Belief in Information Flow

Michael R. Clarkson Andrew C. Myers Fred B. Schneider
Department of Computer Science
Cornell University
{clarkson,andru,fbs}@cs.cornell.edu

Abstract

Measurement of information flow requires a definition of leakage, which traditionally has been defined to occur when an attacker's uncertainty about secret data is reduced. We show that this uncertainty-based approach is inadequate for measuring information flow when an attacker is making assumptions about secret inputs and these assumptions might be incorrect. Moreover, we show that such attacker beliefs are an unavoidable aspect of any satisfactory definition of leakage. To reason about information flow based on beliefs, we develop a model that describes how an attacker's belief changes due to the attacker's observation of the execution of a probabilistic (or deterministic) program. The model leads to a new metric for quantitative information flow that measures accuracy rather than uncertainty of beliefs.

1. Introduction

Qualitative security properties, such as noninterference [9], are inappropriate for some programs, because such properties typically either prohibit any flow of information from a high security level to a lower level, or they allow any amount of flow so long as it passes through some release mechanism. For a program whose correct functioning requires flow from high to low, the former approach is too restrictive and the latter can lead to unbounded leakage of information. Quantitative flow properties, such as "at most k bits leak per run of the program", allow violating flows but with restricted rates. Such properties characterize the security of programs whose nature require that some—but not too much—information be leaked. The prime examples of such programs are guards that sit at the boundary between trusted and untrusted systems, such as password checkers.

Defining the quantity of information flow is more difficult than it might seem. Consider a password checker PWC that sets an authentication flag a by checking a stored password p against a (guessed) password g supplied by the user.

$$PWC: \quad \textbf{if } p = g \textbf{ then } a := 1 \textbf{ else } a := 0$$

For simplicity, suppose that the password is either A, B, or C. Suppose also that the user is actually an attacker attempting to discover the password, and he believes the password is overwhelmingly likely to be A but has a minuscule and equally likely chance to be either B or C. (This need not be an arbitrary assumption on the attacker's part; perhaps the attacker was told by a usually reliable informant that the password is A.) If the attacker experiments by running the program and guessing A, he expects the outcome to be that a is equal to 1. Such a confirmation of the attacker's suspicion does seem to convey some small amount of information. But suppose that the informant was wrong: the real password is C. The outcome of this experiment has a equal to 0, from which the attacker infers that A is not the password. Common sense dictates that his new belief is that B and C each have a 50% chance of being the password. The attacker's belief is greatly changed—he is surprised to discover the password is not A—so this outcome seems to convey a larger amount of information than the previous outcome. Thus, the information conveyed by running PWC depends on what the attacker believes.

How much information flows from p to a in each of the above experiments? Answers to this question traditionally have been based on change in uncertainty [5, 19, 10, 1, 15, 2, 16]: information flow is measured by the reduction in the attacker's uncertainty about secret data. Observe that, in the case where the password was C, the attacker initially is quite certain (though wrong) about the value of the password and after the experiment is rather uncertain about the value of the password; the change from "quite certain" to "rather uncertain" is an increase in uncertainty. So according to the reduction in uncertainty metric, no information flow occurred, which flatly contradicts our intuition. The problem with reduction in uncertainty as a metric is that it does not take accuracy into account. Accuracy and uncertainty are orthogonal properties of the attacker's belief—being certain does not make one correct—and as the password checking example illustrates, the amount of information flow depends on accuracy rather than on uncertainty.

This paper presents a new way of measuring information

flow, based on this insight. Section 2 gives the basic representations and notations for beliefs and programs. Section 3 describes a model that is a precise characterization of the interaction between attackers and systems; it also describes how attackers update their beliefs from observing execution of programs. Section 4 defines a new quantitative flow metric, based on information theory, that successfully captures the amount of information flow due to changes in the accuracy of an attacker's belief. The model and metric are formulated for use with any programming language that can be given a denotational semantics that is compatible with the representation of beliefs, and Section 5 instantiates the model and metric for a particular programming language (**while**-programs plus probabilistic choice). Section 6 discusses related work, and Section 7 concludes.

2. Incorporating Beliefs

Our measure of information flow is based on accuracy of beliefs. A *belief* is a statement an agent makes about the state of the world, accompanied by some measure of how certain the agent is about the truthfulness of the statement. We begin by developing some mathematical structures for representing beliefs.

2.1. Distributions

A *frequency distribution* is a function δ that maps a program state to a *frequency*, where a frequency is a non-negative real number. A frequency distribution is essentially an unnormalized probability distribution over program states; frequency distributions are known to serve better than probability distributions as the basis for a programming language semantics [20]. Henceforth, we write "distribution" to mean "frequency distribution".

The set of all program states is **State**, and the set of all distributions is **Dist**. The structure of **State** is unimportant; it can be instantiated according to the needs of any particular language or system. For our examples, states are maps from variables to values, where domains **Var** and **Val** are both countable sets.

$$\begin{array}{lcl} v \in \mathbf{Var} & & \\ \sigma \in \mathbf{State} & \triangleq & \mathbf{Var} \to \mathbf{Val} \\ \delta \in \mathbf{Dist} & \triangleq & \mathbf{State} \to \mathbb{R}^+ \end{array}$$

The *mass* in a distribution δ is the sum of all the frequencies:

$$mass(\delta) \quad \triangleq \quad \textstyle\sum_\sigma \ \delta(\sigma)$$

A probability distribution has mass 1, but a frequency distribution may have any non-negative mass. A *point mass* is a distribution that maps a single state to 1, and all other states to 0. It is denoted by placing a dot over the state:

$$\dot\sigma \quad \triangleq \quad \lambda\sigma' \,.\, \text{if } \sigma' = \sigma \text{ then } 1 \text{ else } 0$$

2.2. Semantic Functions

Execution of program S is described by a denotational semantics in which the meaning of S is written $[\![S]\!]$, and is a function of type **State** $\to$ **Dist**. We also require that this semantics can be lifted to a function of type **Dist** $\to$ **Dist** by the following definition:

$$[\![S]\!]\delta \quad \triangleq \quad \lambda\sigma \,.\, \textstyle\sum_{\sigma'} \ \delta(\sigma') \cdot ([\![S]\!]\sigma')(\sigma)$$

This is a healthiness condition that requires the meaning of programs as distribution transformers to be determined by the meaning of programs as state transformers. By defining programs in terms of how they operate on distributions we permit analysis of probabilistic programs. Section 5 shows how to build a semantics of this form.

2.3. Labels and Projections

We need a way to identify secret data; *confidentiality labels* satisfy this need. For simplicity, we assume there are only two such labels: a label L that indicates low-confidentiality (public) data, and a label H that indicates high-confidentiality (secret) data. Assume that **State** is a product of two domains $\mathbf{State}_L$ and $\mathbf{State}_H$, which contain the low- and high-labeled data, respectively. A *low state* is an element $\sigma_L \in \mathbf{State}_L$; a *high state* is an element $\sigma_H \in \mathbf{State}_H$. The projection of state $\sigma \in \mathbf{State}$ onto $\mathbf{State}_L$ is written $\sigma \restriction L$; this is the part of σ that is visible to the attacker. Projection onto $\mathbf{State}_H$, the part of σ that is not visible to the attacker, is written $\sigma \restriction H$.

Assume that each variable in a program is labeled to indicate the confidentiality of the information in that variable; for example, x_L is a variable x that contains low information. For convenience, assume that variable l is labeled L and variable h is labeled H. Let $\mathbf{Var}_L$ be the set of variables in a program that are labeled L; then $\mathbf{State}_L = \mathbf{Var}_L \to \mathbf{Val}$. The low projection $\sigma \restriction L$ of a state σ is:

$$\sigma \restriction L \quad \triangleq \quad \lambda v \in \mathbf{Var}_L \,.\, \sigma(v)$$

States σ and σ' are *low-equivalent*, written $\sigma \approx_L \sigma'$, if they have the same low projection:

$$\sigma \approx_L \sigma' \quad \triangleq \quad (\sigma \restriction L) \ = \ (\sigma' \restriction L)$$

Distributions also have projections. Let δ be a distribution and σ_L a low state. Then $(\delta \restriction L)(\sigma_L)$ is the frequency with which any state whose low projection is σ_L occurs in δ:[1]

$$\delta \restriction L \quad \triangleq \quad \lambda\sigma_L \in \mathbf{State}_L \,.\, \textstyle\sum_{\sigma' \ \mid \ (\sigma' \restriction L) \ = \ \sigma_L} \ \delta(\sigma')$$

1 Formula $\star_{x \in D \ \mid \ R} \ P$ is a quantification in which $\star$ is the quantifier (such as $\forall$ or Σ), x is the variable that is bound in R and P, D is the domain of x, R is the range, and P is the body. We omit D, R, and even x when they are clear from context; an omitted range means $R = true$.

High projections and high equivalence are defined by replacing each occurrence of L with H in the definitions above.

2.4. Belief Representation

To reason about beliefs, we must choose a representation for them. Many representations, both quantitative and qualitative, have been developed; Halpern [12] presents several representations that all share the idea of *possible worlds*, the set W of all elementary outcomes about which beliefs can be held. Though we do fix a belief representation, the results presented below are mostly independent of any particular representation.

To be usable in our framework, a representation must have certain natural operations defined. Let b and b' be beliefs about possible worlds W and W', respectively.

1. Belief product $\otimes$ combines b and b' into a new belief $b \otimes b'$ about possible worlds $W \times W'$, where W and W' must be disjoint.
2. Belief update $b|U$ is the belief that results when b is updated to include the new information that the actual world is in a set $U \subseteq W$ of possible worlds.
3. Belief distance $D(b \twoheadrightarrow b')$ is a real number $r \geq 0$ that quantifies the difference between b and b'.

The rest of this paper uses distributions to represent beliefs. We take high states as the possible worlds for beliefs, fixing W to be $\mathbf{State}_H$. We define a belief b as a normalized distribution over high states, i.e. $mass(b) = 1$. Whereas distributions correspond to positive measures, beliefs correspond to probability measures. Probability measures are well-studied as a belief representation [12], and have several advantages here: they are familiar, quantitative, satisfy the required operations, and admit a programming language semantics (as shown in Section 5). There is also a nice justification for the numbers they produce: roughly, $b(\sigma)$ characterizes the amount of money an attacker should be willing to bet that σ is the true state of the system [12].

For belief product $\otimes$, we define a more general distribution product $\otimes$ of two distributions $\delta_1 : A \to \mathbb{R}^+$ and $\delta_2 : B \to \mathbb{R}^+$, where A and B are disjoint, as:

$$\delta_1 \otimes \delta_2 \quad \triangleq \quad \lambda(\sigma_1, \sigma_2) \in A \times B \,.\, \delta_1(\sigma_1) \cdot \delta_2(\sigma_2)$$

It is easy to check that if b and b' are beliefs, then $b \otimes b'$ is too. For belief update $|$, we use distribution conditioning:

$$\delta|U \quad \triangleq \quad \lambda\sigma \,.\, \text{if } \sigma \in U \text{ then } \frac{\delta(\sigma)}{\sum_{\sigma' \in U} \delta(\sigma')} \text{ else } 0$$

For belief distance D we use *relative entropy*, an information-theoretic metric [13] for the distance between distributions.

$$D(b' \twoheadrightarrow b) \quad \triangleq \quad \sum_\sigma b(\sigma) \cdot \log \tfrac{b(\sigma)}{b'(\sigma)}$$

The base of the logarithm in D can be chosen arbitrarily; we use base two and write $\lg$ to indicate $\log_2$, making bits the unit of measurement for distance. The relative entropy of b to b' is the expected inefficiency (that is, the number of additional bits that must be sent) of an optimal code that is constructed by assuming an inaccurate distribution over symbols b' when the real distribution is b [13]. Like an analytic metric, $D(b' \twoheadrightarrow b)$ is always at least zero and $D(b' \twoheadrightarrow b)$ equals zero only when $b = b'$.[2]

Relative entropy has the property that if $b(\sigma) > 0$ and $b'(\sigma) = 0$, then $D(b' \twoheadrightarrow b) = \infty$. An infinite distance between beliefs would cause difficulty in measuring change in accuracy. To avoid this anomaly, beliefs may be required to satisfy certain restrictions. For example, an attacker's belief b might be restricted such that:

$$(\min_{\sigma_H} b(\sigma_H)) \geq \frac{\epsilon}{|\mathbf{State} \restriction H|}$$

for some $\epsilon > 0$, which ensures that b is never off by more than a factor of ϵ from a uniform distribution; we call such beliefs *admissible*. Other admissibility restrictions may be substituted for this one if the analysis context suggests stronger assumptions about how attackers form their beliefs.

3. Experiments

We formalize as an *experiment* how an *attacker*, an agent that reasons about beliefs, revises his beliefs from interaction with a *system*, an agent that executes programs. The attacker should not learn about the high input to the program but is allowed to observe (and perhaps influence) low inputs and outputs. Other agents (a system operator, other users of the system with their own high data, an informant upon which the attacker relies, etc.) might be involved when an attacker interacts with a system; however, it suffices to condense all of these to just the attacker and the system, because the system can act for any other agents.

In a particular experiment, we are chiefly interested in the program S with which the attacker is interacting. We conservatively assume that the attacker knows the source code of S. For simplicity of presentation, we assume that S always terminates and that it never modifies the high state. Section 3.4 discusses how both restrictions can be lifted without significant changes.

2 Unlike an analytic metric, D does not satisfy the triangle inequality. However, there is no compelling reason to assume that the triangle inequality holds for beliefs: perhaps it is easier to rule out a possibility from a belief than to add a new one, or vice-versa.

An experiment $\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L \rangle$ is conducted as follows.

1. The attacker chooses a prebelief b_H about the high state.
2. (a) The system picks a high state σ_H
   (b) The attacker picks a low state σ_L.
3. The system executes the program S, which produces a state $\sigma' \in \Gamma(\delta')$ as output, where $\delta' = [\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}_H)$. The attacker observes the low projection of the output state: $o = \sigma' \upharpoonright L$.
4. The attacker infers a postbelief: $b'_H = (([\![S]\!](\dot{\sigma}_L \otimes b_H)|o)) \upharpoonright H \in \mathcal{B}(\mathcal{E})$

Figure 1. Experiment Protocol

3.1. Experiment Protocol

Formally, an experiment $\mathcal{E}$ is a tuple:

$$\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L \rangle$$

where S is the program, b_H is the attacker's belief, σ_H is the high projection of the initial state, and σ_L is the low projection of the initial state. The protocol for experiments is summarized in Figure 1. Here is a justification for the protocol.

An attacker's *prebelief*, describing his belief at the beginning of the experiment, may be chosen arbitrarily (subject to the admissibility requirement in Section 2.4) or may be informed by previous experiments. In a series of experiments, the *postbelief* from one experiment becomes the prebelief to the next. The attacker might even choose a prebelief b_H that contradicts his true subjective probability distribution on the state, and this gives our analysis additional power, because it allows the attacker to conduct experiments that answer questions such as "What would happen if I were to believe b_H?".

The system chooses σ_H, the high projection of the initial state, and it might remain constant from one experiment to the next or it might vary. For example, Unix passwords do not usually change frequently, but the PINs on RSA SecurID tokens change each minute. We conservatively assume that the attacker chooses all of σ_L, the low projection of the initial state, since this gives him additional power in controlling execution of the program.[3] The attacker's choice of σ_L is likely to be influenced by b_H, but for generality, we do not require there be such a strategy.

Program S is executed only a single time in a single experiment; multiple executions are modeled by multiple experiments. The meaning of S given input $\dot{\sigma}_L \otimes \dot{\sigma}_H$ is an output distribution δ':

$$\delta' = [\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}_H)$$

From δ' the attacker makes an *observation*, which is a low projection of an output state. Probabilistic programs may yield many possible output states, but in a single execution of the program, only one output state can be produced. So in a single experiment, the attacker is allowed only a single observation. A single state is chosen from a distribution by *sampling* operator Γ, where $\Gamma(\delta)$ generates a state σ from the domain of δ with probability $\delta(\sigma)/mass(\delta)$; the mass operator is used to normalize the distribution to a probability distribution. To emphasize the fact that the choice is made randomly, assignment of a sample is written $\sigma \in \Gamma(\delta)$, using $\in$ instead of $=$. The observation o resulting from δ' is:

$$o \in \Gamma(\delta') \upharpoonright L$$

The formula the attacker uses for postbelief b'_H in Figure 1 has the attacker perform two operations. The first operation is to use the semantics of S along with prebelief b_H as the distribution on high input. This "thought experiment" generates the inferences the attacker can make from his knowledge of the program text and low inputs by predicting the output distribution. We define the prediction δ'_A to correlate the output state with the high input state:

$$\delta'_A = [\![S]\!](\dot{\sigma}_L \otimes b_H)$$

The second operation is to condition prediction δ'_A on observation o. This incorporates any additional inferences that can be made from the observation, then restricts that to high inputs:

$$b'_H = (\delta'_A|o) \upharpoonright H$$

The conditioning operator $|$ is a specialization of distribution conditioning. It removes all the mass in distribution δ that is inconsistent with observation o, then normalizes the distribution:

$$\delta|o \quad \triangleq \quad \lambda\sigma\,.\,\text{if } (\sigma \upharpoonright L) \;=\; o \text{ then } \frac{\delta(\sigma)}{(\delta \upharpoonright L)(o)} \text{ else } 0$$

Belief revision operator $\mathcal{B}$, which yields the postbelief from an experiment $\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L \rangle$, compactly represents all of the formulas above:

$$\begin{aligned} \mathcal{B}(\mathcal{E}) \quad \triangleq \quad & (([\![S]\!](\dot{\sigma}_L \otimes b_H)|o)) \upharpoonright H \\ & \text{where } \; o \in \Gamma(\delta') \upharpoonright L \\ & \qquad \delta' = [\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}_H) \end{aligned}$$

Because it uses Γ, operator $\mathcal{B}$ produces values by sampling, so we write $b'_H \in \mathcal{B}(\mathcal{E})$. To select a particular b'_H from $\mathcal{B}$, we provide observation o:

$$\mathcal{B}(\mathcal{E}, o) \quad \triangleq \quad (([\![S]\!](\dot{\sigma}_L \otimes b_H)|o)) \upharpoonright H$$

3 More generally, both the system and the attacker might contribute to σ_L. But since we are concerned with only confidentiality, not integrity, of information, we do not need to distinguish which parts are chosen by which agent.

p	b_H	b'_{H1}	b'_{H2}
A	0.98	1	0
B	0.01	0	0.5
C	0.01	0	0.5

Table 1. Beliefs about p

p	g	a	δ'_A	$\delta'_A\|o_1$	$\delta'_A\|o_2$
A	A	0	0	0	0
A	A	1	0.98	1	0
B	A	0	0.01	0	0.5
B	A	1	0	0	0
C	A	0	0.01	0	0.5
C	A	1	0	0	0
	...		0	0	0

Table 2. Distributions on the output

3.2. Password Checking as an Experiment

Attaching confidentiality labels to the password checker of Section 1 yields:

$$PWC: \quad \textbf{if } p_H = g_L \textbf{ then } a_L := 1 \textbf{ else } a_L := 0$$

Repeating the analysis of PWC in terms of our experiment model allows the informal reasoning previously used to be made precise.

The attacker starts by choosing prebelief b_H, perhaps as formalized in Table 1. The columns of this table give the probability that the belief at the head of the column assigns to the state at the beginning of a row. Next, the system chooses the initial high projection σ_H, and the attacker chooses the initial low projection σ_L. In the first experiment in Section 1, the password was A, so the system chooses $\sigma_H = (p \mapsto A)$. Similarly, the attacker chooses $\sigma_L = (g \mapsto A, a \mapsto 0)$. (The initial value of a is actually irrelevant, since it is never used by the program and a is set along all control paths.) Next, the system runs PWC. The output distribution δ' should clearly be a point mass at the state $\sigma' = (p \mapsto A, g \mapsto A, a \mapsto 1)$; the semantics in Section 5 will validate this intuition. Since σ' is the only state that can be sampled from δ', the attacker's observation o_1 is $\sigma' \restriction L = (g \mapsto A, a \mapsto 1)$.

In the final step of the protocol, the attacker applies the definition of $\mathcal{B}$. He runs a thought experiment, predicting an output distribution $\delta'_A = [\![PWC]\!](\dot{\sigma}_L \otimes b_H)$, given in Table 2. The ellipsis in the final row of the table indicates that all states not shown have frequency 0. This distribution is intuitively correct: the attacker believes that he has a 98% chance of being authenticated, whereas 1% of the time he will fail to be authenticated because the password is B, and another 1% because it is C. The attacker conditions prediction δ'_A on observation o_1, obtaining $\delta'_A|o_1$, also shown in Table 2. Projecting to high yields the attacker's postbelief b'_{H1}, shown in Table 1. This postbelief is what the informal reasoning in Section 1 suggested: the attacker is certain that the password is A.

The second experiment in Section 1 is also correctly modeled by a formal experiment. In it, b_H and σ_L remain the same as before, but σ_H becomes $(p \mapsto C)$. Observation o_2 is therefore the point mass at $(g \mapsto A, a \mapsto 0)$. The prediction δ'_A remains unchanged, and conditioned on o_2 it becomes $\delta'_A|o_2$, shown in Table 2. Projecting to high yields the new postbelief b'_{H2} in Table 1. This postbelief again agrees with the informal reasoning: the attacker believes that there is a 50% chance each for the password to be B or C.

3.3. Bayesian Belief Revision

Postbelief operator $\mathcal{B}$ is an application of *Bayesian inference*, which is a standard technique in applied statistics for making inferences when uncertainty is made explicit through probability models [8]. The fundamental Bayesian method of updating a hypothesis Hyp based on an observation obs is *Bayes' rule*:

$$\Pr(Hyp|obs) \quad = \quad \frac{\Pr(Hyp)\Pr(obs|Hyp)}{\sum_{Hyp'} \Pr(Hyp')\Pr(obs|Hyp')}$$

In our model, the attacker's hypothesis is about the values of high states, so the domain of hypotheses is **State** $\restriction H$. Therefore $\Pr(Hyp)$, the probability he ascribes to a particular hypothesis σ_H, is modeled by $b_H(\sigma_H)$. The probability $\Pr(obs|Hyp)$ the attacker ascribes to an observation given the assumed truth of a hypothesis is modeled by the program semantics: the probability of an observation o given an assumed high input σ_H is $([\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}_H) \restriction L)(o)$. Given experiment $\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L \rangle$, instantiating Bayes rule on these probability models yields $B(\mathcal{E}, o)$, which is $\Pr(\sigma_H|o)$:

$$B(\mathcal{E}, o) = \frac{b_H(\sigma_H) \cdot ([\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}_H) \restriction L)(o)}{\sum_{\sigma'_H} b_H(\sigma'_H) \cdot ([\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}'_H) \restriction L)(o)}$$

With this instantiation, we can show that how an attacker updates his belief according to our experiment protocol is equivalent to Bayesian updating.

Theorem 1

$$\mathcal{B}(\mathcal{E}, o)(\sigma_H) \quad = \quad B(\mathcal{E}, o)$$

Proof. In Appendix A. □

3.4. Mutable High State and Nontermination

Section 3.1 made two simplifying assumptions about program S: it never modifies high input, and it always terminates. We now dispense with these mostly minor technical issues.

To eliminate the first assumption, note that if S were to modify the high state, the attacker's prediction δ'_A would correlate high outputs with low outputs. However, to calculate a postbelief, δ'_A must correlate high *inputs* with low outputs. So the high input state must be preserved in δ'_A. Let the notation b^0_H mean the same distribution as b_H, except that each state of its domain has a 0 as a superscript. So, if b_H ascribes probability p to the state σ, then b^0_H ascribes probability p to the state σ^0. We assume that S cannot modify states with a superscript 0. In the case that states map variables to values, this could be achieved by defining σ^0 to be the same state as σ, but with the superscript 0 attached to variables; for example, if $\sigma(v) = 1$ then $\sigma^0(v^0) = 1$. Note that S cannot modify σ^0 if did not originally contain any variables with superscripts.

Using this notation, the belief revision operator is extended to $\mathcal{B}^!$, which allows S to modify the high state in experiment $\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L \rangle$.

$$\begin{aligned}\mathcal{B}^!(\mathcal{E}) \quad &\triangleq \quad ((\llbracket S \rrbracket(\dot\sigma_L \otimes b_H \otimes b^0_H)|o)) \upharpoonright H^0 \\ &\qquad \text{where } o \in \Gamma(\delta') \upharpoonright L \\ &\qquad\qquad \delta' = \llbracket S \rrbracket(\dot\sigma_L \otimes \dot\sigma_H)\end{aligned}$$

In the first line of the definition, the high input state is preserved by introducing the product with b^0_H, and the attacker's postbelief about the input is recovered by restricting to H^0, the high input state with the superscript 0.

To eliminate the second assumption, note that program S must terminate for an attacker to obtain a low state as an observation when running S. There are two ways to model the observation in the case of nontermination, depending on whether the attacker can detect nontermination.

If the attacker has an oracle that decides nontermination, then nontermination can be modeled in the standard denotational style. Let $\perp$ be the divergent state representing nontermination, $\mathbf{State}_\perp \triangleq \mathbf{State} \cup \{\perp\}$, and $\perp \upharpoonright L \triangleq \perp$. Nontermination is now allowed as an observation, leading to an extended belief revision operator $\mathcal{B}^{!\perp}$:

$$\begin{aligned}\mathcal{B}^{!\perp}(\mathcal{E}) \quad &\triangleq \quad (out_\perp(S, \dot\sigma_L \otimes b_H \otimes b^0_H)|o) \upharpoonright H^0 \\ &\qquad \text{where } o \in \Gamma(\delta') \upharpoonright L \\ &\qquad\qquad \delta' = out_\perp(S, \dot\sigma_L \otimes \dot\sigma_H)\end{aligned}$$

Function $out_\perp(S, \delta)$ produces a distribution that yields the frequency that S terminates, or fails to terminate, on input distribution δ:

$$\begin{aligned}out_\perp(S, \delta) \quad &\triangleq \quad \lambda\sigma : \mathbf{State}_\perp \,.\, \text{if } \sigma = \perp \\ &\qquad\qquad \text{then } mass(\delta) - mass(\llbracket S \rrbracket\delta) \\ &\qquad\qquad \text{else } (\llbracket S \rrbracket\delta)(\sigma)\end{aligned}$$

If S does not terminate on some input states in δ, then output distribution $\llbracket S \rrbracket\delta$ will contain less mass than δ; otherwise, $mass(\delta)$ will equal $mass(\llbracket S \rrbracket\delta)$. Missing mass corresponds to nontermination [20, 17], so $out_\perp$ maps the missing mass to $\perp$.

An attacker that cannot correctly detect nontermination is more difficult to model. At some point during the execution of the program, he will stop waiting for the program to terminate and declare that he has observed nontermination. However, he may be incorrect in doing so—leading to beliefs about nontermination and instruction timings. The interaction of these beliefs with beliefs about high inputs is complex; we leave this for future work.

4. Measuring Information Flow

The informal analysis of PWC in Section 1 suggests that information flow corresponds to an improvement in the accuracy of an attacker's belief. Recall the more accurate belief b is with respect to high state $\dot\sigma_H$, the less the distance $D(b \twoheadrightarrow \dot\sigma_H)$. We use change in accuracy, as measured by distance, to quantify information flow.

4.1. Information Flow from a Report

Given an experiment $\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L \rangle$, a *report* is a pair $\langle \mathcal{E}, b'_H \rangle$ such that $b'_H \in \mathcal{B}(\mathcal{E})$. The accuracy of the attacker's prebelief b_H in a report $\langle \mathcal{E}, b'_H \rangle$ is $D(b_H \twoheadrightarrow \dot\sigma_H)$; the accuracy of the attacker's postbelief b'_H in that report is $D(b'_H \twoheadrightarrow \dot\sigma_H)$. And, the amount of information flow $\mathcal{Q}$ caused by $\langle \mathcal{E}, b'_H \rangle$ is defined as the difference of these two quantities:

$$\mathcal{Q}(\langle \mathcal{E}, b'_H \rangle) \quad \triangleq \quad D(b_H \twoheadrightarrow \dot\sigma_H) - D(b'_H \twoheadrightarrow \dot\sigma_H)$$

Thus the amount of information flow (in bits) in $\mathcal{Q}$ corresponds to the improvement in the accuracy of the attacker's belief, exactly as desired.

Using relative entropy as the distance operator D in the definition of $\mathcal{Q}$ allows us to give a concrete interpretation to the number produced by our definition of $\mathcal{Q}$. Recalling the coding efficiency interpretation of relative entropy in Section 2.4, the amount of information flow $\mathcal{Q}$ is the improvement in the expected inefficiency of the attacker's optimal code for the high input.

With an additional definition from information theory, a more consequential characterization of $\mathcal{Q}$ is possible. Let $\mathcal{I}_\delta(F)$ denote the *information* contained in event F drawn from probability distribution δ:

$$\mathcal{I}_\delta(F) \quad \triangleq \quad -\lg \Pr_\delta(F)$$

Information is sometimes called "surprise" because $\mathcal{I}$ measures how surprising an event is; for example, events that occur with probability 1 have surprise 0.

For an attacker, there are two unknowns in the outcome of an experiment: the initial high state, and the probabilistic choices made by the program. Let $\delta_S = [\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}_H) \upharpoonright L$ be the system's distribution on low outputs, and $\delta_A = [\![S]\!](\dot{\sigma}_L \otimes b_H) \upharpoonright L$ be the attacker's distribution on low outputs. $\mathcal{I}_{\delta_A}(o)$ measures the information contained in o about both unknowns, but $\mathcal{I}_{\delta_S}(o)$ measures only the latter unknown. For programs that make no probabilistic choices, neither quantity can measure any information about probabilistic choices; thus, δ_A contains information about only the initial high state, and δ_S is a point mass at some state σ such that $\sigma \upharpoonright L = o$, so the amount of information $\mathcal{I}_{\delta_S}(o)$ is 0. For probabilistic programs, $\mathcal{I}_{\delta_S}(o)$ is generally not equal to 0; subtracting it removes all the information contained in $\mathcal{I}_{\delta_A}(o)$ that is solely about the outcomes of probabilistic choices, leaving only information about high inputs.

The following theorem states that $\mathcal{Q}$ measures the information about high input σ_H contained in observation o.

Theorem 2

$$\mathcal{Q}(\langle \mathcal{E}, b'_H \rangle) \quad = \quad \mathcal{I}_{\delta_A}(o) - \mathcal{I}_{\delta_S}(o)$$

Proof. In Appendix A. □

As an example, consider the experiments on PWC in Section 3.2. The first experiment $\mathcal{E}_1$ has the attacker correctly guess the password A, so:

$$\mathcal{E}_1 = \langle PWC, b_H, (p \mapsto A), (g \mapsto A, a \mapsto 0) \rangle$$

where b_H (and the other beliefs about to be used) is defined in Table 1. Only one report, $\langle \mathcal{E}_1, b'_{H1} \rangle$, is possible from this experiment. Calculating $\mathcal{Q}(\langle \mathcal{E}_1, b'_{H1} \rangle)$ yields a flow of 0.0291 bits from the report. The small flow makes sense because the report has only confirmed something the attacker already believed to be almost certainly true. In experiment $\mathcal{E}_2$ the attacker guesses incorrectly.

$$\mathcal{E}_2 = \langle PWC, b_H, (p \mapsto C), (g \mapsto A, a \mapsto 0) \rangle$$

Again, only one report $\langle \mathcal{E}_2, b'_{H2} \rangle$ is possible from this experiment, and calculating $\mathcal{Q}(\langle \mathcal{E}_2, b'_{H2} \rangle)$ yields an information flow of 5.6439 bits. A higher information flow makes sense because the attacker's postbelief is much closer to correctly identifying the high state. The attacker's prebelief b_H ascribed a 0.02 probability to the event $[p \neq A]$, and the information of an event with probability 0.02 is 5.6439, the same information flow calculated above. This suggests that $\mathcal{Q}$ is correctly measuring the information about high input contained in the observation.

4.2. Comparing Accuracy and Uncertainty

The information flow in experiment $\mathcal{E}_2$ is surprisingly high; at most two bits are required to store password p in memory, so how can the program leak more than five bits? In brief, the extra bits correct the attacker's misconceptions about the password, but this question also illuminates the difference between measuring information flow based on uncertainty versus based on accuracy. Consider how an uncertainty-based approach would analyze the program.

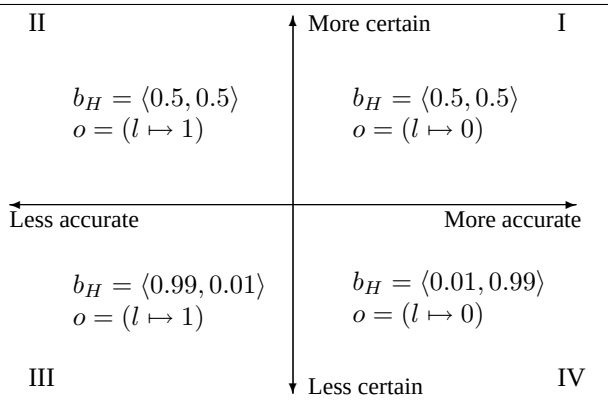

Figure 2. Effect of $FLIP$ on postbelief

The attacker's initial uncertainty about p is $\mathcal{H}(b_H) = 0.1614$ bits, where $\mathcal{H}$ is the information-theoretic measure of *entropy*, or uncertainty, in a probability distribution δ.

$$\mathcal{H}(\delta) \quad \triangleq \quad -\textstyle\sum_\sigma \ \delta(\sigma) \cdot \lg \delta(\sigma)$$

Maximum entropy is achieved by uniform distributions [13], so the maximal uncertainty about p is $\lg 3 \approx 1.6$ bits, the same number of bits required to store p. In the second experiment, the attacker's final uncertainty about p is $\mathcal{H}(b_{H2}) = 1$. The reduction in uncertainty is $0.1614 - 1 = -0.8386$. An uncertainty-based analysis, such as [5] or [16], would interpret this negative quantity as an absence of information flow. But this is clearly not the case—the attacker's belief has been guided much closer to reality by the experiment. The uncertainty-based analysis ignores reality by measuring b_H and b_{H2} against themselves only, instead of against the high state σ_H.

Accuracy and uncertainty are orthogonal properties of beliefs, as shown in Figure 2. The figure shows the change in an attacker's accuracy and uncertainty when an experiment $\mathcal{E} = \langle FLIP, b_H, (h \mapsto 0), (l \mapsto 0) \rangle$ is run, and observation o is generated by the run. The notation $b_H = \langle x, y \rangle$ means that $b_H(h \mapsto 0) = x$ and $b_H(h \mapsto 1) = y$. The program $FLIP$ is:

$$FLIP: \quad l := h \ {}_{0.99}[\!] \ l := \neg h$$

Usually, $FLIP$ sets l to be h, so the attacker will expect this to be the case. Runs that satisfy this expectation will cause his postbelief to be more accurate, but may cause his uncertainty to either increase or decrease, depending on his

Quadrant	h	I	II	III	IV
b_H :	0	0.5	0.5	0.99	0.01
	1	0.5	0.5	0.01	0.99
o		$(l \mapsto 0)$	$(l \mapsto 1)$	$(l \mapsto 1)$	$(l \mapsto 0)$
b'_H :	0	0.99	0.01	0.5	0.5
	1	0.01	0.99	0.5	0.5
Increase in accuracy		$+0.9855$	-5.6439	-0.9855	$+5.6439$
Reduction in uncertainty		$+0.9192$	$+0.9192$	-0.9192	-0.9192

Table 3. Analysis of $FLIP$

	p	
b_H :	A	0.98
	B	0.01
	C	0.01
b'_H :	A	0
	B	0.5
	C	0.5
Increase in accuracy		$+5.6439$
Reduction in uncertainty		-0.8245

Table 4. Analysis of PWC

prebelief; when uncertainty increases, an uncertainty metric would mistakenly say that no flow has occurred.

With probability 0.01, $FLIP$ produces a run that fools the attacker and sets l to be $\neg h$, causing his belief to become less accurate. The decrease in accuracy results in *misinformation*, which is a negative information flow. When the attacker's prebelief is almost completely accurate, such runs will make him more uncertain. But when the attacker's prebelief is uniform, runs that result in misinformation will make him less uncertain; when uncertainty decreases, an uncertainty metric would mistakenly say that flow has occurred. Table 3 demonstrates this phenomenon concretely. The quadrant labels refer to Figure 2. For each quadrant, the attacker's prebelief b_H, observation o, and the resulting postbelief b'_H is given in the top half of the table. In the bottom half, increase in accuracy is calculated using the information flow metric $\mathcal{Q}(\langle \mathcal{E}, b'_H \rangle)$, and reduction in uncertainty is calculated using the difference in entropy $\mathcal{H}(b_H) - \mathcal{H}(b'_H)$.

Finally, recall that when the attacker guessed a password incorrectly in Section 1, his belief became more accurate and more uncertain. Table 4 gives the exact changes in his accuracy and uncertainty, using guess $g = A$ and password $p = C$.

In summary, uncertainty is inadequate as a metric for information flow. By Theorem 2, information flows when an attacker's belief becomes more accurate, but an uncertainty metric can mistakenly measure a flow of zero or less. Inversely, misinformation flows when an attacker's belief becomes less accurate, but an uncertainty metric can mistakenly measure a positive information flow. Hence, accuracy is the correct metric for information flow.

4.3. Expected Information Flow

We expect the results of this paper to be useful in deciding whether a program satisfies a quantitative security property. Since an experiment on a probabilistic program S can produce many reports, it is reasonable to assume that such properties will discuss expected flow over those reports. So we define expected flow $\mathcal{Q}_E$ over all reports from experiment $\mathcal{E}$:

$$\begin{array}{rcl} \mathcal{Q}_E(\mathcal{E}) & \triangleq & E_{o \in \delta' \uparrow L}[\mathcal{Q}(\langle \mathcal{E}, \mathcal{B}(\mathcal{E}, o) \rangle)] \\ & = & \sum_o \ (\delta' \uparrow L)(o) \\ & & \quad \cdot \mathcal{Q}(\langle \mathcal{E}, ([\![S]\!](\dot{\sigma}_L \otimes b_H) | o) \uparrow H \rangle) \end{array}$$

where $\delta' = [\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}_H)$ gives the probability distribution on reports, as in Figure 1, and $E_\delta[X]$ is the expected value of an expression X with respect to distribution δ.

Expected flow is useful in analyzing probabilistic programs that can produce many observations for a single input. Consider a faulty password checker:

$$\begin{array}{rl} FPWC: & \textbf{if } p = g \textbf{ then } a := 1 \textbf{ else } a := 0; \\ & a := \neg a \ {}_{0.1}[\!] \ \textbf{skip} \end{array}$$

With probability 0.1, $FPWC$ flips the authentication flag. Can this program be expected to confound attackers; that is, does $FPWC$ leak less expected information than PWC? This question can be answered by comparing the expected flow from $FPWC$ to the flow of PWC. Table 5 gives the flow of $FPWC$ for experiments $\mathcal{E}_1^F$ and $\mathcal{E}_2^F$, which are identical to $\mathcal{E}_1$ and $\mathcal{E}_2$ from Section 4.1, except that they execute $FPWC$ instead of PWC. Observe that, for both pairs of experiments, the expected flow of $FPWC$ is less than the flow of PWC. The random flip of a makes it more difficult for the attacker to increase the accuracy of his belief.

$\mathcal{E}$	o	$\mathcal{Q}(\langle \mathcal{E}, \mathcal{B}(\mathcal{E}, o)\rangle)$	$\mathcal{Q}_E(\mathcal{E})$
$\mathcal{E}_1$	$(a \mapsto 1)$	0.0291	0.0291
	$(a \mapsto 0)$	impossible	
$\mathcal{E}_1^F$	$(a \mapsto 1)$	0.0258	0.0018
	$(a \mapsto 0)$	-0.2142	
$\mathcal{E}_2$	$(a \mapsto 1)$	impossible	5.6439
	$(a \mapsto 0)$	5.6439	
$\mathcal{E}_2^F$	$(a \mapsto 1)$	-3.1844	2.3421
	$(a \mapsto 0)$	2.9561	

Table 5. Leakage of PWC and $FPWC$

Reports $\langle \mathcal{E}_1^F, (a \mapsto 0)\rangle$ and $\langle \mathcal{E}_2^F, (a \mapsto 1)\rangle$ correspond to an execution where the value of a is flipped. The flow for these reports is negative, indicating that the program is giving the attacker misinformation, as described in Section 4.2.

Calculating expected flow requires a summation over all $o \in \mathbf{State}_L$, which may be a countably infinite set; this is infeasible to calculate either by hand or by machine. Fortunately, expected flow can be conservatively approximated by conditioning on a single distribution rather than conditioning on many observations. Conditioning δ on δ_L has the effect of making the low projection of δ identical to δ_L, while leaving the high projection of δ unchanged.

$$\delta|\delta_L \quad \triangleq \quad \lambda\sigma \,.\, \frac{\delta(\sigma)}{\sum_{\sigma' \mid \sigma \approx_L \sigma'} \delta(\sigma')} \cdot \delta_L(\sigma \upharpoonright L)$$

The bound on expected flow is then calculated as follows.

Theorem 3 *Let:*

$$\begin{array}{rcl} \mathcal{E} & = & \langle S, b_H, \sigma_H, \sigma_L\rangle \\ \delta' & = & [\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}_H) \\ e_H & = & (([\![S]\!](\dot{\sigma}_L \otimes b_H))|(\delta' \upharpoonright L)) \upharpoonright H \end{array}$$

Then:

$$\mathcal{Q}_E(\mathcal{E}) \quad \leq \quad \mathcal{Q}(\langle \mathcal{E}, e_H\rangle)$$

Proof. In Appendix A. □

The experiment model can be extended to increase the applicability of expected flow. Rather than choose a particular low state σ_L, the attacker may more generally choose a distribution over low states, δ_L, which the system samples to produce the initial low state $\sigma_L \in \Gamma(\delta_L)$. This expresses a randomized guessing strategy for the attacker. By taking the expectation in $\mathcal{Q}_E$ with respect to both σ_L and o, the expected flow for the attacker's guessing strategy can be calculated. The initial high state σ_H can be similarly generalized to δ_H and incorporated into $\mathcal{Q}_E$. This could be used, for example, to determine the expected flow of the password checker when users' choice of passwords can be described by a distribution.

Repetition #		1	2
b_H :	A	0.98	0
	B	0.01	0.5
	C	0.01	0.5
$\sigma_L(g)$		A	B
$o(a)$		0	0
b'_H :	A	0	0
	B	0.5	0
	C	0.5	1
$\mathcal{Q}(\langle \mathcal{E}, b'_H\rangle)$		5.6439	1.0

Table 6. Repeated experiments on PWC

4.4. Maximum Information Flow

Designers of quantitative security properties are likely to want to limit maximum information flow. So we define the maximum amount of information flow that program S can cause in a single report as the maximum amount of flow from any report of any experiment $\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L\rangle$ on S:

$$\mathcal{Q}_{\max}(S) \quad \triangleq \quad \max_{\mathcal{E}, b'_H \mid b'_H \in \mathcal{B}(\mathcal{E})} \mathcal{Q}(\langle \mathcal{E}, b'_H\rangle)$$

Consider applying $\mathcal{Q}_{\max}(S)$ to PWC. Assuming that b_H satisfies the admissibility restriction in Section 2.4 and that the attacker guesses an incorrect password yields that PWC can leak at most $-\lg(\epsilon \cdot \frac{n-1}{n})$ bits per report, where n is the number of possible passwords. If $\epsilon = 1$, the attacker is forced to have a uniform distribution over passwords, representing a lack of belief for any particular value for the password. Additionally, if $n = 2^k$ for some k, then we obtain that for k-bit passwords, PWC can leak at most $k - \lg(2^k - 1)$ bits in a report; for $k > 12$ this is less than 0.0001 bits, supporting the intuition that password checking leaks little information.

4.5. Repeated Experiments

Nothing precludes repetition of experiments. The most interesting case has the attacker return to step 2b of the experiment protocol in Figure 1 after updating his belief in step 4; that is, the system keeps the high input to the program constant, and the attacker is allowed to check new low inputs based on the results of previous experiments. Suppose that experiment $\mathcal{E}_1$ from Section 4.1 is run and then repeated with $\sigma_L = (g \mapsto B)$. Then the attacker's belief about the password evolves as shown in Table 6.

Summing the information flow for each repetition yields a total information flow of 6.6439. This total corresponds to what $\mathcal{Q}$ would calculate for a single experiment, if that experiment changed prebelief b_H to postbelief b'_{H2}, where

b'_{H2} is the attacker's postbelief in the second repetition in Table 6:

$$\begin{aligned} D(b_H \twoheadrightarrow \dot{\sigma}_H) - D(b'_{H2} \twoheadrightarrow \dot{\sigma}_H) &= 6.6439 - 0 \\ &= 6.6439 \end{aligned}$$

This example is an instance of a more general theorem stating that the postbelief from a series of experiments, where the postbelief from one experiment becomes the prebelief to the next, contains all the information learned during the series. Let $\mathcal{E}_i = \langle S, b_{H_i}, \sigma_H, \sigma_{L_i} \rangle$ be the i^{th} experiment in the series, and let $r_i = \langle \mathcal{E}_i, b'_{H_i} \rangle$ be a report from $\mathcal{E}_i$. Let $r_1, \ldots, r_n$ be a series of n reports in which prebelief b_{H_i} in experiment $\mathcal{E}_i$ is the postbelief $b'_{H_{i-1}}$ from report $i-1$. Finally, let $b'_{H_0} = b_{H_1}$ be the attacker's prebelief for the entire series.

Theorem 4

$$D(b_{H_1} \twoheadrightarrow \dot{\sigma}_H) - D(b'_{H_n} \twoheadrightarrow \dot{\sigma}_H) \;=\; \textstyle\sum_{i \mid 1 \le i \le n} \; \mathcal{Q}(r_i)$$

Proof. In Appendix A. □

5. Language Semantics

The last piece required for our framework is a semantics $[\![S]\!]$ based on distributions. For our programming language, we use **while**-programs extended with a probabilistic choice construct $\;{}_p[\!]$. The operational semantics for the deterministic subset of this language is standard. Probabilistic choice $S_1 \;{}_p[\!]\; S_2$ executes s_1 with probability p or S_2 with probability $1-p$.

Metavariables S, v, E, and B range over programs, variables, arithmetic expressions, and Boolean expressions, respectively. Evaluation of expressions is assumed side-effect free, but we do not otherwise give their syntax or semantics. The syntax of the language is:

$$\begin{aligned} S \;::=\; & \textbf{skip} \mid v := E \mid S; S \mid \textbf{if } B \textbf{ then } S \textbf{ else } S \\ \mid\; & \textbf{while } B \textbf{ do } S \mid S \;{}_p[\!]\; S \end{aligned}$$

The experiment protocol of Section 3 requires a semantics in which programs are functions that map distributions to distributions. Here we build such a semantics in two stages, as suggested by Section 2.2. First, we build a simpler semantics that maps states to distributions. Second, we lift the simpler semantics so that it operates on distributions.

Our first task then is to define the semantics $[\![S]\!]$: **State** $\to$ **Dist**. This semantics should describe the probability of termination in a given state: if $[\![S]\!]\sigma = \delta$, then the probability of S, when begun in σ, terminating in σ' should be $\delta(\sigma')$. The semantics is given in Figure 3. We assume some semantics $[\![E]\!]$: **State** $\to$ **Val** that gives meaning to expressions, and a semantics $[\![B]\!]$: **State** $\to$ **Bool** that gives meaning to Boolean expressions.

The statements **skip**, **if**, and **while** have essentially the same denotations as in the standard deterministic case.[4] State update $\sigma[v \mapsto V]$, where $V \in$ **Val**, changes the value of v to V in σ. The distribution update $\delta[v \mapsto E]$ in the denotation of assignment represents the result of substituting the meaning of E for v in all the states of δ and is defined as:

$$\delta[v \mapsto E] \quad \triangleq \quad \lambda\sigma\,.\,(\textstyle\sum_{\sigma' \mid \sigma'[v \mapsto [\![E]\!]\sigma'] = \sigma} \; \delta(\sigma'))$$

The sequential composition of two programs, written $S_1; S_2$, is defined using intermediate states. The probability of $S_1; S_2$, starting from σ, reaching a final state σ'' is the sum of the probabilities of all the ways that S_1 can reach some intermediate σ' and then S_2 from that σ' can reach σ''. Note that $([\![S_1]\!]\sigma)(\sigma')$ is the probability that S_1, beginning in σ, terminates in σ', because $[\![S_1]\!]\sigma$ produces a distribution that, when applied to σ', returns the probability of termination in σ'. Similarly, $([\![S_2]\!]\sigma')(\sigma'')$ is the probability that S_2, beginning in σ', terminates in σ''.

The final program construct is probabilistic choice, $S_1 \;{}_p[\!]\; S_2$, where $0 \le p \le 1$. The semantics multiplies the probability of choosing a side S_i with the probability that S_i produces a particular output state σ'. Since the same state σ' might actually be produced by both sides of the choice, the probability of its occurrence is the sum of the probability from either side: $p \cdot ([\![S_1]\!]\sigma)(\sigma') + (1-p) \cdot ([\![S_2]\!]\sigma)(\sigma')$. This formula is simplified to the definition in Figure 3 using $\cdot$ and $+$ as pointwise operators:

$$\begin{aligned} p \cdot \delta \quad &\triangleq \quad \lambda\sigma\,.\,p \cdot \delta(\sigma) \\ \delta_1 + \delta_2 \quad &\triangleq \quad \lambda\sigma\,.\,\delta_1(\sigma) + \delta_2(\sigma) \end{aligned}$$

To show how to lift the semantics in Figure 3 and define $[\![S]\!]$: **Dist** $\to$ **Dist** we use an intuition to what is done for the sequential operator above, where there are many states σ' in which S could begin execution, and all of them could potentially terminate in state σ. So to compute $([\![S]\!]\delta)(\sigma)$, we take a weighted average over all input states σ'. The weights are $\delta(\sigma')$, which describes how likely σ' is to be used as the input state. With σ' as input, S terminates in state σ with frequency $([\![S]\!]\sigma')(\sigma)$. Thus we define $[\![S]\!]\delta$ as:

$$[\![S]\!]\delta \quad \triangleq \quad \lambda\sigma\,.\,\textstyle\sum_{\sigma'} \; \delta(\sigma') \cdot ([\![S]\!]\sigma')(\sigma)$$

This is in accordance with the requirements of Section 2.2.

Applying this definition to the semantics in Figure 3 yields $[\![S]\!]\delta$ for each kind of statement in the language

4 To ensure that the fixed point for **while** exists, we have to verify that **Dist** is a complete partial order (CPO) with a bottom element. In fact, to make this so, we have to extend the definition **Dist** to be **State** $\to [0,1]$. This makes distributions correspond to subprobability measures, and it is easy to check that the semantics always produces subprobability measures as output. The LUB is at most $\lambda\sigma\,.\,1$, and the bottom element is $\lambda\sigma\,.\,0$.

$$
\begin{array}{rcl}
[\![\textbf{skip}]\!]\sigma & = & \dot{\sigma} \\
[\![v := E]\!]\sigma & = & \dot{\sigma}[v \mapsto E] \\
[\![S_1; S_2]\!]\sigma & = & \lambda\sigma'' \,.\, \sum_{\sigma'} ([\![S_1]\!]\sigma)(\sigma') \cdot ([\![S_2]\!]\sigma')(\sigma'') \\
[\![\textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2]\!]\sigma & = & \text{if } [\![B]\!]\sigma \text{ then } [\![S_1]\!]\sigma \text{ else } [\![S_2]\!]\sigma \\
[\![\textbf{while } B \textbf{ do } S]\!]\sigma & = & \text{fix} f : \textbf{Dist} \to \textbf{Dist}. \text{ if } [\![B]\!]\sigma \text{ then } f([\![S]\!]\sigma) \text{ else } \dot{\sigma} \\
[\![S_1 \ {}_p[]\ S_2]\!]\sigma & = & p \cdot [\![S_1]\!]\sigma + (1-p) \cdot [\![S_2]\!]\sigma
\end{array}
$$

Figure 3. Semantics of programs in states

$$
\begin{array}{rcl}
[\![\textbf{skip}]\!]\delta & = & \delta \\
[\![v := E]\!]\delta & = & \delta[v \mapsto E] \\
[\![S_1; S_2]\!]\delta & = & [\![S_2]\!]([\![S_1]\!]\delta) \\
[\![\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2]\!]\delta & = & [\![S_1]\!](\delta \mid B) + [\![S_2]\!](\delta \mid \neg B) \\
[\![\textbf{while } B \textbf{ do } S]\!]\sigma & = & \text{fix} f : \textbf{Dist} \to \textbf{Dist}. \, f([\![S]\!](\delta \mid B)) + (\delta \mid \neg B) \\
[\![S_1 \ {}_p[]\ S_2]\!]\delta & = & [\![S_1]\!]p \cdot \delta + [\![S_2]\!](1-p) \cdot \delta
\end{array}
$$

Figure 4. Semantics of programs in distributions

as shown in Figure 4. This corresponds directly to a semantics given by Kozen [14], which interprets programs as continuous linear operators on measures. Our semantics uses an extension of the distribution conditioning operator $\mid$ to Boolean expressions. Whereas distribution conditioning produces a normalized distribution, Boolean expression conditioning produces an unnormalized distribution:

$$\delta|B \quad \triangleq \quad \lambda\sigma \,.\, \text{if } [\![B]\!]\sigma \text{ then } \delta(\sigma) \text{ else } 0$$

By producing unnormalized distributions as part of the meaning of **if** and **while** statements we are tracking the frequency with which each branch of the statement is chosen.

6. Related Work

We believe our work is the first to address and show the importance of attacker beliefs in quantifying information flow. Perhaps the first connection between information theory and information flow is Denning [5], who demonstrates the analysis of a few particular assignment and **if** statements, using entropy to calculate leakage. Millen [19], using deterministic state machines, proves that a system satisfies noninterference exactly when the mutual information between certain inputs and outputs is zero. He also proposes mutual information as a metric for information flow, but does not show how to compute the amount of flow for programs.

Wittbold and Johnson [25] introduce *nondeducibility on strategies*, an extension of Sutherland's *nondeducibility* [21]. Wittbold and Johnson observe that if a program is run multiple times and feedback between runs is allowed, then information can be leaked by coding schemes across multiple runs. A system that is nondeducible on strategies has no noiseless communication channels between high input and low output, even in the presence of feedback. The flow model (FM) is a security property first given by McLean [18] and later given a quantitative formalization by Gray [10], who called it the Applied Flow Model (AFM). The FM stipulates that the probability of a low output may depend on previous low outputs, but not on previous high outputs. Gray formalizes this in the context of probabilistic state machines, and he relates noninterference to the rate of maximum flow between high and low. Browne [1] develops a novel application of the idea behind the Turing test to characterize information flow: a system passes Browne's Turing test exactly when for all finite lengths of time, the information flow over that time is zero. Halpern and O'Neill [11] construct a framework for reasoning about secrecy that generalizes many previous results on qualitative and probabilistic security.

Volpano [22] gives a type system that can be used to establish the security of password checking and one-way functions such as MD5 and SHA1. Noninterference does not allow such functions to be typed, so this type system is an improvement over previous systems. However, the type system does not allow a general analysis of quantitative information flow. Volpano and Smith [23] give another type system that enforces *relative secrecy*, which requires that well-typed programs cannot leak confidential data in polynomial time.

Weber [24] defines the property *n-limited security*, which allows declassification at a rate that depends, in part, on the size n of a buffer shared by the high and low projections of a state. Lowe [15] defines the *information flow quantity* of a process with two users H and L to be the number of behaviors of H that L can distinguish. When there are n such distinguishable behaviors, H can use them to transmit $\lg n$ bits to L. These both measure the size of channels rather than accuracy of belief.

Di Pierro, Hankin, and Wiklicky [7] relax noninterference to *approximate noninterference*, where "approximate" is a quantified measure of the similarity of two processes in a process algebra. Similarity is measured using the supremum norm over the difference of the probability distributions the processes create on the store. They show how to interpret this quantity as a probability on an attacker's ability to distinguish two processes from a finite number of tests, in the sense of statistical hypothesis testing. Finally, the paper explores how to build an abstract interpretation that allows approximation of the confinement of a process. More recent work [6] has generalized this to measuring approximate confinement in probabilistic transition systems.

Clark, Hunt, and Malacaria [3] apply information theory to the analysis of **while**-programs. They develop a static analysis that provides bounds on the amount of information that can be leaked by a program. The metric for information leakage is based on conditional entropy; the analysis consists of a dataflow analysis, which computes a use-def graph, accompanied by a set of syntax-directed inference rules, which calculate leakage bounds. The analysis of Boolean and arithmetic expressions is somewhat problematic, requiring the introduction of specialized rules that apply only outside of loops. Also, the bounds on **if**-statements are calculated conservatively: the analysis does not make use of any facts that are known about the relative probability of each branch being chosen. Our work solves both of these problems by using a denotational semantics that calculates precise probability distributions; however, we have not developed a static analysis. In other work [2], the same authors investigate other leakage metrics, settling on conditional mutual information as an appropriate metric for measuring flow in probabilistic languages; they do not consider relative entropy. Mutual information is always at least 0, so unlike relative entropy it cannot represent misinformation.

McIver and Morgan [16] calculate the channel capacity of a program using conditional entropy. They add *demonic nondeterminism* as well as probabilistic choice to the language of **while**-programs, and they show that the perfect security (0 bits of leakage) of a program is determined by the behavior of its deterministic refinements. They also consider restricting the power of the demon making the nondeterministic choices, such that it can see all data, or just low data, or no data.

7. Conclusion

This paper presents a model for incorporating attacker belief into the analysis of quantitative information flow in programs. The fundamental insight is that attackers' distributions on high state represent subjective beliefs, not objective facts. A theory based on beliefs reveals that uncertainty, the traditional metric for information flow, is inadequate: it cannot satisfactorily explain even the simple example of password checking. Accuracy is the appropriate metric for information flow, and we have shown how to use it to calculate exact, expected, and maximum flow. A formal model of experiments we give enables precise descriptions of attackers' actions. We have instantiated the model with a probabilistic semantics and have given several examples of applying the model and metric to the measurement of information flow.

Acknowledgments

Stephen Chong participated in an early discussion about the distinction between attacker beliefs and reality. Sigmund Cherem, Jed Liu, Kevin O'Neill, Nathaniel Nystrom, Riccardo Pucella, and Lantian Zheng provided helpful comments on the paper.

This work was supported by the Department of the Navy, Office of Naval Research, ONR Grant N00014-01-1-0968; Air Force Office of Scientific Research, Air Force Materiel Command, USAF, grant number F49620-03-1-0156; and National Science Foundation grants 0208642, 0133302, and 0430161. Michael Clarkson is supported by a National Science Foundation Graduate Research Fellowship; Andrew Myers is supported by an Alfred P. Sloan Research Fellowship. Opinions, findings, conclusions, or recommendations contained in this material are those of the authors and do not necessarily reflect the views of these sponsors. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

References

[1] R. Browne. The Turing test and non-information flow. In *S&P 1991*, pages 375–385, Oakland, CA, 1991. IEEE.

[2] D. Clark, S. Hunt, and P. Malacaria. Quantified interference: Information theory and information flow. Presented at Workshop on Issues in the Theory of Security (WITS'04), April 2004.

[3] D. Clark, S. Hunt, and P. Malacaria. Quantified interference for a while language. *Electronic Notes in Theoretical Computer Science*, 112:149–166, Jan 2005.

[4] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley & Sons, 1991.

[5] D. Denning. *Cryptography and Data Security*. Addison-Wesley, 1982.

[6] A. Di Pierro, C. Hankin, and H. Wiklicky. Measuring the confinement of probabilistic systems. To appear in *Theoretical Computer Science*.

[7] A. Di Pierro, C. Hankin, and H. Wiklicky. Approximate non-interference. *Journal of Computer Security*, 12(1):37–81, 2004.

[8] A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin. *Bayesian Data Analysis*. Chapman and Hall/CRC, 2004.

[9] J. A. Goguen and J. Meseguer. Security policies and security models. In *Proc. IEEE Symposium on Security and Privacy*, pages 11–20, Apr. 1982.

[10] J. W. Gray, III. Toward a mathematical foundation for information flow security. In *S&P 1991*, pages 21–35, Oakland, CA, 1991. IEEE.

[11] J. Halpern and K. O'Neill. Secrecy in multiagent systems. In *CSFW 2002*, pages 32–46, Cape Breton, Nova Scotia, Canada, 2002. IEEE.

[12] J. Y. Halpern. *Reasoning about Uncertainty*. MIT Press, Cambridge, Massachusetts, 2003.

[13] G. A. Jones and J. M. Jones. *Information and Coding Theory*. Springer, 2000.

[14] D. Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 22:328–350, 1981.

[15] G. Lowe. Quantifying information flow. In *CSFW 2002*, pages 18–31, Cape Breton, Nova Scotia, Canada, 2002. IEEE.

[16] A. McIver and C. Morgan. A probabilistic approach to information hiding. In *Programming Methodology*, chapter 20, pages 441–460. Springer, 2003.

[17] A. McIver and C. Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer, 2004.

[18] J. McLean. Security models and information flow. In *S&P 1990*, pages 180–189, Oakland, CA, 1990. IEEE.

[19] J. Millen. Covert channel capacity. In *S&P 1987*, pages 60–66, Oakland, CA, 1987. IEEE.

[20] L. H. Ramshaw. *Formalizing the Analysis of Algorithms*. PhD thesis, Stanford University, 1979. Available as technical report, XEROX PARC, 1981.

[21] D. Sutherland. A model of information. In *Proceedings of the 9th National Computer Security Conference*, pages 175–183, Sep 1986.

[22] D. Volpano. Secure introduction of one-way functions. In *CSFW 2000*, pages 246–254, Cambridge, UK, 2000. IEEE.

[23] D. Volpano and G. Smith. Verifying secrets and relative secrecy. In *POPL 2000*, pages 268–276, Boston, MA, 2000. ACM.

[24] D. G. Weber. Quantitative hook-up security for covert channel analysis. In *CSFW 1988*, pages 58–71, Franconia, NH, 1988. IEEE.

[25] J. T. Wittbold and D. Johnson. Information flow in nondeterministic systems. In *S&P 1990*, pages 144–161, Oakland, CA, 1990. IEEE.

A. Proofs

Theorem 1 Let $\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L \rangle$.

$$\mathcal{B}(\mathcal{E}, o)(\sigma_H) \quad = \quad B(\mathcal{E}, o)$$

Proof.

$$\begin{array}{ll}
& B(\mathcal{E}, o) \\
= & \quad \langle \text{ Definition of } B \rangle \\
& \dfrac{b_H(\sigma_H) \cdot ([\![S]\!](\dot\sigma_L \otimes \dot\sigma_H) \upharpoonright L)(o)}{\sum_{\sigma'_H} b_H(\sigma'_H) \cdot ([\![S]\!](\dot\sigma_L \otimes \dot\sigma'_H) \upharpoonright L)(o)} \\
= & \quad \langle \text{ Definition of } \delta \upharpoonright L, \text{ apply distribution to } o \rangle \\
& \dfrac{b_H(\sigma_H) \cdot (\sum_{\sigma \mid \sigma\upharpoonright L = o} ([\![S]\!](\dot\sigma_L \otimes \dot\sigma_H)(\sigma))}{\sum_{\sigma'_H} b_H(\sigma'_H) \cdot (\sum_{\sigma \mid \sigma\upharpoonright L = o} ([\![S]\!](\dot\sigma_L \otimes \dot\sigma'_H)(\sigma))} \\
= & \quad \langle \text{ Lemma 1.1 } \rangle \\
& \dfrac{b_H(\sigma_H) \cdot (\sum_{\sigma \mid \sigma\upharpoonright L = o} ([\![S]\!](\dot\sigma_L \otimes \dot\sigma_H)(\sigma))}{\sum_{\sigma' \mid \sigma'\upharpoonright L = o} [\![S]\!](\dot\sigma_L \otimes b_H)(\sigma')} \\
= & \quad \langle \text{ Distributivity, one-point rule } \rangle \\
& \dfrac{\sum_{\sigma \mid \sigma\upharpoonright L = o \,\wedge\, \sigma\upharpoonright H = \sigma_H} \sum_{\sigma'_H} b_H(\sigma_H) \cdot [\![S]\!](\dot\sigma_L \otimes \dot\sigma_H)(\sigma)}{\sum_{\sigma' \mid \sigma'\upharpoonright L = o} [\![S]\!](\dot\sigma_L \otimes b_H)(\sigma')} \\
= & \quad \langle \text{ Lemma 1.1 } \rangle \\
& \dfrac{\sum_{\sigma \mid \sigma\upharpoonright L = o \,\wedge\, \sigma\upharpoonright H = \sigma_H} [\![S]\!](\dot\sigma_L \otimes b_H)(\sigma)}{\sum_{\sigma' \mid \sigma'\upharpoonright L = o} [\![S]\!](\dot\sigma_L \otimes b_H)(\sigma')} \\
= & \quad \langle \text{ Distributivity } \rangle \\
& \sum_{\sigma \mid \sigma\upharpoonright L = o \,\wedge\, \sigma\upharpoonright H = \sigma_H} \dfrac{[\![S]\!](\dot\sigma_L \otimes b_H)(\sigma)}{\sum_{\sigma' \mid \sigma'\upharpoonright L = o} [\![S]\!](\dot\sigma_L \otimes b_H)(\sigma')} \\
= & \quad \langle \text{ Definition of } \delta \upharpoonright L \rangle \\
& \sum_{\sigma \mid \sigma\upharpoonright H = \sigma_H} (([\![S]\!](\dot\sigma_L \otimes b_H))|o)(\sigma) \\
= & \quad \langle \text{ Definition of } \delta \upharpoonright H, \text{ applying distribution to } \sigma_H \rangle \\
& ((([\![S]\!](\dot\sigma_L \otimes b_H))|o) \upharpoonright H)(\sigma_H) \\
= & \quad \langle \text{ Definition of } \mathcal{B}(\mathcal{E}, o) \rangle \\
& \mathcal{B}(\mathcal{E}, o)(\sigma_H)
\end{array}$$

□

Lemma 1.1 Let $\sigma \upharpoonright L = o$.

$$[\![S]\!](\dot\sigma_L \otimes b_H)(\sigma) \;=\; \sum_{\sigma_H} b_H(\sigma_H) \cdot [\![S]\!](\dot\sigma_L \otimes \dot\sigma_H)(\sigma)$$

Proof.

$$\begin{array}{ll}
& [\![S]\!](\dot\sigma_L \otimes b_H)(\sigma) \\
= & \quad \langle \text{ Definition of } [\![S]\!]\delta \rangle \\
& \sum_{\sigma'} (\dot\sigma_L \otimes b_H)(\sigma') \cdot ([\![S]\!]\sigma')(\sigma) \\
= & \quad \langle \text{ Definition of point mass } \rangle \\
& \sum_{\sigma' \mid \sigma'\upharpoonright L = \sigma_L} b_H(\sigma' \upharpoonright H) \cdot ([\![S]\!]\sigma')(\sigma) \\
= & \quad \langle \text{ Let } \sigma = \langle \sigma_L, \sigma_H \rangle, \text{ nesting, one-point rule } \rangle \\
& \sum_{\sigma_H} b_H(\sigma_H) \cdot [\![S]\!](\dot\sigma_L \otimes \dot\sigma_H)(\sigma)
\end{array}$$

□

Theorem 2 Let $\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L \rangle$.

$$\mathcal{Q}(\langle \mathcal{E}, b'_H \rangle) \quad = \quad \mathcal{I}_{\delta_A}(o) - \mathcal{I}_{\delta_S}(o)$$

Proof.

$$\begin{array}{ll} & \mathcal{Q}(\langle \mathcal{E}, b'_H \rangle) \\ = & \quad \langle \text{ Definition of } \mathcal{Q} \rangle \\ & D(b_H \twoheadrightarrow \dot{\sigma}_H) - D(b'_H \twoheadrightarrow \dot{\sigma}_H) \\ = & \quad \langle \text{ Definitions of } D \text{ and point mass } \rangle \\ & -\lg b_H(\sigma_H) + \lg b'_H(\sigma_H) \\ = & \quad \langle \text{ Lemma 2.1, properties of } \lg \rangle \\ & -\lg \Pr_{\delta_A}(o) + \lg \Pr_{\delta_S}(o) \\ = & \quad \langle \text{ Definition of } \mathcal{I} \rangle \\ & \mathcal{I}_{\delta_A}(o) - \mathcal{I}_{\delta_S}(o) \end{array}$$

□

Lemma 2.1

$$b'_H(\sigma_H) = b_H(\sigma_H) \cdot \frac{\delta_S(o)}{\delta_A(o)}$$

Proof.

$$\begin{array}{ll} & b'_H(\sigma_H) \\ = & \quad \langle \text{ Definition of } \mathcal{B} \rangle \\ & ((\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H)|o) \upharpoonright H)(\sigma_H) \\ = & \quad \langle \text{ Definition of } \delta \upharpoonright H \rangle \\ & \sum_{\sigma \mid \sigma \upharpoonright H = \sigma_H} (\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H)|o)(\sigma) \\ = & \quad \langle \text{ Definition of } \delta|o \rangle \\ & \sum_{\sigma \mid \sigma \upharpoonright H = \sigma_H \wedge \sigma \upharpoonright L = o} \frac{\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H)(\sigma)}{(\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H) \upharpoonright L)(o)} \\ = & \quad \langle \text{ One-point rule: } \sigma = \langle o, \sigma_H \rangle \rangle \\ & \frac{\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H)(\langle o, \sigma_H \rangle)}{(\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H) \upharpoonright L)(o)} \\ = & \quad \langle \text{ Definition of } \delta_A \rangle \\ & \frac{1}{\delta_A(o)} \cdot \llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H)(\langle o, \sigma_H \rangle) \\ = & \quad \langle \text{ Definition of } \llbracket S \rrbracket \delta \rangle \\ & \frac{1}{\delta_A(o)} \cdot \sum_{\sigma'} (\dot{\sigma}_L \otimes b_H)(\sigma') \cdot (\llbracket S \rrbracket \sigma')(\dot{o} \otimes \dot{\sigma}_H) \\ = & \quad \langle \text{ Definition of } \otimes, \text{ point mass } \rangle \\ & \frac{1}{\delta_A(o)} \cdot \sum_{\sigma' \mid \sigma' \upharpoonright L = \sigma_L} b_H(\sigma' \upharpoonright H) \\ & \qquad \cdot (\llbracket S \rrbracket(\dot{\sigma}_L \otimes (\dot{\sigma}' \upharpoonright H)))(\dot{o} \otimes \dot{\sigma}_H) \\ = & \quad \langle \text{ High input is immutable } \rangle \\ & \frac{1}{\delta_A(o)} \cdot \sum_{\sigma' \mid \sigma' \upharpoonright L = \sigma_L \wedge \sigma' \upharpoonright H = \sigma_H} b_H(\sigma' \upharpoonright H) \\ & \qquad \cdot (\llbracket S \rrbracket(\dot{\sigma}_L \otimes (\dot{\sigma}' \upharpoonright H)))(\dot{o} \otimes \dot{\sigma}_H) \\ = & \quad \langle \text{ One-point rule: } \sigma' = \langle \sigma_L, \sigma_H \rangle \rangle \\ & \frac{1}{\delta_A(o)} \cdot b_H(\sigma_H) \cdot (\llbracket S \rrbracket(\dot{\sigma}_L \otimes \dot{\sigma}'_H))(\dot{o} \otimes \dot{\sigma}_H) \\ = & \quad \langle \text{ High input is immutable, Definition of } \delta \upharpoonright L \rangle \\ & \frac{1}{\delta_A(o)} \cdot b_H(\sigma_H) \cdot ((\llbracket S \rrbracket(\dot{\sigma}_L \otimes \dot{\sigma}'_H)) \upharpoonright L)(o) \\ = & \quad \langle \text{ Definition of } \delta_S \rangle \\ & b_H(\sigma_H) \cdot \frac{\delta_S(o)}{\delta_A(o)} \end{array}$$

Note that the immutability of high input can be dispensed with using the technique of Section 3.4. □

Theorem 3 Let:

$$\begin{array}{rcl} \mathcal{E} & = & \langle S, b_H, \sigma_H, \sigma_L \rangle \\ \delta' & = & \llbracket S \rrbracket(\dot{\sigma}_L \otimes \dot{\sigma}_H) \\ e_H & = & ((\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H))|(\delta' \upharpoonright L)) \upharpoonright H \end{array}$$

Then:

$$\mathcal{Q}_E(\mathcal{E}) \quad \leq \quad \mathcal{Q}(\langle \mathcal{E}, e_H \rangle)$$

Proof.

$$\begin{array}{ll} & \mathcal{Q}_E(\mathcal{E}) \\ = & \quad \langle \text{ Definition of } \mathcal{Q}_E \rangle \\ & E_{o \in \delta' \upharpoonright L}[\mathcal{Q}(\langle \mathcal{E}, \mathcal{B}(\mathcal{E}, o) \rangle)] \\ = & \quad \langle \text{ Definition of } \mathcal{Q}, \text{ let } b'_H = \mathcal{B}(\mathcal{E}, o) \rangle \rangle \\ & E_{o \in \delta' \upharpoonright L}[D(b_H \twoheadrightarrow \dot{\sigma}_H) - D(b'_H \twoheadrightarrow \dot{\sigma}_H)] \\ = & \quad \langle \text{ Linearity of } E \rangle \\ & D(b_H \twoheadrightarrow \dot{\sigma}_H) - E_{o \in \delta' \upharpoonright L}[D(b'_H \twoheadrightarrow \dot{\sigma}_H)] \\ \leq & \quad \langle \text{ Jensen's inequality and convexity of } D, \text{ see [4] } \rangle \\ & D(b_H \twoheadrightarrow \dot{\sigma}_H) - D(E_{o \in \delta' \upharpoonright L}[b'_H] \twoheadrightarrow \dot{\sigma}_H) \\ & \quad = \quad \langle \text{ Lemma 3.1 } \rangle \\ & \quad D(b_H \twoheadrightarrow \dot{\sigma}_H) - D(e_H \twoheadrightarrow \dot{\sigma}_H) \\ & \quad = \quad \langle \text{ Definition of } \mathcal{Q} \rangle \\ & \quad \mathcal{Q}(\langle \mathcal{E}, e_H \rangle) \end{array}$$

□

Lemma 3.1 Let $\mathcal{E}$, δ', e_H be defined as in Theorem 3. Let $b'_H = \mathcal{B}(\mathcal{E}, o)$ and assume the range of o is always $\delta' \upharpoonright L$. Then:

$$E_o[b'_H] \quad = \quad e_H$$

Proof.

$$\begin{array}{ll} & E_o[b'_H](\sigma_H) \\ = & \quad \langle \text{ Definitions of } E, b'_H \rangle \\ & (\sum_o (\delta' \upharpoonright L)(o) \cdot \mathcal{B}(\mathcal{E}, o)(\sigma_H) \\ = & \quad \langle \text{ Definition of } \mathcal{B}(\mathcal{E}, o) \rangle \\ & \sum_o (\delta' \upharpoonright L)(o) \cdot ((((\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H))|o) \upharpoonright H)(\sigma_H) \\ = & \quad \langle \text{ Definition of } \delta \upharpoonright H, \text{ applying distribution to } \sigma_H \rangle \\ & \sum_o (\delta' \upharpoonright L)(o) \\ & \quad \cdot (\sum_{\sigma' \mid \sigma' \upharpoonright H = \sigma_H} ((\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H))|o)(\sigma')) \end{array}$$

$$
\begin{array}{ll}
= & \langle \text{ Definition of } \delta|o, \text{ applying distribution to } \sigma' \ \rangle \\
& \sum_o (\delta' \upharpoonright L)(o) \\
& \quad \cdot (\sum_{\sigma' \mid \sigma' \upharpoonright H = \sigma_H \ \wedge \ \sigma' \upharpoonright L = o} \ \frac{([\![S]\!](\dot\sigma_L \otimes b_H))(\sigma')}{([\![S]\!](\dot\sigma_L \otimes b_H) \upharpoonright L)(o)}) \\
= & \langle \text{ One-point rule } \rangle \\
& \sum_o (\delta' \upharpoonright L)(o) \cdot \frac{([\![S]\!](\dot\sigma_L \otimes b_H))(\langle o, \sigma_H \rangle)}{([\![S]\!](\dot\sigma_L \otimes b_H) \upharpoonright L)(o)} \\
= & \langle \text{ Definition of } \delta \upharpoonright L, \text{ applied to } o \ \rangle \\
& \sum_o (\delta' \upharpoonright L)(o) \cdot \frac{([\![S]\!](\dot\sigma_L \otimes b_H))(\langle o, \sigma_H \rangle)}{\sum_{\sigma' \mid \sigma' \upharpoonright L = o} [\![S]\!](\dot\sigma_L \otimes b_H)(\sigma')} \\
= & \langle \text{ Let } \sigma = \langle o, \sigma_H \rangle, \text{ change of dummy: } o := \sigma, \\
& \quad \text{definition of } \approx_L \ \rangle \\
& \sum_{\sigma \mid \sigma \upharpoonright H = \sigma_H} (\delta' \upharpoonright L)(o) \\
& \quad \cdot \frac{([\![S]\!](\dot\sigma_L \otimes b_H))(\sigma)}{\sum_{\sigma' \mid \sigma' \approx_L \sigma} [\![S]\!](\dot\sigma_L \otimes b_H)(\sigma')} \\
= & \langle \text{ Definition of } \delta|\delta_L, \text{ applied to } \sigma \ \rangle \\
& \sum_{\sigma \mid \sigma \upharpoonright H = \sigma_H} ([\![S]\!](\dot\sigma_L \otimes b_H)|(\delta' \upharpoonright L))(\sigma) \\
= & \langle \text{ Definition of } \delta \upharpoonright H, \text{ applied to } \sigma_H \ \rangle \\
& ([\![S]\!](\dot\sigma_L \otimes b_H)|(\delta' \upharpoonright L)) \upharpoonright H)(\sigma_H) \\
= & \langle \text{ Definition of } e_H \ \rangle \\
& e_H(\sigma_H)
\end{array}
$$

Therefore $E_o[b'_H] = e_H$ by extensionality.
□

Theorem 4

$$D(b_{H_0} \twoheadrightarrow \dot\sigma_H) - D(b'_{H_n} \twoheadrightarrow \dot\sigma_H) \quad = \quad \sum_i \ \mathcal{Q}(r_i)$$

Proof.

$$
\begin{array}{ll}
& \sum_{i \mid 1 \le i \le n} \ \mathcal{Q}(r_i) \\
= & \langle \text{ Definition of } \mathcal{Q} \ \rangle \\
& \sum_{i \mid 1 \le i \le n} \ D(b_{H_i} \twoheadrightarrow \dot\sigma_H) - D(b'_{H_i} \twoheadrightarrow \dot\sigma_H) \\
= & \langle \text{ Lemma 4.1, } f(i) = D(b_{H_i} \twoheadrightarrow \dot\sigma_H), \\
& \quad f'(i) = D(b'_{H_i} \twoheadrightarrow \dot\sigma_H), b_{H_i} = b'_{H_{i-1}}, \\
& \quad b'_{H_0} = b_{H_1} \ \rangle \\
& D(b_{H_1} \twoheadrightarrow \dot\sigma_H) - D(b'_{H_n} \twoheadrightarrow \dot\sigma_H)
\end{array}
$$

□

Lemma 4.1 Assume for a pair of functions f and f' that $\forall_{i \mid 1 \le i \le n} \ f(i) = f'(i-1)$, $n \ge 2$, and $f(1) = f'(0)$. Then:

$$(\sum_{i \mid 1 \le i \le n} \ f(i) - f'(i)) = f(1) - f'(n)$$

Proof.

$$
\begin{array}{ll}
& \sum_{i \mid 1 \le i \le n} \ f(i) - f'(i) \\
= & \langle \ f(i) = f'(i-1) \ \rangle \\
& \sum_{i \mid 1 \le i \le n} \ f'(i-1) - f'(i) \\
= & \langle \text{ Distributivity } \rangle \\
& \sum_{i \mid 1 \le i \le n} \ f'(i-1) \\
& - \sum_{i \mid 1 \le i \le n} \ f'(i) \\
= & \langle \text{ Change of dummy: } i := i-1 \ \rangle \\
& \sum_{i \mid 0 \le i \le n-1} \ f'(i) \\
& - \sum_{i \mid 1 \le i \le n} \ f'(i) \\
= & \langle \text{ Split off term, } n \ge 2 \ \rangle \\
& f'(0) + (\sum_{i \mid 1 \le i \le n-1} \ f'(i)) \\
& -(\sum_{i \mid 1 \le i \le n-1} \ f'(i)) - f'(n) \\
= & \langle \text{ Arithmetic } \rangle \\
& f'(0) - f'(n) \\
= & \langle \ f(1) = f'(0) \ \rangle \\
& f(1) - f'(n)
\end{array}
$$

□