Quantitative Information Flow, Relations and Polymorphic Types

DAVID CLARK, *Department of Computer Science, Kings College London, Strand, London WC1R 2LS, UK.*
E-mail: david@dcs.kcl.ac.uk

SEBASTIAN HUNT, *Department of Computing, School of Informatics, City University, London EC1V OHB, UK.*
E-mail: seb@soi.city.ac.uk

PASQUALE MALACARIA, *Department of Computer Science, Queen Mary, University of London, London E1 4NS, UK.*
E-mail: pm@dcs.qmul.ac.uk

Abstract

This paper uses Shannon's information theory to give a quantitative definition of information flow in systems that transform inputs to outputs. For deterministic systems, the definition is shown to specialize to a simpler form when the information source and the known inputs jointly determine all inputs uniquely. For this special case, the definition is related to the classical security condition of non-interference and an equivalence is established between non-interference and independence of random variables. Quantitative information flow for deterministic systems is then presented in relational form. With this presentation, it is shown how relational parametricity can be used to derive upper and lower bounds on information flows through families of functions defined in the second-order lambda calculus.

Keywords: Information flow, information theory, lambda calculus, program analysis, polymorphism, parametricity.

1 Introduction

The general aim of our research is to provide a quantitative analysis of the information flows within computational systems. The original motivation for this work lies in a security setting (specifically issues relating to confidentiality). Much research in this area in recent years has focused on *non-interference* properties [16, 23, 17] (see [30] for a wide-ranging survey of literature in this and related areas). Roughly speaking, non-interference is said to hold when the behaviour of a system as observed by unauthorized users is invariant with respect to variation of the confidential inputs and data maintained by the system. More general work uses relations (in particular, equivalence relations) to capture different classes of observation and categories of input (see [19, 31]; see also Section 5 below).

The need for a quantitative approach arises naturally when we consider situations in which non-interference fails to hold between parts of a system, and we ask *how much* information flows between them. This applies, for example, in 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. However, such systems violate non-interference, because observation of the success or failure of a login

Vol. 15 No. 2,
doi:10.1093/logcom/exi009

attempt is clearly not invariant with respect to variations in the password database. Furthermore, the non-interference security community is aware of limitations of non-interference [29]. Since real programs often do leak confidential information [44] it seems sensible to try to measure that leakage as best we can. To date, we have shown that the idea is feasible [5], and have described an implementable analysis for a While language which handles loops, branching, equality tests and arithmetic expressions [6, 7].

We note two related limitations of the work presented in this paper. Relaxing these limitations is the subject of ongoing work.

First, we ignore any intermediate states in programs and consider systems which simply transform inputs into outputs. The definitions in Section 3 apply to any system treated in this way, irrespective of the language in which it is written. Examples may be written in a While language, or some functional language such as PCF or Haskell. The key thing is that the semantics of the system considered is a transformation of inputs to outputs. We refer to such systems as transformational systems (see Section 3.1 for the formal definition). At this level of abstraction it is clearly not possible to talk about timing issues or to deal adequately with reactive systems.

Second, we assume that our transformational systems define total mappings on finite spaces. We do not regard the restriction to finite primitive datatypes, such as k-bit twos-complement integers, to be a major handicap, since these are precisely the datatypes used in many programming languages. But the inability to deal directly with non-termination and unbounded structured types, such as lists and trees, is obviously unsatisfactory.

1.1 Overview

Section 2 summarizes the basic definitions and notation we require from Shannon's information theory. In Section 3 information theory is used to give a quantitative definition of information flow in transformational systems. The definition is shown to specialize to a simpler form for deterministic systems when the information source and the known inputs jointly determine all inputs uniquely. In Section 4, quantitative information flow is related to the classical security condition of non-interference and an equivalence is established between non-interference and independence of random variables. In Section 5, the quantitative definition of information flow is presented in relational form for deterministic systems. With this presentation, it is shown in Section 6 how relational parametricity can be used to derive upper and lower bounds on information flows through families of functions defined in the second-order lambda calculus.

The remainder of the current section discusses related work.

1.2 Related work

Non-interference is a property guaranteeing the absence of unwanted information flows in a software system. This idea was given a formal definition by Goguen and Messeguer in 1982 [16] within the context of a discussion of security policies. Since that time the study of non-interference has been of ongoing interest to the security and programming language design and analysis communities.

In the last decade or so there has been significant activity around the question of how to construct programs that demonstrably have the non-interference property. Two approaches have been an *a priori* one via type systems and an *a posteriori* one via program analysis. Volpano and Smith have successfully applied a types based approach to a variety of systems [42, 39, 40, 34]. Examples of work that use the program analysis approach include Bodei's work with the Nielsens [2], Malacaria and Hankin's work using control flow analysis via games [21], Clark, Hankin and Hunt's application

of flow logic to idealized Algol [4].

Our work, in contrast, is concerned with using information theoretic quantities to measure *the amount of the flow of information* caused by interference between inputs and outputs.

What little there is of the early work in this area refers to state-based systems. One precursor is that of Denning in the early 1980s. [11] gives examples that contain a few basic intuitions about information theoretic measures of information flow through imperative program constructs and also proposes an information theoretic definition for measuring interference between program variables evaluated at two different program points s and s':

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

This formula is intended to measure the flow of information from x into y as the result of any changes to y along the program execution from s to s'. The meaning of the formula is as follows (see Section 2 for the formal definitions): it measures the difference between an observer's uncertainty about the value of x at s, given knowledge of the value of y at s, and the observer's uncertainty about the value which x had at s, after observing the value of y on reaching s'. As discussed in [8], this definition is quite closely related to our definition of information flow (see Section 3.2) but is actually incorrect.

Other, closely related work, earlier than ours, is that of McLean and Gray. McLean presented a very general flow model in [23], related to Sutcliffe's approach to information flow in [36]. On the basis of this flow model, McLean gave a probabilistic definition of security (with respect to flows) of a system. Gray presented a less general and more detailed elaboration of McLean's flow model in [17], making an explicit connection with information theory through his definition of the channel capacity of flows between confidential and non-confidential variables.

Although our work is the only such that measures the amount of information (in an information theoretic sense) that flows along channels of interference in a program, other researchers have recently taken quantitative approaches to information flow.

In [26] Di Pierro, Hankin and Wiklicky define probabilistic measures on flows in a probabilistic concurrent constraint setting where the interference comes via probabilistic operators. They use this to derive a quantitative measure of the similarity between agents written in this probabilistic concurrent constraint language, however in contrast to our work they do not measure quantities of information. Gavin Lowe has measured information flow in CSP by counting refusals [20] and Volpano and Smith have relaxed strict non-interference and developed type systems in which a well typed program will not leak its secret in polynomial time [41].

2 Information and conditional information

In this section we introduce the quantity we intend to measure: the amount of information that flows from (part of) the input to (part of) the output and we show how this flow is dependent on the likelihood of a particular input occurring, i.e. on the probability distribution on the inputs. To do this we use Shannon's information theory [33]. 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 [33]). The total information

carried by a set of n events is then taken as the weighted sum of their surprisal values:

$$\mathcal{H} = \sum_{i=1}^{n} p_i \log \frac{1}{p_i} \tag{2.1}$$

(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.

It is not easy to give reliable intuitions about the meaning of this definition. Ultimately, this formal notion of information is properly understood through Shannon's coding theorems. The basic idea of coding is a follows. Suppose we wish to record or transmit a message specifying which of the n events has occurred. By clever choice of representation we wish to minimize the average space consumed by the message: this minimum is $\mathcal{H}$. In the simplest case, if we have 2^k events, each equally likely, it is clear that we will require k bits, and it is easily verified that indeed $\mathcal{H} = k$ in this case. This case actually turns out to be the worst case: for any other distribution, $\mathcal{H} < k$. The reason is that we can optimize our representation so that it generates longer messages for less likely events and shorter ones for more likely events. When this idea is properly formalized it turns out that $\mathcal{H}$ is, in fact, the expected length in bits for any optimal encoding. A very clear account of these ideas is given in [37].

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 surjective map $X : D \to \mathcal{R}(X)$, where D is a finite set with a specified probability distribution and $\mathcal{R}(X)$ is the (necessarily finite) range of X. (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*.) For the remainder of this section, let D and its probability distribution be fixed, identically for all random variables under discussion. For each $d \in D$, we write $p(d)$ for its probability. We adopt the following conventions for random variables:

1. If X is a random variable we let x range over $\mathcal{R}(X)$ and we write $p(x)$ to mean the probability that X yields the value x, that is $p(x) \stackrel{\text{def}}{=} \sum_{d \in X^{-1}(x)} p(d)$; where any confusion might otherwise arise, we write this more verbosely as $P(X = x)$.
2. For a vector of (possibly dependent) random variables $(X_1, \ldots, X_n)$, we write $p(x_1, \ldots, x_n)$ for the probability that the joint random variable $\langle X_1, \ldots, X_n \rangle : d \mapsto (X_1(d), \ldots, X_n(d))$ takes the value $(x_1, \ldots, x_n)$.
3. When summing over the range of a random variable, we write $\sum_x f(x)$ to mean $\sum_{x \in \mathcal{R}(X)} f(x)$; 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 (2.1), as:

$$\mathcal{H}(X) = \sum_x p(x) \log \frac{1}{p(x)}. \tag{2.2}$$

Note that, for the purposes of calculating entropy, the only relevant characteristic of X is its set of inverse images, since $p(x)$ is just the sum in D of the probabilities of the elements of $X^{-1}(x)$. If two maps X and Y are similar in this respect, we write $X \simeq Y$; formally:

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

It is easily verified that if $X \simeq Y$ then $\mathcal{H}(X) = \mathcal{H}(Y)$.

Another 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

$$\mathcal{H}(X|Y) = \mathcal{H}(X,Y) - \mathcal{H}(Y). \tag{2.3}$$

(We follow standard practice in suppressing the pairing brackets within information measures; thus $\mathcal{H}(X,Y)$ is shorthand for $\mathcal{H}(\langle X,Y\rangle)$, etc.)

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:

$$\mathcal{I}(X;Y) = \mathcal{H}(X) + \mathcal{H}(Y) - \mathcal{H}(X,Y). \tag{2.4}$$

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

$$\mathcal{I}(X;Y|Z) = \mathcal{H}(X|Z) + \mathcal{H}(Y|Z) - \mathcal{H}(X,Y|Z). \tag{2.5}$$

For more detail on the above fundamental definitions of information theory, see [33, 9].

3 Quantifying information flow

In this section we start by defining a measure of information flow appropriate in a quite general computational setting. We go on to consider the special case of flows in deterministic systems (allowing a purely functional semantics) where all inputs are accounted for, and show how the general definition simplifies in this case.

3.1 Transformational systems

A *transformational system* $\mathcal{S}$ is specified by the following:

1. $D_{\mathcal{S}}$ - a finite set;
2. a probability distribution on $D_{\mathcal{S}}$;
3. a random variable $I_{\mathcal{S}} : D_{\mathcal{S}} \to \mathcal{R}(I_{\mathcal{S}})$ which defines the inputs of the system;
4. a random variable $O_{\mathcal{S}} : D_{\mathcal{S}} \to \mathcal{R}(O_{\mathcal{S}})$ which defines the outputs of the system.

Note that, in any real system of interest, we would expect the outputs of the system to be determined, to some extent, by the inputs, and so $I_{\mathcal{S}}$ and $O_{\mathcal{S}}$ will not normally be independent. Note also that we could, without loss of generality, fix $D_{\mathcal{S}}$ as $\mathcal{R}(I_{\mathcal{S}}) \times \mathcal{R}(O_{\mathcal{S}})$, taking $I_{\mathcal{S}}$ and $O_{\mathcal{S}}$ to be the first and second projections, respectively. However, it is technically convenient not to do this, especially in the case of purely deterministic systems.

Given a transformational system $\mathcal{S}$, we are concerned with two classes of observation:

- An *input observation* is a surjective map $X : \mathcal{R}(I_{\mathcal{S}}) \to \mathcal{R}(X)$.
- An *output observation* is a surjective map $Y : \mathcal{R}(O_{\mathcal{S}}) \to \mathcal{R}(Y)$.

Observations induce random variables $X^{\mathrm{in}} : D_{\mathcal{S}} \to \mathcal{R}(X)$ and $Y^{\mathrm{out}} : D_{\mathcal{S}} \to \mathcal{R}(Y)$ by composition with the relevant component of $\mathcal{S}$:

- $X^{\mathrm{in}} \stackrel{\mathrm{def}}{=} X \circ I_{\mathcal{S}}$.
- $Y^{\mathrm{out}} \stackrel{\mathrm{def}}{=} Y \circ O_{\mathcal{S}}$.

3.1.1 Examples

In the following, let N be some finite subset of the integers, such as those representable in a k-bit twos-complement representation.

1. Consider the system defined by the swap function $\lambda(x, y).(y, x)$ restricted to $N \times N$. In this case it is natural to take $D_{\mathcal{S}}$ to be $N \times N$ and $I_{\mathcal{S}}$ to be the identity. Given a probability distribution on $N \times N$, we then have a transformational system where $I_{\mathcal{S}}$ is the identity and $O_{\mathcal{S}}$ is just swap. In this case we also have $\mathcal{R}(O_{\mathcal{S}}) = N \times N = \mathcal{R}(I_{\mathcal{S}})$. Possible input and output observations include the projections π_1, π_2.
2. Consider terminating programs written in a simple imperative language defined over a finite set of variables Var and containing a `coin` operator, which evaluates randomly to 0 or 1 with equal probability. Let $\Sigma \stackrel{\text{def}}{=} \text{Var} \to N$. Assume given a probability distribution on input states $\sigma \in \Sigma$, writing $p(\sigma)$ for the probability that the input state is σ.

   Any such program P can be viewed as a transformational system taking $D_{\mathcal{S}} \stackrel{\text{def}}{=} \Sigma \times \Sigma$ to be the space of input/output pairs, $I_{\mathcal{S}}(\sigma, \sigma') = \sigma$ and $O_{\mathcal{S}}(\sigma, \sigma') = \sigma'$. The probability distribution on $D_{\mathcal{S}}$ is induced by the standard semantics of P. For example, if P is the program `y := coin` then we have:

   $$\begin{array}{lcll} p(\sigma, \sigma') & = & p(\sigma)/2 & \text{if } \sigma' = \sigma[y \mapsto 0] \text{ or } \sigma' = \sigma[y \mapsto 1] \\ p(\sigma, \sigma') & = & 0 & \text{otherwise.} \end{array}$$

   The obvious input and output observations to consider in this setting are projections on the state, that is, observations of the values of one or more program variables. For any program variable `x`, the corresponding observation is given by $X : \sigma \mapsto \sigma(x)$.

3.2 Information flow

We are interested in flows of information from system inputs to system outputs. We use some input observation $X : \mathcal{R}(I_{\mathcal{S}}) \to \mathcal{R}(X)$ and output observation $Y : \mathcal{R}(O_{\mathcal{S}}) \to \mathcal{R}(Y)$, as defined in Section 3.1, to pick out the parts of the input and output we wish to focus on. In this context, we refer to X as the *information source*. In the rest of this section, assume given some transformational system $\mathcal{S}$.

A natural information-theoretic quantity to view as the flow from X to Y is the mutual information between the two corresponding random variables: $\mathcal{I}(X^{\text{in}}; Y^{\text{out}})$. We say this seems natural because it is a direct formalization of the idea that the quantity of information flowing from X to Y is the amount of information given by input observation X which is shared with output observation Y.

However, despite its intuitive appeal, this formalisation is flawed as it stands. To see why, consider the following case: $Y = X \text{ XOR } Z$ (true when exactly one of the arguments is true) with X and Z independent random variables uniformly distributed over the Booleans. Since Y is the value of a function with argument X, and since variation in X clearly can cause variation in Y, we might expect the presence of an information flow from X to Y, But this is not shown in $\mathcal{I}(X; Y)$; indeed we have

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

At first sight this is surprising but the explanation is straightforward: XOR is here providing perfect encryption of X, with Z as the key. An observer can learn nothing about X from Y *provided the observer does not know* Z. This shows very clearly that a satisfactory definition of flow must take account of the observer's prior knowledge of the context. The right way to do this is via *conditional*

mutual information. In the XOR example, if we assume knowledge of Z and account for this by conditioning on Z, we find

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

By applying standard information theory equalities, we find that the rhs is $\mathcal{H}(X) - \mathcal{H}(Y|Z,Y) = 1 - 0 = 1$.

Prior knowledge may also *decrease* the flow of information as measured by an observer. For example, an observer of the swap system who already knows Z learns nothing new about $\langle X, Z\rangle$ by observing $\pi_1(\text{swap}(X,Z))$, whereas an observer who knew nothing to start with would measure a flow of $\mathcal{H}(Z)$ bits.

We therefore modify our definition of information flow thus. Let X and Z be input observations, let Y be an output observation. Then we define the information flow from X to Y given knowledge of Z as

$$\mathcal{F}_Z(X \rightsquigarrow Y) \stackrel{\text{def}}{=} \mathcal{I}(X^{\text{in}}; Y^{\text{out}}|Z^{\text{in}}). \tag{3.1}$$

There are two natural questions arising from this basic definition:

- What is the meaning of $\mathcal{F}_Z(X \rightsquigarrow Y) = 0$? This captures the special case when *no* information flows from X to Y. In Corollary 1 we show that, in the deterministic case, this is equivalent to the standard notion of non-interference between X and Y (provided X,Z constitute the whole input).
- What is the meaning of $\mathcal{F}_Z(X \rightsquigarrow Y) = n$ for $n > 0$? First, what is the maximum value n can take, i.e. how much information is initially 'undiscovered'? One of the basic information theoretic inequalities is $\mathcal{I}(U;V|W) \leq \min(\mathcal{H}(U|W), \mathcal{H}(V|W))$. So we know that, whatever the details of how the system transforms inputs to outputs, and whatever the choice of output observation, $n \leq \mathcal{H}(X^{\text{in}}|Z^{\text{in}})$. When n achieves this maximum, the observation is revealing everything: *all* the undiscovered information in X is flowing to Y. When n falls short of the maximum, the observation is incomplete, leaving $(\mathcal{H}(X^{\text{in}}|Z^{\text{in}}) - n)$ bits of information still unknown. One possible operational interpretation of this 'gap' is that it provides a measure of how hard it remains to guess the actual value of X once Y is known. This is formalized in [22] (note, however, that this 'guessing game' is an idealized one in which it is assumed that the encoding of information about X in Y is invertible in constant time; it has nothing to say about the computational difficulty of recovering knowledge of X from Y when this does not hold).

3.3 Deterministic information flow

We now restrict attention to the case of *deterministic* systems, by which we mean systems for which there exists a function f such that $O_{\mathcal{S}} = f \circ I_{\mathcal{S}}$. This will be the case, for example, in systems defined by programs written in a simple imperative language (without non-deterministic constructs) or in a functional language (see Section 6). Now consider flows of the form $\mathcal{F}_Z(X \rightsquigarrow Y)$ in the special case that observations X and Z jointly determine the inputs, i.e. $\langle X, Z\rangle$ is injective on inputs, or, equivalently, $\langle X, Z\rangle^{\text{in}} \simeq I_{\mathcal{S}}$. For example, in a security setting we may be interested in flows of the form $\mathcal{F}_L(H \rightsquigarrow L)$ where program variables are partitioned into the high-security set (input observation H) and the low-security set (input observation L). Such a flow measures what a low-security observer (who can only observe low-security variables) can learn about high-security inputs as a result of information flow into the low-security outputs. Since H, L partition the set of all program variables they jointly provide a complete observation of the input.

As shown by the following proposition, this special case allows a simplified definition of flow:

PROPOSITION 3.1
Assume a deterministic system $\mathcal{S}$. Let X and Z be input observations and let Y be an output observation. If $\langle X, Z\rangle$ is injective on $\mathcal{R}(I_{\mathcal{S}})$ then:

$$\mathcal{F}_Z(X \rightsquigarrow Y) = \mathcal{H}(Y^{\text{out}}|Z^{\text{in}}). \tag{3.2}$$

PROOF. By determinism and injectivity of $\langle X, Z\rangle$, we have $Y^{\text{out}} = f \circ \langle X^{\text{in}}, Z^{\text{in}}\rangle$, for some f. In what follows, let $A = X^{\text{in}}, B = Y^{\text{out}}, C = Z^{\text{in}}$. By the definitions (see (3.1) and Section 2), we must show $\mathcal{H}(A|C) + \mathcal{H}(B|C) - \mathcal{H}(A, B|C) = \mathcal{H}(B|C)$, that is, we must show $\mathcal{H}(A|C) = \mathcal{H}(A, B|C)$. Expanding both sides according to the definitions, we must show $\mathcal{H}(A, B, C) - \mathcal{H}(C) = \mathcal{H}(A, C) - \mathcal{H}(C)$. But $B = f \circ \langle A, C\rangle$ implies $\langle A, B, C\rangle \simeq \langle A, C\rangle$, so we are done. ∎

Given its relative simplicity, it may be tempting to consider (3.2) as an alternative *general* definition of flow. However, in general, it is not adequate. Consider again the example program `y := coin` from Section 3.1.1. Consider some other program variable `x` (the choice is arbitrary) and define $X : \sigma \mapsto \sigma(\mathtt{x})$ and $Y : \sigma \mapsto \sigma(\mathtt{y})$ and let Z be any input observation. Clearly, no information flows from X to Y, since the value assigned to `y` does not depend on any part of the store. This is confirmed using (3.1), which gives a flow of $\mathcal{I}(X^{\text{in}}; Y^{\text{out}}|Z^{\text{in}}) = 0$ (one of the basic identities of information theory, since X^{in} and Y^{out} are independent). By contrast, applying (3.2) would give a flow of $\mathcal{H}(Y^{\text{out}}|Z^{\text{in}}) = \mathcal{H}(Y^{\text{out}}) = \frac{1}{2}\log 2 + \frac{1}{2}\log 2 = 1$.

Another striking difference between the specialized setting and the more general probabilistic setting, is that in the specialized case an upper bound on the flow into a collection of outputs can be determined by considering the outputs separately.

PROPOSITION 3.2
Let $\mathcal{S}$ be a deterministic system. Let X and Z be input observations. Let Y_1 and Y_2 be output observations and let $Y = \langle Y_1, Y_2\rangle$ (thus Y is also an output observation). If $\langle X, Z\rangle$ is injective on inputs then $\mathcal{F}_Z(X \rightsquigarrow Y) \leq \mathcal{F}_Z(X \rightsquigarrow Y_1) + \mathcal{F}_Z(X \rightsquigarrow Y_2)$.

PROOF. By Proposition 3.1, it suffices to establish the general inequality

$$\mathcal{H}(A, B|C) \leq \mathcal{H}(A|C) + \mathcal{H}(B|C). \tag{3.3}$$

It is easy to show (for example, by Venn diagram, see [45]) that $\mathcal{H}(A, B|C) = \mathcal{H}(A|C) + \mathcal{H}(B|C) - \mathcal{I}(A; B|C)$. Since all of the Shannon measures are non-negative, the inequality follows. ∎

But the conclusion of this proposition does *not* hold in general when the injectiveness condition is dropped. The reason is essentially the one used to motivate the use of conditional mutual information in the definition of flow in Section 3.2: knowledge of one input can *increase* the apparent flow from another. Consider the program $\lambda(x, z).(x \text{ XOR } z, z)$ defining a deterministic system with $\mathcal{R}(I_{\mathcal{S}}) = \mathcal{R}(O_{\mathcal{S}}) = \mathit{bool} \times \mathit{bool}$. Let X and Z be the input observations π_1 and π_2, respectively. Similarly, let Y_1, Y_2 be the output observations π_1, π_2. Now suppose the distribution for $\mathcal{S}$ is such that X^{in} and Z^{in} are independent and uniform. We are concerned with flows having X as the information source. Instead of taking Z as the observer's prior knowledge (which would satisfy the injectiveness condition of the proposition) take some *constant* function W (representing ignorance), in which case, injectiveness clearly fails. Conditioning on a constant has no effect, thus $\mathcal{I}(X^{\text{in}}; Y_i^{\text{out}}|W^{\text{in}}) = \mathcal{I}(X^{\text{in}}; Y_i^{\text{out}})$, hence $\mathcal{F}_W(X \rightsquigarrow Y_i) = \mathcal{I}(X^{\text{in}}; Y_i^{\text{out}})$. Simple calculations then give the following:

- $\mathcal{F}_W(X \rightsquigarrow Y_1) = 0$
- $\mathcal{F}_W(X \rightsquigarrow Y_2) = 0$

- $\mathcal{F}_W(X \rightsquigarrow \langle Y_1, Y_2\rangle) = 1$.

So in this case it is *not* sufficient to calculate the flows to the outputs separately. The reason is clear: the two outputs are, respectively, a perfect encryption of X and its key. Observing either one by itself reveals nothing about X but observing both reveals everything.

4 Non-interference

In this section we consider only deterministic systems $\mathcal{S}$ and we assume that the inputs and outputs are structured as vectors, whose elements we refer to as the *components* (equivalently, we may take inputs and outputs to be finite maps whose domains index the components, as in the case of the store for a simple imperative language). Thus we are assuming the existence of a set of input observations $\{X_1, \ldots, X_n\}$ such that $I_{\mathcal{S}} = \langle X_1, \ldots, X_n\rangle^{\text{in}}$ and a set of output observations $\{Y_1, \ldots, Y_m\}$ such that $O_{\mathcal{S}} = \langle Y_1, \ldots, Y_m\rangle^{\text{out}} = f \circ \langle X_1, \ldots, X_n\rangle^{\text{in}}$, for some f. **Note:** it is a simple consequence of the definitions that $\langle X_1, \ldots, X_n\rangle^{\text{in}} = \langle X_1^{\text{in}}, \ldots, X_n^{\text{in}}\rangle$, and similarly for output observations.

In the setting of security, information flow is of particular relevance when considering confidentiality properties. Leakage of confidential information is a particular case of information flow where the source of information is a high-security part of the input and the target a low-security part of the output. In general when there is information flow from inputs to outputs, the inputs are said to *interfere* with the outputs, whereas the absence of any such flow is known 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 [16]. Roughly speaking, a deterministic program is said to satisfy non-interference if its low-security outputs depend only on its low-security inputs (hence *not* on its high-security inputs).

More formally, in the deterministic case, a generalized definition of non-interference can be formalized by modelling different categories of observation (e.g. high-security, low-security) by equivalence relations. Given relations R and S, a function f is said to *map R into S*, written $f : R \Rightarrow S$, iff $\forall x, x'.(x, x') \in R \Rightarrow (f(x), f(x')) \in S$. In what follows, R and S will always be equivalence relations, though the general construction (known as logical relations [35, 27]) does not require this. Given a set of components X, let $=_X$ be the equivalence relation which relates two inputs just when they agree on all components in X. Suppose that the input (resp. output) components are partitioned into the low security components L (resp. L') and the high-security components H (resp. H'). Then non-interference is defined as $f : (=_L) \Rightarrow (=_{L'})$ ('low-equivalence' is mapped into 'low-equivalence'). More generally, a collection of input components A *interferes* with an output component B iff $f : (=_{\overline{A}}) \not\Rightarrow (=_B)$, where $\overline{A}$ is the *complement* of A, i.e. all those input components *not* in A. (So $\overline{H} = L$ and H interferes with L precisely when $f : (=_L) \not\Rightarrow (=_{L'})$.)

In Section 5 we show how our quantitative approach to information flow can also be expressed in this relational form. Here we go on to explore the relationship between non-interference and information theory.

4.1 Non-interference and independence

First recall that two random variables X and Y are independent iff for all x, y, $P(X = x, Y = y) = P(X = x)P(Y = y)$. An immediate consequence of the definition is that for two independent random variables X and Y, $\mathcal{H}(Y, X) = \mathcal{H}(Y) + \mathcal{H}(X)$, which provides a proof for the following:

PROPOSITION 4.1
Random variables X and Y are independent iff $\mathcal{I}(Y; X) = 0$.

As the XOR example suggests, simple random variable independence is not enough to capture the absence of information flows. The correct probabilistic characterization of non-interference is via *conditional* independence.

PROPOSITION 4.2
Let Y be an output component, hence (given the assumptions of this section) $Y^{\text{out}} = f \circ \langle X_1, \ldots, X_n \rangle^{\text{in}}$ for some f. Assume a probability distribution such that[1], for all $(x_1, \ldots, x_n)$, $P(X_1^{\text{in}} = x_1, \ldots, X_n^{\text{in}} = x_n) \neq 0$. Let $i \leq n$. Then $X_1, \ldots, X_i$ are non-interfering with Y iff

$$\mathcal{I}(Y^{\text{out}}; X_1^{\text{in}}, \ldots, X_i^{\text{in}} | X_{i+1}^{\text{in}}, \ldots, X_n^{\text{in}}) = 0.$$

PROOF. **Note:** this proof uses notation and results from Section 5.

In the following we use A for $\{X_1, \ldots, X_i\}$ and Z for $\{X_{i+1}, \ldots, X_n\}$. Note that $Z = \overline{A}$. From Proposition 3.1 we know that $\mathcal{I}(Y^{\text{out}}; A^{\text{in}} | Z^{\text{in}}) = \mathcal{H}(Y^{\text{out}} | Z^{\text{in}})$ so all we have to prove is that A does not interfere with Y iff $\mathcal{H}(Y^{\text{out}} | Z^{\text{in}}) = 0$

($\Rightarrow$) : Assume A does not interfere with Y, i.e. $f : (=_Z) \Rightarrow (=_Y)$. Thus, by Lemma 5.3, $(=_Z) \subseteq \ker(Y^{\text{out}})$ and so, by Lemma 5.2, $\mathcal{H}(\text{var}(=_Z) | Z^{\text{in}}) \geq \mathcal{H}(Y^{\text{out}} | Z^{\text{in}})$. But $\text{var}(=_Z) \simeq Z^{\text{in}}$, hence $\mathcal{H}(\text{var}(=_Z) | Z^{\text{in}}) = \mathcal{H}(Z^{\text{in}} | Z^{\text{in}}) = 0$, hence $\mathcal{H}(Y^{\text{out}} | Z^{\text{in}}) = 0$.

($\Leftarrow$) : Assume A interferes with Y, i.e. there exist x, x', z such that $f(x, z) \neq f(x', z)$. Then given only the Z components we will not know the value Y^{out} will yield, i.e. we will have uncertainty in Y^{out} given Z^{in}, i.e. $\mathcal{H}(Y^{\text{out}} | Z^{\text{in}}) > 0$. ∎

COROLLARY 4.3

1. $\mathcal{F}_{\overline{A}}(A \rightsquigarrow Y) = 0$ iff A does not interfere with Y.
2. When $n = 1$ we have non-interference iff $\mathcal{I}(X^{\text{in}}; Y^{\text{out}}) = 0$, that is, iff X^{in} and Y^{out} are independent random variables.

5 Relational information flow

In this section we begin by observing the (standard) correspondence between equivalence relations and surjective maps. We go on to explore how this correspondence can be used to derive a relational presentation of information for deterministic transformational systems.

5.1 Equivalence relations and surjective maps

As observed in Section 2, random variables with identical sets of inverse images necessarily have the same entropy. 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 input (in D) belongs to $X^{-1}(x)$. Thus X effectively partitions the underlying space D into sets whose elements are indistinguishable by an observer who only sees the value of X. We can state this relationally. Given $X : D \to \mathcal{R}(X)$, the *kernel* of X is defined to be the equivalence relation $\ker(X)$ on D, defined thus:

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

The inverse images of X are the equivalence classes of $\ker(X)$, hence:

[1]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.

LEMMA 5.1
$X \simeq Y$ iff $\ker(X) = \ker(Y)$.

Conversely, any equivalence relation R on a set D induces an obvious surjective map from D onto its quotient by R. We write D/R for the set of equivalence classes of D wrt R and we write $[d]_R$ for the R-equivalence class $\{d' \in D | (d, d') \in R\}$. We write the quotient map for R as $[\_]_R : D \to D/R$. Clearly, $\ker([\_]_R) = R$.

We note that the above correspondences form the basis of the work presented in [19], which explicitly identifies the informal notion of information with the formal one of equivalence relations. However, that work is not quantitative and makes no use of information theory.

5.2 Equivalence relations and information flow

For the rest of this section assume given a deterministic transformational system $\mathcal{S}$ in which $O_{\mathcal{S}} = f \circ I_{\mathcal{S}}$. Let $D = \mathcal{R}(I_{\mathcal{S}})$, thus $f : D \to \mathcal{R}(f)$, with D finite. Now, given an equivalence relation R on D, we may define a corresponding random variable:

$$\mathrm{var}^{\mathrm{in}}(R) \stackrel{\mathrm{def}}{=} [\_]^{\mathrm{in}}_R.$$

Similarly, given S on $\mathcal{R}(f)$, we define:

$$\mathrm{var}^{\mathrm{out}}(S) \stackrel{\mathrm{def}}{=} [\_]^{\mathrm{out}}_S.$$

Where the domain of a relation R (either D or $\mathcal{R}(f)$) is clear from the context, we will drop the superscript and write just $\mathrm{var}(R)$.

Since equivalence relations are naturally combined by intersection and compared by inclusion, we may ask what this means for the corresponding random variables.

LEMMA 5.2
Let R_1, R_2 be equivalence relations, either both on D or both on $\mathcal{R}(f)$. Let V, W be random variables (on $D_{\mathcal{S}}$). Then:

1. $\mathrm{var}(R_1 \cap R_2) \simeq \langle \mathrm{var}(R_1), \mathrm{var}(R_2) \rangle$.
2. $R_1 \subseteq R_2 \Rightarrow \mathcal{H}(\mathrm{var}(R_1)) \geq \mathcal{H}(\mathrm{var}(R_2))$.
3. $R_1 \subseteq R_2 \Rightarrow \mathcal{H}(\mathrm{var}(R_1)|W) \geq \mathcal{H}(\mathrm{var}(R_2)|W)$.
4. $R_1 \subseteq R_2 \Rightarrow \mathcal{I}(\mathrm{var}(R_1); V|W) \geq \mathcal{I}(\mathrm{var}(R_2); V|W)$.

PROOF. Part 1 is more or less immediate from the definitions. Since $R_1 \subseteq R_2$ implies $R_1 \cap R_2 = R_1$, we have (by Part 1) $\mathcal{H}(R_1) = \mathcal{H}(\mathrm{var}(R_1), \mathrm{var}(R_2))$ and Part 2 follows from the standard inequality $\mathcal{H}(X, Y) \geq \mathcal{H}(Y)$. Parts 1 and 2 entail $\mathcal{H}(\mathrm{var}(R_1) \geq \mathcal{H}(\mathrm{var}(R_1), \mathrm{var}(R_3))$ from which Part 3 follows, using $\mathcal{H}(X|Y) = \mathcal{H}(X, Y) - \mathcal{H}(Y)$. ∎

Return now to the relational definition of non-interference in Section 4. Given any equivalence relation S on $\mathcal{R}(f)$, it is easily seen that there is a *coarsest possible* equivalence relation on D with respect to which f satisfies non-interference. We denote this relation $f^{-1}(S)$, defined thus:

$$d \; f^{-1}(S) \; d' \text{ iff } f(d) \; S \; f(d').$$

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

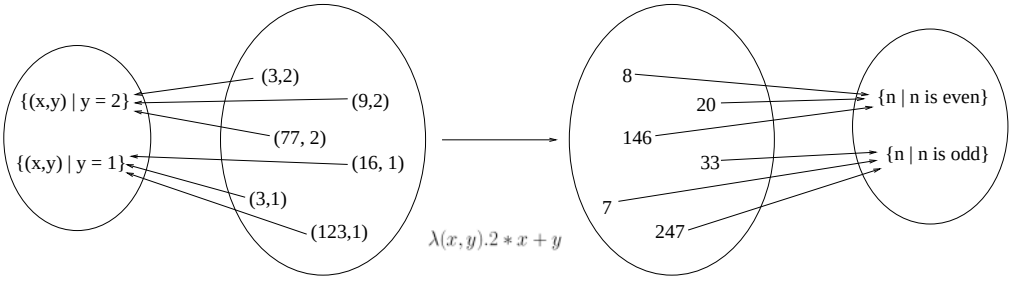

FIGURE 1. Equivalence relation and its inverse image

LEMMA 5.3
Let $f : D \to \mathcal{R}(f)$ with R and S equivalence relations on A and B, respectively. Then $f : R \Rightarrow S$ iff $R \subseteq f^{-1}(S)$.

Figure 1 illustrates the idea. Here D is a small set of pairs of integers in which the second component is always either 1 or 2. The function f is $\lambda(x, y).2 * x + y$ and S (illustrated on the right) relates integers with equal parity. The inverse image of S (illustrated on the left) relates pairs whose second components are equal and it is easily seen that this is the coarsest R such that $f : R \Rightarrow S$.

It is a straightforward consequence of the above correspondences and definitions that random variable $\mathrm{var}^{\mathrm{out}}(S)$ captures exactly the same information as $\mathrm{var}^{\mathrm{in}}(f^{-1}(S))$:

LEMMA 5.4
$\mathrm{var}^{\mathrm{in}}(f^{-1}(S)) \simeq \mathrm{var}^{\mathrm{out}}(S)$.

We can now present quantitative flow in relational form for deterministic systems. The following theorem tells us that, to calculate an upper bound on the information flow through a function f, it suffices to approximate an output observation with kernel S by an *input* observation with kernel R provided we can establish $f : R \Rightarrow S$. In the next section we see how this can be applied in a concrete example in the second-order lambda calculus.

THEOREM 5.5
Let R, S be equivalence relations such that $f : R \Rightarrow S$. Let V, W be random variables (on D_S). Then

$$\mathcal{I}(V; \mathrm{var}^{\mathrm{in}}(R)|W) \geq \mathcal{I}(V; \mathrm{var}^{\mathrm{out}}(S)|W).$$

PROOF. By Lemmas 5.2, 5.3 and 5.4. ∎

COROLLARY 5.6
Let X, Z be input observations and let Y be an output observation. If $f : R \Rightarrow \ker(Y)$ then $\mathcal{F}_Z(X \rightsquigarrow Y) \leq \mathcal{I}(X^{\mathrm{in}}; \mathrm{var}^{\mathrm{in}}(R)|Z^{\mathrm{in}})$.

6 Information flow and polymorphic types

In this section we illustrate how, combining Theorem 5.5 with the principle of parametricity, polymorphic types can be used to derive bounds on information flow which are valid for all lambda terms sharing the type.

The second-order lambda calculus [14, 15, 28] (also known as System F) extends the type language of the simply typed calculus with type variables and a universal quantifier and extends the

term language with abstraction over type variables and application to types. The syntax is standard and we do not reproduce it here. For convenience, we use a syntax extended with useful base types (such as the integers `int` and Booleans `bool`) and list types $<T>$ for each type T (no new theory is required as all these types are definable in the calculus). In examples, we omit explicit types from terms and elide the application of terms to types (in fact, our examples all stay within the Hindley–Milner fragment [18, 24, 10], for which principal types may always be inferred). We refer to functions definable in the second-order lambda calculus as $\lambda 2$ functions.

6.1 Parametricity

The key to the results of this section is Reynolds' relational treatment of parametric polymorphism [28], as applied to specific functions [43]. In Reynolds' treatment, each term e is given an essentially standard funtional semantics $[\![e]\!]$ whereas each type T is given a *relational* interpretation $[\![T]\!]$.

Reynolds shows that, in such models (e.g. Frame models [3, 25]), $e : T$ implies $([\![e]\!], [\![e]\!]) \in [\![T]\!]$. This is known as *relational parametricity*. In the following we give a quick summary of Wadler's presentation [43].

The relational interpretation of basic types is the identity relation on the semantics of those types; `bool` is for example interpreted by $\mathbf{I}_{\texttt{bool}} \subseteq [\![\texttt{bool}]\!] \times [\![\texttt{bool}]\!]$.

The relational interpretation of $\rightarrow$ is that functions are related if they map related arguments to related values. Thus $(f, g) \in (R \rightarrow S)$ iff $\forall (x, x') \in R.\ (f\,x, g\,x') \in S$. (Note that $f : R \Rightarrow S$ (Section 4) iff $(f, f) \in R \rightarrow S$.) The (derived) relational interpretation of list types is that $(l, l') \in <R>$ iff there exists n such that $l = <x_1, \ldots, x_n>$ and $l' = <x'_1, \ldots, x'_n>$ and $\forall i.(x_i, x'_i) \in R$ (viewing a list as a map from $\{0, \ldots, n\}$ to a set of values, the above definition $(l, l') \in <R>$ is hence the same as $(l, l') \in (Id_{\{0,\ldots,n\}} \rightarrow R)$).

To give a relational interpretation of $\forall$ we start by considering a function from relations to relations (these are intended as relations over the universe of types in a frame model) $\mathcal{F}(\mathcal{X})$, i.e. for any relation $\mathcal{A} \subseteq A \times A'$, $\mathcal{F}(\mathcal{A}) \subseteq F(A) \times F(A')$ is a relation ($F(A) \times F(A')$ is the support of the relation image of $\mathcal{A}$ under $\mathcal{F}$). Then $(f, g) \in \forall\mathcal{X}.\mathcal{F}(\mathcal{X})$ iff for any relation $\mathcal{A} \subseteq A \times A'$, $(f_A, g_{A'}) \in \mathcal{F}(\mathcal{A})$. You can see (f, g) as a map, $(f, g)(\mathcal{A}) = (f_A, g_{A'}) \in \mathcal{F}(\mathcal{A})$. The meaning of the definition is that 'type abstractions are related if they map related types into related results'.

Consider the $\lambda 2$ function *double* : $\forall\alpha.(\alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha$, defined as[2] $\lambda f.\lambda x.f\,(f\,x)$. Now suppose $f : T \rightarrow T$ and R are such that $f : R \Rightarrow R$. We have then (*double* f) $: R \Rightarrow R$, since $(x, x') \in R$ implies $(f\,x, f\,x') \in R$ implies $(f\,(f\,x), f\,(f\,x')) \in R$. However, parametricity allows us to derive this property solely from the type, thus extending it to *all* $\lambda 2$ functions δ with type $\forall\alpha.(\alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha$. The proof is straightforward. Substitute R for α, giving.

$$(\delta, \delta) \in (R \rightarrow R) \rightarrow R \rightarrow R. \tag{6.1}$$

Assume $f : R \Rightarrow R$ (that is, $(f, f) \in R \rightarrow R$). We need to show $(\delta\,f, \delta\,f) \in R \rightarrow R$; this is immediate from (6.1) and the definition of $\rightarrow$ on relations. By reiterating this style of argument it is possible to show that all elements of type $\forall\alpha.(\alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha$ are *iterators*, i.e. Church numerals.

[2] In order to keep notation light here and elsewhere we will ignore type quantifiers and type applications, i.e. $\lambda f.\lambda x.f\,(f\,x)$ stands really for $\Lambda\alpha\lambda f.\lambda x.f\,(f\,x)$. Similarly (*double* f) stands for (*double*$[T]\,f$).

6.2 Calculating bounds on flows

In the remainder of this section we present two examples showing how the relational interpretation of types can be exploited to derive both upper and lower bounds on information flow through functions with polymorphic types. In both examples, bounds are derived which hold for *all* functions with the given polymorphic type. In the examples we implicitly treat each function f of interest as a deterministic transformational system. This involves restricting the functions (which, in full generality, are defined over infinite domains) to some finite input set D of interest and then specifying a distribution on that set in the form of a random variable X with range D. For simplicity, we consider only observers with no prior knowledge of the inputs and we assume the input and output observations to be the identity, i.e. the observer can see the whole output and we measure the total flow of information from the whole input. Formally then, the flows we consider have the form $\mathcal{F}_Z(\text{id} \rightsquigarrow \text{id})$, with Z a constant function. Applying Proposition 3.1 shows that this reduces to $\mathcal{H}(f \circ X)$. We slightly abuse the notation, writing this more suggestively as $\mathcal{F}(X \rightsquigarrow f(X))$.

6.2.1 Example: the *map* family

Recall the *map* function with type

$$\forall\alpha.\forall\beta.(\alpha \to \beta) \to <\alpha> \to <\beta>.$$

By the *map* family we mean the set of all $\lambda 2$ functions sharing this type. The *map* function itself maps a function f over a list, thus:

$$\mathit{map}\ f <a_1, \ldots, a_n> = <f\ a_1, \ldots, f\ a_n>.$$

Consider that we have a secret b of some primitive type and a program which attempts to guess its value by evaluating $c = b$ for some value c. Note that even the outcome `false` reveals *some* information about the secret, since it narrows down the possibilities. However, if the possible values of b form a large, uniformly distributed set, a single guess reveals only a small amount of information. This is essentially the example of attempting to guess a PIN number: if there are sufficient digits and PIN numbers are well distributed, a guess is highly unlikely to succeed and leaves the set of possible values almost (but not quite) unchanged.

We can use *map* in two ways to iterate the basic guessing program:

1. Given a list of secrets l as input, we can try a given guess c on all of them by mapping $(=c)$ over l: $\lambda l.\mathit{map}\ (=c)\ l$.
2. Given a secret b as input, we can test it against a fixed list of guesses by mapping $(=b)$ over the list: $\lambda b.\mathit{map}\ (=b)\ \langle c_1, \ldots, c_n\rangle$.

Here we consider just the first iteration (though the second gives essentially the same results, the form of reasoning is closer to that used in the *fold* example - see below). In what follows we derive results which allow us to place an upper bound on the amount of information revealed. Concretely, for secrets drawn from a set of 2^{32} possibilities, each with equal probability, and for lists of one million elements, the information flowing through $\lambda l.\mathit{map}\ (=c)\ l$ is less than 0.01 bits. Moreover, parametricity immediately generalizes this result to *all* similar programs $\lambda l.\mu\ (=c)\ l$ using other members μ of the *map* family.

For the remainder of this section, let μ be any member of the *map* family.

We assume all lists of secrets input to the program to have the same length, n and we assume their elements to be chosen independently from a finite set. We may thus assume $I_{\mathcal{S}}$ to have the

form $X = \langle X_1, \ldots, X_n \rangle$. It is easily seen that, for all c, $(=c) : R_{=c} \Rightarrow \mathrm{Id}$, where $R_{=c} \stackrel{\text{def}}{=} \{(c,c)\} \cup \{(a,a') | a \neq c \neq a'\}$. Thus, instantiating the type of μ and noting that $<\mathrm{Id}>$ is just Id (on lists), we have

$$\mu\,(=c) : <R_{=c}> \Rightarrow \mathrm{Id}.$$

Note that $R_{=c}$ has just two equivalence classes, which we will denote $[c] \stackrel{\text{def}}{=} \{c\}$ and $[\overline{c}] \stackrel{\text{def}}{=} \{a \in T | a \neq c\}$.

Theorem 5.5 then tells us that the total flow is bounded above by the entropy of the input distribution quotiented by $<R_{=c}>$. Quotienting by $<R_{=c}>$ gives $\langle X'_1, \ldots, X'_n \rangle$ where each X'_i is derived from X_i by quotienting its range by $R_{=c}$. Theorem 5.5, together with the fundamental inequality $\mathcal{H}(Y,Z) \leq \mathcal{H}(Y) + \mathcal{H}(Z)$, then yields

$$\mathcal{F}(X \rightsquigarrow (\mu\,(=c))(X)) \leq \sum_{1 \leq i \leq n} \mathcal{H}(X'_i). \tag{6.2}$$

As shown in [5], we may calculate bounds on $\mathcal{H}(X'_i)$ knowing only $\mathcal{H}(X_i)$ and the cardinality of T. Specifically:

$$\mathcal{H}(X_i) \leq \mathcal{H}(X'_i) + P(X'_i = [\overline{c}]) \log(|T| - 1). \tag{6.3}$$

(A derivation is given in [5] but, in fact, it turns out to be just a special case of Fano's inequality [9].) At first sight, this seems to promise a lower bound, rather than an upper bound, on $\mathcal{H}(X'_i)$. However, let $q_i \stackrel{\text{def}}{=} P(X'_i = [c])$ and note that $\mathcal{H}(X'_i) = \mathcal{H}(q_i, 1-q_i)$; thus (6.3) may be rewritten

$$\mathcal{H}(X_i) \leq U(q_i), \tag{6.4}$$

where $U(q_i) \stackrel{\text{def}}{=} \mathcal{H}(q_i, 1-q_i) + (1-q_i)\log(|T|-1)$. Now we obtain an upper bound by maximizing $\mathcal{H}(q_i, 1-q_i)$ over all q_i for which (6.4) holds.

For an example of how this works, see Figure 2. The figure plots $\mathcal{H}(X'_i)$ and $U(q_i)$ against q_i for the case $|T| = 2^4$. As shown, if $\mathcal{H}(X_i) \geq 3.75$, then $q_i \leq 0.25$. Furthermore, for $q_i \leq 0.25$, $\mathcal{H}(X'_i)$ takes its maximum value (~ 0.81) when $q_i = 0.25$.

Return now to the concrete instance described at the start of this section. We suppose our inputs to be lists of length 10^6 with each element a secret chosen uniformly from a set of 2^{32} possibilities. Then $|T| = 2^{32}$ and $\mathcal{H}(X_i) = 32$ for each X_i. Some simple numerical approximations then derive $\mathcal{H}(X'_i) \leq 7.79 \times 10^{-9}$, hence, by (6.2), $\mathcal{F}(X \rightsquigarrow (\mu\,(=c))(X)) \leq 0.0079$.

6.2.2 Example: the *fold* family

In the previous example, note that we needed a *lower* bound on $\mathcal{H}(X_i)$ to derive an upper bound on $\mathcal{H}(X'_i)$. Thus, when analysing quantitative flows, we may need to be able to derive lower bounds for flows through sub-programs, even if we are interested overall only in upper bounds (as in the case of security, for example).

In this example we show how parametricity can be used to establish that all members of the *fold* family (see below) preserve a lower-bound information flow property of addition. Recall the function *foldr* with type

$$\forall\alpha\forall\beta.(\alpha \to \beta \to \beta) \to <\alpha> \to \beta \to \beta.$$

By the *fold* family we mean the set of all $\lambda 2$ functions sharing this type. The *foldr* function itself is defined such that, for any $\odot$:

$$\mathit{foldr}\,(\odot) <a_1, \ldots, a_n> y = a_1 \odot (a_2 \odot (\cdots (a_n \odot y) \cdots)).$$

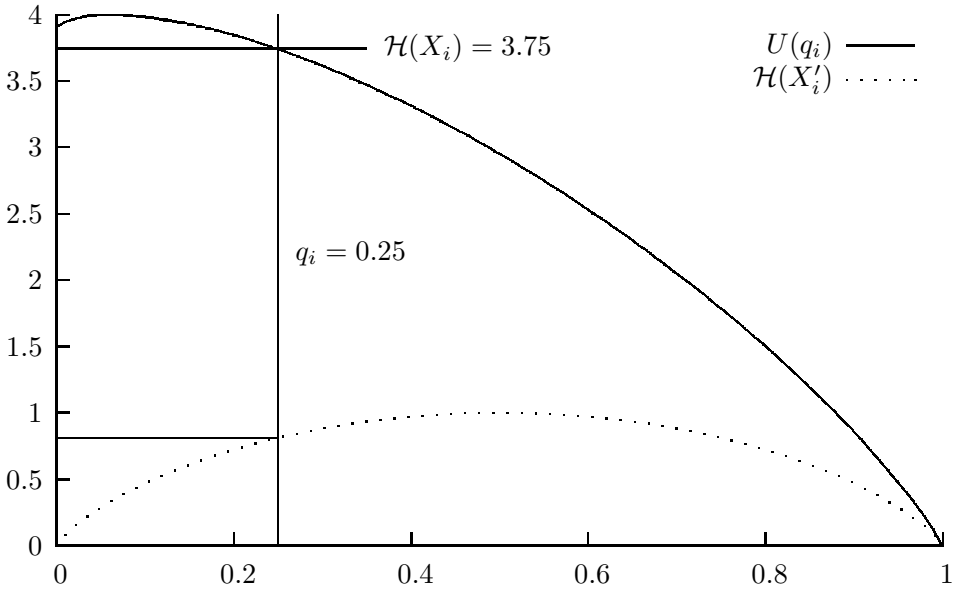

FIGURE 2. deriving an upper bound on $\mathcal{H}(X_i')$

We will use the specific properties of *foldr* to derive a property which holds for *all* members of the *fold* family. In the remainder of this section let ϕ be any such function.

Consider an application such as $\phi\,(+)\,l$, where l is a constant list of integers. What can we say about the flow $\mathcal{F}(Y \rightsquigarrow \phi\,(+)\,l\,Y)$? The Data Processing theorem [9] states that the information obtainable from the outputs of a deterministic system cannot exceed that obtainable from its inputs. Thus we know that

$$\mathcal{F}(Y \rightsquigarrow \phi\,(+)\,l\,Y) \leq \mathcal{H}(Y), \tag{6.5}$$

where Y is the random variable corresponding to the integer input. This leaves open the possibility that entropy is lost; we will show that it is not.

As shown by Theorem 5.5, properties of the form $f : R \Rightarrow S$ allow us to derive upper bounds on the amount of information which flows through f. However, such properties do *not* allow us to place lower bounds on the information flow. Lemma 5.3 shows that $f : R \Rightarrow S$ is equivalent to $R \subseteq f^{-1}(S)$. To derive lower bounds on entropy we need properties of the dual form, $R \supseteq f^{-1}(S)$. This is the route we take to showing that (6.5) can be strengthened to an equality.

First we show that *foldr* in particular preserves 'post fixpoint' properties of the form $R \supseteq f^{-1}(R)$.

LEMMA 6.1
Let $\odot$, R be such that, for all a, $R \supseteq (\odot\, a)^{-1}(R)$. Then for all l, $R \supseteq (\textit{foldr}\,(\odot)\,l)^{-1}(R)$.

PROOF. By induction on the length of l. When l is empty we have $\textit{foldr}\,(\odot)\,[\,] = \mathrm{id}$ and clearly $R \supseteq \mathrm{id}^{-1}(R)$. For the inductive step, observe that $\textit{foldr}\,(\odot)\,(x : l) = (\odot\, x) \circ (\textit{foldr}\,(\odot)\,l)$ and then use the (easily verified) fact that $R \supseteq f^{-1}(R)$ and $R \supseteq g^{-1}(R)$ implies $R \supseteq (f \circ g)^{-1}(R)$. ∎

Now we use parametricity to lift this result to all members of the *fold* family.[3]

[3]In fact the result can be extended further, since it relies only on polymorphism in type β (see [13]).

PROPOSITION 6.2
Let $\odot$, R be such that, for all a, $R \supseteq (\odot\, a)^{-1}(R)$. Then, for all l, $R \supseteq (\phi\,(\odot)\, l)^{-1}(R)$.

PROOF. As shown in [43] (Section 3.6), parametricity ensures the following equation, for arbitrary $\odot$, l:

$$\phi\,(\odot)\, l = \mathit{foldr}\,(\odot)\,(\phi\,(:)\, l\,[\,])$$

where $(:)$ and $[\,]$ are the list 'cons' operator and empty list, respectively. Since $\phi\,(:)\, l\,[\,]$ is a list uniquely determined by l, the result follows immediately from the lemma. ∎

From the proposition it follows easily that $\mathcal{F}(Y \rightsquigarrow \phi\,(+)\, l\, Y) = \mathcal{H}(Y)$. The key observation is that $\mathcal{F}(Y \rightsquigarrow \phi\,(+)\, l\, Y) = \mathcal{H}(Y)$ if $(\phi\,(+)\, l)$ is injective. But $(+\, a)$ is injective, for all a, and, in general, f is injective iff Id $\supseteq f^{-1}$(Id), so the proposition applies.

7 Conclusions

We have defined a quantitative measure of information flow, $\mathcal{F}_Z(X \rightsquigarrow Y)$ as the information theoretic quantity $\mathcal{I}(X^{\mathrm{in}}; Y^{\mathrm{out}} | Z^{\mathrm{in}})$, sufficiently general to account for flows in finite-domain systems featuring probabilistic non-determinism. For deterministic systems we have proved that, in the special case when X, Z completely determine the input, this quantity is the same as the simpler $\mathcal{H}(Y^{\mathrm{out}} | Z^{\mathrm{in}})$ and that in this context the flow into separate components of the output can safely be calculated one component at a time.

We have shown how equivalence relations can be interpreted as random variables 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 above-mentioned special case, non-interference is equivalent to a form of conditional probabilistic independence.

Finally, we have shown how Reynolds' notion of relational parametricity can be used to derive upper and lower bounds on information flows for families of functions sharing the same polymorphic type in the second order lambda calculus.

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 [32]). In view of the use of parametricity in Section 6, recent work by Tse and Zdancewic [38] suggests that there may be interesting connections with Abadi *et al.*'s dependency calculus [1].

Although security was the original motivation for this work, it 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 [12].

References

[1] M. Abadi, A. Banerjee, N. Heintze and J. G. Riecke. A core calculus of dependency. In *Proc. 26th Annual ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages, POPL'99*, pp. 147–160, San Antonio, Texas, USA, 1999. ACM Press.

[2] C. Bodei, P. Degano, H. Riis Nielson and F. Nielson. Static analysis for secrecy and non-interference in networks of processes. In *Proc. PACT'01*, number 2127 in Lecture Notes in Computer Science, pp. 27–41. Springer-Verlag, 2001.

[3] K. B. Bruce and A. R.Meyer. The semantics of second-order polymorphic lambda calculus. In *Semantics of Data Types*, Khan, MacQueen, and Plotkin, eds, volume 173 of *Lecture Notes in Computer Science*. Springer-Verlag, 1984.

[4] D. Clark, C. Hankin and S. Hunt. Information flow for Algol-like languages. *Computer Languages (Special Issue: Computer Languages and Security)*, **28**, 3–28, 2002.

[5] D. Clark, S. Hunt and P. Malacaria. Quantitative analysis of the leakage of confidential data. *Electronic Notes in Theoretical Computer Science*, **59**, 1–14, 2002.

[6] D. Clark, S. Hunt and P. Malacaria. Quantified interference for a While language. Technical Report TR-03-07, Department of Computer Science, King's College London, October 2003.

[7] D. Clark, S. Hunt and P. Malacaria. Quantified interference for a While language. In *Proceedings the Second Workshop on Quantitative Aspects of Programming Languages (QAPL04)*, A. Cerone and A. di Pierro, eds, Barcelona, April 2004.

[8] D. Clark, S. Hunt and P. Malacaria. Quantified interference: Information theory and information flow. In *Proceedings of the Fourth Workshop in Issues in the Theory of Security (WITS04)*, P. Y. A. Ryan, ed., Barcelona, April 2004.

[9] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley Interscience, 1991.

[10] L. Damas and R. Milner. Principal type-schemes for functional programs. In *Proc. 9th Annual ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages, POPL'82*, Albuquerque, New Mexico, USA, January 1982. ACM Press.

[11] D. E. R. Denning. *Cryptography and Data Security*. Addison-Wesley, 1982.

[12] R. Giacobazzi and I. Mastroeni. Abstract non-interference: parameterizing non-interference by abstract interpretation. In *Proc. 31st Annual ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages, POPL 2004*. ACM Press, 2004.

[13] A. Gill, J. Launchbury and S. L. Peyton Jones. A short cut to deforestation. In *Functional Programming and Computer Architecture, Copenhagen, FPCA'93*, pp. 223–232, Copenhagen, Denmark, June 1993.

[14] J.-Y. Girard. *Interprétation functionelle et élimination des coupures de l'arithmétique d'ordre supérieur*. PhD thesis, Université Paris VII, 1972.

[15] J.-Y. Girard, Y. Lafont and P. Taylor. *Proofs and Types*. Cambridge Tracts in Theoretical Computer Science, 7. Cambridge University Press, 1990.

[16] J. A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, pp. 11–20. IEEE Computer Society Press, 1982.

[17] J. W. Gray, III. Toward a mathematical foundation for information flow security. In *Proceedings of the 1991 IEEE Symposium on Security and Privacy*, pp. 21–34, Oakland, California, 1991.

[18] J. R. Hindley. The principal type scheme of an object in combinatory logic. *Transactions of the American Mathematical Society*, **146**, 29–60, 1969.

[19] J. Landauer and T. Redmond. A lattice of information. In *Proc. of the 6th IEEE Computer Security Foundations Workshop*, pp. 65–70, June 1993.

[20] G. Lowe. Quantifying information flow. In *Proceedings of the Workshop on Automated Verification of Critical Systems*, 2001.

[21] P. Malacaria and C. Hankin. Non-deterministic games and program analysis: an application to security. In *Proceedings of Logic in Computer Science (LICS)*. IEEE Press, 1999.

[22] J. L. Massey. Guessing and entropy. In *Proc. IEEE International Symposium on Information Theory*, Trondheim, Norway, 1994.

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

[24] R. Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, **17**, 348–375, 1978.

[25] J. C. Mitchell and A. R. Meyer. Second-order logical relations. In *Logics of Programs*, R. Parikh, ed., volume 193 of *Lecture Notes in Computer Science*. Springer-Verlag, 1985.

[26] A. Di Pierro, C. Hankin and H. Wiklicky. Approximate non-interference. In *CSFW'02 – 15th IEEE Computer Security Foundation Workshop*, I. Cervesato, ed. IEEE Computer Society Press, June 2002.

[27] G. Plotkin. Lambda definability and logical relations. Technical Report Memorandum SAI-RM-4, Department of Artificial Intelligence, University of Edinburgh, 1973.

[28] J. C. Reynolds. Types, abstraction and parametric polymorphism. *Information Processing*, **83**, 513–523, 1983.

[29] P. Y. A. Ryan, J. McLean, J. Millen, and V. Gilgor. Non-interference, who needs it? In *Proceedings of the 14th IEEE Security Foundations Workshop*, Cape Breton, Nova Scotia, Canada, June 2001. IEEE.

[30] A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communication, special issue on Formal Methods for Security*, **21**, 5–19, 2003.

[31] A. Sabelfeld and D. Sands. A per model of secure information flow in sequential programs. In *Proc. European Symposium on Programming*, Amsterdam, The Netherlands, March 1999. ACM Press.

[32] A. Sabelfeld and D. Sands. Probabilistic noninterference for multi-threaded programs. In *Proc. 13th IEEE Computer Security Foundations Workshop*, Cambridge, England, July 2000. IEEE Computer Society Press.

[33] C. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, **27**, 379–423 and 623–656, 1948. Available on-line at http://cm.bell-labs.com/cm/ms/what/shannonday/paper.html.

[34] G. Smith and D. Volpano. Secure information flow in a multi-threaded imperative language. In *Proc. 25th ACM Symposium on Principles of Programming Languages*, pp. 355–364, San Diego, CA, January 1998.

[35] R. Statman. Logical relations and the typed lambda calculus. *Information and Control*, **65**, 85–97, 1985.

[36] D. Sutherland. A model of information. In *Proceedings of the 9th National Computer Security Conference*, Washington, DC, September 1986.

[37] F. Topsøe. Entropy and codes. In *Eichstaetter Kolloquium zur Didaktik der mathematik*, volume 17, pp. 1–20. Katholische Universitaet Eichstaett, 2001. http://www.math.ku.dk/ma/kurser/informationsteori/entropy.ps.

[38] S. Tse and S. Zdancewic. Translating dependency into parametricity. Technical Report MS-CIS-04-01, Department of Computer and Information Science, University of Pennsylvania, 2004.

[39] D. Volpano and G. Smith. Eliminating covert flows with minimum typings. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop*, pp. 156–168, Rockport, MA, 1997.

[40] D. Volpano and G. Smith. A type-based approach to program security. In *Proceedings of TAPSOFT '97 (Colloquium on Formal Approaches in Software Engineering)*, pp. 607–621, number 1214 in *Lecture Notes in Computer Science*, Springer, 1997.

[41] D. Volpano and G. Smith. Verifying secrets and relative secrecy. In *Proc. 27th ACM Symposium on Principles of Programming Languages*, pp. 268–276, Boston, MA, 2000.

[42] D. Volpano, G. Smith and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, **4**, 167–187, 1996.

[43] P. Wadler. Theorems for free! In *Proc. 4th International Symposium on Functional Programming Languages and Computer Architecture*. Springer-Verlag, 1989.

[44] D. G. Weber. Quantitative hookup security for covert channel analysis. In *Proceedings of the 1988 Workshop on the Foundations of Computer Security*, Fanconia, New Hampshire, USA, 1988.

[45] Raymond W. Yeung. A new outlook on shannon's information measures. *IEEE Transactions on Information Theory*, **37**, 466–474, 1991.

Received 21 June 2004