Dynamic Enforcement of Knowledge-based Security Policies

Piotr Mardziel, Stephen Magill, Michael Hicks
University of Maryland, College Park

Mudhakar Srivatsa
IBM T.J. Watson Research Laboratory

***Abstract*—This paper explores the idea of *knowledge-based security policies*, which are used to decide whether to answer a query over secret data based on an estimation of the querier's (possibly increased) knowledge given the result. Limiting knowledge is the goal of existing information release policies that employ mechanisms such as noising, anonymization, and redaction. Knowledge-based policies are more general: they increase flexibility by not fixing the means to restrict information flow. We enforce a knowledge-based policy by explicitly tracking a model of a querier's belief about secret data, represented as a probability distribution. We then deny any query that could increase knowledge above a given threshold. We implement query analysis and belief tracking via abstract interpretation using a novel domain we call *probabilistic polyhedra*, whose design permits trading off precision with performance while ensuring estimates of a querier's knowledge are sound. Experiments with our implementation show that several useful queries can be handled efficiently, and performance scales far better than would more standard implementations of probabilistic computation based on sampling.**

I. Introduction

Facebook, Twitter, Flickr, and other successful on-line services enable users to easily foster and maintain relationships by sharing information with friends and fans. These services store users' personal information and use it to customize the user experience and to generate revenue. For example, Facebook third-party applications are granted access to a user's "basic" data (which includes name, profile picture, gender, networks, user ID, and list of friends [1]) to implement services like birthday announcements and horoscopes, while Facebook selects ads based on age, gender, and even sexual preference [2]. Unfortunately, once personal information is collected, users have limited control over how it is used. For example, Facebook's EULA grants Facebook a non-exclusive license to any content a user posts [3]. MySpace, another social network site, has recently begun to sell its users' data [4].

Some researchers have proposed that, to keep tighter control over their data, users could use a storage server (e.g., running on their home network) that handles personal data requests, and only responds when a request is deemed safe [5], [6]. The question is: which requests are safe? The standard answer is to defer to some kind of user-determined access control policy. The problem is that such policies are unnecessarily restrictive when the goal is to maximize the customized personal experience. Consider the following example: a horoscope or "happy birthday" application operates on birth month and day; music recommendation algorithms typically operate on the birth year. Access control at the granularity of the entire birth date could preclude both of these applications, while choosing only to release birth year or birth day precludes access to one application or the other. But in fact the user may not care much about these particular bits of information, but rather about what can be deduced from them. For example, it has been reported that zip code, birthday, and gender are sufficient information to uniquely identify 63% of Americans in the 2000 U.S. census [7]. So the user may be perfectly happy to reveal any one of these bits of information in its entirety as long as a querier gains no better than a $1/n$ chance to guess the entire group, for some parameter n. We call such a policy a *knowledge-based security policy*.

This paper explores one design and implementation strategy for knowledge-based policies. In our model, a user agent U responds to queries involving secret data. For each querying principal Q, agent U maintains a probability distribution over the secret data, representing Q's *belief* of the data's likely values. For example, U may model a social networking site X's otherwise uninformed knowledge of a user's birthday according to a likely demographic: the birth month and day are uniformly distributed, while the birth year is most likely between 1956 and 1992 [8]. Each querier Q is also assigned a knowledge-based policy, expressed as a set of thresholds, each applying to a different group of (potential overlapping) data. For example, U's policy for site X might be a threshold of $1/100$ for the entire tuple $(birthday, zipcode, gender)$, and $1/5$ for just birth month and day. U will not respond to any queries that it determines could increase Q's chances of guessing a secret above the assigned threshold. If deemed safe, U returns the query's (exact) result and updates Q's modeled belief appropriately. Note that if there is a perceived risk of collusion, a single distribution may be used to model a set of principals' collective beliefs.

To implement our model, we need (1) an algorithm to check whether answering a query could violate a knowledge-based policy, (2) a method for revising a querier's belief according to the answer that is given, and (3) means to implement (1) and (2) efficiently. We build on the work of Clarkson et al. [9] (reviewed in Section III), which works out the theoretical basis for (2) and gives us a head start on (1). The main contributions of this paper, therefore, in addition to the idea of knowledge-based policies, are our solutions to problems (1) and (3).

Given a solution to the second problem, a solution to the first problem seems deceptively simple: U runs the query, tentatively revises Q's belief based on the result, and then responds with the answer only if Q's revised belief about the secrets does not exceed the prescribed thresholds. The problem with this approach is that the decision to deny depends on the actual secret, so a rejection could leak information. We give an example in the next section that shows how the entire secret could be revealed. Therefore, we propose that a query should be rejected if *any* secret value Q believes is possible would produce an output whereby the revised belief would exceed the threshold. This idea, given in detail in Section IV, is inspired by Smith's proposal to use min-entropy, rather than Shannon entropy, to characterize a program's security [10].

To implement belief tracking and revision, our first thought was to use languages for probabilistic computation and conditioning, which provide the foundational elements of the approach. Most languages we know of—IBAL [11], Probabilistic Scheme [12], and several other systems [13], [14], [15]—are implemented using sampling. Unfortunately, we found these implementations to be inadequate because they either underestimate the querier's knowledge when sampling too little, or run too slowly when the state space is large.

Instead of using sampling, we have developed an implementation based on abstract interpretation. In Section V we develop a novel abstract domain called a *probabilistic polyhedron*, which extends the standard domain of convex polyhedra [16] with measures of probability. We represent beliefs as a set of probabilistic polyhedra. While some prior work has explored probabilistic abstract interpretation [17], this work does not support belief revision, which is required to track how observation of outputs affects a querier's belief. Support for revision requires that we maintain both under- and over-approximations of the querier's belief, whereas [17] deals only with over-approximation. We have developed an implementation of our approach based on Parma [18] and LattE [19], which we present in Section VII along with some experimental measurements of its performance. We find that while the performance of Probabilistic Scheme degrades significantly as the input space grows, our implementation scales much better, and can be orders of magnitude faster.

The next section presents a technical overview of the rest of the paper, whose main results are contained in Sections III–VII. We compare our approach to related work, and discuss future work and limitations, in Sections VIII and IX.

II. Overview

Knowledge-based policies and beliefs. User Bob would like to enforce a knowledge-based policy on his data so that advertisers do not learn too much about him. Suppose Bob considers his birthday of September 27, 1980 to be relatively private; variable $bday$ stores the calendar day (a number between 0 and 364, which for Bob would be 270) and $byear$ stores the birth year (which would be 1980). To $bday$ he assigns a *knowledge threshold* $t_d = 0.2$ stating that he does not want an advertiser to have better than a 20% likelihood of guessing his birth day. To the pair $(bday, byear)$ he assigns a threshold $t_{dy} = 0.05$, meaning he does not want an advertiser to be able to guess the combination of birth day *and* year together with better than a 5% likelihood.

Bob runs an agent program to answer queries about his data on his behalf. This agent models an estimated *belief* of queriers as a probability distribution δ, which is conceptually a map from secret states to positive real numbers representing probabilities (in range $[0, 1]$). Bob's secret state is the pair $(bday = 270, byear = 1980)$. The agent represents a distribution as a set of probabilistic polyhedra. For now, we can think of a probabilistic polyhedron as a standard convex polyhedron C with a probability mass m, where the probability of each integer point contained in C is $m/\|C\|$, where $\|C\|$ is the number of integer points contained in the polyhedron C. Shortly we present a more involved representation.

Initially, the agent might model an advertiser X's belief using the following rectangular polyhedron C, where each point contained in it is considered equally likely ($m = 1$):

$$C = bday \geq 0, bday \leq 364, byear \geq 1956, byear \leq 1992$$

Enforcing knowledge-based policies safely. Suppose X wants to identify users whose birthday falls within the next week, to promote a special offer. X sends Bob's agent the following program.

Example 1.

```
$today$ := 260;
if $bday \geq today \land bday < (today + 7)$ then
   $output$ := True;
```

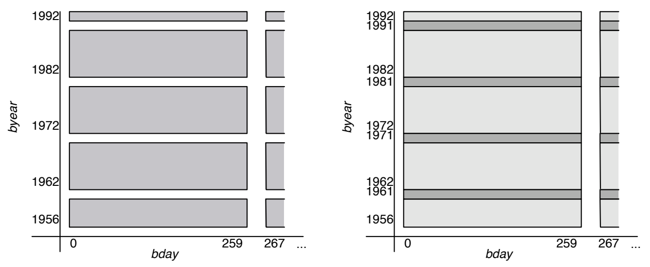
This program refers to Bob's secret variable $bday$, and also uses non-secret variables $today$, which represents the current day and is here set to be 260, and $output$, which is set to True if the user's birthday is within the next seven days (we assume $output$ is initially False).

The agent must decide whether returning the result of running this program will potentially increase X's knowledge about Bob's data above the prescribed threshold. We explain how it makes this determination shortly, but for the present we can see that answering the query is safe: the returned $output$ variable will be False which essentially teaches the querier that Bob's birthday is not within the next week, which still leaves many possibilities. As such, the agent *revises* his model of the querier's belief to be the following *pair* of rectangular polyhedra C_1, C_2, where, again, all points in each are equally likely ($m_1 = 0.726, m_2 = 0.274$):

$$C_1 = bday \geq 0, bday < 260, byear \geq 1956, byear < 1993$$
$$C_2 = bday \geq 267, bday < 365, byear \geq 1956, byear < 1993$$

Ignoring $byear$, there are 358 possible values for $bday$ and each is equally likely. Thus the probability of any one is $1/358 = 0.0028 \leq t_d = 0.2$.

Suppose the next day the same advertiser sends the same program to Bob's user agent, but with $today$ set to 261. Should the agent run the program? At first glance, doing so seems OK. The program will return False, and the revised belief will be the same as above but with constraint $bday \geq 267$ changed



(a) $output =$ False (b) $output =$ True

Fig. 1. Example 2: most precise revised beliefs

to $bday \geq 268$, meaning there is still only a $1/357 = 0.0028$ chance to guess $bday$.

But suppose Bob's birth day was actually 267, rather than 270. The first query would have produced the same revised belief as before, but since the second query would return True (since $bday = 267 < (261 + 7)$), the querier can deduce Bob's birth day exactly: $bday \geq 267$ (from the first query) and $bday < 268$ (from the second query) together imply that $bday = 267$! But the user agent is now stuck: it cannot simply refuse to answer the query, because the querier knows that with $t_d = 0.05$ (or indeed, any reasonable threshold) the only good reason to refuse is when $bday = 267$. As such, refusal essentially tells the querier the answer.

The lesson is that the decision to refuse a query must not be based on the effect of running the query on the actual secret, because then a refusal could leak information. In Section IV we propose that an agent should reject a program if there exists *any* possible secret that could cause a program answer to increase querier knowledge above the threshold. As such we would reject the second query regardless of whether $bday = 270$ or $bday = 267$.

Full probabilistic polyhedra. Now suppose, having run the first query and rejected the second, the user agent receives the following program from X.

Example 2.

```
$age$ := 2011 − $byear$;
if $age = 20 \vee ... \vee age = 60$ then
    $output$ := True;
pif 0.1 then $output$ := True;
```

This program attempts to discover whether this year is a "special" year for the given user, who thus deserves a special offer. The program returns True if either the user's age is (or will be) an exact decade, or if the user wins the luck of the draw (one chance in ten), as implemented by the probabilistic if statement.

Running this program reveals nothing about $bday$, but does reveal something about $byear$. In particular, if $output =$ False then the querier knows that $byear \notin \{1991, 1981, 1971, 1961\}$, but all other years are equally likely. We could represent this new knowledge, combined with the knowledge gained from the first query, as shown in Figure 1(a), where each shaded box is a polyhedron containing equally likely points. On the other hand, if $output =$ True then either $byear \in \{1991, 1981, 1971, 1961\}$ or the user got lucky. We represent the querier's knowledge in this case as in Figure 1(b). Darker shading indicates higher probability; thus, all years are still possible, though some are much more likely than others. With the given threshold of $t_{dy} = 0.05$, the agent will permit the query; when $output =$ False, the likelihood of any point in the shaded region is $1/11814$; when $output =$ True, the points in the dark bands are the most likely, with probability $5/13067$. Since both outcomes are possible with Bob's $byear = 1980$, the revised belief will depend on the result of the probabilistic if statement.

This example illustrates a potential problem with the simple representation of probabilistic polyhedra mentioned earlier: when $output =$ False we will jump from using two probabilistic polyhedra to ten, and when $output =$ True we jump to using eighteen. Allowing the number of polyhedra to grow without bound will result in performance problems. To address this concern, we need a way to abstract our belief representation to be more concise. Section V shows how to represent a probabilistic polyhedron P as a seven-tuple, $(C, \mathrm{s}^{\min}, \mathrm{s}^{\max}, \mathrm{p}^{\min}, \mathrm{p}^{\max}, \mathrm{m}^{\min}, \mathrm{m}^{\max})$ where $\mathrm{s}^{\min}$ and $\mathrm{s}^{\max}$ are lower and upper bounds on the number of points with non-zero probability in the polyhedron C (called the *support points* of C); the quantities $\mathrm{p}^{\min}$ and $\mathrm{p}^{\max}$ are lower and upper bounds on the probability mass per support point; and $\mathrm{m}^{\min}$ and $\mathrm{m}^{\max}$ give bounds on the total probability mass. Thus, polyhedra modeled using the simpler representation (C, m) given earlier are equivalent to ones in the more involved representation with $\mathrm{m}^{\max} = \mathrm{m}^{\min} = m$, $\mathrm{p}^{\max} = \mathrm{p}^{\min} = m/\|C\|$, and $\mathrm{s}^{\max} = \mathrm{s}^{\min} = \|C\|$.

With this representation, we could choose to collapse the sets of polyhedron given in Figure 1. For example, we could represent Figure 1(a) with two probabilistic polyhedra P_1 and P_2 containing polyhedra C_1 and C_2 defined above, respectively, essentially drawing a box around the two groupings of smaller boxes in the figure. The other parameters for P_1 would be as follows:

$$\begin{aligned} \mathrm{p}_1^{\min} &= \mathrm{p}_1^{\max} = 9/135050 \\ \mathrm{s}_1^{\min} &= \mathrm{s}_1^{\max} = 8580 \\ \mathrm{m}_1^{\min} &= \mathrm{m}_1^{\max} = 7722/13505 \end{aligned}$$

Notice that $\mathrm{s}_1^{\min} = \mathrm{s}_2^{\max} = 8580 < \|C_1\| = 9620$, illustrating that the "bounding box" of the polyhedron covers more area than is strictly necessary. In this representation the probabilities may not be normalized, which improves both performance and precision. For this example, P_2 happens to have $\mathrm{m}_2^{\min} = \mathrm{m}_2^{\max} = 3234/13505$ so we can see $\mathrm{m}_1^{\max} + \mathrm{m}_2^{\max} = (10956/13505) \neq 1$.

While all minimums and maximums above are equal, if we consider the representation of Figure 1(b) in a similar manner, using the same two polyhedra C_1 and C_2, the other parameters for C_1 are as follows:

$$\begin{aligned} \mathrm{p}_1^{\min} &= 1/135050 & \mathrm{p}_1^{\max} &= 10/135050 \\ \mathrm{s}_1^{\min} &= 9620 & \mathrm{s}_1^{\max} &= 9620 \\ \mathrm{m}_1^{\min} &= 26/185 & \mathrm{m}_1^{\max} &= 26/185 \end{aligned}$$

$$\begin{array}{llll}
\textit{Variables} & x & \in & \mathbf{Var} \\
\textit{Integers} & n & \in & \mathbb{Z} \\
\textit{Rationals} & q & \in & \mathbb{Q} \\
\textit{Arith.ops} & \textit{aop} & ::= & + \mid \times \mid - \\
\textit{Rel.ops} & \textit{relop} & ::= & \leq \mid < \mid = \mid \neq \mid \cdots \\
\textit{Arith.exps} & E & ::= & x \mid n \mid E_1 \ \textit{aop} \ E_2 \\
\textit{Bool.exps} & B & ::= & E_1 \ \textit{relop} \ E_2 \mid \\
 & & & B_1 \wedge B_2 \mid B_1 \vee B_2 \mid \neg B \\
\textit{Statements} & S & ::= & \mathsf{skip} \mid x \ := \ E \mid \\
 & & & \mathsf{if} \ B \ \mathsf{then} \ S_1 \ \mathsf{else} \ S_2 \mid \\
 & & & \mathsf{pif} \ q \ \mathsf{then} \ S_1 \ \mathsf{else} \ S_2 \mid \\
 & & & S_1 \ ; \ S_2 \mid \mathsf{while} \ B \ \mathsf{do} \ S
\end{array}$$

Fig. 2. Core language syntax

In this case $\mathrm{s}_1^{\min} = \mathrm{s}_1^{\max} = \|C_1\|$, meaning that all covered points are possible, but $\mathrm{p}_1^{\min} \neq \mathrm{p}_1^{\max}$ as some points are more probable than others (i.e., those in the darker band).

The key property of probabilistic polyhedra, and a main technical contribution of this paper, is that this abstraction can be used to make sound security policy decisions. To accept a query, we must check that, for all possible outputs, the querier's revised, normalized belief of any of the possible secrets is below the threshold t. In checking whether the revised beliefs in our example are acceptable, the agent will try to find the maximum probability the querier could ascribe to a state, for each possible output. In the case $\textit{output} = \mathsf{True}$, the most probable points are those in the dark bands, which each have probability mass $10/135050 = \mathrm{p}_1^{\max}$ (the dark bands in P_2 have the same probability). To find the maximum normalized probability of these points, we divide by the minimum possible total mass, as given by the lower bounds in our abstraction. In our example, this results in $\mathrm{p}_1^{\max}/(\mathrm{m}_1^{\min} + \mathrm{m}_2^{\min}) = (10/135050)/(26/185 + 49/925) = 0.0004 \leq t_d = 0.05$.

As just shown, the bound on minimum total mass is needed in order to soundly normalize distributions in our abstraction. The maintenance of such lower bounds on probability mass is a key component of our abstraction that is missing from prior work. Each of the components of a probabilistic polyhedron play a role in producing the lower bound on total mass. While $\mathrm{s}_1^{\min}, \mathrm{s}_1^{\max}, \mathrm{p}_1^{\min}$, and $\mathrm{m}_1^{\max}$ do not play a role in making the final policy decision, their existence allows us to more accurately update belief during the query evaluation that precedes the final policy check. Nevertheless, the choice of the number of probabilistic polyhedra to use impacts both precision and performance, so choosing the right number is a challenge. For the examples given in this section, our implementation can often answer queries in a few of seconds; more details are in Sections V–VII.

III. Tracking beliefs

This section reviews Clarkson et al.'s method of revising a querier's belief of the possible valuations of secret variables based on the result of a query involving those variables [9].

A. Core language

The programming language we use for queries is given in Figure 2. A computation is defined by a statement S whose standard semantics can be viewed as a relation between states: given an input state σ, running the program will produce an output state σ'. States are maps from variables to integers:

$$\sigma, \tau \in \mathbf{State} \overset{\text{def}}{=} \mathbf{Var} \to \mathbb{Z}$$

Sometimes we consider states with domains restricted to a subset of variables V, in which case we write $\sigma_V \in \mathbf{State}_V \overset{\text{def}}{=} V \to \mathbb{Z}$. We may also *project* states to a set of variables V:

$$\sigma \upharpoonright V \overset{\text{def}}{=} \lambda x \in \mathbf{Var}_V . \sigma(x)$$

The language is essentially standard. We limit the form of expressions to support our abstract interpretation-based semantics (Section V). The semantics of the statement form $\mathsf{pif}\ q\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2$ is non-deterministic: the result is that of S_1 with probability q, and S_2 with probability $1 - q$.

B. Probabilistic semantics for tracking beliefs

To enforce a knowledge-based policy, a user agent must be able to estimate what a querier could learn from the output of his query. To do this, the agent keeps a distribution δ that represents the querier's *belief* of the likely valuations of the user's secrets. More precisely, a distribution is a map from states to positive real numbers, interpreted as probabilities (in range $[0, 1]$).

$$\delta \in \mathbf{Dist} \overset{\text{def}}{=} \mathbf{State} \to \mathbb{R}+$$

We sometimes focus our attention on distributions over states of a fixed set of variables V, in which case we write $\delta_V \in \mathbf{Dist}_V$ to mean $\mathbf{State}_V \to \mathbb{R}+$. Projecting distributions onto a set of variables is as follows:

$$\delta \upharpoonright V \overset{\text{def}}{=} \lambda \sigma_V \in \mathbf{State}_V . \sum_{\sigma' \mid (\sigma' \upharpoonright V = \sigma_V)} \delta(\sigma')$$

The *mass* of a distribution, written $\|\delta\|$ is the sum of the probabilities ascribed to states, $\sum_\sigma \delta(\sigma)$. A *normalized distribution* is one such that $\|\delta\| = 1$. A normalized distribution can be constructed by scaling a distribution according to its mass:

$$\text{normal}(\delta) \overset{\text{def}}{=} \frac{1}{\|\delta\|} \cdot \delta$$

The *support* of a distribution is the set of states which have non-zero probability: *support*$(\delta) \overset{\text{def}}{=} \{\sigma \mid \delta(\sigma) > 0\}$.

The agent evaluates a query in light of the querier's initial belief using a probabilistic semantics. Figure 3 defines a semantic function $[\![\cdot]\!]$ whereby $[\![S]\!]\delta = \delta'$ indicates that, given an input distribution δ, the semantics of program S is the output distribution δ'. The semantics is defined in terms of operations on distributions, including *assignment* $\delta\,[v \to E]$ (used in the rule for $v := E$), *conditioning* $\delta | B$ and *addition* $\delta_1 + \delta_2$ (used in the rule for if), and *scaling* $q \cdot \delta$ where q is a rational (used for pif). The rules are standard, so we omit discussion of them here; the appendix provides a brief review.

$$\begin{array}{rcl}
[\![\mathsf{skip}]\!]\delta & = & \delta \\
[\![x\ :=\ E]\!]\delta & = & \delta\,[x \to E] \\
[\![\mathsf{if}\ B\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2]\!]\delta & = & [\![S_1]\!](\delta|B) + [\![S_2]\!](\delta|\neg B) \\
[\![\mathsf{pif}\ q\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2]\!]\delta & = & [\![S_1]\!](q\cdot\delta) + [\![S_2]\!]((1-q)\cdot\delta) \\
[\![S_1\ ;\ S_2]\!]\delta & = & [\![S_2]\!]([\![S_1]\!]\delta) \\
[\![\mathsf{while}\ B\ \mathsf{do}\ S]\!]\delta & = & \mathrm{fix}(\lambda f : \mathbf{Dist} \to \mathbf{Dist}.\lambda\delta. \\
 & & \quad f([\![S]\!](\delta|B) + (\delta|\neg B)))
\end{array}$$

where

$$\begin{array}{ll}
\delta\,[x \to E] & \stackrel{\text{def}}{=} \lambda\sigma.\, \sum_{\tau\ |\ \tau[x\to[\![E]\!]\tau]=\sigma} \delta(\tau) \\
\delta_1 + \delta_2 & \stackrel{\text{def}}{=} \lambda\sigma.\ \delta_1(\sigma) + \delta_2(\sigma) \\
\delta|B & \stackrel{\text{def}}{=} \lambda\sigma.\ \textbf{if}\ [\![B]\!]\sigma\ \textbf{then}\ \delta(\sigma)\ \textbf{else}\ 0 \\
p\cdot\delta & \stackrel{\text{def}}{=} \lambda\sigma.\, p\cdot\delta(\sigma)
\end{array}$$

Fig. 3. Probabilistic semantics for the core language

C. Belief and security

Clarkson et al. [9] describe how a belief about possible values of a secret, expressed as a probability distribution, can be revised according to an experiment using the actual secret. Such an experiment works as follows.

The values of the set of secret variables H are given by the hidden state σ_H. The attacker's initial belief as to the possible values of σ_H is represented as a distribution δ_H. A query is a program S that makes use of variables H and possibly other, non-secret variables from a set L; the final values of L, after running S, will be made visible to the attacker. Let σ_L be an arbitrary initial state of these variables such that *domain*$(\sigma_L) = L$. Then we take the following steps:

Step 1. Evaluate S probabilistically using the attacker's belief about the secret to produce an output distribution δ', which amounts to the attacker's prediction of the possible output states. This is computed as $\delta' = [\![S]\!]\delta$, where δ, a distribution over variables $H \uplus L$, is defined as $\delta = \delta_H \times \dot{\sigma}_L$. Here, we make use of the distribution product operator and point operator. That is, given δ_1, δ_2, which are distributions over states having disjoint domains, the *distribution product* is

$$\delta_1 \times \delta_2 \stackrel{\text{def}}{=} \lambda(\sigma_1,\sigma_2).\ \delta_1(\sigma_1)\cdot\delta_2(\sigma_2)$$

where (σ_1, σ_2) is the "concatenation" of the two states, which is itself a state and is well-defined because the two states' domains are disjoint. And, given a state σ, the *point distribution* $\dot{\sigma}$ is a distribution in which only σ is possible:

$$\dot{\sigma} \stackrel{\text{def}}{=} \lambda\tau.\ \textbf{if}\ \sigma = \tau\ \textbf{then}\ 1\ \textbf{else}\ 0$$

Thus, the initial distribution δ is the attacker's belief about the secret variables combined with an arbitrary valuation of the public variables.

Step 2. Using the actual secret σ_H, evaluate S "concretely" to produce an output state $\hat{\sigma}_L$, in three steps. First, we have $\hat{\delta}' = [\![S]\!]\hat{\delta}$, where $\hat{\delta} = \dot{\sigma}_H \times \dot{\sigma}_L$. Second, we have $\hat{\sigma} \in \Gamma(\hat{\delta})$ where Γ is a sampling operator that produces a state σ from the domain of a distribution δ with probability $\delta(\sigma)/\|\delta\|$. Finally, we extract the attacker-visible output of the sampled state by projecting away the high variables: $\hat{\sigma}_L = \hat{\sigma} \upharpoonright L$.

Step 3. Revise the attacker's initial belief δ_H according to the observed output $\hat{\sigma}_L$, yielding a new belief $\hat{\delta}_H = \delta'|\hat{\sigma}_L \upharpoonright H$. Here, δ' is *conditioned* on the output σ_L, which yields a new distribution, and this distribution is then projected to the variables H. The conditioning operation is defined as follows:

$$\delta|\sigma_V \stackrel{\text{def}}{=} \lambda\sigma.\ \textbf{if}\ \sigma \upharpoonright V = \sigma_V\ \textbf{then}\ \delta(\sigma)\ \textbf{else}\ 0$$

Note that this protocol assumes that S always terminates and does not modify the secret state. The latter assumption can be eliminated by essentially making a copy of the state before running the program, while eliminating the former differs depending on the observer's ability to detect nontermination [9].

IV. Enforcing knowledge-based policies

When presented with a query over a user's data σ_H, the user's agent should only answer the query if doing so will not reveal too much information. More precisely, given a query S, the agent will only return the public output σ_L resulting from running S on σ_H if the agent deems that from this output the querier cannot guess the secret state σ_H beyond some level of doubt, identified by a threshold t. If this threshold could be exceeded, then the agent declines to run S. We call this security check *knowledge threshold security*.

Definition 3 (Knowledge Threshold Security)**.** Let $\delta' = [\![S]\!]\delta$, where δ is the model of the querier's initial belief. Then query S is *threshold secure* iff for all $\sigma_L \in$ *support*$(\delta' \upharpoonright L)$ and all $\sigma'_H \in \mathbf{State}_H$ we have $(\text{normal}(\delta'|\sigma_L \upharpoonright H))(\sigma'_H) \leq t$ for some threshold t.

This definition can be related to the experiment protocol defined in Section III-C. First, δ' in the definition is the same as δ' computed in the first step of the protocol. Step 2 in the protocol produces a concrete output $\hat{\sigma}_L$ based on executing S on the actual secret σ_H, and Step 3 revises the querier's belief based on this output. Definition 3 generalizes these two steps: instead of considering a single concrete output based on the actual secret it considers *all possible* concrete outputs, as given by *support*$(\delta' \upharpoonright L)$, and ensures that the revised belief in each case for *all possible* secret states must assign probability no greater than t.

This definition considers a threshold for the whole secret state σ_H. As described in Section II we can also enforce thresholds over portions of a secret state. In particular, a threshold that applies only to variables $V \subseteq H$ requires that all $\sigma'_V \in \mathbf{State}_V$ result in $(\text{normal}(\delta'|\sigma_L \upharpoonright V))(\sigma'_V) \leq t$.

The two "foralls" in the definition are critical for ensuring security. The reason was shown by the first example in Section II: If we used the flawed approach of just running the experiment protocol and checking if $\hat{\delta}_H(\sigma_H) > t$ then rejection depends on the value of the secret state and could reveal information about it. Indeed, even a more general policy $\forall\sigma_L \in$ *support*$(\delta' \upharpoonright L).\,(\text{normal}(\delta'|\sigma_L \upharpoonright H))(\sigma_H) \leq t$, which would sidestep the problem in the example, could reveal information because it, too, depends on the actual secret σ_H.

(An example illustrating the problem in this case is given in the appendix.) Definition 3 avoids any inadvertent information leakage because rejection is not based on the actual secret: if there exists *any* secret such that a possible output would reveal too much, the query is rejected. Definition 3 resembles *min-entropy*, since the security decision is made based on the most likely secret from the attacker's point of view [10]. In fact, the use of a simple threshold t corresponds to a minimum relative entropy 2^{-t} between revised belief and the true belief [9].

V. Belief revision via abstract interpretation

Consider how we might implement belief tracking and revision to enforce the threshold security property given in Definition 3. A natural choice would be to evaluate queries using a probabilistic programming language with support for conditioning, such as IBAL [11], Probabilistic Scheme [12], or another system [13], [14], [15]. In these languages, normalization (following conditioning) is implemented by sampling. In particular, they select a random set of input states X and compute the sum $\Sigma_{\sigma \in X} (\delta | \sigma_V)(\sigma)$. The probabilities of *support*$(\delta | \sigma_V)$ are scaled by this sum. Unfortunately, to get a reasonable estimate of the total sum requires sampling over the entire input space, which could be quite large. If insufficient coverage is achieved, then the threshold check in Definition 3 could either be unsound or excessively conservative, depending in which direction an implementation errs.

To avoid sampling, we have developed a new means to perform probabilistic computation based on abstract interpretation, for which conditioning and normalization are relatively inexpensive. In the next two sections, we present two abstract domains. This section presents the first, denoted $\mathbb{P}$, for which an abstract element is a single *probabilistic polyhedron*, which is a convex polyhedron [16] combined with information about probabilities of its points. Because using a single polyhedron will accumulate precision after multiple queries, in our implementation we actually use a different domain, denoted $\mathcal{P}_n(\mathbb{P})$, for which an abstract element consists of a set of at most n probabilistic polyhedra (whose construction is inspired by powersets of polyhedra [20], [21]). This domain, described in the next section, allows us to retain precision at the cost of increased execution time. By adjusting n, the user can trade off efficiency and precision.

A. Polyhedra

We first review *convex polyhedra*, a common technique for representing sets of program states. We use the meta-variables $\beta, \beta_1, \beta_2, \ldots$ to denote linear inequalities. We write $fv(\beta)$ to be the set of variables occurring in β; we also extend this to sets, writing $fv(\{\beta_1, \ldots, \beta_n\})$ for $fv(\beta_1) \cup \ldots \cup fv(\beta_n)$.

Definition 4. A *convex polyhedron* C is a set of linear inequalities $\{\beta_1, \ldots, \beta_m\}$, interpreted conjunctively. We write $\mathbb{C}$ for the set of all convex polyhedra. A polyhedron C represents a set of states, denoted $\gamma_{\mathbb{C}}(C)$, as follows, where $\sigma \models \beta$ indicates that the state σ satisfies the inequality β.

$$\gamma_{\mathbb{C}}(C) \stackrel{\text{def}}{=} \{\sigma \mid \mathit{domain}(\sigma) \supseteq fv(C) \land \forall \beta \in C.\ \sigma \models \beta\}$$

Given a state σ and an ordering on the variables in *domain*(σ), we can view σ as a point in an N-dimensional space, where $N = |\mathit{domain}(\sigma)|$. The set $\gamma_{\mathbb{C}}(C)$ can then be viewed as the integer-valued lattice points in an N-dimensional polyhedron. Due to this correspondence, we use the words *point* and *state* interchangeably. We will also allow ourselves to write linear equalities $x = f(\vec{y})$ as an abbreviation for the pair of inequalities $x \leq f(\vec{y})$ and $x \geq f(\vec{y})$.

Convex polyhedra support the following operations.

- Polyhedron size, or $\#(C)$, is the number of integer points in the polyhedron, i.e., $|\gamma_{\mathbb{C}}(C)|$. We will always consider bounded polyhedra, ensuring that $\#(C)$ is finite.
- Expression Evaluation, $\langle\langle B \rangle\rangle\, C$ returns a convex polyhedron containing at least all points in C that satisfy B.
- Expression Count, $C \# B$ returns an upper bound on the number of integer points in C that satisfy B.
- Meet, $C_1 \sqcap_{\mathbb{C}} C_2$ is the convex polyhedron representing the set of points in the intersection of $\gamma_{\mathbb{C}}(C_1), \gamma_{\mathbb{C}}(C_2)$.
- Join, $C_1 \sqcup_{\mathbb{C}} C_2$ is the smallest convex polyhedron containing both $\gamma(C_1)$ and $\gamma(C_2)$.
- Comparison, $C_1 \sqsubseteq_{\mathbb{C}} C_2$ is a partial order whereby $C_1 \sqsubseteq_{\mathbb{C}} C_2$ if and only if $\gamma(C_1) \subseteq \gamma(C_2)$.
- Affine transform, $C\,[x \to E]$, where $x \in fv(E)$, computes an affine transformation of C. This scales the dimension corresponding to x by the coefficient of x in E and shifts the polyhedron. For example, $\{x \leq y, y = 2z\}\,[y \to z + y]$ evaluates to $\{x \leq y - z, y - z = 2z\}$.
- Forget, $\mathrm{f}_x(C)$, projects away x. That is, $\mathrm{f}_x(C) = \pi_{fv(C) - \{x\}}(C)$, where $\pi_V(C)$ is a polyhedron C' such that $\gamma_{\mathbb{C}}(C') = \{\sigma \mid \sigma' \in \gamma_{\mathbb{C}}(C) \land \sigma = \sigma' \restriction V\}$. So $C' = \mathrm{f}_x(C)$ implies $x \notin fv(C')$.

We write $\bot$ to denote the empty polyhedron. We have $\gamma_{\mathbb{C}}(\bot) = \emptyset$ and any polyhedron with an inconsistent set of linear constraints is equivalent to $\bot$.

B. Probabilistic Polyhedra

We take this standard representation of sets of program states and extend it to a representation for sets of distributions over program states. We define *probabilistic polyhedra*, the core element of our abstract domain, as follows.

Definition 5. A *probabilistic polyhedron* P is a tuple $(C, \mathrm{s}^{\min}, \mathrm{s}^{\max}, \mathrm{p}^{\min}, \mathrm{p}^{\max}, \mathrm{m}^{\min}, \mathrm{m}^{\max})$. We write $\mathbb{P}$ for the set of probabilistic polyhedra. The quantities $\mathrm{s}^{\min}$ and $\mathrm{s}^{\max}$ are lower and upper bounds on the number of support points in the polyhedron C. The quantities $\mathrm{p}^{\min}$ and $\mathrm{p}^{\max}$ are lower and upper bounds on the probability mass *per support point*. The $\mathrm{m}^{\min}$ and $\mathrm{m}^{\max}$ components give bounds on the total probability mass. Thus P represents the *set* of distributions $\gamma_{\mathbb{P}}(P)$ defined below.

$$\begin{aligned} \gamma_{\mathbb{P}}(P) \stackrel{\text{def}}{=} \{\delta \mid\ & \mathit{support}(\delta) \subseteq \gamma_{\mathbb{C}}(C) \land \\ & \mathrm{s}^{\min} \leq |\mathit{support}(\delta)| \leq \mathrm{s}^{\max} \land \\ & \mathrm{m}^{\min} \leq \|\delta\| \leq \mathrm{m}^{\max} \land \\ & \forall \sigma \in \mathit{support}(\delta).\ \mathrm{p}^{\min} \leq \delta(\sigma) \leq \mathrm{p}^{\max}\} \end{aligned}$$

Note the set $\gamma_{\mathbb{P}}(P)$ is singleton exactly when $\mathrm{s}^{\min} = \mathrm{s}^{\max} = \#(C)$ and $\mathrm{p}^{\min} = \mathrm{p}^{\max}$, and $\mathrm{m}^{\min} = \mathrm{m}^{\max}$. In such a case $\gamma_{\mathbb{P}}(P)$ is the uniform distribution where each state in $\gamma_{\mathbb{C}}(C)$ has probability $\mathrm{p}^{\min}$. Distributions represented by a probabilistic polyhedron are not necessarily normalized (as was true in Section III-B). In general, there is a relationship between $\mathrm{p}^{\min}, \mathrm{s}^{\min}$, and $\mathrm{m}^{\min}$, in that $\mathrm{m}^{\min} \geq \mathrm{p}^{\min} \cdot \mathrm{s}^{\min}$ (and $\mathrm{m}^{\max} \leq \mathrm{p}^{\max} \cdot \mathrm{s}^{\max}$), and the combination of the three can yield more information than any two in isolation.

Our convention will be to always use $C_1, \mathrm{s}_1^{\min}, \mathrm{s}_1^{\max}$, etc. for the components associated with probabilistic polyhedron P_1 and to use subscripts to name different probabilistic polyhedra.

In [9], distributions are ordered point-wise. That is, $\delta_1 \leq \delta_2$ if and only if $\forall \sigma.\ \delta_1(\sigma) \leq \delta_2(\sigma)$. To extend this to our abstract domain, we say that $P_1 \sqsubseteq_{\mathbb{P}} P_2$ if and only if $\forall \delta_1 \in P_1.\ \exists \delta_2 \in P_2.\ \delta_1 \leq \delta_2$. This corresponds to the following definition, given in terms of the components of P_1 and P_2

Definition 6. $P_1 \sqsubseteq_{\mathbb{P}} P_2$ if and only if $C_1 \sqsubseteq_{\mathbb{C}} C_2$, $\mathrm{m}_1^{\max} \leq \mathrm{m}_2^{\max}$, $\mathrm{s}_1^{\max} \leq \mathrm{s}_2^{\max}$, and $\mathrm{p}_1^{\max} \leq \mathrm{p}_2^{\max}$.

The least element is then the probabilistic polyhedron P with $C = \bot$, $\mathrm{m}^{\max} = 0$, $\mathrm{s}^{\max} = 0$, $\mathrm{p}^{\max} = 0$, which represents the zero distribution.

In a standard abstract domain, termination of the fixed point computation for loops is often ensured by use of a widening operator. This allows abstract fixed points to be computed in fewer iterations and also permits analysis of loops that may not terminate. In our setting, non-termination may reveal information about secret values. As such, we would like to reject queries that may be non-terminating.

We enforce this by not introducing a widening operator. Our abstract interpretation then has the property that it will not terminate if a loop in the query may be non-terminating (and, since it is an over-approximate analysis, it may also fail to terminate even for some terminating computations). We then reject all queries for which our analysis fails to terminate. Loops do not play a major role in any of our examples, and so this approach has proved sufficient so far. We leave for future work the development of an abstract domain with widening that soundly accounts for non-termination behavior.

Following standard abstract interpretation terminology, we will refer to $\mathcal{P}(\mathbf{Dist})$ (sets of distributions) as the *concrete domain*, $\mathbb{P}$ as the *abstract domain*, and $\gamma_{\mathbb{P}} : \mathbb{P} \to \mathcal{P}(\mathbf{Dist})$ as the *concretization function* for $\mathbb{P}$.

C. Abstract Semantics for $\mathbb{P}$

In order to support execution in the abstract domain just defined, we need to provide abstract implementations of the basic operations of assignment, conditioning, addition, and scaling used in the concrete semantics given in Figure 3. We will overload notation and use the same syntax for the abstract operators as we did for the concrete operators.

As we present each operation, we will also state the associated soundness theorem which shows that the abstract operation is an over-approximation of the concrete operation. Proofs are given in a forthcoming technical report. The abstract program semantics is then exactly the semantics from Figure 3, but making use of the abstract operations defined here, rather than the operations on distributions defined in Section III-B. We will write $\langle\langle S \rangle\rangle\, P$ to denote the result of executing S using the abstract semantics. The main soundness theorem we obtain is the following.

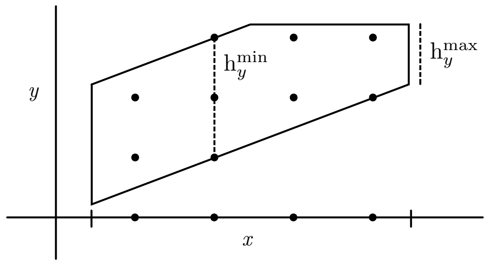


Fig. 4. An example of a forget operation in the abstract domain $\mathbb{P}$. In this case, $\mathrm{h}_y^{\min} = 1$ and $\mathrm{h}_y^{\max} = 3$. Note that $\mathrm{h}_y^{\max}$ is precise while $\mathrm{h}_y^{\min}$ is an under-approximation. If $\mathrm{s}_1^{\min} = \mathrm{s}_1^{\max} = 9$ then we have $\mathrm{s}_2^{\min} = 3$, $\mathrm{s}_2^{\max} = 4$, $\mathrm{p}_2^{\min} = \mathrm{p}_1^{\min} \cdot 1$, $\mathrm{p}_2^{\max} = \mathrm{p}_2^{\max} \cdot 4$.

Theorem 7. *If $\delta \in \gamma_{\mathbb{P}}(P)$ and $([\![S]\!]\delta) = \delta'$ then $\delta' \in \gamma_{\mathbb{P}}(\langle\langle S \rangle\rangle\, P)$.*

We now present the abstract operations.

1) Forget: We first describe the abstract forget operator $\mathrm{f}_y(P_1)$, which is used in implementing assignment. When we forget variable y, we collapse any states that are equivalent up to the value of y into a single state. In order to do this correctly, we must find an upper bound $\mathrm{h}_y^{\max}$ and a lower bound $\mathrm{h}_y^{\min}$ on the number of different points that share the same value of x (this may be visualized of as the min and max height of C_1 in the y dimension). Once these are obtained, we have that $\mathrm{f}_y(P_1) \overset{\text{def}}{=} P_2$ where the following hold of P_2.

$$\begin{array}{lll} C_2 & = & \mathrm{f}_y(C_1) \\ \mathrm{p}_2^{\min} & = & \mathrm{p}_1^{\min} \cdot \max\big(\mathrm{h}_y^{\min} - (\#(C_1) - \mathrm{s}_1^{\min}),\ 1\big) \\ \mathrm{p}_2^{\max} & = & \mathrm{p}_1^{\max} \cdot \min\big(\mathrm{h}_y^{\max},\ \mathrm{s}_1^{\max}\big) \end{array}$$

$$\begin{array}{lll|lll} \mathrm{s}_2^{\min} & = & \lceil \mathrm{s}_1^{\min} / \mathrm{h}_y^{\max} \rceil & \mathrm{m}_2^{\min} & = & \mathrm{m}_1^{\min} \\ \mathrm{s}_2^{\max} & = & \min\big(\#(\mathrm{f}_y(C_1)),\ \mathrm{s}_1^{\max}\big) & \mathrm{m}_2^{\max} & = & \mathrm{m}_1^{\max} \end{array}$$

Figure 4 gives an example of a forget operation and illustrates the quantities $\mathrm{h}_y^{\max}$ and $\mathrm{h}_y^{\min}$. The upper bound $\mathrm{h}_y^{\max}$ can be found by maximizing $y - y'$ subject to the constraints $C_1 \cup C_1[y'/y]$, where y' is a fresh variable and $C_1[y'/y]$ represents the set of constraints obtained by substituting y' for y in C_1. As our points are integer-valued, this is an integer linear programming problem (and can be solved by ILP solvers). A less precise upper bound can be found by simply taking the extent of the polyhedron C_1 along y, which is given by $\#(\pi_y(C_1))$.

For the lower bound, it is always sound to use 1, and this what our implementation does. A more precise estimate can be obtained by checking each vertex to find the vertex with minimal height along dimension y. Call this distance u. Since the shape is convex, all other points will have y height greater than or equal to u. We then find the smallest number of integer points that can be covered by a line segment of length u. This is given by $\lceil u \rceil - 1$. This value can be taken as $\mathrm{h}_y^{\min}$.

Since the forget operator is related to projection, we state soundness in terms of the projection operation on distributions.

Lemma 8. *If $\delta \in \gamma_{\mathbb{P}}(P)$ then $\delta \restriction (fv(\delta) - \{y\}) \in \gamma_{\mathbb{P}}(\mathrm{f}_y(P))$.*

We can define an abstract version of projection using forget:

Definition 9. Let $\mathrm{f}_{\{x_1,x_2,\ldots,x_n\}}(P) = \mathrm{f}_{\{x_2,\ldots,x_n\}}(\mathrm{f}_{x_1}(P))$. Then $P \restriction V' = \mathrm{f}_{(domain(P)-V')}(P)$.

That is, in order to project onto the set of variables V', we forget all variables not in V'.

2) Assignment: We have two cases for abstract assignment. If the assignment is invertible, then the result of the assignment $P_1\,[x \to E]$ is the probabilistic polyhedron P_2 where $C_2 = C_1\,[x \to E]$ and all other components are unchanged.

If the assignment $x := E$ is not invertible, then information about the previous value of x is lost. In this case, we use the forget operation to project onto the other variables and then add a new constraint on x. Let $P_2 = \mathrm{f}_x(P_1)$. Then $P_1\,[x \to E]$ is the probabilistic polyhedron P_3 where all values are as in P_2 except that $C_3 = C_2 \cup \{x = E\}$.

Lemma 10. *If $\delta \in \gamma_{\mathbb{P}}(P)$ then $\delta\,[v \to E] \in \gamma_{\mathbb{P}}(P\,[v \to E])$.*

The soundness of assignment relies on the fact that our language of expressions does not include division. An invariant of our representation is that $\mathrm{s}^{\max} \leq \#(C)$. When E contains only multiplication and addition the above rules preserve this invariant; an E containing division would violate it. Division should result in multiple points collapsing to one and would be dealt with in a manner similar to that used for projection, which accounts for the same possibility.

3) Plus: In order to compute the effect of plus on the number of support points and probability per support point, we need to consider how the support points contained in the polyhedra being added may overlap. That is, we need to determine the minimum and maximum number of points in the intersection that may be a support point for both P_1 and for P_2. We refer to these counts as the *pessimistic overlap* and *optimistic overlap*, respectively, and define them below.

Definition 11. Given two distributions δ_1, δ_2, we refer to the set of states that are in the support of both δ_1 and δ_2 as the *overlap* of δ_1, δ_2. The *pessimistic overlap* of P_1 and P_2, denoted $P_1 \frown P_2$, is the cardinality of the smallest possible overlap for any distributions $\delta_1 \in \gamma_{\mathbb{P}}(P_1)$ and $\delta_2 \in \gamma_{\mathbb{P}}(P_2)$. The *optimistic overlap* $P_1 \smile P_2$ is the cardinality of the largest possible overlap. Formally, we define these as follows. $n_3 \stackrel{\text{def}}{=} \#(C_1 \sqcap_{\mathbb{C}} C_2)$, $n_1 \stackrel{\text{def}}{=} \#(C_1) - n_3$, and $n_2 \stackrel{\text{def}}{=} \#(C_2) - n_3$. Then

$$P_1 \frown P_2 \stackrel{\text{def}}{=} \max((\mathrm{s}_1^{\min} - n_1) + (\mathrm{s}_2^{\min} - n_2) - n_3,\ 0)$$
$$P_1 \smile P_2 \stackrel{\text{def}}{=} \min(\mathrm{s}_1^{\max}, \mathrm{s}_2^{\max}, n_3)$$

We now use these concepts to define the abstract version of distribution addition.

Definition 12. $P_1 + P_2$ is the probabilistic polyhedron $P_3 = (C_3, \mathrm{s}_3^{\min}, \mathrm{s}_3^{\max}, \mathrm{p}_3^{\min}, \mathrm{p}_3^{\max})$ defined as follows.

$$\begin{aligned}
C_3 &= C_1 \sqcup_{\mathbb{C}} C_2 \\
\mathrm{p}_3^{\min} &= \begin{cases} \mathrm{p}_1^{\min} + \mathrm{p}_2^{\min} & \text{if } P_1 \frown P_2 = \#(C_3) \\ \min(\mathrm{p}_1^{\min}, \mathrm{p}_2^{\min}) & \text{otherwise} \end{cases} \\
\mathrm{p}_3^{\max} &= \begin{cases} \mathrm{p}_1^{\max} + \mathrm{p}_2^{\max} & \text{if } P_1 \smile P_2 > 0 \\ \max(\mathrm{p}_1^{\max}, \mathrm{p}_2^{\max}) & \text{otherwise} \end{cases} \\
\mathrm{s}_3^{\min} &= \mathrm{s}_1^{\min} + \mathrm{s}_2^{\min} - P_1 \smile P_2 \\
\mathrm{s}_3^{\max} &= \mathrm{s}_1^{\max} + \mathrm{s}_2^{\max} - P_1 \frown P_2 \\
\mathrm{m}_3^{\min} &= \mathrm{m}_1^{\min} + \mathrm{m}_2^{\min} \quad | \quad \mathrm{m}_3^{\max} = \mathrm{m}_1^{\max} + \mathrm{m}_2^{\max}
\end{aligned}$$

Lemma 13. *If $\delta_1 \in \gamma_{\mathbb{P}}(P_1)$ and $\delta_2 \in \gamma_{\mathbb{P}}(P_2)$ then $\delta_1 + \delta_2 \in \gamma_{\mathbb{P}}(P_1 + P_2)$.*

4) Product: When evaluating the product $P_3 = P_1 \times P_2$, we assume that the domains of P_1 and P_2 are disjoint. This implies that C_1 and C_2 are polyhedra involving disjoint sets of variables. Let $fv(C_1) = V_1$ and $fv(C_2) = V_2$. The polyhedron $C_1 \cup C_2$ is the Cartesian product of C_1 and C_2 and contains all those states σ for which $\sigma \restriction V_1 \in \gamma_{\mathbb{C}}(C_1)$ and $\sigma \restriction V_2 \in \gamma_{\mathbb{C}}(C_2)$. What remains is to ensure that the probabilities and support point counts are correctly accounted for. The full list of constraints on P_3 is given below.

$$C_3 = C_1 \cup C_2$$
$$\begin{array}{rcl|rcl}
\mathrm{p}_3^{\min} &=& \mathrm{p}_1^{\min} \cdot \mathrm{p}_2^{\min} & \mathrm{p}_3^{\max} &=& \mathrm{p}_1^{\max} \cdot \mathrm{p}_2^{\max} \\
\mathrm{s}_3^{\min} &=& \mathrm{s}_1^{\min} \cdot \mathrm{s}_2^{\min} & \mathrm{s}_3^{\max} &=& \mathrm{s}_1^{\max} \cdot \mathrm{s}_2^{\max} \\
\mathrm{m}_3^{\min} &=& \mathrm{m}_1^{\min} \cdot \mathrm{m}_2^{\min} & \mathrm{m}_3^{\max} &=& \mathrm{m}_1^{\max} \cdot \mathrm{m}_2^{\max}
\end{array}$$

Lemma 14. *If $\delta_1 \in \gamma_{\mathbb{P}}(P_1)$ and $\delta_2 \in \gamma_{\mathbb{P}}(P_2)$ then $\delta_1 \times \delta_2 \in \gamma_{\mathbb{P}}(P_1 \times P_2)$.*

In our examples we often find it useful to express uniformly distributed data directly, rather than encoding it using pif. In particular, consider extending statements S to include the statement form uniform $x\ n_1\ n_2$ whose semantics is to define variable x as possibly having values uniformly distributed between n_1 and n_2. Its semantics is as follows.

$$[\![\mathsf{uniform}\ x\ n_1\ n_2]\!]P_1 = \mathrm{f}_x(P_1) \times P_2$$

Here, P_2 has $\mathrm{p}_2^{\min} = \mathrm{p}_2^{\max} = \frac{1}{n_2 - n_1 + 1}$, $\mathrm{s}_2^{\min} = \mathrm{s}_2^{\max} = n_2 - n_1 + 1$, $\mathrm{m}_2^{\min} = \mathrm{m}_2^{\max} = 1$, and $C_2 = \{x \geq n_1, x \leq n_2\}$.

5) Conditioning: Distribution conditioning for probabilistic polyhedra serves the same role as meet in the classic domain of polyhedra in that each is used to perform abstract evaluation of a conditional expression in its respective domain.

Definition 15. Consider the probabilistic polyhedron P_1 and Boolean expression B. Let $n, \overline{n}$ be such that $n = C_1 \# B$ and $\overline{n} = C_1 \# (\neg B)$. The value n is an over-approximation of the number of points in C_1 that satisfy the condition B and $\overline{n}$ is an over-approximation of the number of points in C_1 that do not satisfy B. Then $P_1 \mid B$ is the probabilistic polyhedron P_2

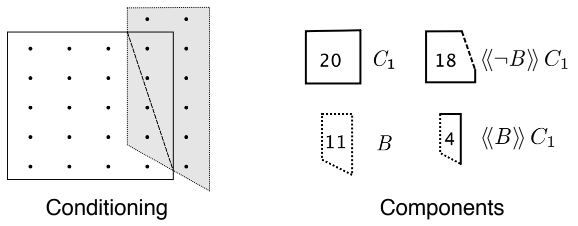

20
C_1
18
$\langle\langle \neg B \rangle\rangle\, C_1$
11
B
4
$\langle\langle B \rangle\rangle\, C_1$
Conditioning
Components

Fig. 5. An example of distribution conditioning in the abstract domain $\mathbb{P}$.

defined as follows.

$$\begin{array}{ll|ll} \mathrm{p}_2^{\min} & = \mathrm{p}_1^{\min} & \mathrm{s}_2^{\min} & = \max(\mathrm{s}_1^{\min} - \overline{n}, 0) \\ \mathrm{p}_2^{\max} & = \mathrm{p}_1^{\max} & \mathrm{s}_2^{\max} & = \min(\mathrm{s}_1^{\max}, n) \end{array}$$

$$\begin{array}{ll} \mathrm{m}_2^{\min} & = \max\big(\mathrm{p}_2^{\min} \cdot \mathrm{s}_2^{\min},\ \mathrm{m}_1^{\min} - \mathrm{p}_1^{\max} \cdot \min\big(\mathrm{s}_1^{\max}, \overline{n}\big)\big) \\ \mathrm{m}_2^{\max} & = \min\big(\mathrm{p}_2^{\max} \cdot \mathrm{s}_2^{\max},\ \mathrm{m}_1^{\max} - \mathrm{p}_1^{\min} \cdot \max\big(\mathrm{s}_1^{\min} - n, 0\big)\big) \\ C_2 & = \langle\langle B \rangle\rangle\, C_1 \end{array}$$

The maximal and minimal probability per point are unchanged, as conditioning simply selectively retains points from the original distribution. To compute the minimal number of points in P_2, we assume that as many points as possible from C_1 fall in the region satisfying $\neg B$. The maximal number of points is obtained by assuming that a maximal number of points fall within the region satisfying B.

The total mass calculations are more complicated. There are two possible approaches to computing $\mathrm{m}_2^{\min}$ and $\mathrm{m}_2^{\max}$. The bound $\mathrm{m}_2^{\min}$ can never be less than $\mathrm{p}_2^{\min} \cdot \mathrm{s}_2^{\min}$, and so we can always safely choose this as the value of $\mathrm{m}_2^{\min}$. Similarly, we can always choose $\mathrm{p}_2^{\max} \cdot \mathrm{s}_2^{\max}$ as the value of $\mathrm{m}_2^{\max}$. However, if $\mathrm{m}_1^{\min}$ and $\mathrm{m}_1^{\max}$ give good bounds on total mass (i.e., $\mathrm{m}_1^{\min}$ is much higher than $\mathrm{p}_1^{\min} \cdot \mathrm{s}_1^{\min}$ and similarly for $\mathrm{m}_1^{\max}$), then it can be advantageous to reason starting from these bounds.

We can obtain a sound value for $\mathrm{m}_2^{\min}$ by considering the case where a maximal amount of mass from C_1 fails to satisfy B. To do this, we compute $\overline{n} = C_1 \# \neg B$, which provides an over-approximation of the number of points within C_1 but outside the area satisfying B. We bound $\overline{n}$ by $\mathrm{s}_1^{\max}$ and then assign each of these points maximal mass $\mathrm{p}_1^{\max}$, and subtract this from $\mathrm{m}_1^{\min}$, the previous lower bound on total mass.

By similar reasoning, we can compute $\mathrm{m}_2^{\max}$ by assuming a minimal amount of mass m is removed by conditioning, and subtracting m from $\mathrm{m}_1^{\max}$. This m is given by considering an underapproximation of the number of points falling outside the area of overlap between C_1 and B and assigning each point minimal mass as given by $\mathrm{p}_1^{\min}$. This m is given by $\max\big(\mathrm{s}_1^{\min} - n, 0\big)$.

Figure 5 demonstrates the components that affect the conditioning operation. The figure depicts the integer-valued points present in two polyhedra—one representing C_1 and the other representing B (shaded). As the set of points in C_1 satisfying B is convex, this region is precisely represented by $\langle\langle B \rangle\rangle\, C_1$. By contrast, the set of points in C_1 that satisfy $\neg B$ is not convex, and thus $\langle\langle \neg B \rangle\rangle\, C_1$ is an over-approximation. The icons beside the main image indicate which shapes correspond to which components and the numbers within the icons give the total count of points within those shapes.

Suppose the components of P_1 are as follows.

$$\begin{array}{lll} \mathrm{s}_1^{\min} = 19 & \mathrm{p}_1^{\min} = 0.01 & \mathrm{m}_1^{\min} = 0.75 \\ \mathrm{s}_1^{\max} = 20 & \mathrm{p}_1^{\max} = 0.05 & \mathrm{m}_1^{\max} = 0.9 \end{array}$$

Then $n = 4$, $\overline{n} = 18$, and $\underline{n} = 16$. We have the following for the first four numeric components of P_2.

$$\begin{array}{ll} \mathrm{s}_2^{\min} = \max(19 - 18, 0) = 1 & \mathrm{p}_2^{\min} = 0.01 \\ \mathrm{s}_2^{\max} = \min(20, 4) = 4 & \mathrm{p}_2^{\max} = 0.05 \end{array}$$

For the $\mathrm{m}_2^{\min}$ and $\mathrm{m}_2^{\max}$, we have the following for the method of calculation based on $\mathrm{p}_2^{\min/\max}$ and $\mathrm{s}_2^{\min/\max}$.

$$\mathrm{m}_2^{\min} = 0.01 \cdot 1 = 0.01 \qquad \mathrm{m}_2^{\max} = 0.05 \cdot 4 = 0.2$$

For the method of computation based on $\mathrm{m}_1^{\min/\max}$, we have

$$\begin{array}{lllll} \mathrm{m}_2^{\min} & = & 0.75 - 0.05 \cdot 18 & = & 0.04 \\ \mathrm{m}_2^{\max} & = & 1.0 - 0.01 \cdot 16 & = & 0.74 \end{array}$$

In this case, the calculation based on subtracting from total mass provides a tighter estimate for $\mathrm{m}_2^{\min}$, while the method based on multiplying $\mathrm{p}_2^{\max}$ and $\mathrm{s}_2^{\max}$ is better for $\mathrm{m}_2^{\max}$.

Lemma 16. *If $\delta \in \gamma_{\mathbb{P}}(P)$ then $\delta | B \in \gamma_{\mathbb{P}}(P \mid B)$.*

6) Scalar Product: The scalar product is straightforward, as it just scales the mass per point and total mass.

Definition 17. Given a scalar p in $[0, 1]$, we write $p \cdot P_1$ for the probabilistic polyhedron P_2 specified below.

$$\begin{array}{lll|lll} \mathrm{s}_2^{\min} & = & \mathrm{s}_1^{\min} & \mathrm{p}_2^{\min} & = & p \cdot \mathrm{p}_1^{\min} \\ \mathrm{s}_2^{\max} & = & \mathrm{s}_1^{\max} & \mathrm{p}_2^{\max} & = & p \cdot \mathrm{p}_1^{\max} \\ \mathrm{m}_2^{\min} & = & p \cdot \mathrm{m}_1^{\min} & C_2 & = & C_1 \\ \mathrm{m}_2^{\max} & = & p \cdot \mathrm{m}_1^{\max} & & & \end{array}$$

Lemma 18. *If $\delta_1 \in \gamma_{\mathbb{P}}(P_1)$ then $p \cdot \delta_1 \in \gamma_{\mathbb{P}}(p \cdot P_1)$.*

7) Normalization: If a probabilistic polyhedron P has $\mathrm{m}^{\min} = 1$ and $\mathrm{m}^{\max} = 1$ then it represents a normalized distribution. We define below an abstract counterpart to distribution normalization, capable of transforming an arbitrary probabilistic polyhedron into one containing only normalized distributions, which is important for policy evaluation.

Definition 19. We write normal(P_1) for the probabilistic polyhedron P_2 specified below.

$$\begin{array}{lll|lll} \mathrm{p}_2^{\min} & = & \mathrm{p}_1^{\min} / \mathrm{m}_1^{\max} & \mathrm{s}_2^{\min} & = & \mathrm{s}_1^{\min} \\ \mathrm{p}_2^{\max} & = & \mathrm{p}_1^{\max} / \mathrm{m}_1^{\min} & \mathrm{s}_2^{\max} & = & \mathrm{s}_1^{\max} \\ \mathrm{m}_2^{\min} & = & \mathrm{m}_2^{\max} = 1 & C_2 & = & C_1 \end{array}$$

Lemma 20. *If $\delta_1 \in \gamma_{\mathbb{P}}(P_1)$ then normal$(\delta_1) \in \gamma_{\mathbb{P}}($normal$(P_1))$.*

D. Policy Evaluation

Here we show how to implement the threshold test given as Definition 3 using probabilistic polyhedra.

Definition 21. Suppose we have some initial probabilistic polyhedron P_1. Let $P_2 = \langle\langle S \rangle\rangle\, P_1$. Let $P_3 = P_2 \upharpoonright L$. If, for all $\sigma \in \gamma_{\mathbb{C}}(C_3)$ we have $\mathrm{p}_4^{\max} \leq t$ where $P_4 = \text{normal}(P_2 \mid \bigwedge_{x \in L} x = \sigma(x))$, then we write $\mathit{tsecure}_t(S, P_1)$.

Here we condition on a boolean expression corresponding to the assumed output state and not the output state directly, as per the definition for distributions.

Now we state the main soundness theorem for abstract interpretation using probabilistic polyhedra. This theorem states that the abstract interpretation just described can be used to soundly determine whether to accept a query.

Theorem 22. *Let δ be an attacker's initial belief. If $\delta \in \gamma_{\mathbb{P}}(P_1)$ and $\mathit{tsecure}_t(S, P_1)$, then S is threshold secure for threshold t when evaluated with initial belief δ.*

VI. Powerset of Probabilistic Polyhedra

This section presents the $\mathcal{P}_n(\mathbb{P})$ domain, an extension of the $\mathbb{P}$ domain that abstractly represents a set of distributions as at most n probabilistic polyhedra, elements of $\mathbb{P}$.

Definition 23. A *probabilistic (polyhedral) set* Δ is a set of probabilistic polyhedra, or $\{P_i\}$. We write $\mathcal{P}_n(\mathbb{P})$ for the domain of probabilistic polyhedral powersets composed of no more than n probabilistic polyhedra.

The concretization function for $\mathcal{P}_n(\mathbb{P})$ is defined as follows:

$$\gamma_{\mathcal{P}_n(\mathbb{P})}(\{P_1, \ldots P_m\}) \overset{\text{def}}{=} \{\Sigma_{i=1}^m \delta_i \mid \delta_i \in \gamma_{\mathbb{P}}(P_i)\}$$

A. Abstract Semantics for $\mathcal{P}_n(\mathbb{P})$

With a few exceptions, the abstract implementations of the basic operations for the powerset domain are extensions of operations defined on the base probabilistic polyhedra domain.

Theorem 24. *For all δ, P, S, Δ,*

$$\text{if } \delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta) \text{ then } [\![S]\!]\delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\langle\langle S \rangle\rangle\, \Delta).$$

Definition 25. The *powerset simplification* transforms a set containing potentially more than n elements into one containing no more than n. The simplest approach involves repeated use of abstract plus in the base domain $\mathbb{P}$.

$$\lfloor \{P_i\}_{i=1}^m \rfloor_n \overset{\text{def}}{=} \begin{cases} \{P_i\}_{i=1}^m & \text{if } m \leq n \\ \lfloor \{P_i\}_{i=1}^{m-2} \cup \{P_{m-1} + P_m\} \rfloor_n & \text{otherwise} \end{cases}$$

Lemma 26. $\gamma_{\mathcal{P}_n(\mathbb{P})}(\{P_i\}) \subseteq \gamma_{\mathcal{P}_n(\mathbb{P})}(\lfloor \{P_i\} \rfloor_n)$.

The order in which simplification acts on individual probabilistic polyhedra makes no difference for the soundness of the approach but may have significant impact on the accuracy of the resulting abstraction.

Many of the operations and lemmas for the powerset domain are simple liftings of the corresponding operations and lemmas for single probabilistic polyhedra. For these operations (operations 1-4 given below), we simply list the definition.

1) Forget: $\mathrm{f}_y(\{P_i\}) \overset{\text{def}}{=} \{\mathrm{f}_y(P_i)\}$

2) Assignment: $\{P_i\}\,[x \to E] \overset{\text{def}}{=} \{P_i\,[x \to E]\}$

3) Scalar product: $p \cdot \{P_i\} \overset{\text{def}}{=} \{p \cdot P_i\}$

4) Product: The product operation is only required for the special uniform statement and only applies to the product of a probabilistic set with a single probabilistic polyhedron. $\{P_i\} \times P' \overset{\text{def}}{=} \{P_i \times P'\}$

5) Plus: The abstract plus operation involves simplifying the combined contributions from two sets into one bounded set: $\Delta_i + \Delta'_i \overset{\text{def}}{=} \lfloor \Delta_i \cup \Delta'_i \rfloor_n$

Lemma 27. *If $\delta_1 \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_1)$ and $\delta_2 \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_2)$ then $\delta_1 + \delta_2 \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_1 + \Delta_2)$.*

6) Conditioning: To take best advantage of the richer abstraction, we need to better approximate the state cover of boolean expressions. Instead of approximating them with a single polyhedron, we can also approximate them with a set of polyhedra, interpreted as disjuncts. The *disjunctive abstract* operation formalizes this operation.

Definition 28. The *disjunctive abstract* of a boolean expression B, denoted $a^m(B)$, is a set of polyhedra $\mathcal{C} = \{C_i\}$ with $|\mathcal{C}| \leq m$, and $\{\sigma \mid \sigma \models B\} \subseteq \bigcup_i \gamma_{\mathbb{C}}(C_i)$.

Though not part of the definition, we are naturally interested in the smallest abstractions for boolean expressions.

Definition 29. *Conditioning on polyhedron*. Consider a probabilistic polyhedron P_1 and a polyhedron C. Let $n = \#(C_1 \sqcap_{\mathbb{C}} C)$ and $\overline{n} = \#(C_1) - n$. Then $P_1 \mid C$ is the probabilistic polyhedron P_2 defined as follows.

$$\begin{array}{ll} C_2 & = C_1 \sqcap_{\mathbb{C}} C \\ \mathrm{p}_2^{\min} = \mathrm{p}_1^{\min} & \mathrm{s}_2^{\min} = \max(\mathrm{s}_1^{\min} - \overline{n}, 0) \\ \mathrm{p}_2^{\max} = \mathrm{p}_1^{\max} & \mathrm{s}_2^{\max} = \min(\mathrm{s}_1^{\max}, n) \\ \mathrm{m}_2^{\min} & = \max\big(\mathrm{p}_2^{\min} \cdot \mathrm{s}_2^{\min},\ \mathrm{m}_1^{\min} - \mathrm{p}_1^{\max} \cdot \min\big(\mathrm{s}_1^{\max}, \overline{n}\big)\big) \\ \mathrm{m}_2^{\max} & = \min\big(\mathrm{p}_2^{\max} \cdot \mathrm{s}_2^{\max},\ \mathrm{m}_1^{\max} - \mathrm{p}_1^{\min} \cdot \max\big(\mathrm{s}_1^{\min} - n, 0\big)\big) \end{array}$$

Similarly, we can condition on a set of disjuncts. If $\mathcal{C} = \{C_i\}$ then $P_1 \mid \mathcal{C} \overset{\text{def}}{=} \{P_1 \mid C_i\}$, a member of $\mathcal{P}_n(\mathbb{P})$.

Finally, given a set $\{P_i\}$ and a boolean expression B, we can define a set of probabilistic polyhedra conditioned on B:

$$\{P_i\} \mid B \overset{\text{def}}{=} \Big\lfloor \bigcup_i \big(P_i \mid a^m(B)\big) \Big\rfloor_n$$

Note that the parameter m bounding the number of disjuncts in the abstraction of B need not be related to n.

Lemma 30. *If $\delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta)$ then $\delta | B \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta \mid B)$.*

7) Normalization: Since in the $\mathbb{P}$ domain, the over(under) approximation of the total mass is not contained in any single probabilistic polyhedron, the normalization must scale each component of a set by the overall total. The minimum (maximum) mass of a probabilistic polyhedra set $\Delta = \{P_i\}$ is defined as follows.

$$\mathrm{M}^{\min}(\{P_i\}) \overset{\text{def}}{=} \textstyle\sum_i \mathrm{m}_i^{\min} \quad \Big| \quad \mathrm{M}^{\max}(\{P_i\}) \overset{\text{def}}{=} \textstyle\sum_i \mathrm{m}_i^{\max}$$

Definition 31. The scaling of a probabilistic polyhedra P_1 by minimal total mass $\underline{m}$ and maximal total mass $\overline{m}$, written $\mathrm{normal}(P)(\underline{m}, \overline{m})$ is the probabilistic polyhedron P_2 satisfying the conditions below.

$$\begin{array}{rcl|rcl} \mathrm{p}_2^{\min} & = & \mathrm{p}_1^{\min}/\overline{m} & \mathrm{s}_2^{\min} & = & \mathrm{s}_1^{\min} \\ \mathrm{p}_2^{\max} & = & \mathrm{p}_1^{\max}/\underline{m} & \mathrm{s}_2^{\max} & = & \mathrm{s}_1^{\max} \\ \mathrm{m}_2^{\min} & = & \mathrm{m}_1^{\min}/\overline{m} & C_2 & = & C_1 \\ \mathrm{m}_2^{\max} & = & \mathrm{m}_1^{\max}/\underline{m} & & & \end{array}$$

Finally, the normalization of a set of probabilistic polyhedra can be defined.

$$\mathrm{normal}(\{P_i\}) \overset{\mathrm{def}}{=} \left\{\mathrm{normal}(P_i)(\mathrm{M}^{\min}(\{P_i\}), \mathrm{M}^{\max}(\{P_i\}))\right\}$$

Lemma 32. *If $\delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta)$ then normal$(\delta) \in \gamma_{\mathcal{P}_n(\mathbb{P})}($normal$(\Delta))$.*

Definition 33. We write $\overline{P}(\sigma)$ to mean the maximum probability of a state σ according to a probabilistic polyhedron P, and $\overline{\{P_i\}}(\sigma)$ to mean the maximum probability of a state σ according to a probabilistic polyhedron set $\{P_i\}$. The definitions are as follows.

$$\overline{P}(\sigma) = \begin{cases} \mathrm{p}^{\max} & \text{if } \sigma \in \gamma_{\mathbb{C}}(C) \\ 0 & \text{otherwise} \end{cases} \quad \Big| \quad \overline{\{P_i\}}(\sigma) = \textstyle\sum_i \overline{P_i}(\sigma)$$

Lemma 34. $\overline{\{P_i\}}(\sigma) \leq \overline{(\Sigma_i P_i)}(\sigma)$

Determining the maximal probabilty of any state represented by a single probabilistic polyhedron is a simple as checking the $\mathrm{p}^{\max}$ value in the normalized version of the probabilistic polyhedron. In the domain of probabilistic polyhedron sets, however, the situation is more complex, as polyhedra may overlap and thus a state's probability could involve multiple probabilistic polyhedra.

A complex approach would produce a disjoint set of probabilistic polyhedra, but a simple estimate can be computed by abstractly adding all the probabilistic polyhedra in the set, and finding the $\mathrm{p}^{\max}$ value of the result.

$$\max_{\sigma} \overline{\{P_i\}}(\sigma) \leq \mathrm{p}_a^{\max} \qquad \text{where } P_a = \Sigma_i P_i$$

This is the approach we adopt in the implementation.

B. Policy Evaluation

We begin by defining concretization for sets of polyhedra.

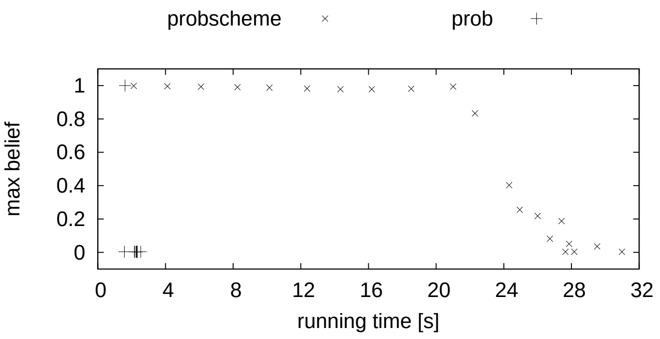
Definition 35. A set of convex polyhedra $\{C_i\}$ represents all states that are in at least one of the polyhedra.

$$\gamma_{\mathcal{P}(\mathbb{C})}(\{C_i\}) \overset{\mathrm{def}}{=} \bigcup_i \gamma_{\mathbb{C}}(C_i)$$

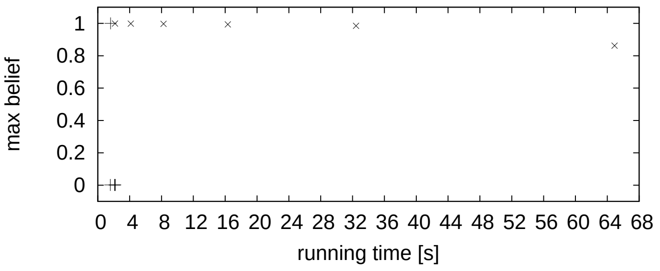
We now define *tsecure* for $\mathcal{P}_n(\mathbb{P})$:

Definition 36. Suppose we have some initial probabilistic polyhedron set Δ_1. Let $\Delta_2 = \langle\langle S \rangle\rangle \Delta_1$. Let $\Delta' = \{P'_i\} = \Delta_2 \upharpoonright L$. If, for all $\sigma \in \gamma_{\mathcal{P}(\mathbb{C})}(\{C'_i\})$ we have $\overline{\Delta_3}(\sigma) \leq t$ where $\Delta_3 = \mathrm{normal}(\Delta_2 \mid \bigwedge_{x \in L} x = \sigma(x))$, then we write $\mathit{tsecure}_t(S, \Delta_1)$.

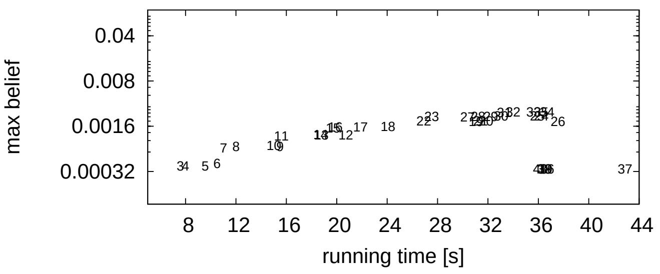
Below we state the main soundness theorem for abstract interpretation using probabilistic polyhedron sets. This theorem



(a) birthday query (Example 1)



(b) birthday query (Example 1), larger state space



(c) special year query (Example 2)

Fig. 6. Query evaluation comparison

states that the abstract interpretation just described can be used to soundly determine whether to accept a query.

Theorem 37. *Let δ be an attacker's initial belief. If $\delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta)$ and tsecure$_t(S, \Delta)$, then S is threshold secure for threshold t when evaluated with initial belief δ.*

VII. Implementation and experiments

We have implemented an interpreter for the core language based on the probabilistic polyhedra powerset domain. The base manipulations of polyhedra are done using the Parma Polyhedra Library [18]. Size calculations are done using the LattE lattice point counter [19]. LattE is also used for the integer linear programming problem involved in the abstract forget operation. The interpreter itself is written in OCaml.

To compare the abstract interpretation approach to an existing, sampling-based approach, we used our implementation to vet the query given in Example 1, Section II, and an implementation based on Probabilistic Scheme [12], which is capable of sound probability estimation after partial enumeration.

Figure 6(a) illustrates the result when run on 2.5 GHz Intel Core 2 Duo MacBook Pro, using OS X v10.5.8 with 4 GB of RAM. Each $\times$ plots Probscheme's maximum probability value (the y axis)—that is, the probability it assigns to the most likely secret state—when given a varying amount of time for sampling (the x axis). We can see the precision improves steadily until it reaches the exact value of 1/259 at around 30 seconds. Each $+$ plots our implementation's maximum probability value when given an increasing number of probabilistic polyhedra; with a polyhedral bound of 2 (or more), we obtain the exact value in less than 3 seconds. The advantage of our approach is more evident in Figure 6(b) where we use the same program but allow $byear$ to span 1910 to 2010 rather than 1956 to 1992. In this case ProbScheme makes little progress even after a minute, and eventually runs out of memory. Our approach, however, is unaffected by this larger state space and produces the exact maximum belief after around 3 seconds when using only 2 probabilistic polyhedra.

Figure 6(c) shows the result of our implementation assessing the special query (Example 2) with initial belief matching that following the first birthday query. Each plotted point is the number of polyhedra allowed. The result demonstrates that more complex queries, specifically ones with many disjunctions in their conditionals, not only slow our approach down, but also reduce the precision of the maximum probability value. The example requires 36 polyhedra for exact calculations though as little as 3 produce probabilities near exact. The precision worsens as the number of polyhedra is increased until 36 are allowed. We conjecture this is a side-effect of an overly simple means of deciding which polyhedra to merge when performing abstract simplification and plan to investigate this closely in future work.

VIII. Related Work

Prior work aimed at controlling access to users' private data has focused on access control policies. For example, Persona [6] users can store personal data on distributed storage servers that use attribute-based encryption; only those parties that have the attribute keys for particular data items may see them. Our approach relaxes the access control model to offer more fine-grained information release policies by directly modeling an attacker's belief.

However, explicit belief modeling can be both a strength and a limitation. As shown in this paper, explicit modeling is a strength because it supports intuitive policies, i.e., those designed to control the likelihood of correctly guessing a secret value, and these policies can be enforced without an a priori limit on the number or form of queries. On the other hand, a secret's owner must produce a reasonably accurate estimate of the attacker's knowledge for the technique to be effective; gaps in that estimate (e.g., due to collusion) could lead to leaks. This problem was the motivation for work on differential privacy [22], which aims to bound the increase in attacker knowledge without having to model knowledge explicitly. Unfortunately, the secret owner is still left with deciding what this bound should be, and ultimately this decision must be made with some assumption about the attacker's knowledge; this fact is shown explicitly by Rastogi et al [23]. Another problem with differential privacy is that because knowledge is not modeled explicitly, in general each query's knowledge release is assumed independent, and once the query limit is reached, as determined by the bound, *no further queries are permitted.* In contrast, our approach permits an unlimited number of queries as long as knowledge gained from them never exceeds the threshold. Finally, differential privacy applies to queries over databases of structurally-similar records, so its application to individual personal data, as is our interest, is not clear.

Several quantitative approaches to track program information leakage have been proposed by past approaches. McCamant and Ernst [24] model the program as a covert channel whose output is a *lossy* encoding over the program's input symbols; Backes et al. [25] model information leakage in a program as (the size of) an equivalence relationship between the set of input and output symbols. However, the measures for information proposed by past approaches are coarse-grained. For example, [24] use of channel capacity as a metric for information leakage. However, channel capacity is determined by the worst case probability distribution over input symbols that maximizes leakage. While such worst case estimates may offer bounds on leakage, we believe that our approach offers a more fine-grained (per query), semantically-richer means to measure release.

Köpf and Rybalchenko [26] present an analysis that uses under- and over-approximation to obtain bounds on information flow. However this analysis is focused on tracking entropy and is incapable of reasoning about belief distributions. Mu and Clark [27] present a similar analysis that uses over-approximation only. This work is also focused solely on entropy measures and cannot represent beliefs. Monniaux [28] gives an abstract interpretation for probabilistic programs based on over-approximating probabilities. That work contains no treatment of distribution conditioning and normalization, which are crucial for belief-based information flow analysis. Under-approximations are needed to soundly handle conditioning and normalization and this use of under-approximations is unique to our approach.

IX. Conclusion

This paper has explored the idea of *knowledge-based security policies*: given a query over some secret data, that query should only be answered if doing so will not increase the querier's knowledge above a fixed threshold. We enforce knowledge-based policies by explicitly tracking a model of a querier's belief about secret data, represented as a probability distribution, and we deny any query that could increase knowledge above a the threshold. Our denial criterion is independent of the actual secret, so denial does not leak information. We implement query analysis and belief tracking via abstract interpretation using a novel domain we developed called *powersets of probabilistic polyhedra*. Compared to typical approaches to implementing belief revision, our implementation using this domain is more efficient and scales better.

References

[1] "Facebook developers," http://developers.facebook.com/, 2011, see the policy and docs/guides/canvas directories for privacy information.

[2] S. Guha, B. Cheng, and P. Francis, "Challenges in measuring online advertising systems," in *10th ACM SIGCOMM conference on Internet measurement conference*, Nov. 2010.

[3] "Statement of rights and responsibilities," http://www.facebook.com/terms.php, Oct. 2010.

[4] D. Worthington, "Myspace user data for sale," *PC World on-line*, Mar. 2010, http://www.pcworld.com/article/191716/myspace_user_data_for_sale.html.

[5] S.-W. Seong, J. Seo, M. Nasielski, D. Sengupta, S. Hangal, S. K. Teh, R. Chu, B. Dodson, and M. S. Lam, "PrPl: a decentralized social networking infrastructure," in *1st International Workshop on Mobile Cloud Computing and Services: Social Networks and Beyond*, Jun. 2010, invited Paper.

[6] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin, "Persona: an online social network with user-defined privacy," in *SIGCOMM*, 2009.

[7] P. Golle, "Revisiting the uniqueness of simple demographics in the us population," in *WPES*, 2006.

[8] "Facebook demographics and statistics report 2010," http://www.istrategylabs.com/2010/01/facebook-demographics-and-statistics-report-2010-145-growth-in-1-year/, 2010.

[9] M. R. Clarkson, A. C. Myers, and F. B. Schneider, "Quantifying information flow with beliefs," *J. Comput. Secur.*, vol. 17, no. 5, 2009.

[10] G. Smith, "On the foundations of quantitative information flow," in *FOSSACS*, 2009.

[11] A. Pfeffer, "The design and implementation of IBAL: A general-purpose probabilistic language," in *Statistical Relational Learning*, L. Getoor and B. Taskar, Eds. MIT Press, 2007.

[12] A. Radul, "Report on the probabilistic language Scheme," in *DLS*, 2007.

[13] S. Park, F. Pfenning, and S. Thrun, "A probabilistic language based on sampling functions," *TOPLAS*, vol. 31, pp. 4:1–4:46, 2008.

[14] N. D. Goodman, V. K. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum, "Church: a language for generative models," in *UAI*, 2008.

[15] O. Kiselyov and C.-C. Shan, "Embedded probabilistic programming," in *DSL*, 2009.

[16] P. Cousot and N. Halbwachs, "Automatic discovery of linear restraints among variables of a program," in *POPL*, 1978.

[17] D. Monniaux, "Abstract interpretation of probabilistic semantics," in *Seventh International Static Analysis Symposium (SAS'00)*, no. 1824, 200, pp. 322–339.

[18] "PPL: The Parma polyhedral library," http://www.cs.unipr.it/ppl/, 2011.

[19] J. Loera, D. Haws, R. Hemmecke, P. Huggins, J. Tauzer, and R. Yoshida, "Latte," http://www.math.ucdavis.edu/latte/, 2008.

[20] R. Bagnara, P. M. Hill, and E. Zaffanella, "Widening operators for powerset domains," *Int. J. Softw. Tools Technol. Transf.*, vol. 8, pp. 449–466, 2006.

[21] C. Popeea and W. Chin, "Inferring disjunctive postconditions," in *In ASIAN CS Conference*, 2006.

[22] C. Dwork, "Differential privacy," in *ICALP*, 2006.

[23] V. Rastogi, M. Hay, G. Miklau, and D. Suciu, "Relationship privacy: output perturbation for queries with joins," in *PODS*, 2009.

[24] S. McCamant and M. D. Ernst, "Quantitative information flow as network flow capacity," in *PLDI 2008, Proceedings of the ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation*, Tucson, AZ, USA, June 9–11, 2008, pp. 193–205.

[25] M. Backes, B. Köpf, and A. Rybalchenko, "Automatic discovery and quantification of information leaks," in *Security and Privacy*, 2009.

[26] B. Köpf and A. Rybalchenko, "Approximation and randomization for quantitative information-flow analysis," in *Computer Security Foundations Symposium (CSF), 2010 23rd IEEE*, Jul. 2010, pp. 3 –14.

[27] C. Mu and D. Clark, "An interval-based abstraction for quantifying information flow," *Electron. Notes Theor. Comput. Sci.*, vol. 253, pp. 119–141, November 2009.

[28] D. Monniaux, "Analyse de programmes probabilistes par interprétation abstraite," Thèse de doctorat, Université Paris IX Dauphine, 2001, résumé étendu en fran cais. Contents in English.

Appendix
Concrete probabilistic semantics

Here we briefly explain the concrete probabilistic semantics given in Figure 3. More details can be found in Clarkson et al. [9].

The semantics of skip is straightforward: it is the identity on distributions. The semantics of sequences S_1 ; S_2 is also straightforward: the distribution that results from executing S_1 with δ is given as input to S_2 to produce the result.

The semantics of assignment is $\delta\,[x \to E]$, which is defined as follows:

$$\delta\,[x \to E] \stackrel{\text{def}}{=} \lambda\sigma.\ \sum_{\tau\ \mid\ \tau[x \to [\![E]\!]\tau] = \sigma} \delta(\tau)$$

In words, the result of substituting an expression E for x is a distribution where state σ is given a probability that is the sum of the probabilities of all states τ that are equal to σ when x is mapped to the distribution on E in τ. For implementation purposes, it will be useful to consider separately the case where assignment is invertible.

When $x \to E$ is an invertible transformation, the formula for assignment can be simplified to the following, where $x \to E'$ is the inverse of $x \to E$.

$$\delta\,[x \to E] \stackrel{\text{def}}{=} \lambda\sigma.\ \delta(\sigma\,[x \to [\![E']\!]\sigma])$$

When $x \to E$ is not invertible, the original definition is equivalent to a projection followed by an assignment. Let $V' = \mathit{domain}(\delta) - \{x\}$ and let $\delta' = \delta \upharpoonright V'$. Then we have the following for a non-invertible assignment.

$$\delta\,[x \to E] \stackrel{\text{def}}{=} \lambda\sigma.\ \textbf{if}\ \sigma(x) = [\![E]\!]\sigma\ \textbf{then}\ \delta'(\sigma \upharpoonright V')\ \textbf{else}\ 0$$

In the appendix, we show that this definition by cases is equivalent to the original definition (Theorem **??**).

The semantics for conditionals makes use of two operators on distributions which we now define. First, given distributions δ_1 and δ_2 we define the *distribution sum* as follows:

$$\delta_1 + \delta_2 \stackrel{\text{def}}{=} \lambda\sigma.\ \delta_1(\sigma) + \delta_2(\sigma)$$

In words, the probability mass for a given state σ of the summed distribution is just the sum of the masses from the input distributions for σ. Second, given a distribution δ and a boolean expression B, we define the *distribution conditioned on B* to be

$$\delta|B \stackrel{\text{def}}{=} \lambda\sigma.\ \textbf{if}\ [\![B]\!]\sigma\ \textbf{then}\ \delta(\sigma)\ \textbf{else}\ 0$$

In short, the resulting distribution retains only the probability mass from δ for states σ in which B holds.

With these two operators, the semantics of conditionals can be stated simply: the resulting distribution is the sum of the distributions of the two branches, where the first branch's distribution is conditioned on B being true, while the second branch's distribution is conditioned on B being false.

The semantics for probabilistic conditionals like that of conditionals but makes use of *distribution scaling*, which is defined as follows: given δ and some scalar p in $[0,1]$, we have

$$p \cdot \delta \overset{\text{def}}{=} \lambda\sigma.\, p \cdot \delta(\sigma)$$

In short, the probability ascribed to each state is just the probability ascribed to that state by δ but multiplied by p. For probabilistic conditionals, we sum the distributions of the two branches, scaling them according to the odds q and $1-q$.

The semantics of a single iteration of a while loop is essentially that of if B then S else skip and the semantics of the entire loop is the fixpoint of a function that composes the distributions produced by each iteration. That such a fixpoint exists is proved by Clarkson et al. [9].

Finally, the semantics of uniform x n_1 n_2, introduced in Section V is given as

$$[\![\mathsf{uniform}\ x\ n_1\ n_2]\!]\delta \;=\; (\delta \restriction V - \{x\}) \times \delta'$$

Where V is the set of variables of δ, and δ' is defined as follows.

$$\delta' = \lambda\sigma.\ \textbf{if}\ n_1 \leq \sigma(x) \leq n_2\ \textbf{then}\ \frac{1}{n_2 - n_1 + 1}\ \textbf{else}\ 0$$

Appendix

Alternative (flawed) threshold security policy

As an alternative to Definition 3, suppose we used the following instead:

$$\forall \sigma_L \in \mathit{support}(\delta' \restriction L).\, (\mathrm{normal}(\delta' | \sigma_L \restriction H))(\sigma_H) \leq t$$

Here is an example that illustrates why this definition is not safe, as it could underestimate the information a querier can learn.

Suppose Bob's threshold for his birth year byear is $t = 0.05$. He models a social networking site X as believing his age is more likely between 20 and 40 than between 40 and 60, e.g., $1971 \leq \mathit{byear} < 1991$ with probability 0.6 (thus, 0.03 per possibility) and $1951 \leq \mathit{byear} < 1971$ with probability 0.4 (thus, 0.02 per possibility). If user Bob was born in 1965, then X's believes his is actual birth year not as likely a more recent year, say 1975; in any case X does not currently believe any possibility above Bob's threshold. Now suppose X submits program S that determines whether Bob's birth year is even. The revised belief will include only even (when $\mathit{output} = \mathsf{True}$) or odd (when $\mathit{output} = \mathsf{False}$) birthdays, increasing the likelihood of years in the range $[1971, 1991)$ to be 0.06 per point, and the likelihood of years in the range $[1951, 1971)$ to be 0.04 per point. Bob's birthday is 1965, and its probability 0.04 is less than t, so according to the flawed definition the agent would respond to this query. But if this query result is returned, X will see that there are ten possibilities of birth year that are above Bob's threshold. X can deduce that none of these possibilities is Bob's actual birth year, or else the query would have been rejected. Excluding these possibilities, he knows that Bob's birth year is one of ten possibilities between 1951 and 1971 ascribing to each a probability 0.1 which exceeds Bob's threshold of 0.05.