, \dots, X_i | X_{i+1}, \dots, X_n) = 0$.*

Corollary 3.

1. Zero leakage is just a particular case of this general notion of non-interference where X and Z are the High and the Low inputs.
2. When $n = 1$ we have Y, X are non-interfering iff $\mathcal{I}(Y; X) = 0$, that is, iff Y and X are independent random variables.

We can now state a modified version of Shannon’s theorem which, in view of the previous corollary, can be understood as follows: If a deterministic program’s Low output has strictly more information than its Low input, then it leaks. Formally:

Theorem 3. *Let X, Y, Z be random variables such that $\mathcal{I}(X; Y|Z) = 0$ and $\mathcal{H}(X|Y, Z) = 0$. Then $\mathcal{H}(Z) \geq \mathcal{H}(X)$.*

To use a While language example, consider a program whose output is a 16 bit variable and whose Low input is a 8 bit variable and assume that all outputs are roughly equiprobable. Then the program leaks.

We can see from this example that the Perfect Secrecy theorem has a simple intuitive explanation: All information in the output of a deterministic program has to come from the input and if it cannot be provided by the Low input then it has to be provided by the High input.

Notice that the result doesn’t rely on any knowledge of the internal structure of the program; it is not possible, at this level of generality, to determine how and where the leakage occurs.

⁵ This constraint is to avoid f being a “constant in disguise” i.e. f could assume theoretically more than one value but in practice only one value is possible as the inputs for the other values have probability 0.

5 Conclusions

We have defined a measure of information flow, $\mathcal{L} = \mathcal{I}(H^\iota; X^\omega | L^\iota)$, sufficiently general to account for flows in languages with probabilistic non-determinism and have shown its relationship with earlier work. We have proved that for deterministic languages this quantity is the same as the simpler $\mathcal{L}' = \mathcal{H}(X^\omega | L^\iota)$ and that in this context leakage into a set of variables can safely be calculated one variable at a time.

We have shown that equivalence relations and random variables can be used interchangeably when calculating Shannon's information measures and that this simple insight reveals strong relationships between our approach and non-interference. In particular, we have shown that, in the deterministic setting, non-interference is equivalent to a form of conditional probabilistic independence.

Future work along the lines of this paper will develop the relational approach in an effort to quantify information flows for richer languages and more sophisticated models of observation (for example, the threads-based language and the use of probabilistic bisimulation described in [16]).

Security may not be the only application of interest for quantitative analysis of information flow. Other possibilities may appear, such as measuring propagation of meaning in models of natural language, measuring the tightness of coupling between parallel components of a system, or even as a guide to computing optimal fixed points in security related program analyses [17].

References

1. Ryan, P.Y.A., McLean, J., Millen, J., Gilgor, V.: Non-interference, who needs it? In: Proceedings of the 14th IEEE Security Foundations Workshop, Cape Breton, Nova Scotia, Canada, IEEE (2001)
2. Weber, D.G.: Quantitative hookup security for covert channel analysis. In: Proceedings of the 1988 Workshop on the Foundations of Computer Security, Fanconia, New Hampshire, U.S.A. (1988)
3. Clark, D., Hunt, S., Malacaria, P.: Quantitative analysis of the leakage of confidential data. *Electronic Notes in Theoretical Computer Science* **59** (2002)
4. Clark, D., Hunt, S., Malacaria, P.: Quantified interference for a while language. Technical Report Technical Report TR-03-07, Department of Computer Science, King's College London (2003)
5. Denning, D.E.R.: *Cryptography and Data Security*. Addison-Wesley (1982)
6. McLean, J.: Security models and information flow. In: Proceedings of the 1990 IEEE Symposium on Security and Privacy, Oakland, California (1990)
7. W. Gray, III, J.: Toward a mathematical foundation for information flow security. In: Proc. 1991 IEEE Symposium on Security and Privacy, Oakland, CA (1991) 21–34
8. Millen, J.: Covert channel capacity. In: Proc. 1987 IEEE Symposium on Research in Security and Privacy, IEEE Computer Society Press (1987)
9. Denning, D.E.R.: A lattice model of secure information flow. *Communications of the ACM* **19** (1976)

10. Roscoe, Woodcock, Wulf: Non-interference through determinism. In: ESORICS 94, Third European Symposium on Research in Computer Security. Volume 875 of Lecture Notes in Computer Science., Springer (1994)
11. Shannon, C.: A mathematical theory of communication. The Bell System Technical Journal **27** (1948) 379–423 and 623–656 Available on-line at <http://cm.bell-labs.com/cm/ms/what/shannonday/paper.html>.
12. Pierro, A.D., Hankin, C., Wiklicky, H.: Approximate non-interference. In Cervesato, I., ed.: CSFW'02 – 15th IEEE Computer Security Foundation Workshop, IEEE Computer Society Press (2002)
13. Sutherland, D.: A model of information. In: Proceedings of the 9th National Computer Security Conference. (1986)
14. Goguen, J., Meseguer, J.: Security policies and security models. In: IEEE Symposium on Security and Privacy, IEEE Computer Society Press (1982) 11–20
15. Statman, R.: Logical relations and the typed lambda calculus. Information and Control **65** (1985)
16. Sabelfeld, A., Sands, D.: Probabilistic noninterference for multi-threaded programs. In: Proceedings of the 13th IEEE Computer Security Foundations Workshop, Cambridge, England, IEEE Computer Society Press (2000)
17. Giacobazzi, R., Mastroeni, I.: Abstract non-interference: parameterizing non-interference by abstract interpretation. In: POPL 2004, ACM Press (2004) to appear.