Quantifying Information Flow with Beliefs

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

Abstract

To reason about information flow, a new model is developed that describes how attacker beliefs change due to the attacker's observation of the execution of a probabilistic (or deterministic) program. The model enables compositional reasoning about information flow from attacks involving sequences of interactions. The model also supports a new metric for quantitative information flow that measures accuracy of an attacker's beliefs. Applying this new metric reveals inadequacies of traditional information flow metrics, which are based on reduction of uncertainty. However, the new metric is sufficiently general that it can be instantiated to measure either accuracy or uncertainty. The new metric can also be used to reason about misinformation; deterministic programs are shown to be incapable of producing misinformation. Additionally, programs in which nondeterministic choices are made by insiders, who collude with attackers, can be analyzed.

1 Introduction

Qualitative security properties, such as noninterference [13], typically either prohibit any flow of information from a high security level to a lower level, or they allow any information to flow provided it passes through some release mechanism. For a program whose correctness requires flow from high to low, the former property is too restrictive and the latter can lead to unbounded leakage of information. Quantitative flow policies, such as "at most k bits leak per execution of the program", allow information flows but at restricted rates. Such policies are useful when analyzing programs whose nature requires that some—but not too much—information be leaked. Examples of these programs include guards, which sit at the boundary between trusted and untrusted systems, and password checkers.

Quantifying information flow is more difficult than it might seem. Consider a password checker PWC that sets an authentication flag a after 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.) If the attacker experiments by executing PWC and guessing A, he expects to observe that a equals 1 upon termination. Such a confirmation of the attacker's belief would seem to convey some small amount of information. But suppose that the informant was wrong: the real password is C. Then the attacker observes that a is equal to 0 and 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 has greatly changed—he is surprised to discover the password is not A—so the outcome of this experiment conveys more information than the previous outcome. Thus, the information conveyed by executing PWC depends on what the attacker initially believed.

How much information flows from p to a in each of the above experiments? Answers to this question have traditionally been based on change in uncertainty [8, 31, 14, 2, 24, 3, 27]: information flow is measured by the reduction in uncertainty about secret data. Observe that, in the case where the password is 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 a metric based on reduction in uncertainty, no information flow occurred, which contradicts our intuition.

The problem with metrics based on uncertainty is twofold. First, they do 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. Second, uncertainty-based metrics are concerned with some unspecified agent's uncertainty rather than an attacker's. The unspecified agent is able to observe a probability distribution over secret input values but cannot observe the particular secret input used in the program execution. If the attacker were the unspecified agent, then there would be no reason in general to assume that the probability distribution the attacker uses is correct. Because the attacker's probability distribution is therefore subjective, it must be treated as a belief. Beliefs are thus an essential—though until now uninvestigated—component of information flow.

This paper presents a new way of measuring information flow, based on these insights. Section 2 gives basic representations for beliefs and programs. Section 3 describes a model of the interaction between attackers and systems; it also describes how attackers update beliefs by observing execution of programs. Section 4 defines a new quantitative flow metric, based on information theory, that characterizes the amount of information flow due to changes in the accuracy of an attacker's belief. The metric can also be instantiated to measure change in uncertainty, and thus it generalizes previous information-flow metrics. The model and metric are formulated for use with any programming model that can be given a denotational semantics compatible with the representation of beliefs, and Section 5 illustrates with a particular programming language (**while**-programs plus probabilistic choice). Section 6 extends the model to programs in which nondeterministic choices are resolved by insiders, who are allowed to observe secret values. Section 7 discusses related work, and Section 8 concludes. This paper revises and expands an earlier version [6] that appeared in CSFW'05.

2 Incorporating 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. Agents will reason about probabilistic programs, so we begin by developing mathematical structures for representing probabilistic programs and 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; it is easier to define a programming language semantics using frequency distributions than using probability distributions [32].[1] 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 mostly unimportant; it can be instantiated according to the needs of any particular language or system. For our examples, states map variables to values, where **Var** and **Val** are both countable sets.

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

We write a state as a list of mappings; for example, $(g \mapsto A, a \mapsto 0)$ is a state in which variable g has value A and a has value 0.

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

$$\|\delta\| \quad \triangleq \quad \textstyle\sum_\sigma \, \delta(\sigma)$$

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

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

2.2 Programs

Execution of program S is described by a denotational semantics in which the meaning $[\![S]\!]$ of S is a function of type **State** $\to$ **Dist**. This semantics describes the frequency of termination in a given state: if $[\![S]\!]\sigma = \delta$, then the frequency of S, when begun

[1] The distribution notation in this paper is used in lieu of traditional random variable notation. A distribution is the probability density function associated with a random variable. When defining basic operations, the random variable notation might look more familiar. In our more advanced definitions, such as belief distance and program semantics, the distribution notation turns out to be superior because those definitions are most naturally expressed using the density function. Indeed, in most of the paper, it is the density function with which we are concerned, and that is why we use the distribution notation throughout.

in σ, terminating in σ' is $\delta(\sigma')$. This semantics can be lifted to a function of type **Dist** $\rightarrow$ **Dist** by the following definition:

$$[\![S]\!]\delta \quad \triangleq \quad \sum_{\sigma} \delta(\sigma) \cdot [\![S]\!]\sigma$$

Thus, the meaning of S over a distribution of inputs is completely determined by the meaning of S given a state as input. By defining programs in terms of how they operate on distributions we permit analysis of probabilistic programs. Section 5 shows how to build such a semantics.

Our examples use **while**-programs extended with a probabilistic choice construct. Let 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 prescribe their syntax or semantics. The syntax of the language is:

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

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 $1 - p$, where $0 \leq p \leq 1$.

2.3 Labels and Projections

We need a way to identify secret data; *confidentiality labels* serve this purpose. For simplicity, assume there are only two 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 $\textbf{State}_L$ and $\textbf{State}_H$, which contain the low- and high-labeled data, respectively. A *low state* is an element $\sigma_L \in \textbf{State}_L$; a *high state* is an element $\sigma_H \in \textbf{State}_H$. The projection of state $\sigma \in \textbf{State}$ onto $\textbf{State}_L$ is denoted $\sigma \restriction L$; this is the part of σ visible to the attacker. Projection onto $\textbf{State}_H$, the part of σ not visible to the attacker, is denoted $\sigma \restriction H$.

Each variable in a program is subscripted by a label to indicate the confidentiality of the information stored in that variable; for example, x_L is a variable that contains low information. For convenience, let variable l be labeled L and variable h be labeled H. $\textbf{Var}_L$ is the set of variables in a program that are labeled L, so $\textbf{State}_L = \textbf{Var}_L \rightarrow \textbf{Val}$. The *low projection* $\sigma \restriction L$ of state σ is:

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

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 combined frequency of those states whose low projection is σ_L:[2]

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

[2]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 $R \equiv \mathit{true}$.

High projection and high equivalence are defined by replacing occurrences of L with H in the definitions above.

2.4 Belief Representation

To be usable in our framework, a belief representation must support certain natural operations. Let b and b' be beliefs ranging over sets of possible worlds W and W', respectively, where a *world* is an elementary outcome about which beliefs can be held.

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' are disjoint.
2. *Belief update* $b|U$ is the belief that results when b is updated to include 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 $r \geq 0$ quantifying differences between b and b'.

While the results in this paper are, for the most part, independent of any particular representation, the rest of this paper uses distributions to represent beliefs. High states are the possible worlds for beliefs, and a belief is a probability distribution over high states.

$$b \in \textbf{Belief} \quad \triangleq \quad \textbf{State}_H \to \mathbb{R}^+ \quad \text{s.t. } \|b\| = 1$$

Whereas distributions correspond to positive measures, beliefs correspond to probability measures. Probability measures are well-studied as a belief representation [18], and they have several advantages here: they are familiar, quantitative, support the operations required above, 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 actual state of the system [18]. Other choices of belief representation could include belief functions or sets of probability measures [18]. While these are more expressive than probability measures, it is more complicated to define the required operations for them.

For belief product $\otimes$, we employ a distribution product $\otimes$ of two distributions $\delta_1 : A \to \mathbb{R}^+$ and $\delta_2 : B \to \mathbb{R}^+$, with A and B disjoint:

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

It is easy to check that if b and b' are beliefs, $b \otimes b'$ 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 [20] for the distance between distributions.

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

The base of the logarithm in D can be chosen arbitrarily; we use base 2 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' [7]. Like an analytic metric, $D(b \twoheadrightarrow b')$ is always at least zero and $D(b \twoheadrightarrow b')$ equals zero only when $b = b'$.[3]

Relative entropy has the property that if $b'(\sigma) > 0$ and $b(\sigma) = 0$, then $D(b \twoheadrightarrow b') = \infty$. Intuitively, b' is "infinitely surprising" because it regards σ as possible whereas b regards σ as impossible. To avoid this anomaly, beliefs may be required to satisfy an *admissibility restriction*, which ensures that attackers do not initially believe that certain states are impossible. For example, a belief might be restricted such that it never differs by more than a factor of ϵ from a uniform distribution. Or, the attacker's belief may be required to be a maximal entropy distribution [7] with respect to attacker-specified constraints. Other admissibility restrictions may be substituted for these when stronger assumptions can be made about attacker beliefs.

3 Experiments

We formalize as an *experiment* how an *attacker*, an agent that reasons about secret data, 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.

We are chiefly interested in the program S with which the attacker is interacting, and 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.

3.1 Experiment Protocol

Formally, an experiment $\mathcal{E}$ is described by 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, which uses some notation defined below, is summarized in Figure 1. Here is a justification for the protocol.

[3] Unlike an analytic metric, D does not satisfy symmetry or the triangle inequality. However, it seems unreasonable to assume that either of these properties holds for beliefs, since it can be easier to rule out a possibility from a belief than to add a new one, or vice-versa.

Figure 1 Experiment protocol

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 attacker predicts the output distribution: $\delta'_A = [\![S]\!](\dot{\sigma}_L \otimes b_H)$.
4. The system executes the program S, which produces a state $\sigma' \in \delta'$ as output, where $\delta' = [\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}_H)$. The attacker observes the low projection of the output state: $o = \sigma' \restriction L$.
5. The attacker infers a postbelief: $b'_H = (\delta'_A | o) \restriction H$.

An attacker's *prebelief*, describing his belief at the beginning of the experiment (step 1), may be chosen arbitrarily (subject to an admissibility restriction as in Section 2.4) or may be informed by previous experiments. In a series of experiments, the *postbelief* from one experiment typically becomes the prebelief to the next. The attacker might even choose a prebelief b_H that contradicts his true subjective probability distribution for the state, and this gives our analysis additional power by allowing the attacker to conduct experiments to answer questions such as "What would happen if I were to believe b_H?".

The system chooses σ_H (step 2a), the high projection of the initial state, and this part of the state might remain constant from one experiment to the next or might vary. For example, Unix passwords do not usually change frequently, but the output displayed on an RSA SecurID token changes each minute. We conservatively assume that the attacker chooses all of σ_L (step 2b), the low projection of the initial state.[4] This gives the attacker additional power in controlling execution of the program, which he can use to attempt to maximize the amount of information flow. The attacker's choice of σ_L is thus likely to be influenced by b_H, but for generality, we do not require there be such a strategy.

Using the semantics of S along with prebelief b_H as a distribution on high input, the attacker conducts a "thought experiment" to generate a *prediction* of the output distribution (step 3). We define prediction δ'_A to correlate the output state with the high input state:

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

Program S is executed (step 4) only once in each experiment; multiple executions are modeled by multiple experiments. The meaning of S given inputs σ_L and σ_H is an

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

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 is actually produced. This output state σ' is produced with frequency $\delta'(\sigma')$. We write:

$$\sigma' \in \delta'$$

to denote that σ' is in the support of (i.e., has positive frequency according to) δ'. In a single experiment, the attacker is allowed only a single observation. The observation o resulting from σ' is:

$$o = \sigma' \restriction L$$

Finally, the attacker incorporates any new inferences that can be made from observation o by conditioning prediction δ'_A. The result is projected to H to produce the attacker's postbelief b'_H (step 5):

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

Here, conditioning operator $\delta|o$ is defined in terms of conditioning operator $\delta|U$. The new operator removes all mass in distribution δ that is inconsistent with observation o, then normalizes the result:

$$\begin{array}{rcl} \delta|o & \triangleq & \delta|\{\sigma' \mid \sigma' \restriction L = o\} \\ & = & \lambda\sigma\,.\,\text{if } (\sigma \restriction L) \;=\; o \text{ then } \frac{\delta(\sigma)}{(\delta\restriction L)(o)} \text{ else } 0 \end{array}$$

3.2 Password Checking as an Experiment

Our experiment model allows the informal reasoning in Section 1 to be made precise. For example, consider the password checker; adding confidentiality labels yields:

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

The attacker begins an experiment by choosing prebelief b_H, perhaps as specified in the column labeled b_H of Table 1. Next, the system chooses initial high projection σ_H, and the attacker chooses 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 executes PWC. Output distribution δ' is 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 $\sigma' \restriction L = (g \mapsto A, a \mapsto 1)$.

Finally, the attacker infers a postbelief. He conducts 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

Table 1 Beliefs about p_H

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

Table 2 Distributions on PWC output

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

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 can also be formalized. 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 $(g \mapsto A, a \mapsto 0)$. 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

The formula the attacker uses to infer a postbelief 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 [12]. The attacker therefore reasons rationally, according to Halpern's rationality axioms [17], though the literature on human behavior shows that this is not always the same as human reasoning [21, 22].

Let belief revision operator $\mathcal{B}$ yield the postbelief from an experiment $\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L \rangle$, given observation o:

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

We write $b'_H \in \mathcal{B}(\mathcal{E})$ to denote that there exists some o for which $b'_H = \mathcal{B}(\mathcal{E}, o)$.

Recall Bayes' rule for updating a hypothesis Hyp with an observation obs:

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

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 the attacker 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 Bayesian inference $BI(\mathcal{E}, o)$, which is $\Pr(\sigma_H|o)$:

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

With this instantiation, we can show that the experiment protocol leads an attacker to update his belief according to Bayesian inference.

Theorem 1

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

Proof. In Appendix B. □

3.4 Mutable High State and Nontermination

Section 3.1 invokes two simplifying assumptions about program S: it never modifies high input, and it always terminates. We now dispense with these 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 (in step 5), δ'_A must correlate high *inputs* with low outputs. So our experiment protocol requires the high input state be preserved in δ'_A. Informally, we can do this by keeping a copy of the initial high inputs in the program state. This copy is never modified by the program. Thus, the copy is preserved in the final output state, and the attacker can again establish a correlation between high inputs and low outputs. Technical details are given in Appendix A.

To eliminate the second assumption, note that program S must terminate for an attacker to obtain a low state as an observation when executing 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 with a state $\perp$ representing the divergent state. Details of this approach are given in Appendix A. An attacker that cannot detect nontermination is more difficult to model. At some point

during the execution of the program, he may 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. We use change in accuracy, as measured by belief distance D, to quantify information flow.

4.1 Information Flow from an Outcome

Given an experiment $\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L \rangle$, an *outcome* is a postbelief b'_H such that $b'_H \in \mathcal{B}(\mathcal{E})$, where $\mathcal{B}$ is the belief revision operator from Section 3.3. Recall from Section 2.4 that $D(b \twoheadrightarrow b')$ is the distance from belief b to belief b'. The accuracy of the attacker's prebelief b_H in experiment $\mathcal{E}$ is $D(b_H \twoheadrightarrow \dot{\sigma}_H)$; the accuracy of outcome b'_H, the attacker's postbelief, is $D(b'_H \twoheadrightarrow \dot{\sigma}_H)$. We define the amount of information flow $\mathcal{Q}$ caused by outcome b'_H of experiment $\mathcal{E}$ as the difference of these two quantities:

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

Thus the amount of information flow $\mathcal{Q}$ is the improvement in the accuracy of the attacker's belief. This amount can positive or negative; we defer discussion of negative flow to Section 4.3. Since D is instantiated with relative entropy, the unit of measurement for $\mathcal{Q}$ is (information-theoretic) bits.

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, when an event that has probability 1 occurs, no information (0 bits) is conveyed because the occurrence is completely unsurprising.

For an attacker, the outcome of an experiment involves two unknowns: the initial high state σ_H 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 probabilistic choices made by the program.[5] For programs that make no probabilistic choices, 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,

[5]The technique used in Section 3.4 for modeling nontermination ensures that δ_A and δ_S are probability distributions. Thus, $\mathcal{I}_{\delta_A}$ and $\mathcal{I}_{\delta_S}$ are well-defined.

$\mathcal{I}_{\delta_S}(o)$ is generally not equal to 0; subtracting it removes all the information contained in $\mathcal{I}_{\delta_A}(o)$ that is solely about the results of probabilistic choices, leaving information about high inputs only.

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

Theorem 2

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

Proof.

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

□

As an example, consider the experiments involving 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 Table 1 defines b_H and the other beliefs about to be used. Only one outcome, b'_{H1}, is possible from this experiment. We calculate the amount of flow from this outcome, letting $\sigma_H = (p \mapsto A)$:

$$\begin{array}{rcl} \mathcal{Q}(\mathcal{E}_1, b'_{H1}) & = & D(b_H \twoheadrightarrow \dot{\sigma}_H) - D(b'_{H1} \twoheadrightarrow \dot{\sigma}_H) \\ & = & \sum_{\sigma'_H} \dot{\sigma}_H(\sigma'_H) \cdot \lg \frac{\dot{\sigma}_H(\sigma'_H)}{b_H(\sigma'_H)} \; - \; \sum_{\sigma'_H} \dot{\sigma}_H(\sigma'_H) \cdot \lg \frac{\dot{\sigma}_H(\sigma'_H)}{b'_{H1}(\sigma'_H)} \\ & = & -\lg b_H(\sigma_H) + \lg b'_{H1}(\sigma_H) \\ & = & 0.0291 \end{array}$$

This small flow makes sense because the outcome 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 outcome is possible from this experiment, and calculating $\mathcal{Q}(\mathcal{E}_2, b'_{H2})$ yields an information flow of 5.6439 bits. This 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. This suggests that $\mathcal{Q}$ is the right metric for the information about high input contained in the observation.

The information flow of 5.6439 bits in experiment $\mathcal{E}_2$ might seem surprisingly high. At most two bits are required to store password p in memory, so why does the program leak more than five bits? Here, the greater leakage occurs because the attacker's belief is not uniform. A uniform prebelief (ascribing $1/3$ probability to each password A, B, and C) would, in a series of experiments, cause the attacker to learn a total of $\lg 3 \approx 1.6$ bits. However, belief b_H is more erroneous than the uniform belief, so a larger amount of information is required to correct it.

An uncertainty-based definition for information flow does not produce a reasonable leakage for this experiment. The attacker's initial uncertainty about p is $\mathcal{H}(b_H) = 0.1614$ bits, where $\mathcal{H}$ is the information-theoretic metric of *entropy*, or uncertainty, in a probability distribution δ:

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

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 Denning's [8], 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 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.

4.2 Interpreting Metric $\mathcal{Q}$

According to Theorem 2, metric $\mathcal{Q}$ correctly measures the amount of information flow, in bits. But what does it mean to leak one bit of information? The next theorem states that k bits of leakage correspond to a k-fold doubling of the probability that the attacker ascribes to reality.

Theorem 3 *Let* $\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L \rangle$.

$$\mathcal{Q}(\mathcal{E}, b'_H) = k \quad \equiv \quad b'_H(\sigma_H) = 2^k \cdot b_H(\sigma_H)$$

Proof. In Appendix B. □

Suppose an attacker were to guess what reality is by sampling from his belief b_H; the probability he guesses correctly is $b_H(\sigma_H)$. Thus, by Theorem 3, one bit of leakage makes the attacker twice as likely to guess correctly. This reveals an interesting analogy with the uncertainty-based definition. In it, one bit of leakage corresponds to the attacker becoming twice as certain about the high state, though he may, as the example in Section 4.1 shows, become certain about the wrong high state. However, one bit of leakage in our accuracy-based definition corresponds to the attacker becoming twice as certain about the *correct* high state.

Figure 2 Effect of $FLIP$ on postbelief

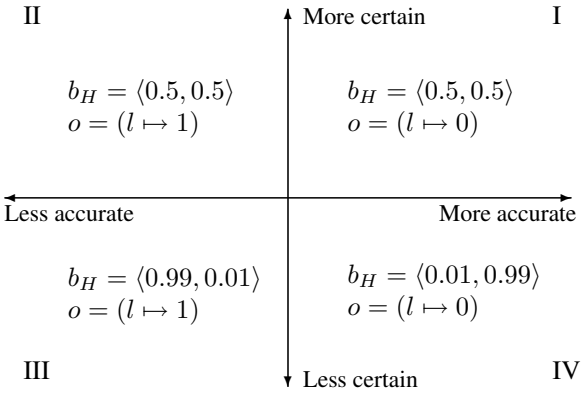

Table 3 Analysis of $FLIP$

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

4.3 Accuracy, Uncertainty, and Misinformation

Accuracy and uncertainty are orthogonal properties of beliefs, as depicted in Figure 2. The figure shows the change in an attacker's accuracy and uncertainty when the program

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

is analyzed with experiment $\mathcal{E} = \langle FLIP, b_H, (h \mapsto 0), (l \mapsto 0)\rangle$ and observation o is generated by the experiment. The notation $b_H = \langle x, y\rangle$ in Figure 2 means that $b_H(h \mapsto 0) = x$ and $b_H(h \mapsto 1) = y$.

Usually, $FLIP$ sets l to be h, so the attacker will expect this to be the case. Executions in which this occurs will cause his postbelief to be more accurate, but may cause his uncertainty to either increase or decrease, depending on his prebelief; when uncertainty increases, an uncertainty metric would mistakenly say that no flow has occurred.

With probability 0.01, $FLIP$ produces an execution 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 executions will make him more uncertain. But when the attacker's prebelief is uniform, executions that result in misinformation will make him less uncertain; when uncertainty decreases, an uncertainty metric would mistakenly say that flow has occurred.

Table 3 concretely demonstrates the orthogonality of accuracy and uncertainty. The quadrant labels refer to Figure 2. The attacker's prebelief b_H, observation o, and resulting postbelief b'_H are given in the top half of the table. In the bottom half of the table, increase in accuracy is calculated using information flow metric $\mathcal{Q}$, and reduction in uncertainty is calculated using the difference in entropy $\mathcal{H}(b_H) - \mathcal{H}(b'_H)$. The symmetries in the bottom half of the table are a result of the symmetries between prebeliefs and postbeliefs. Quadrants II and IV, for example, have exchanged these beliefs, which for both metrics has the effect of negating the amount of information flow.

The probabilistic choice in $FLIP$ is essential for producing misinformation, as shown by the following theorem. Let **Det** be the set of syntactically deterministic programs, i.e., programs that do not contain any probabilistic choice. Because they lack a source of randomness, these programs cannot decrease the accuracy of an attacker's belief.

Theorem 4

$$S \in \textbf{Det} \quad \Rightarrow \quad \forall \mathcal{E}, b'_H \in \mathcal{B}(\mathcal{E}) \,.\, \mathcal{Q}(\mathcal{E}, b'_H) \geq 0$$

Proof. In Appendix B. □

4.4 Emulating Uncertainty

The accuracy metric of Section 4.1 generalizes uncertainty metrics. Informally, this is because uncertainty metrics recognize only two distributions (belief before and after execution), whereas our framework recognizes these plus one additional distribution (reality). By ignoring reality, our framework can produce the same results as many uncertainty metrics. Here we show how to emulate the metric of Clark, Hunt, and Malacaria [5]. Their metric states that the amount of information flow $\mathcal{L}$ from high input H_{in} into low output L_{out}, given low input L_{in}, is:

$$\mathcal{L}(H_{in}, L_{in}, L_{out}) \quad \triangleq \quad \mathcal{H}(H_{in}|L_{in}) - \mathcal{H}(H_{in}|L_{in}, L_{out})$$

where $\mathcal{H}$ is the generalization of the entropy function from Section 4.1 to conditional entropy [7].[6]

First, to instantiate our framework to that of Clark et al., we force our framework to ignore reality by introducing an admissibility restriction (cf. Section 2.4): prebeliefs must be identical to the system's chosen high input distribution. This means that prebeliefs must be correct; there can be no error in the attacker's estimate of the probability distribution on high inputs.

[6]Their metric more generally allows the measurement of information flow into any subset of the output variables. The approach we give here can similarly be generalized.

Second, we adjust the definition of belief. The uncertainty model of Clark et al. calculates information flow as an expectation over a probability distribution on both low and high inputs. We could model this using the techniques about to be introduced in Sections 4.5 and 4.6, but because of the admissibility restriction just made, it is equivalent and simpler to allow beliefs to range over low state as well as high state. As before, we assume that high state remains constant using the copying technique of Section 3.4. Since beliefs now include low state, we must also apply this technique to assure that the initial values of low variables are preserved in the state. Let the low input component of the state be denoted L^0. Assume that the attacker's prebelief b ranges over $L^0 \cup H^0$, whereas his postbelief b' ranges over $L^0 \cup H^0 \cup L \cup H$.

We want to establish that accuracy metric $\mathcal{Q}$ yields the same result as uncertainty metric $\mathcal{L}$ for any outcome. Recall that $\mathcal{Q}$ is defined in terms of distance function D. Our previous instantiation of D as relative entropy yielded an accuracy metric. Now we reinstantiate D using (non-relative) entropy:

$$D(b \rightarrowtriangle b') \quad = \quad \mathcal{H}(b \upharpoonright (L \cup L^0 \cup H^0)) - \mathcal{H}(b \upharpoonright (L \cup L^0))$$

Observe that this instantiation ignores argument b', the belief representing reality. This yields that amount of information flow $\mathcal{Q}$ is the same as uncertainty metric $\mathcal{L}$.

Theorem 5

$$\mathcal{Q}(\mathcal{E}, b') \quad = \quad \mathcal{L}(H_{in}, L_{in}, L_{out})$$

Proof. In Appendix B. □

4.5 Expected Flow for an Experiment

Since an experiment on a probabilistic program can produce many observations, and therefore many outcomes, it is desirable to characterize expected flow over those outcomes. So we define expected flow $\mathcal{Q}_{\mathsf{E}}$ over all observations from experiment $\mathcal{E}$:

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

where $\delta' \upharpoonright L = [\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}_H) \upharpoonright L$ gives the distribution on observations; $\mathsf{E}_{\sigma \in \delta}[X(\sigma)]$ is the expected value, with respect to distribution δ, of expression X with free variable σ; and $\mathcal{B}$ is the belief revision operator from Section 3.3.

Expected flow is useful in analyzing probabilistic programs. 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, $FPWC$ inverts the authentication flag. Can this program be expected to confound attackers—does $FPWC$ leak less expected information than PWC? This question can be answered by comparing the expected flow from $FPWC$

Table 4 Leakage of PWC and $FPWC$

$\mathcal{E}$	o	$\mathcal{Q}(\mathcal{E}, \mathcal{B}(\mathcal{E}, o))$	$\mathcal{Q}_{\mathsf{E}}(\mathcal{E})$
$\mathcal{E}_1$	$(a \mapsto 1)$	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	

to the flow of PWC. Table 4 gives information flows from $FPWC$ 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. Observations $(a \mapsto 0)$ and $(a \mapsto 1)$ correspond to an execution where the value of a is inverted. The flow for the outcomes resulting from these observations is negative, indicating that the program is giving the attacker misinformation. Note that, for both pairs of experiments in Table 4, the expected flow of $FPWC$ is less than the flow of PWC. We have confirmed that the random corruption of a makes it more difficult for the attacker to increase the accuracy of his belief.

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 $\delta|\sigma_L$ unchanged for all σ_L.

$$\delta|\delta_L \quad \triangleq \quad \lambda\sigma \,.\, \frac{\delta(\sigma)}{(\delta \restriction L)(\sigma \restriction L)} \cdot \delta_L(\sigma \restriction L)$$

A bound on expected flow is then calculated as follows. Given experiment $\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L \rangle$, let δ' be the distribution that results from the system executing S as in step 4 of the experiment protocol, i.e., $\delta' = [\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}_H)$. In the experiment protocol, an attacker would observe the low projection of a state from δ'. But suppose that the attacker instead observed the low projection of δ' itself. (This projection is the distribution over observations that the attacker would approach if he continued to repeat $\mathcal{E}$.) Let e_H be the postbelief that results from conditioning on this distribution, as in step 5 of the protocol: $e_H = (([\![S]\!](\dot{\sigma}_L \otimes b_H))|(\delta' \restriction L)) \restriction H$. Intuitively, e_H is the attacker's expected postbelief with respect to $\delta' \restriction L$. The amount of information flow from expected postbelief e_H then bounds the expected amount of information flow, as stated in the following theorem.

Theorem 6 *Let:*

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

Then:

$$\mathcal{Q}_{\mathsf{E}}(\mathcal{E}) \quad \leq \quad \mathcal{Q}(\mathcal{E}, e_H)$$

Proof. In Appendix B. □

As an example, consider experiment $\mathcal{E}_2^F$. Calculating the attacker's expected postbelief e_H in this experiment yields $e_H = \langle 0.8601, 0.0699, 0.0699 \rangle$, using the postbelief notation from Section 4.3. Bound $\mathcal{Q}(\mathcal{E}, e_H)$ from Theorem 6 is thus 6.4264 bits, which is indeed greater than expected flow $\mathcal{Q}_{\mathsf{E}}$ as calculated in Table 4.

4.6 Expected Flow over All Experiments

Uncertainty-based metrics typically consider the expected information flow over all experiments, rather than the flow in a single experiment. An analysis, like ours, based on single experiments allows a more expressive language of security properties in which particular inputs or experiments can be considered. Moreover, our analysis can be extended to calculate expected flow over all experiments.

Rather than choosing particular high and low input states σ_H and σ_L, the system and the attacker may choose distributions δ_H and δ_L over high and low states, respectively. These distributions are sampled to produce the initial input state. Taking the expectation in $\mathcal{Q}_{\mathsf{E}}$ with respect to σ_H, σ_L and o then yields the expected flow over all experiments.

This extension also increases the expressive power of the experiment model. A distribution over low inputs allows the attacker to use a randomized guessing strategy. His distribution might also be a function of his belief, though we leave investigation of such attacker strategies as future work. A distribution over high inputs 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.

4.7 Maximum Information Flow

System designers are likely to want to limit the maximum possible information flow. We characterize the maximum amount of information flow that program S can cause in a single outcome as the maximum amount of flow from any outcome of any experiment $\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L \rangle$ on S:

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

Consider applying $\mathcal{Q}_{\max}$ to PWC. Assume that b_H is a uniform distribution, representing a lack of belief for any particular password, over k-bit passwords. If the attacker guesses correctly, then according to $\mathcal{Q}_{\max}$, the maximum leakage is k bits. But if the attacker guesses incorrectly, PWC can leak at most $k - \lg(2^k - 1)$ bits in an outcome; for $k > 12$ this is less than 0.0001 bits.

Uncertainty metrics typically declare that the maximum possible information flow is $\lg |\mathbf{State}_H|$; this is the number of bits necessary to store the high state. This was true for the example of k-bit passwords above. However, as experiment $\mathcal{E}_2$ from Section 4.1 shows, this declaration is valid only if the attacker's prebelief is no more inaccurate

Table 5 Repeated experiments on PWC

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
$\mathcal{Q}(\mathcal{E}, b'_H)$		5.6439	1.0

than the uniform distribution. Thus uncertainty metrics make an implicit restriction on attacker beliefs that our accuracy metric does not.

4.8 Repeated Experiments

Nothing precludes performing a series 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 5; 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 $\mathcal{E}_2$ from Section 4.1 is conducted and repeated with $\sigma_L = (g \mapsto B)$. Then the attacker's belief about the password evolves as shown in Table 5. Summing the information flow for each experiment 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 final postbelief in Table 5:

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

This example suggests a 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 b'_{H_i} be the outcome from $\mathcal{E}_i$. Let prebelief b_{H_i} in experiment $\mathcal{E}_i$ be chosen as postbelief $b'_{H_{i-1}}$ from experiment $\mathcal{E}_{i-1}$. Let b_{H_1} be the attacker's prebelief for the entire series. Let n be the length of the series.

Theorem 7

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

Proof. Immediate by the definition of $\mathcal{Q}$ and arithmetic. □

4.9 Number of Experiments

Attackers conduct experiments to refine their beliefs. This suggests another measurement of the security of a program: the number of experiments required for an attacker to refine his belief to within some distance of reality. For simplicity, assume that program S is deterministic[7], and thus that only one observation is possible from an experiment. Then belief revision $\mathcal{B}$ (from Section 3.3) can be used as a function from experiments to postbeliefs. Let $\mathcal{A} : \textbf{Belief} \rightarrow \textbf{State}_L$ be the attacker's *strategy* for choosing low inputs based on his beliefs. Define the i^{th} iteration of $\mathcal{B}$ as $\mathcal{B}^i$:

$$\begin{array}{rcl} \mathcal{B}^i(S, b_H, \sigma_H, \mathcal{A}) & \triangleq & \mathcal{B}(S, b'_H, \sigma_H, \mathcal{A}(b'_H)) \\ & & \text{where } b'_H = \mathcal{B}^{i-1}(S, b_H, \sigma_H, \mathcal{A}) \\ \mathcal{B}^1(S, b_H, \sigma_H, \mathcal{A}) & \triangleq & \mathcal{B}(S, b_H, \sigma_H, \mathcal{A}(b_H)) \end{array}$$

Then the number of experiments $\mathcal{N}$ needed to achieve a postbelief within distance ϵ of reality is:

$$\mathcal{N}(S, b_H, \sigma_H, \mathcal{A}) \triangleq \min\{i \mid D(\mathcal{B}^i(S, b_H, \sigma_H, \mathcal{A}) \twoheadrightarrow \sigma_H) \leq \epsilon\}$$

As discussed in Section 4.2, when an attacker's belief is k bits distant from reality, the probability he ascribes to the correct high state is $1/2^k$. If the attacker were to "guess" a high state by sampling from his belief, he would therefore guess correctly with probability $1/2^\epsilon$ after $\mathcal{N}$ experiments.

If a metric M on $\textbf{State}_H$ is available, then the attacker may approximate reality by establishing a high certainty on a state near to the correct high state. For example, if the high state is a Cartesian coordinate, we may wish to establish the security policy that the attacker should never have a high degree of certainty on the correct coordinate nor any coordinates within some Cartesian distance. Let $ball(\sigma_H)$ be all the high states within distance γ of σ_H:

$$ball(\sigma_H) \triangleq \{\sigma'_H \mid M(\sigma'_H \twoheadrightarrow \sigma_H) \leq \gamma\}$$

Then the number of experiments needed to achieve some distance ϵ from some ball γ around reality is:

$$\begin{array}{rcl} \mathcal{N}(S, b_H, \sigma_H, \mathcal{A}) & \triangleq & \min\{i \mid \sigma'_H \in ball(\sigma_H) \\ & & \qquad \wedge\, D(\mathcal{B}^i(S, b_H, \sigma_H, \mathcal{A}) \twoheadrightarrow \sigma'_H) \leq \epsilon\} \end{array}$$

5 Language Semantics

The last piece required for our framework is a semantics $[\![S]\!]$ in which programs denote functions that map distributions to distributions. Here we build such a semantics in two stages. First, we build a simpler semantics that maps states to distributions. Second, we lift the simpler semantics so that it operates on distributions.

[7] If program S is probabilistic, $\mathcal{B}(\mathcal{E})$ can instead be defined as a random variable giving the probability with which the attacker holds a postbelief. This allows the definition of the *expected* number of experiments to achieve a distance from reality.

Figure 3 Semantics of programs in states

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

Figure 4 Semantics of programs in distributions

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

Our first task then is to define the semantics $[\![S]\!] : \textbf{State} \to \textbf{Dist}$. That semantics is given in Figure 3. We assume a semantics $[\![E]\!] : \textbf{State} \to \textbf{Val}$ that gives meaning to expressions, and a semantics $[\![B]\!] : \textbf{State} \to \textbf{Bool}$ that gives meaning to Boolean expressions.

The statements **skip** and **if** have essentially the same denotations as in the standard deterministic case. State update $\sigma[v \mapsto V]$, where $V \in \textbf{Val}$, 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:

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

The semantics of **while** and sequential composition $S_1; S_2$ use lifting operator *, which lifts function $d : \textbf{State} \to \textbf{Dist}$ to function $d^* : \textbf{Dist} \to \textbf{Dist}$, as suggested by Section 2.2:

$$
\begin{array}{rcl}
d^* & \triangleq & \lambda\delta \,. \textstyle\sum_\sigma \delta(\sigma) \cdot d(\sigma) \\
 & = & \lambda\delta \,. \, \lambda\sigma \,. \textstyle\sum_{\sigma'} \delta(\sigma') \cdot d(\sigma')(\sigma)
\end{array}
$$

where the equality follows from η-reduction, and $\cdot$ and $+$ are used as pointwise operators:

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

Lifted d^* is thus the expected value (which is a distribution) of d with respect to distribution δ.

To ensure that the fixed point for **while** exists, we must verify that **Dist** is a complete partial order with a bottom element and that $[\![\cdot]\!]$ is continuous. We omit the proof here,

as it is a consequence of a theorem proved by Kozen [23]. But we note that a key step is to strengthen the definition of **Dist** from Section 2.1 to be $\{\delta \mid \delta \in \textbf{State} \to [0,1] \ \wedge \ \|\delta\| \leq 1\}$. This makes distributions correspond to subprobability measures, and it is easy to check that the semantics produces subprobability measures as output. The bottom element is then $\lambda\sigma\,.\,0$, and the ordering relation on distributions is pointwise. Note that the definition of **Belief** from Section 2.4 remains unchanged, since it did not depend on **Dist**. Thus beliefs still correspond to probability measures. Anywhere that the result of the program semantics must be upgraded to a belief (i.e., from a subprobability to a probability), we rely on the technique of Section 3.4 to handle nontermination. The most important occurrence of this is in step 5 of the experiment protocol in Figure 1.

The final program construct is probabilistic choice, $S_1 \ {}_p[\!] \ S_2$, where $0 \leq p \leq 1$. The semantics multiplies the probability of choosing a side S_i with the frequency 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 frequency of its occurrence is the sum of the frequency from either side: $p \cdot (\llbracket S_1 \rrbracket\sigma)(\sigma') + (1-p) \cdot (\llbracket S_2 \rrbracket\sigma)(\sigma')$, which can be simplified to the formula in Figure 3.

To lift the semantics in Figure 3 and define $\llbracket S \rrbracket : \textbf{Dist} \to \textbf{Dist}$, we again employ lifting operator *:

$$\begin{array}{lcl} \llbracket S \rrbracket\delta & \triangleq & \llbracket S \rrbracket^*\delta \\ & = & \lambda\sigma\,.\,\sum_{\sigma'} \ \delta(\sigma') \cdot (\llbracket S \rrbracket\sigma')(\sigma) \end{array}$$

Interpreting this definition, note there are many states σ' in which S could begin execution, and all of them could potentially terminate in state σ. So to compute $(\llbracket S \rrbracket\delta)(\sigma)$, 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 $(\llbracket S \rrbracket\sigma')(\sigma)$.

Applying this definition to the semantics in Figure 3 yields $\llbracket S \rrbracket\delta$, shown in Figure 4. This lifted semantics corresponds directly to a semantics given by Kozen [23], 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:

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

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

6 Insider Choice

The experiment protocol in Section 3 involved two agents, the attacker and the system. Consider a third agent called the *insider*, whose goal is to help the attacker learn secret information. The insider and attacker may initially communicate to establish a strategy to achieve this goal. Once execution begins, the insider cannot directly communicate

with the attacker, but the insider is able to observe and influence the execution of programs. He can observe both the high and low components of the program state. His ability to influence execution is modeled by a new programming language construct, *insider choice*, denoted $S_1 \ [] \ S_2$:

$$S \quad ::= \quad \ldots \mid S_1 \ [] \ S_2$$

The insider, rather than the system, is the entity who executes this kind of choice. The insider chooses either S_1 or S_2 and execution continues with the chosen program.

As an example of insider choice, consider program $L1$:

$$\begin{array}{ll} L1: & h := h \bmod 2; \\ & l := 0 \ [] \ l := 1 \end{array}$$

The second line of $L1$ allows the insider to choose between two values for variable l. Since the insider is allowed to observe the high component of the state, he can observe the parity of h and choose to set l equal to it, thus leaking the parity of h.

The insider in this example made a deterministic choice. More generally, insiders may also make probabilistic choices. For example, an insider could flip a fair coin then choose the left side on heads or the right side on tails. This can be seen as an extension of probabilistic choice, in which the probability is a function of the program state rather than just a constant. Thus insider choice can also represent the behavior of normal programs without the influence of an insider.

6.1 Insider Functions

Formally, an insider is a function $I \in \textbf{Insider}$ where:

$$\textbf{Insider} \quad \triangleq \quad \textbf{State} \to [0..1]$$

The value of $I(\sigma)$ is the probability with which the left-hand side of the insider choice is taken. For example, insider function I_{L1} leaks the value of h in program $L1$ with probability 0.99:

$$I_{L1}(\sigma) \quad = \quad \text{if } \sigma(h) = 0 \text{ then } 0.99 \text{ else } 0.01$$

In a program with multiple syntactic occurrences of insider choice, a single insider function can encode different probabilities for each such occurrence, assuming that the program state encodes the program counter.

Insider functions are able to model a range of powers that an insider might have, if those powers can be encoded into the program state. For example, suppose that the operational semantics of the language guarantees that for every variable x, the previous value of x (that is, the value that was assigned to it before its current value was assigned) is preserved in a variable $\grave{x}$. Then insider functions can make decisions based on past state by reading previous values. This is similar to the idea of *history variables* [1]. In the following program, the insider leaks the initial parity of h.

$$\begin{array}{ll} LP: & h := h \bmod 2; \\ & h := 0; \\ & l := 0 \ [] \ l := 1 \end{array}$$

Figure 5 Insider semantics

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

Figure 6 Insider semantics for distributions

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

The insider function that accomplishes this is:

$$I_{LP}(\sigma) \quad = \quad \text{if } \sigma(\grave{h}) = 0 \text{ then } 1 \text{ else } 0$$

Note that without access to history variable $\grave{h}$, the insider is unable to leak the initial parity of h because this information is removed from the state when h is assigned the value 0. Similarly, insiders who can predict the values of variables in the future are modeled by the addition of *prophecy variables* [1].

Insiders with limited computational resources can be modeled by further restricting **Insider**. For example, suppose that insiders are allowed only polynomial time to make a choice. Then insider functions could be replaced by polynomially time-bounded Turing machines, where the input to the machine is the input σ to the insider function, and the output of the machine is used as the output of the insider function.

6.2 Semantics and Experiments

Formal semantics $[\![S]\!]$: **Insider** $\to$ **State** $\to$ **Dist** is given in Figure 5. The only place in the semantics that the insider function is used is in the semantics of $S_1 \ [\!]\ S_2$, and the semantics never modifies the insider function. Due to this second-class nature of insider functions, and for improved readability, we use a subscript notation for the insider

Figure 7 Experiment protocol with insider

An experiment $\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L, I \rangle$ 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 attacker predicts the output distribution: $\delta'_A = [\![S]\!]_I(\dot{\sigma}_L \otimes b_H)$.
4. The system and insider execute the program S, which produces a state $\sigma' \in \delta'$ as output, where $\delta' = [\![S]\!]_I(\dot{\sigma}_L \otimes \dot{\sigma}_H)$. The attacker observes the low projection of the output state: $o = \sigma' \upharpoonright L$.
5. The attacker infers a postbelief: $b'_H = (\delta'_A | o) \upharpoonright H$.

function I in semantics $[\![S]\!]_I$. We can lift the semantics to operate on distributions as shown in Figure 6. The lifted insider function is defined as:

$$\begin{array}{rcl} I(\delta) & \triangleq & \lambda\sigma \,.\, I(\sigma) \cdot \delta(\sigma) \\ \overline{I}(\delta) & \triangleq & \lambda\sigma \,.\, (1 - I(\sigma)) \cdot \delta(\sigma) \end{array}$$

The experiment protocol in Section 3.1 can be extended to include insiders, as shown in Figure 7. Note that the attacker uses insider function I when conducting the thought-experiment. This function thus encodes choices that the insider and attacker have agreed upon in advance.

6.3 Security Conditions

Observational determinism [30, 33, 41] is a security condition for nondeterministic systems that generalizes noninterference [13]. We can state a probabilistic generalization of observational determinism that is applicable to our insider model. Let the set of programs satisfying observational determinism be denoted **ObsDet**. A program S satisfies observational determinism exactly when S behaves as a function from a low input state to a low output distribution, with respect to any insider and high input:

$$\textbf{ObsDet} \quad \triangleq \quad \{S \mid \forall I \,.\, \forall \sigma_L \,.\, \exists \delta_L \,.\, \forall \sigma_H \,.\, [\![S]\!]_I(\dot{\sigma}_L \otimes \dot{\sigma}_H) \upharpoonright L = \delta_L\}$$

Observational determinism is equivalent to zero information flow in the insider model. That is, a program S satisfies observational determinism exactly when all experiments over S leak exactly 0 bits of information.

Theorem 8

$$S \in \textbf{ObsDet} \quad \equiv \quad \forall \mathcal{E}, b'_H \in \mathcal{B}(\mathcal{E}) \,.\, \mathcal{Q}(\mathcal{E}, b'_H) = 0$$

Proof. In Appendix B. □

This theorem provides evidence that observational determinism is the right absolute security condition for nondeterministic systems. (On the other hand, this theorem also shows that observational determinism is too strong to be applicable to programs that require information flow, such as PWC.)

Other nondeterministic security conditions, such as generalized noninterference (GNI) [26], are already known to allow leakage of information [36]. Our model of insider choice allows this leakage to be quantified, which further demonstrates the weakness of such security conditions. A program S satisfies GNI when S behaves as a relation on a low input state and low output distributions, with respect to any insider and high input:

$$\textbf{GNI} \quad \triangleq \quad \{S \mid \forall \sigma_L \,.\, \exists \Delta_L \,.\, \forall \sigma_H \,.\, \bigcup_I ([\![S]\!]_I(\dot{\sigma}_L \otimes \dot{\sigma}_H) \upharpoonright L) = \Delta_L\}$$

Consider program LH, which can be shown to be in **GNI**:

$$LH: \quad l := h \;\; [\!] \;\; (l := 0 \;\; [\!] \;\; l := 1)$$

Using insider function $I_{LH}(\sigma) = 1$, this program always leaks the value of h. Unless the attacker already has a perfectly accurate belief about h, this is a positive (and nonzero) amount of leakage. So even though the program is secure according to **GNI**, an insider can refine the program to be insecure. This weakness is known as the *refinement paradox* [33]. Insiders therefore introduce a kind of nondeterminism that is not secure under refinement.

7 Related Work

We believe the work reported herein is the first to address attacker beliefs in quantifying information flow. Perhaps the earliest published connection between information theory and information flow is Denning [8], which demonstrates the analysis of a few particular assignment and **if** statements by using entropy to calculate leakage. Millen [31], 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 he does not show how to compute the amount of flow for programs.

Wittbold and Johnson [40] introduce *nondeducibility on strategies*, an extension of Sutherland's *nondeducibility* [34]. 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. Our insider framework can quantify the leakage due to strategies that are encodable as insider functions.

The flow model (FM) is a security property first given by McLean [29] and later given a quantitative formalization by Gray [14], who called it the Applied Flow Model.

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 [2] 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.

Volpano [35] 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 type systems. However, the type system does not allow a general analysis of quantitative information flow. Volpano and Smith [37] give another type system that enforces *relative secrecy*, which requires that well-typed programs cannot leak confidential data in polynomial time.

Weber [38] defines 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 [24] 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.

Halpern and Tuttle [19] introduce a framework for reasoning about knowledge and probability based on three kinds of adversaries: adversaries who make nondeterministic choices, adversaries who represent the knowledge of the opponent, and adversaries who control timing. Our insiders can be seen as an instantiation of this framework. The insider choice and insider function constitute an adversary who makes nondeterministic choices, and each of the models of the insider's power in Section 6.1 correspond to an adversary representing the knowledge of the opponent. Gray and Syverson [15] apply the Halpern-Tuttle framework to reason about qualitative security of probabilistic systems. They relate their security condition to probabilistic noninterference [14] and information theory. Halpern and O'Neill [16] construct a framework for reasoning about secrecy that generalizes many previous results on qualitative and probabilistic, but not quantitative, security. Their framework, like ours, uses subjective probability distributions.

Di Pierro, Hankin, and Wiklicky [9] 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 memory. 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. Their more recent work [10] generalizes this to measuring approximate confinement in probabilistic transition systems.

Clark, Hunt, and Malacaria [4] 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. In other work [3], 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. As noted in Section 4.4, this uncertainty-based definition requires a strong admissibility restriction: the attacker's prebelief must be the same distribution from which the system generates the high input. Malacaria [25] extends this line of work by classifying the rate of leakage of loops. His basic definition of amount of leakage is equivalent to [4], so it is an instance of our own definition, as shown in Section 4.4. For the same reason, Malacaria's model is no more precise than our own model [6]. Rate of leakage could be defined in our own model, using Malacaria's techniques, much like the other parameters of leakage given in Section 4.

McIver and Morgan [27] 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.

Evfimievski, Gehrke, and Srikant [11] quantify *privacy breaches* in data mining. In their framework, randomized operators are applied to confidential data before the data is released. A privacy breach occurs when release of the randomized data causes a large change in an attacker's probability distribution on a property of the confidential data. They use Bayesian reasoning, based on observation of randomized data, to update the attacker's distribution. Their distributions are similar to our beliefs, but have the same strong admissibility restriction as Clark et al. [4]. They also show that relative entropy can be used to bound the maximum privacy breach for a randomized operator.

8 Conclusion

This paper presents a model for incorporating attacker beliefs into analysis of quantitative information flow. Our theory reveals that uncertainty, the traditional metric for information flow, is inadequate: it cannot satisfactorily explain even the simple example of password checking. 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, in the presence of beliefs, accuracy is the correct metric for information flow.

We have shown how to use an accuracy metric to calculate exact, expected, and maximum information flow; other parameters of information flow, such as variance, median, and rate, could be defined in the same way. We have demonstrated that our metric generalizes uncertainty metrics. Our formal model of experiments enables precise, compositional reasoning about attackers' actions and beliefs. We have instantiated this model with a probabilistic semantics and have shown that probabilistic choice is essential to producing misinformation. We have also extended the model to enable analysis of information flow due to insiders, who collude with attackers.

Acknowledgments

Sigmund Cherem, Stephen Chong, Sebastian Hunt, Jed Liu, Carroll Morgan, Michael Mislove, Kevin O'Neill, Nathaniel Nystrom, Riccardo Pucella, Lantian Zheng, and the anonymous referees for *JCS* and *CSFW'05* provided helpful comments on this work.

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 F9550-06-0019; National Science Foundation grants 0208642, 0133302, 0430161; AF-TRUST (Air Force Team for Research in Ubiquitous Secure Technology for GIG/NCES), which receives support from the DAF Air Force Office of Scientific Research (FA9550-06-1-0244), National Science Foundation (CCF-0424422), Cisco, British Telecom, ESCHER, HP, IBM, iCAST, Intel, Microsoft, ORNL, Pirelli, Qualcomm, Sun, Symantec, Telecom Italia, and United Technologies; and a grant from Intel Corporation. Michael Clarkson was supported by a National Science Foundation Graduate Research Fellowship; Andrew Myers was supported by an Alfred P. Sloan Research Fellowship. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of these organizations or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

References

[1] Martín Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, 1991.

[2] Randy Browne. The Turing test and non-information flow. In *Proc. IEEE Symp. on Security and Privacy*, pages 375–385, Oakland, CA, 1991.

[3] David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantified interference: Information theory and information flow. Presented at *Workshop on Issues in the Theory of Security*, April 2004.

[4] David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantified interference for a while language. *Electronic Notes in Theoretical Computer Science*, 112:149–166, January 2005.

[5] David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantitative information flow, relations and polymorphic types. *Journal of Logic and Computation*, 18(2):181–199, 2005.

[6] Michael R. Clarkson, Andrew C. Myers, and Fred B. Schneider. Belief in information flow. In *Proc. 18th IEEE Computer Security Foundations Workshop*, pages 31–45, Aix-en-Provence, France, June 2005.

[7] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley & Sons, 1991.

[8] Dorothy Denning. *Cryptography and Data Security*. Addison-Wesley, 1982.

[9] Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky. Approximate non-interference. *Journal of Computer Security*, 12(1):37–81, 2004.

[10] Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky. Measuring the confinement of probabilistic systems. *Theoretical Computer Science*, 340(1):3–56, 2005.

[11] Alexandre Evfimievski, Johannes Gehrke, and Ramakrishnan Srikant. Limiting privacy breaches in privacy preserving data mining. In *Proc. ACM Symp. on Principles of Database Systems*, pages 211–222, San Diego, CA, 2003.

[12] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis*. Chapman and Hall/CRC, 2004.

[13] Joseph A. Goguen and Jose Meseguer. Security policies and security models. In *Proc. IEEE Symp. on Security and Privacy*, pages 11–20, April 1982.

[14] James W. Gray, III. Toward a mathematical foundation for information flow security. In *Proc. IEEE Symp. on Security and Privacy*, pages 21–35, Oakland, CA, 1991.

[15] James W. Gray, III and Paul F. Syverson. A logical approach to multilevel security of probabilistic systems. *Distributed Computing*, 11(2):73–90, 1998.

[16] Joseph Halpern and Kevin O'Neill. Secrecy in multiagent systems. In *Proc. 15th IEEE Computer Security Foundations Workshop*, pages 32–46, Cape Breton, Nova Scotia, Canada, 2002.

[17] Joseph Halpern and Kevin O'Neill. Anonymity and information hiding in multiagent systems. In *Proc. 16th IEEE Computer Security Foundations Workshop*, pages 75–88, Pacific Grove, CA, 2003.

[18] Joseph Y. Halpern. *Reasoning about Uncertainty*. MIT Press, Cambridge, Massachusetts, 2003.

[19] Joseph Y. Halpern and Mark R. Tuttle. Knowledge, probability, and adversaries. *Journal of the ACM*, 40(4):917–962, 1993.

[20] Gareth A. Jones and J. Mary Jones. *Information and Coding Theory*. Springer, 2000.

[21] Daniel Kahneman and Amos Tversky. Subjective probability: A judgment of representativeness. *Cognitive Psychology*, 3:430–454, 1972.

[22] Johnathan J. Koehler. The base rate fallacy reconsidered: Descriptive, normative, and methodological challenges. *Behavioral and Brain Sciences*, 19(1):1–53, 1996.

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

[24] Gavin Lowe. Quantifying information flow. In *Proc. 15th IEEE Computer Security Foundations Workshop*, pages 18–31, Cape Breton, Nova Scotia, Canada, 2002.

[25] Pasquale Malacaria. Assessing security threats of looping constructs. In *Proc. 34th ACM Symp. on Principles of Programming Languages*, pages 225–235, Nice, France, January 2007.

[26] Daryl McCullough. Specifications for multi-level security and a hook-up property. In *Proc. IEEE Symp. on Security and Privacy*, Oakland, CA, 1987.

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

[28] Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer, 2004.

[29] John McLean. Security models and information flow. In *Proc. IEEE Symp. on Security and Privacy*, pages 180–189, Oakland, CA, 1990.

[30] John McLean. Proving noninterference and functional correctness using traces. *Journal of Computer Security*, 1(1), 1992.

[31] Jonathan Millen. Covert channel capacity. In *Proc. IEEE Symp. on Security and Privacy*, pages 60–66, Oakland, CA, 1987.

[32] Lyle Harold Ramshaw. *Formalizing the Analysis of Algorithms*. PhD thesis, Stanford University, 1979. Available as technical report, XEROX PARC, 1981.

[33] A. W. Roscoe. CSP and determinism in security modelling. In *Proc. IEEE Symp. on Security and Privacy*, pages 114–127, Oakland, CA, 1995.

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

[35] Dennis Volpano. Secure introduction of one-way functions. In *Proc. 13th IEEE Computer Security Foundations Workshop*, pages 246–254, Cambridge, UK, 2000.

[36] Dennis Volpano and Geoffrey Smith. Confinement properties for programming languages. *SIGACT News*, 29(3):33–42, September 1998.

[37] Dennis Volpano and Geoffrey Smith. Verifying secrets and relative secrecy. In *Proc. 27th ACM Symp. on Principles of Programming Languages*, pages 268–276, Boston, MA, 2000.

[38] Douglas G. Weber. Quantitative hook-up security for covert channel analysis. In *Proc. First IEEE Computer Security Foundations Workshop*, pages 58–71, Franconia, NH, 1988.

[39] Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction*. MIT Press, Cambridge, Massachusetts, 1993.

[40] J. Todd Wittbold and Dale Johnson. Information flow in nondeterministic systems. In *Proc. IEEE Symp. on Security and Privacy*, pages 144–161, Oakland, CA, 1990.

[41] Steve Zdancewic and Andrew C. Myers. Observational determinism for concurrent program security. In *Proc. 16th IEEE Computer Security Foundations Workshop*, pages 29–43, Pacific Grove, CA, 2003.

A Relaxing Restrictions on Programs

Mutable high inputs. To allow mutable high inputs, as discussed in Section 3.4, let the notation b_H^0 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_H^0 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 $\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L \rangle$:

$$\mathcal{B}^!(\mathcal{E}, o) \quad \triangleq \quad ((\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H \otimes b_H^0)|o)) \upharpoonright H^0$$

In this definition, the high input state is preserved by introducing the product with b_H^0, and the attacker's postbelief about the input is recovered by restricting to H^0, the high input state with the superscript 0.

Nontermination. To allow nonterminating programs, let $\mathbf{State}_\perp \triangleq \mathbf{State} \cup \{\perp\}$, and $\perp \upharpoonright L \triangleq \perp$. Nontermination is now allowed as an observation, leading to an extended belief revision operator $\mathcal{B}^{!\perp}$:

$$\mathcal{B}^{!\perp}(\mathcal{E}, o) \quad \triangleq \quad (out_\perp(S, \dot{\sigma}_L \otimes b_H \otimes b_H^0)|o) \upharpoonright H^0$$

Observation o is now produced from output distribution $\delta' = out_\perp(S, \dot{\sigma}_L \otimes \dot{\sigma}_H)$. Function $out_\perp(S, \delta)$ produces a distribution which yields the frequency that S terminates, or fails to terminate, on input distribution δ:

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

If S does not terminate on some input states in δ, then output distribution $[\![S]\!]\delta$ will contain less mass than δ; otherwise, $\|\delta\| = \|[\![S]\!]\delta\|$. Missing mass corresponds to nontermination [32, 28], so $out_\perp$ maps the missing mass to $\perp$.

B Proofs

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

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

Proof.

$$
\begin{array}{ll}
 & BI(\mathcal{E}, o) \\
= & \quad \langle \text{ Definition of } BI \ \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)
\end{array}
$$

$=$ $\langle$ Definition of $\mathcal{B}(\mathcal{E}, o)$ $\rangle$

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

□

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

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

Proof.

$[\![S]\!](\dot{\sigma}_L \otimes b_H)(\sigma)$

$=$ $\langle$ Definition of $[\![S]\!]\delta$ $\rangle$

$\sum_{\sigma'}\; (\dot{\sigma}_L \otimes b_H)(\sigma') \cdot ([\![S]\!]\sigma')(\sigma)$

$=$ $\langle$ Definition of point mass $\rangle$

$\sum_{\sigma' \mid \sigma' \restriction L = \sigma_L}\; b_H(\sigma' \restriction H) \cdot ([\![S]\!]\sigma')(\sigma)$

$=$ $\langle$ Let $\sigma = \sigma_L \cup \sigma_H$, nesting, one-point rule $\rangle$

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

□

Lemma 2.1

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

Proof.

$b'_H(\sigma_H)$

$=$ $\langle$ Definition of b'_H in experiment protocol $\rangle$

$(([\![S]\!](\dot{\sigma}_L \otimes b_H)|o) \restriction H)(\sigma_H)$

$=$ $\langle$ Definition of $\delta \restriction H$ $\rangle$

$\sum_{\sigma \mid \sigma \restriction H = \sigma_H}\; ([\![S]\!](\dot{\sigma}_L \otimes b_H)|o)(\sigma)$

$=$ $\langle$ Definition of $\delta|o$ $\rangle$

$$
\begin{array}{ll}
 & \sum_{\sigma \mid \sigma \restriction H = \sigma_H \,\wedge\, \sigma \restriction L = o} \frac{[\![S]\!](\dot\sigma_L \otimes b_H)(\sigma)}{([\![S]\!](\dot\sigma_L \otimes b_H) \restriction L)(o)} \\
= & \quad \langle \text{ One-point rule: } \sigma = o \cup \sigma_H \rangle \\
 & \frac{[\![S]\!](\dot\sigma_L \otimes b_H)(o \cup \sigma_H)}{([\![S]\!](\dot\sigma_L \otimes b_H) \restriction L)(o)} \\
= & \quad \langle \text{ Definition of } \delta_A \rangle \\
 & \frac{1}{\delta_A(o)} \cdot [\![S]\!](\dot\sigma_L \otimes b_H)(o \cup \sigma_H) \\
= & \quad \langle \text{ Definition of } [\![S]\!]\delta \rangle \\
 & \frac{1}{\delta_A(o)} \cdot \sum_{\sigma'} (\dot\sigma_L \otimes b_H)(\sigma') \cdot ([\![S]\!]\sigma')(o \cup \sigma_H) \\
= & \quad \langle \text{ Definition of } \otimes, \text{ point mass } \rangle \\
 & \frac{1}{\delta_A(o)} \cdot \sum_{\sigma' \mid \sigma' \restriction L = \sigma_L} b_H(\sigma' \restriction H) \cdot ([\![S]\!](\dot\sigma_L \otimes (\dot\sigma' \restriction H)))(o \cup \sigma_H) \\
= & \quad \langle \text{ High input is immutable } \rangle \\
 & \frac{1}{\delta_A(o)} \cdot \sum_{\sigma' \mid \sigma' \restriction L = \sigma_L \,\wedge\, \sigma' \restriction H = \sigma_H} b_H(\sigma' \restriction H) \\
 & \quad \cdot ([\![S]\!](\dot\sigma_L \otimes (\dot\sigma' \restriction H)))(o \cup \sigma_H) \\
= & \quad \langle \text{ One-point rule: } \sigma' = \sigma_L \cup \sigma_H \rangle \\
 & \frac{1}{\delta_A(o)} \cdot b_H(\sigma_H) \cdot ([\![S]\!](\dot\sigma_L \otimes \dot\sigma'_H))(o \cup \sigma_H) \\
= & \quad \langle \text{ High input is immutable, Definition of } \delta \restriction L \rangle \\
 & \frac{1}{\delta_A(o)} \cdot b_H(\sigma_H) \cdot (([\![S]\!](\dot\sigma_L \otimes \dot\sigma'_H)) \restriction L)(o) \\
= & \quad \langle \text{ Definition of } \delta_S \rangle \\
 & b_H(\sigma_H) \cdot \frac{\delta_S(o)}{\delta_A(o)}
\end{array}
$$

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

Theorem 3 Let $\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L \rangle$. Then:

$$\mathcal{Q}(\mathcal{E}, b'_H) = k \quad \equiv \quad b'_H(\sigma_H) = 2^k \cdot b_H(\sigma_H)$$

Proof.

$$
\begin{array}{ll}
 & \mathcal{Q}(\mathcal{E}, b'_H) = k \\
\equiv & \quad \langle \text{ Definition of } \mathcal{Q} \rangle
\end{array}
$$

$$
\begin{array}{ll}
& D(b_H \twoheadrightarrow \dot{\sigma}_H) - D(b'_H \twoheadrightarrow \dot{\sigma}_H) = k \\
\equiv & \quad \langle \text{ Definition of } D \ \rangle \\
& -(\lg b_h(\sigma_H) - \lg b'_H(\sigma_H)) = k \\
\equiv & \quad \langle \text{ Arithmetic, properties of } \log \ \rangle \\
& b'_H(\sigma_H) = 2^k \cdot b_H(\sigma_H)
\end{array}
$$

□

Theorem 4

$$S \in \textbf{Det} \quad \Rightarrow \quad \forall \mathcal{E}, b'_H \in \mathcal{B}(\mathcal{E}) \,.\, \mathcal{Q}(\mathcal{E}, b'_H) \geq 0$$

Proof. Assume $S \in$ **Det** and let $\mathcal{E}$, b'_H be arbitrary.

$$
\begin{array}{ll}
& \mathcal{Q}(\mathcal{E}, b'_H) \geq 0 \\
\equiv & \quad \langle \text{ Definition of } \mathcal{Q}, \text{ arithmetic } \rangle \\
& D(b_H \twoheadrightarrow \dot{\sigma}_H) \geq D(b'_H \twoheadrightarrow \dot{\sigma}_H) \\
\equiv & \quad \langle \text{ Definition of } D, \text{ arithmetic } \rangle \\
& \lg b(\sigma_H) \leq \lg b'(\sigma_H) \\
\equiv & \quad \langle \text{ Lemma 4.1, } \lg \text{ is monotonic on } (0, 1], \text{ admissibility of } b \ \rangle \\
& \mathit{true}
\end{array}
$$

□

Lemma 4.1 Assume $S \in$ **Det** and let $\mathcal{E}$, b'_H be arbitrary. Then:

$$b(\sigma_H) \leq b'(\sigma_H)$$

Proof. Let o be the observation producing b'. It is straightforward to check that if $S \in$ **Det**, then $[\![S]\!]\sigma$ is the point mass at σ', where σ' is the state produced by the standard denotational semantics of **while** programs, such as Winskel's [39]. So the output of $[\![S]\!](\sigma_L \cup \sigma_H)$ is the point mass at $o \cup \sigma_H$.

$$
\begin{array}{ll}
& b'(\sigma_H) \\
= & \quad \langle \text{ Definition of } b' \ \rangle
\end{array}
$$

$$
\begin{aligned}
& (\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H) | o \upharpoonright H)(\sigma_H) \\
= \quad & \langle \text{ Definition of } \upharpoonright H, \text{ application to } \sigma_H, \text{ one-point rule } \rangle \\
& \textstyle\sum_{\sigma'_L} (\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H) | o)(\sigma'_L \cup \sigma_H) \\
= \quad & \langle \text{ Definition of } |, \text{ one-point rule } \rangle \\
& \tfrac{\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H)(o \cup \sigma_H)}{(\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H) \upharpoonright L)(o)} \\
= \quad & \langle \text{ High input is immutable } \rangle \\
& \tfrac{b(\sigma_H) \cdot \llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H)(o \cup \sigma_H)}{(\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H) \upharpoonright L)(o)} \\
= \quad & \langle \text{ Output of } S \text{ is a point mass as argued above, let } x \text{ be the denominator } \rangle \\
& \tfrac{b(\sigma_H) \cdot 1}{x} \\
\geq \quad & \langle \text{ Admissibility of } b \text{ implies } x \in (0, 1], \text{ arithmetic } \rangle \\
& b(\sigma_H)
\end{aligned}
$$

□

Theorem 5

$$
\mathcal{Q}(\mathcal{E}, b') \quad = \quad \mathcal{L}(H_{in}, L_{in}, L_{out})
$$

Proof.

$$
\begin{aligned}
& \mathcal{Q}(\mathcal{E}, b') \\
= \quad & \langle \text{ Definition of } \mathcal{Q} \rangle \\
& D(b \twoheadrightarrow \dot{\sigma}_H) - D(b' \twoheadrightarrow \dot{\sigma}_H) \\
= \quad & \langle \text{ Definition of } D \rangle \\
& \mathcal{H}(b \upharpoonright (L \cup L^0 \cup H^0)) - \mathcal{H}(b \upharpoonright (L \cup L^0)) \\
& - (\mathcal{H}(b' \upharpoonright (L \cup L^0 \cup H^0)) - \mathcal{H}(b' \upharpoonright (L \cup L^0))) \\
= \quad & \langle \text{ Definition of domain of } b \rangle \\
& \mathcal{H}(b \upharpoonright (L^0 \cup H^0)) - \mathcal{H}(b \upharpoonright L^0) \\
& - (\mathcal{H}(b' \upharpoonright (L \cup L^0 \cup H^0)) - \mathcal{H}(b' \upharpoonright (L \cup L^0))) \\
= \quad & \langle \ H_{in} = H^0, L_{in} = L^0, L_{out} = L;
\end{aligned}
$$

admissibility restriction makes b' an output distribution $\rangle$

$\mathcal{H}(H_{in}, L_{in}) - \mathcal{H}(L_{in}) - (\mathcal{H}(H_{in}, L_{in}, L_{out}) - \mathcal{H}(L_{in}, L_{out})$

$=$ $\langle$ Definition of conditional entropy $\rangle$

$\mathcal{H}(H_{in}|L_{in}) - \mathcal{H}(H_{in}|L_{in}, L_{out})$

$=$ $\langle$ Definition of $\mathcal{L}$ $\rangle$

$\mathcal{L}(H_{in}, L_{in}, L_{out})$

□

Theorem 6 Given experiment $\mathcal{E} = \langle S, b_H, \sigma_H, \sigma_L \rangle$, let:

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

Then:

$$\mathcal{Q}_{\mathsf{E}}(\mathcal{E}) \quad \leq \quad \mathcal{Q}(\mathcal{E}, e_H)$$

Proof.

$\mathcal{Q}_{\mathsf{E}}(\mathcal{E})$

$=$ $\langle$ Definition of $\mathcal{Q}_{\mathsf{E}}$ $\rangle$

$\mathsf{E}_{o \in \delta' \upharpoonright L}[\mathcal{Q}(\mathcal{E}, \mathcal{B}(\mathcal{E}, o))]$

$=$ $\langle$ Definition of $\mathcal{Q}$, let $b'_H = \mathcal{B}(\mathcal{E}, o)\rangle)$ $\rangle$

$\mathsf{E}_{o \in \delta' \upharpoonright L}[D(b_H \twoheadrightarrow \dot{\sigma}_H) - D(b'_H \twoheadrightarrow \dot{\sigma}_H)]$

$=$ $\langle$ Linearity of E $\rangle$

$D(b_H \twoheadrightarrow \dot{\sigma}_H) - \mathsf{E}_{o \in \delta' \upharpoonright L}[D(b'_H \twoheadrightarrow \dot{\sigma}_H)]$

$\leq$ $\langle$ Jensen's inequality and convexity of D, see [7] $\rangle$

$D(b_H \twoheadrightarrow \dot{\sigma}_H) - D(\mathsf{E}_{o \in \delta' \upharpoonright L}[b'_H] \twoheadrightarrow \dot{\sigma}_H)$

$=$ $\langle$ Lemma 6.1 $\rangle$

$D(b_H \twoheadrightarrow \dot{\sigma}_H) - D(e_H \twoheadrightarrow \dot{\sigma}_H)$

$=$ $\langle$ Definition of $\mathcal{Q}$ $\rangle$

$\mathcal{Q}(\mathcal{E}, e_H)$

□

Lemma 6.1 Let $\mathcal{E}$, δ', e_H be defined as in Theorem 6. Let $b'_H = \mathcal{B}(\mathcal{E}, o)$, where $o \in \delta' \upharpoonright L$. Then:

$$\mathsf{E}_{o \in \delta' \upharpoonright L}[b'_H] \quad = \quad e_H$$

Proof. (by extensionality)

$\mathsf{E}_{o \in \delta' \upharpoonright L}[b'_H](\sigma_H)$

$=$ $\langle$ Definitions of E, b'_H $\rangle$

$(\sum_o (\delta' \upharpoonright L)(o) \cdot \mathcal{B}(\mathcal{E}, o)(\sigma_H)$

$=$ $\langle$ Definition of $\mathcal{B}(\mathcal{E}, o)$ $\rangle$

$\sum_o (\delta' \upharpoonright L)(o) \cdot ((([\![S]\!](\dot{\sigma}_L \otimes b_H))|o) \upharpoonright H)(\sigma_H)$

$=$ $\langle$ Definition of $\delta \upharpoonright H$, applying distribution to σ_H $\rangle$

$\sum_o (\delta' \upharpoonright L)(o) \quad \cdot (\sum_{\sigma' \mid \sigma' \upharpoonright H = \sigma_H} (([\![S]\!](\dot{\sigma}_L \otimes b_H))|o)(\sigma'))$

$=$ $\langle$ Definition of $\delta|o$, applying distribution to σ' $\rangle$

$\sum_o (\delta' \upharpoonright L)(o) \quad \cdot (\sum_{\sigma' \mid \sigma' \upharpoonright H = \sigma_H \wedge \sigma' \upharpoonright L = o} \frac{([\![S]\!](\dot{\sigma}_L \otimes b_H))(\sigma')}{([\![S]\!](\dot{\sigma}_L \otimes b_H) \upharpoonright L)(o)})$

$=$ $\langle$ One-point rule $\rangle$

$\sum_o (\delta' \upharpoonright L)(o) \cdot \frac{([\![S]\!](\dot{\sigma}_L \otimes b_H))(o \cup \sigma_H)}{([\![S]\!](\dot{\sigma}_L \otimes b_H) \upharpoonright L)(o)}$

$=$ $\langle$ Definition of $\delta \upharpoonright L$, applied to o $\rangle$

$\sum_o (\delta' \upharpoonright L)(o) \cdot \frac{([\![S]\!](\dot{\sigma}_L \otimes b_H))(o \cup \sigma_H)}{\sum_{\sigma' \mid \sigma' \upharpoonright L = o} [\![S]\!](\dot{\sigma}_L \otimes b_H)(\sigma')}$

$=$ $\langle$ Let $\sigma = o \cup \sigma_H$, change of dummy: $o := \sigma$, definition of $\approx_L$ $\rangle$

$\sum_{\sigma \mid \sigma \upharpoonright H = \sigma_H} (\delta' \upharpoonright L)(o) \quad \cdot \frac{([\![S]\!](\dot{\sigma}_L \otimes b_H))(\sigma)}{\sum_{\sigma' \mid \sigma' \approx_L \sigma} [\![S]\!](\dot{\sigma}_L \otimes b_H)(\sigma')}$

$=$ $\langle$ Definition of $\delta|\delta_L$, applied to σ $\rangle$

$\sum_{\sigma \mid \sigma \upharpoonright H = \sigma_H} ([\![S]\!](\dot{\sigma}_L \otimes b_H)|(\delta' \upharpoonright L))(\sigma)$

$=$ $\langle$ Definition of $\delta \upharpoonright H$, applied to σ_H $\rangle$

$$
\begin{aligned}
& ((\llbracket S \rrbracket(\dot{\sigma}_L \otimes b_H)|(\delta' \restriction L)) \restriction H)(\sigma_H) \\
= \quad & \quad \langle \text{ Definition of } e_H \rangle \\
& e_H(\sigma_H)
\end{aligned}
$$

□

Theorem 8

$$S \in \textbf{ObsDet} \quad \equiv \quad \forall \mathcal{E}, b'_H \in \mathcal{B}(\mathcal{E}) \,.\, \mathcal{Q}(\mathcal{E}, b'_H) = 0$$

Proof. By mutual implication.

(⇒) Assume $S \in$ **ObsDet**. Let $\mathcal{E} = \langle S, \sigma_L, \sigma_H, b_H, I \rangle$ and $b'_H \in \mathcal{B}(\mathcal{E})$ be arbitrary.

$$
\begin{aligned}
& \mathcal{Q}(\mathcal{E}, b'_H) = 0 \\
\equiv \quad & \quad \langle \text{ Definition of } \mathcal{Q}, \text{ arithmetic } \rangle \\
& D(b_H \twoheadrightarrow \sigma_H) = D(b'_H \twoheadrightarrow \sigma_H) \\
\equiv \quad & \quad \langle \text{ Definition of } D, \text{ arithmetic } \rangle \\
& b_H(\sigma_H) = b'_H(\sigma_H) \\
\equiv \quad & \quad \langle \text{ Lemma 8.2 } \rangle \\
& \mathit{true}
\end{aligned}
$$

This concludes the forward direction (⇒) of the proof.

(⇐) By contrapositive. Assume $S \notin$ **ObsDet**. We need to show:

$$\exists \mathcal{E} = \langle S, \sigma_L, \sigma_H, b_H, I \rangle, b'_H \in \mathcal{B}(\mathcal{E}) \,.\, \mathcal{Q}(\mathcal{E}, b'_H \neq 0$$

We calculate:

$$
\begin{aligned}
& S \notin \textbf{ObsDet} \\
\equiv \quad & \quad \langle \text{ Definition of } \textbf{ObsDet} \rangle \\
& \neg \forall I, \sigma_L \exists \delta_L \forall \sigma_H \,.\, \llbracket S \rrbracket_I(\dot{\sigma}_L \otimes \dot{\sigma}_H) \restriction L = \delta_L \\
\equiv \quad & \quad \langle \text{ Predicate calculus, change of dummy } \rangle \\
& \exists \tilde{I}, \tilde{\sigma}_L \forall \tilde{\delta}_L \exists \tilde{\sigma}_H \,.\, \llbracket S \rrbracket_{\tilde{I}}(\dot{\tilde{\sigma}}_L \otimes \dot{\tilde{\sigma}}_H) \restriction L \neq \tilde{\delta}_L \qquad (8.1)
\end{aligned}
$$

Make the following definitions:

$$\begin{array}{rcl} I & = & \tilde{I} \\ \sigma_L & = & \tilde{\sigma}_L \\ \sigma'_H & = & \text{arbitrary} \\ \delta' & = & [\![S]\!]_I(\dot{\sigma}_L \otimes \dot{\sigma'}_H) \\ \delta'_L & = & \delta' \upharpoonright L \\ \sigma_H & = & \text{the } \tilde{\sigma}_H \text{ guaranteed by formula (8.1) when } \tilde{\delta}_L = \delta'_L \\ \delta & = & [\![S]\!]_I(\dot{\sigma}_L \otimes \dot{\sigma}_H) \\ \delta_L & = & \delta \upharpoonright L \end{array}$$

And let b_H be the belief mapping σ_H to $1/2$ and σ'_H to $1/2$.

We have now defined all the variables in experiment $\mathcal{E}$, but we need to define $b'_H \in \mathcal{B}(\mathcal{E})$. To that end, we calculate the attacker's prediction δ_A:

$$\begin{array}{rl} & \delta_A \\ = & \quad \langle \text{ Definition of prediction } \rangle \\ & [\![S]\!]_I(\dot{\sigma}_L \otimes b_H) \\ = & \quad \langle \text{ Definition of } [\![S]\!]\delta \rangle \\ & 1/2 \cdot [\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}_H) + 1/2 \cdot [\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}'_H) \\ = & \quad \langle \text{ Definition of } \delta,\delta' \rangle \\ & 1/2 \cdot (\delta + \delta') \end{array}$$

To define b'_H, we also need an observation o. Note that, by formula (8.1), $\delta_L \neq \delta'_L$, so there is some low state σ'_L such that $\delta_L(\sigma'_L) \neq \delta'_L(\sigma'_L)$. Assume, without loss of generality, that $\delta_L(\sigma'_L) > \delta'_L(\sigma'_L)$. Let o be σ'_L. But in order for o to be an observation, it must be that $o \in [\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}_H)$, which implies that $[\![S]\!](\dot{\sigma}_L \otimes \dot{\sigma}_H)(o) > 0$. This is guaranteed by the fact that $\delta_L(o) > \delta'_L(o)$, and that $\delta'_L(o) \geq 0$.

We can now calculate b'_H:

$$\begin{array}{rl} & b'_H \\ = & \quad \langle \text{ Definition of } b'_H \text{ experiment protocol } \rangle \\ & \delta_A|o \upharpoonright H \\ = & \quad \langle \text{ Definition of } \delta_A \rangle \\ & 1/2 \cdot (\delta + \delta')|o \upharpoonright H \end{array}$$

With all these definitions, we can prove the desired result:

$$\mathcal{Q}(\mathcal{E}, b'_H) \neq 0$$

$\equiv$ $\langle$ Definition of $\mathcal{Q}$, arithmetic $\rangle$

$D(b_H \rightarrowtriangle \sigma_H) \neq D(b'_H \rightarrowtriangle \sigma_H)$

$\equiv$ $\langle$ Definition of D, arithmetic $\rangle$

$b_H(\sigma_H) \neq b'_H(\sigma_H)$

$\equiv$ $\langle$ Lemma 8.4 $\rangle$

$true$

□

Lemma 8.1

$S \in \textbf{ObsDet} \quad \Rightarrow \quad \forall I \,.\, \forall \sigma_L \,.\, \exists \delta_L \,.\, \forall \delta_H \,.\, \|\delta_H\| = 1 \;\Rightarrow\; [\![S]\!]_I(\dot{\sigma}_L \otimes \dot{\sigma}_H) \upharpoonright L = \delta_L$

Proof. Assume $S \in$ **ObsDet**. Let I, σ_L be arbitrary. Let δ_L be the distribution guaranteed to exist by the definition of **ObsDet**. Let δ_H be arbitrary with $\|\delta_H\| = 1$.

$[\![S]\!]_I(\dot{\sigma}_L \otimes \dot{\sigma}_H) \upharpoonright L$

$=$ $\langle$ Definition of $[\![S]\!]\delta$ $\rangle$

$(\sum_{\sigma_H} \delta_H(\sigma_H) \cdot [\![S]\!]_I(\sigma_L \cup \sigma_H)) \upharpoonright L$

$=$ $\langle$ $\upharpoonright L$ distributes over $+, \cdot$ $\rangle$

$\sum_{\sigma_H} \delta_H(\sigma_H) \cdot [\![S]\!]_I(\sigma_L \cup \sigma_H) \upharpoonright L$

$=$ $\langle$ $S \in$ **ObsDet**, definition of δ_L $\rangle$

$\sum_{\sigma_H} \delta_H(\sigma_H) \cdot \delta_L$

$=$ $\langle$ Distributivity, definition of $\|\delta\|$ $\rangle$

$\delta_L \cdot \|\delta_H\|$

$=$ $\langle$ Assumed $\|\delta_H\| = 1$ $\rangle$

δ_L

□

Lemma 8.2 Assume $S \in$ **ObsDet**. Let $\mathcal{E} = \langle S, \sigma_L, \sigma_H, b_H, I \rangle$ and $b'_H \in \mathcal{B}(\mathcal{E})$ be arbitrary. Then:

$$b_H \quad = \quad b'_H$$

Proof. Let $\delta_A = [\![S]\!]_I(\dot{\sigma}_L \otimes b_H)$. Let $o \in [\![S]\!]_I(\dot{\sigma}_L \otimes \dot{\sigma}_H) \upharpoonright L$.

$$
\begin{array}{ll}
 & b'_H \\
= & \quad \langle \text{ Definition of } b'_H \text{ in experiment protocol } \rangle \\
 & (\delta_A|o) \upharpoonright H \\
= & \quad \langle \text{ Definition of } \upharpoonright H \ \rangle \\
 & \lambda\sigma_H \,.\, \sum_{\sigma' \mid \sigma' \upharpoonright H = \sigma_H} (\delta_A|o)(\sigma') \\
= & \quad \langle \text{ Definition of } \delta|o \ \rangle \\
 & \lambda\sigma_H \,.\, \sum_{\sigma' \mid \sigma' \upharpoonright H = \sigma_H} \text{ if } (\sigma' \upharpoonright L) = o \text{ then } \frac{\delta_A(\sigma')}{(\delta_A \upharpoonright L)(o)} \text{ else } 0 \\
= & \quad \langle \text{ Lemma 8.1 } \rangle \\
 & \lambda\sigma_H \,.\, \sum_{\sigma' \mid \sigma' \upharpoonright H = \sigma_H} \text{ if } (\sigma' \upharpoonright L) = o \text{ then } \frac{\delta_A(\sigma')}{\delta_L(o)} \text{ else } 0 \\
= & \quad \langle \text{ One-point rule } \rangle \\
 & \lambda\sigma_H \,.\, \frac{\delta_A(o \cup \sigma_H)}{\delta_L(o)} \\
= & \quad \langle \text{ Lemma 8.3 } \rangle \\
 & \lambda\sigma_H \,.\, \frac{b_H(\sigma_H) \cdot \delta_L(o)}{\delta_L(o)} \\
= & \quad \langle \text{ Arithmetic, } \eta\text{-reduction } \rangle \\
 & b_H
\end{array}
$$

□

Lemma 8.3 Assume the definitions in Lemma 8.2 and its proof. Then:

$$\delta_A(o \cup \sigma_H) \quad = \quad b_H(\sigma_H) \cdot \delta_L(o)$$

Proof.

$$
\begin{array}{ll}
 & \delta_A(o \cup \sigma_H) \\
= & \quad \langle \text{ Definition of } \delta_A \ \rangle \\
 & [\![S]\!]_I(\dot{\sigma}_L \otimes b_H)(o \cup \sigma_H) \\
= & \quad \langle \text{ Definition of } [\![S]\!]\delta \ \rangle \\
 & \sum_{\sigma'} (\dot{\sigma}_L \otimes b_H)(\sigma') \cdot ([\![S]\!]_I \sigma')(o \cup \sigma_H) \\
= & \quad \langle \text{ Definition of } \otimes, \text{ one-point rule } \rangle
\end{array}
$$

$$
\begin{array}{ll}
 & \sum_{\sigma'_H} b_H(\sigma'_H) \cdot ([\![S]\!]_I(\sigma_L \cup \sigma'_H))(o \cup \sigma_H) \\
= & \quad \langle \text{ Immutable high input, one-point rule } \rangle \\
 & b_H(\sigma_H) \cdot ([\![S]\!]_I(\sigma_L \cup \sigma_H))(o \cup \sigma_H) \\
= & \quad \langle \text{ Immutable high input, definition of } \restriction L \rangle \\
 & b_H(\sigma_H) \cdot (([\![S]\!]_I(\sigma_L \cup \sigma_H)) \restriction L)(o) \\
= & \quad \langle\ S \in \textbf{ObsDet}, \text{ definition of } \delta_L \rangle \\
 & \quad b_H(\sigma_H) \cdot \delta_L(o)
\end{array}
$$

□

Lemma 8.4 Assume the definitions in the contrapositive proof of Theorem 8. Then:

$$b_H(\sigma_H) \quad \neq \quad b'_H(\sigma_H)$$

Proof. First we calculate $b'_H(\sigma_H)$:

$$
\begin{array}{ll}
 & b'_H(\sigma_H) \\
= & \quad \langle \text{ Definition of } b'_H \rangle \\
 & (\delta_A | o \restriction H)(\sigma_H) \\
= & \quad \langle \text{ Calculation of } \delta_A \text{ in Theorem 8} \rangle \\
 & (1/2 \cdot (\delta + \delta')|o \restriction H)(\sigma_H) \\
= & \quad \langle \text{ Definition of } \delta \restriction H, \text{ one-point rule, } D \text{ defined below} \rangle \\
 & \sum_{\sigma_L} (1/2 \cdot (\delta + \delta')|o)(\sigma_L \cup \sigma_H)/D \\
= & \quad \langle \text{ Definition of } \delta|o, \text{ one-point rule} \rangle \\
 & 1/2 \cdot (\delta + \delta')(o \cup \sigma_H)/D \\
= & \quad \langle \text{ Definition of } + \text{ for distributions} \rangle \\
 & 1/2 \cdot (\delta(o \cup \sigma_H) + \delta'(o \cup \sigma'_H))/D \\
= & \quad \langle \text{ Definition of } \delta', \text{ immutability of H input} \rangle \\
 & 1/2 \cdot \delta(o \cup \sigma_H)/D
\end{array}
$$

where $D = 1/2 \cdot (\delta(o \cup \sigma_H) + \delta'(o \cup \sigma'_H))$. Similarly, we can calculate $b'_H(\sigma'_H) = 1/2 \cdot \delta(o \cup \sigma'_H)/D$. Also, we calculate $\delta_L(o)$:

$$
\begin{aligned}
& \delta_L(o) \\
= \quad & \langle \text{ Definition of } \delta_L \text{ and projection } \rangle \\
& \textstyle\sum_{\sigma_H} \ \delta(o \cup \sigma_H) \\
= \quad & \langle \text{ Definition of } \delta \text{, immutability of high input, one-point rule } \rangle \\
& \delta(o \cup \sigma_H)
\end{aligned}
$$

Similarly, $\delta'_L(o) = \delta'(o \cup \sigma'_H)$. By the definition of o we have $\delta_L(o) \neq \delta'_L(o)$, thus $\delta(o \cup \sigma_H) \neq \delta'(o \cup \sigma'_H)$. Thus:

$$
\begin{aligned}
& b'_H(\sigma'_H) \\
= \quad & \langle \text{ Calculated value of } b'_H(\sigma'_H) \rangle \\
& 1/2 \cdot \delta(o \cup \sigma'_H)/D \\
\neq \quad & \langle \text{ Above inequality } \rangle \\
& 1/2 \cdot \delta(o \cup \sigma_H)/D \\
= \quad & \langle \text{ Calculated value of } b'_H(\sigma_H) \rangle \\
& b'_H(\sigma_H)
\end{aligned}
$$

Finally, note that by the immutability of high input, the only high states with non-zero mass in b'_H are σ_H and σ'_H. If $b'_H(\sigma_H) = 1/2$, then because the mass in a belief must sum to 1, we would be forced to conclude $b'_H(\sigma'_H) = 1/2$. But this would contradict the previous calculation. So $b'_H(\sigma_H) \neq 1/2$. Thus, since $b_H(\sigma_H) = 1/2$, we conclude $b_H(\sigma_H) \neq b'_H(\sigma_H)$.
□