Dynamic Enforcement of Knowledge-based Security Policies using Probabilistic Abstract Interpretation

Piotr Mardziel[†], Stephen Magill, Michael Hicks[†], Mudhakar Srivatsa[⋆]

[†] University of Maryland, College Park
[⋆] IBM T.J. Watson Research Laboratory

January 24, 2013

Abstract

This paper explores the idea of *knowledge-based security policies*, which are used to decide whether to answer queries over secret data based on an estimation of the querier's (possibly increased) knowledge given the results. 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, and denying any query that could increase knowledge above a given threshold. We implement query analysis and belief tracking via abstract interpretation, which allows us to trade off precision and performance through the use of abstraction. We have developed an approach to augment standard abstract domains to include probabilities, and thus define distributions. We focus on developing *probabilistic polyhedra* in particular, to support numeric programs. While probabilistic abstract interpretation has been considered before, our domain is the first whose design supports sound conditioning, which is required to ensure that estimates of a querier's knowledge are accurate. Experiments with our implementation show that several useful queries can be handled efficiently, particularly compared to exact (i.e., sound) inference involving sampling. We also show that, for our benchmarks, restricting constraints to *octagons* or *intervals*, rather than full polyhedra, can dramatically improve performance while incurring little to no loss in precision.

1 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 [5]) to implement services like birthday announcements and horoscopes, while Facebook selects ads based on age, gender, and even sexual preference [27]. 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 [2]. MySpace, another social network site, has begun to sell its users' data [56].

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 [50, 8]. The question is: which requests are safe? While deferring to user-defined access control policies seems an obvious approach, such policies are unnecessarily restrictive when the goal is to maximize the customized personal experience. To see why, consider two example applications: a horoscope or "happy birthday" application that operates on birth month and day, and a music recommendation algorithm that considers birth year (age). 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, birth date, and gender are sufficient information to uniquely identify 87% of Americans in the 1990 U.S. census [55] and 63% in the 2000 census [25]. So the user may be perfectly happy to reveal any one of these bits of information as long as a querier gains no better than a $1/n$ chance to guess the entire group, for some parameter n.

This paper explores the design and implementation for enforcing what we call *knowledge-based security policies*. In our model, a user U's agent responds to queries involving secret data. For each querying principal Q, the agent maintains a probability distribution over U's secret data, representing Q's *belief* of the data's likely values. For example, to mediate queries from a social networking site X, user U's agent may model X's otherwise uninformed knowledge of U'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 [1]. Each querier Q is also assigned a knowledge-based policy, expressed as a set of thresholds, each applying to a different group of (potentially overlapping) data. For example, U's policy for X might be a threshold of $1/100$ for the entire tuple $(birthdate, zipcode, gender)$, and $1/5$ for just birth date. U's agent refuses any queries that it determines could increase Q's ability to guess a secret above the assigned threshold. If deemed safe, U's agent returns the query's (exact) result and updates Q's modeled belief appropriately.

Throughout the paper we use users' personal information protection when interacting with services like Facebook (or its advertisers) as a running example, but knowledge-based security policies have other applications as well. For example, they can be used to decide whether a principal should participate in a *secure multiparty computation* involving multiple principals each with its own secrets, such as their current location or available resources. We have explored this application in some detail in recent work [34]. Knowledge-based policies could also be used to protect against browser

fingerprinting, which aims to uniquely identify individuals based on environmental indicators visible to Javascript programs [10]. Users could set a threshold policy over the tuple of the most sensitive indicators, and prevent the execution of the Javascript program (or execute only a modified version) if the threshold is exceeded. Another example would be application to privacy-preserving smart metering [49]. Here, the proposal is that rather than report fine-grained power usage information back to the power company, which could compromise privacy, the pricing algorithm is run locally on the meter, with only the final charge returned. Our work could be combined with this work to ensure that knowledge that can be inferred from the output indeed preserves privacy sufficiently. We elaborate on these and other examples in Section 8.

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. [14] (reviewed in Section 3), which works out the theoretical basis for (2). The main contributions of this paper, therefore, in addition to the idea of knowledge-based policies, are our solutions to (1) and (3).

Given a means to revise querier beliefs based on prior answers, it seems obvious how to check that a query does not reveal too much: 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. Unfortunately, with this approach the decision to accept or reject 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 there exists *any* possible secret value that could induce an output whereby the revised belief would exceed the threshold. This idea is described in detail in Section 4.

The foundational elements of our approach, belief tracking and revision, can be implemented using languages for probabilistic computation. However, existing languages of this variety—IBAL [45], Church [26], Fun [11], and several other systems [47, 44, 30, 38]— are problematic because they are either unsound or too inefficient. Systems that use *exact* inference have no flexibility of approximation to efficiently handle large or complex state spaces. Systems that use approximate inference are more efficient, but the nature of the approximation is under-specified, and thus there is no guarantee of soundness.

We have developed an implementation based on abstract interpretation [17] that is capable of approximate inference, but is sound relative to our policies. In particular, our implementation ensures that, despite the use of abstraction, the probabilities we ascribe to the querier's belief are never less than the true probabilities. At the center of our implementation is a new abstract domain we call a *probabilistic polyhedra*, described in Section 5, which extends the standard convex polyhedron abstract domain [19] with measures of probability. We represent beliefs as a set of probabilistic polyhedra (as developed in Section 6). Our approach can easily be adapted to any abstract domain that supports certain common operations; our implementation includes support for intervals [16] and octagons [39].

While some prior work has explored probabilistic abstract interpretation [40], 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 probabilities in the querier's belief, whereas prior work deals only with over-approximation. We have developed an implementation of our approach based on Parma [6], an abstract interpretation library, and LattE [21], a tool for counting the integer points contained in a polygon. We discuss the implementation in Section 7 along with some experimental measurements of its performance. We find that the varying the number of polyhedra permitted for performing belief tracking constitutes a useful precision/performance tradeoff, and that reasonable performance can be had while maintaining good precision.

Knowledge-based policies aim to ensure that an attacker's knowledge of a secret does not increase much when learning the result of a query. Much prior work aims to enforce similar properties by tracking information leakage quantitatively [37, 53, 7, 32, 48]. Our approach is more precise (but also more resource-intensive) because it maintains an on-line model of adversary knowledge. An alternative to knowledge-based privacy is *differential privacy* [24] (DP), which requires that a query over a database of individuals' records produces roughly the same answer whether a particular individual's data is in the database or not—the possible knowledge of the querier, and the impact of the query's result on it, need not be directly considered. As such, DP avoids the danger of mismodeling a querier's knowledge and as a result inappropriately releasing information. DP also need not maintain a per-querier belief representation for answering subsequent queries. However, DP applies once an individual has released his personal data to a trusted third party's database, a release we are motivated to avoid. Moreover, applying DP to queries over an individual's data, rather than a population, introduces so much noise that the results are often useless. We discuss these issues along with other related work in Section 9.

A preliminary version of this paper was published at CSF'11 [35]. The present version expands the formal description of probabilistic polyhedra and details of their implementation, and expands our experimental evaluation. We have refined some definitions to improve performance (e.g., the forget operation in Section 5.3.1) and implemented two new abstract domains (Section 5.5). We have also expanded our benchmark suite to include several additional programs (Section 7.1 and Appendix A). We discuss the performance/precision tradeoffs across the different domains/benchmarks (Section 7.3). Proofs, omitted for space reasons, appear in a companion technical report [36].

2 Overview

The next section presents a technical overview of the paper through a running example. Full details of the approach are presented in Sections 3–7.

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 ($m = 1$):

$$C = 0 \leq bday < 365,\ 1956 \leq byear < 1993$$

An initial belief such as this one could be derived from several sources. For example, Facebook publishes demographics of its users [1], and similar sorts of personal demographics could be drawn from census data, surveys, employee records, etc. Demographics are also relevant to other applications; e.g., initial beliefs for hiding web browser footprints can be based on browser surveys, likely movie preferences can be found from IMDB, and so on. In general, we observe that understanding the capabilities and knowledge of a potential adversary is necessary no matter the kind of security policy used; often this determination is implicit, rather than explicit as in our approach. We defer further discussion on this topic to Sections 9 and 8.

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.

$$\begin{array}{l} today\ :=\ 260; \\ \mathsf{if}\ bday \geq today \wedge bday < (today + 7)\ \mathsf{then} \\ \quad output\ :=\ \mathsf{True}; \end{array}$$

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 (with probability masses $m_1 = \frac{260}{358} \approx 0.726, m_2 = \frac{98}{358} \approx$

0.274):

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

Ignoring $byear$, there are 358 possible values for $bday$ and each is equally likely. Thus the probability of any one is $1/358 \approx 0.0028 \leq t_d = 0.2$. More explicitly, the same quantity can be computed using the definition of conditional probability: $Pr(A|B) = Pr(A \wedge B)/Pr(B)$ where event A refers to $bday$ having one particular value (e.g., *not in the next week*) and event B refers to the query returning False. Thus $1/358 = \frac{1}{365}/\frac{358}{365}$.

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 to $bday \geq 268$, meaning there is still only a $1/357 \approx 0.0028$ 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 $t_d = 0.2$ (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 4 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 $bday = 267$. This makes the policy decision *simulatable* [29]: given knowledge of the current belief model and the belief-tracking implementation being used, the querier can determine whether his query will be rejected on his own.

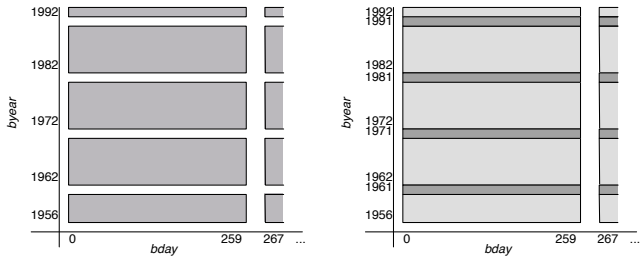
Full probabilistic polyhedra. Now suppose, having run the first query and rejected the second, the user agent receives the following program from X.

Example 2.

$age := 2011 - byear;$
if $age = 20 \vee age = 30 \vee ... \vee age = 60$ 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



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

Figure 1: Example 2: most precise revised beliefs

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 $\left(\frac{9}{10} * \frac{1}{37*358}\right) / \left(\frac{9}{10} * \frac{33}{37}\right) = 1/(33 * 358)$; when $output =$ True, the points in the dark bands are the most likely, with probability $\left(\frac{1}{37*358}\right) / \left(\frac{9}{10} * \frac{4}{37} + \frac{1}{10}\right) = 10/(73 * 358)$ (this and the previous calculation are also just instantiations of the definition of conditional probability). 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 5 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 the seven-tuple representation, we could choose to collapse the sets of polyhedra 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 (explained below):

$$\begin{aligned}
&\mathrm{p}_1^{\min} = \mathrm{p}_1^{\max} = 9/135050 = \tfrac{1}{37*365} * \tfrac{9}{10} \\
&\mathrm{s}_1^{\min} = \mathrm{s}_1^{\max} = 8580 = 260 * 33 \\
&\mathrm{m}_1^{\min} = \mathrm{m}_1^{\max} = 7722/13505 = \mathrm{p}_1^{\min} * \mathrm{s}_1^{\min}
\end{aligned}$$

Notice that $\mathrm{s}_1^{\min} = \mathrm{s}_1^{\max} = 8580 < \#(C_1) = 9620$, illustrating that the "bounding box" of the polyhedron covers more area than is strictly necessary. The other thing to notice is that the probabilities and probability mass are not normalized; we do this for efficiency and precision considerations made clear in Section 5. Non-normalized probabilities arise during conditioning—instead of performing $Pr(A|B) = Pr(A \wedge B)/Pr(B)$ we instead only perform the $Pr(A \wedge B)$ component and delay normalization until making a security decision. In this representation, after Example 1 returns False the probability of each $bday, byear$ combination in the polyhedron would be stored as $\frac{1}{37*365}$ instead of the full conditional probability $\frac{1}{37*365}/\frac{358}{365}$. After the next query, Example 2, returns False, we would store $\frac{1}{37*365} * \frac{9}{10}$, which is given above. This probability corresponds to the probability $Pr(A \wedge B \wedge C)$, where A is the event of having a particular non-special $bday, byear$ not within the next week, B corresponds to the event that the first query returns False and C corresponds to the event that the second query returns False. To compute probabilities $Pr(A|B \wedge C)$, we normalize by dividing $Pr(A \wedge B \wedge C)$ by $Pr(B \wedge C)$, which we can conveniently recover from the total mass components of the probabilistic polyhedron.

Now 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:

$$\begin{aligned}
&\mathrm{p}_1^{\min} = 1/135050 = \tfrac{1}{37*365} * \tfrac{1}{10} \quad \mathrm{p}_1^{\max} = 10/135050 = \tfrac{1}{37*365} \\
&\mathrm{s}_1^{\min} = 9620 = 260 * 37 \qquad\qquad\quad \mathrm{s}_1^{\max} = 9620 = 260 * 37 \\
&\mathrm{m}_1^{\min} = 26/185 \qquad\qquad\qquad\quad \mathrm{m}_1^{\max} = 26/185
\end{aligned} \tag{1}$$

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). An astute reader might notice that here $\mathrm{m}_1^{\min} \neq \mathrm{p}_1^{\min} * \mathrm{s}_1^{\min}$ and $\mathrm{m}_1^{\max} \neq \mathrm{p}_1^{\max} * \mathrm{s}_1^{\max}$. The benefit of these seemingly redundant total mass quantities in the representation is that they can sometimes be computed precisely. In this case $\mathrm{m}_1^{\min} = \mathrm{m}_1^{\max} = \frac{4}{37} * \frac{260}{365} + \frac{1}{10} * \frac{33}{37} * \frac{260}{365}$. This quantity is the probability of the query returning True while having a special year (first term) plus not having a special year (second term).

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 $output =$ 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 conditional, or *normalized*, probability of these points, we divide by the minimum possible total mass, as given by the lower

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

Figure 2: Core language syntax

bounds in our abstraction. In our example, this results in $\mathrm{p}_1^{\max}/(\mathrm{m}_1^{\min} + \mathrm{m}_2^{\min}) = \frac{10}{135050} / \left(\frac{26}{185} + \frac{49}{925}\right) \approx 0.0004 \leq t_d = 0.05$.

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. The choice of the number of probabilistic polyhedra to use impacts both precision and performance, so choosing the right number is a challenge.

Another precision/performance tradeoff coincides with the choice of the *kind* of polyhedron used to represent constraints. If we restrict constraints to always form *intervals* [16] (a.k.a. boxes) or *octagons* [39] we can speed up performance by simplifying some of the abstract operations (e.g., counting points), but at the possible cost of precision, since some polyhedral shapes are now approximated. For the queries given in this section, using probabilistic polyhedra produces answers in a few seconds, while using probabilistic intervals produces answers in a few milliseconds, with no loss of precision. Details are given in Section 7.

3 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 [14]. We retain Clarkson et al.'s notation for consistency.

3.1 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:

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

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

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

The language is essentially standard, though we limit the form of expressions to support our abstract interpretation-based semantics (Section 5). 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 $1 - q$.

Note that in our language, variables have only integer values and the syntax is missing a division operator. Furthermore, we will restrict arithmetic expressions to be of a linear forms only, that is, multiplication of two variables will be disallowed. These restrictions ease implementation considerations. Easing these constraints is an aspect of our future work.

3.2 Probabilistic semantics for tracking beliefs

To enforce a knowledge-based policy, an 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. A distribution is a map from states to positive real numbers, interpreted as probabilities (in range $[0, 1]$).

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

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 a function $\mathbf{State}_V \to \mathbb{R}+$. The variables of a state, written $\mathit{fv}(\sigma)$ is defined by $\mathit{domain}(\sigma)$, sometimes we will refer to this set as just the *domain* of σ. We will also use the this notation for distributions; $\mathit{fv}(\delta) \stackrel{\text{def}}{=} \mathit{domain}(\mathit{domain}(\delta))$. In the context of distributions, *domain* will also refer to the set $\mathit{fv}(\delta)$ as opposed to $\mathit{domain}(\delta)$.

Projecting distributions onto a set of variables is as follows:[1]

$$\delta \upharpoonright V \stackrel{\text{def}}{=} \lambda \sigma_V \in \mathbf{State}_V.\ \sum_{\tau\ :\ \tau \upharpoonright V = \sigma_V} \delta(\tau)$$

We will often project away a single variable. We will call this operation *forget*. Intuitively the distribution *forgets* about a variable x.

$$\mathrm{f}_x(\delta) \stackrel{\text{def}}{=} \delta \upharpoonright (\mathit{fv}(\delta) - \{x\})$$

[1] The notation $\sum_{x\ :\ \pi} \rho$ can be read *ρ is the sum over all x such that formula π is satisfied* (where x is bound in ρ and π).

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

where

$$
\begin{array}{ll}
\delta\,[x \to E] & \stackrel{\text{def}}{=} \lambda\sigma.\ \sum_{\tau\ |\ \tau[x \to [\![E]\!]\tau] = \sigma} \delta(\tau) \\
\delta_1 + \delta_2 & \stackrel{\text{def}}{=} \lambda\sigma.\ \delta_1(\sigma) + \delta_2(\sigma) \\
\delta \wedge B & \stackrel{\text{def}}{=} \lambda\sigma.\ \mathbf{if}\ [\![B]\!]\sigma\ \mathbf{then}\ \delta(\sigma)\ \mathbf{else}\ 0 \\
p \cdot \delta & \stackrel{\text{def}}{=} \lambda\sigma.\ p \cdot \delta(\sigma) \\
\|\delta\| & \stackrel{\text{def}}{=} \sum_\sigma \delta(\sigma) \\
\mathrm{normal}(\delta) & \stackrel{\text{def}}{=} \frac{1}{\|\delta\|} \cdot \delta \\
\delta|B & \stackrel{\text{def}}{=} \mathrm{normal}(\delta \wedge B) \\
\delta_1 \times \delta_2 & \stackrel{\text{def}}{=} \lambda(\sigma_1, \sigma_2).\ \delta_1(\sigma_1) \cdot \delta_2(\sigma_2) \\
\dot{\sigma} & \stackrel{\text{def}}{=} \lambda\tau.\ \mathbf{if}\ \sigma = \tau\ \mathbf{then}\ 1\ \mathbf{else}\ 0 \\
\sigma \upharpoonright V & \stackrel{\text{def}}{=} \lambda x \in \mathbf{Var}_V.\ \sigma(x) \\
\delta \upharpoonright V & \stackrel{\text{def}}{=} \lambda\sigma_V \in \mathbf{State}_V.\ \sum_{\tau\ :\ \tau \upharpoonright V = \sigma_V} \delta(\tau) \\
\mathrm{f}_x(\delta) & \stackrel{\text{def}}{=} \delta \upharpoonright (\mathit{fv}(\delta) - \{x\}) \\
\mathit{support}(\delta) & \stackrel{\text{def}}{=} \{\sigma\ :\ \delta(\sigma) > 0\}
\end{array}
$$

Figure 3: Probabilistic semantics for the core language and index of state/distribution operations

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:

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

Normalization requires the mass of a distribution to be non-zero. We will only be dealing with distributions of finite mass but some of the theory presented later makes use of zero-mass distributions. There is one such distribution for each domain; when the domain is understood from the context we will label its zero-mass distribution as $0_{\mathbf{Dist}}$.

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

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.Here we briefly

explain the concrete probabilistic semantics.

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 other 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.

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 other 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

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

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 is 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

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

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 while-loop iteration is essentially that of if B then S else skip; the semantics of the entire loop is the fixed point of a function that composes the distributions produced by each iteration. That such a fixed point exists is proved by Clarkson et al. [14]. For an implementation, however, the evaluation of a loop can be performed naively, by repeatedly evaluating the loop body until the mass of $\delta \wedge B$ becomes zero. This process has a chance of diverging, signifying an infinite loop on some $\sigma \in \textit{support}(\delta)$.

In Section 5 we make use of an additional convenience statement, uniform $x\ n_1\ n_2$ (equivalent to a series of probabilistic conditionals) intended to assign a uniform value in the range $\{n_1, ..., n_2\}$ to the variable x.

$$[\![\mathsf{uniform}\ x\ n_1\ n_2]\!]\delta \quad = \quad \mathrm{f}_x(\delta) \times \delta'$$

Here we use the *distribution product* operator, which is defined for two distributions with disjoint domains (sharing no variables):

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

The notation (σ_1, σ_2) is the "concatenation" of two states with disjoint domains. In the definition of $\mathsf{uniform}\ x\ n_1\ n_2$, δ' is defined over just the variable x (removed from δ by the forget operator) as follows.

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

3.3 Belief and security

Clarkson et al. [14] 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, are made visible to the attacker. Let σ_L be an arbitrary initial state of these variables. Then we take the following steps:

Step 1. Evaluate S probabilistically using the querier'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 $H \cup L$, is defined as $\delta = \delta_H \times \dot{\sigma}_L$. Here we write $\dot{\sigma}$ to denote the *point distribution* for 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 $\mathrm{normal}(\delta)(\sigma)$. Finally, we extract the attacker-visible output of the sampled state by projecting away the high variables: $\hat{\sigma}_L = \hat{\sigma} \restriction L$. The sampling here is needed because S may include probabilistic if statements, and so $\hat{\delta}'$ may not be a point distribution.

Step 3. Revise the attacker's initial belief δ_H according to the observed output $\hat{\sigma}_L$, yielding a new belief $\hat{\delta}_H = (\delta' \wedge \hat{\sigma}_L) \restriction H$. Here, δ' is *conditioned* on the output $\hat{\sigma}_L$, which yields a new distribution, and this distribution is then projected to the variables H. The conditioned distribution $\hat{\delta}_H$ is the non-normalized representation of

the attacker's belief about the secret variables, after observing the final values of low variables. In can be turned into a true distribution by normalizing it.

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 depends on the observer's ability to detect nontermination [14].

4 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 return the public output σ_L resulting from running S on σ_H if the agent deems that from this output the querier cannot know 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 $(\delta'|\sigma_L \upharpoonright H)(\sigma'_H) \leq t$.

This definition can be related to the experiment protocol defined in Section 3.3. 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 2 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 $(\delta'|\sigma_L \upharpoonright V)(\sigma'_V) \leq t$.

The two "foralls" in the definition are critical for ensuring security. The reason was shown by the first example in Section 2: 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. The more general policy $\forall \sigma_L \in$ *support*$(\delta' \upharpoonright L).(\delta'|\sigma_L \upharpoonright H)(\sigma_H) \leq t$, would sidestep the problem in the example, but this policy could still reveal information because it too depends on the actual secret σ_H.

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 is equivalent to a worst-case *conditional vulnerability* (upper) bound or alternatively a worst-case *conditional min-entropy* (lower) bound. Min-entropy measures the expected likelihood of an adversary guessing the secret value [53]; the stronger worst-case used in our definition does away with

expectation and bounds this likelihood regardless of what the secret is. Such worst-case measures were considered in [31] as a means of providing a stronger security guarantee. In our case, however, the extra strength is a side-effect of our need for a simulatable policy. See Section 9 for further details.

5 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, of which there are many [45, 47, 44, 26, 30, 12, 38, 11]. Such languages are largely ineffective for use in ensuring security guarantees. Approximate inference in these languages cannot ensure security guarantees, while exact inference, due to its lack of abstraction facilities, can be too inefficient when the state space is large.

We have developed a new means to perform probabilistic computation based on abstract interpretation. In this approach, execution time depends on the complexity of the query rather than the size of the input space. In the next two sections, we present two abstract domains. This section presents the first, denoted $\mathbb{P}$, where an abstract element is a single *probabilistic polyhedron*, which is a convex polyhedron [19] with information about the probabilities of its points.

Because using a single polyhedron will accumulate imprecision 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 [9, 46]). 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. An important element of our approach is the ability to soundly evaluate the knowledge-threshold policies, even under approximate inference.

5.1 Polyhedra

We first review *convex polyhedra*, a common technique for representing sets of program states. We use the meta-variables β, β_1, β_2, etc. to denote linear inequalities. We write $\mathit{fv}(\beta)$ to be the set of variables occurring in β; we also extend this to sets, writing $\mathit{fv}(\{\beta_1, \ldots, \beta_n\})$ for $\mathit{fv}(\beta_1) \cup \ldots \cup \mathit{fv}(\beta_n)$.

Definition 4. A *convex polyhedron* $C = (B, V)$ is a set of linear inequalities $B = \{\beta_1, \ldots, \beta_m\}$, interpreted conjunctively, over dimensions V. 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.

$$\gamma_{\mathbb{C}}((B, V)) \stackrel{\text{def}}{=} \{\sigma \;:\; \mathit{fv}(\sigma) = V,\ \forall \beta \in B.\ \sigma \models \beta\}$$

Naturally we require that $\mathit{fv}(\{\beta_1, \ldots, \beta_n\}) \subseteq V$. We write $\mathit{fv}((B, V))$ to denote the set of variables V of a polyhedron.

Given a state σ and an ordering on the variables in $fv(\sigma)$, we can view σ as a point in an N-dimensional space, where $N = |fv(\sigma)|$. 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 sometimes write linear equalities $x = f(\vec{y})$ as an abbreviation for the pair of inequalities $x \leq f(\vec{y})$ and $x \geq f(\vec{y})$.

Let $C = (B, V)$. 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 when determining their size, ensuring that $\#(C)$ is finite.
- (Logical) expression evaluation, $\langle\!\langle B \rangle\!\rangle\, C$ returns a convex polyhedron containing at least the points in C that satisfy B. Note that B may or may not have disjuncts.
- Expression count, $C \# B$ returns an upper bound on the number of integer points in C that satisfy B. Note that this may be more precise than $\#(\langle\!\langle B \rangle\!\rangle\, C)$ if B has disjuncts.
- Meet, $C_1 \sqcap_{\mathbb{C}} C_2$ is the convex polyhedron containing exactly 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 $\gamma_{\mathbb{C}}(C_1) \subseteq \gamma_{\mathbb{C}}(C_2)$.
- Affine transform, $C\,[x \to E]$, where $x \in fv(C)$, 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, $(\{x \leq y, y = 2z\}, V)\,[y \to z + y]$ evaluates to $(\{x \leq y - z, y - z = 2z\}, V)$.
- 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 $\gamma_{\mathbb{C}}(C') = \{\sigma \;:\; \tau \in \gamma_{\mathbb{C}}(C) \wedge \sigma = \tau \restriction V\}$. So $C' = \mathrm{f}_x(C)$ implies $x \notin fv(C')$. The projection of C to variables V, written $C \restriction V$ is defined as the forgetting of all the dimensions of C other than V.
- Linear partition $C_A \between C_B$ of two (possibly overlapping) polyhedra C_A, C_B is a set of equivalent disjoint polyhedra $\{C_i\}_{i=1}^{n}$. That is, $\cup_i \gamma_{\mathbb{C}}(C_i) = \gamma_{\mathbb{C}}(C_A) \cup \gamma_{\mathbb{C}}(C_B)$ and $\gamma_{\mathbb{C}}(C_i) \cap \gamma_{\mathbb{C}}(C_j) = \emptyset$ for $i \neq j$. When C_A and C_B do not overlap then $C_A \between C_B = \{C_A, C_B\}$.

We write $isempty(C)$ *iff* $\gamma_{\mathbb{C}}(C) = \emptyset$.

5.2 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.

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

We will write $\mathit{fv}(P) \stackrel{\text{def}}{=} \mathit{fv}(C)$ to denote the set of variables used in the probabilistic polyhedron.

Note the set $\gamma_{\mathbb{P}}(P)$ is a 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 $\gamma_{\mathbb{P}}(P)$ contains only the uniform distribution where each state in $\gamma_{\mathbb{C}}(C)$ has probability $\mathrm{p}^{\min}$. In general, however, the concretization of a probabilistic polyhedron will have an infinite number of distributions. For example, the pair of probabilistic polyhedra in Section 2, Equation 1 admits an infinite set of distributions, with per-point probabilities varied somewhere in the range $\mathrm{p}_1^{\min}$ and $\mathrm{p}_1^{\max}$. The representation of the non-uniform distribution in that example is thus approximate, but the security policy can still be checked via the $\mathrm{p}_1^{\max}$ (and $\mathrm{m}_2^{\max}$) properties of the probabilistic polyhedron.

Distributions represented by a probabilistic polyhedron are not necessarily normalized (as was true in Section 3.2). 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 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.

Ordering. Distributions are ordered point-wise [14]. That is, $\delta_1 \leq \delta_2$ if and only if $\forall \sigma.\ \delta_1(\sigma) \leq \delta_2(\sigma)$. For our abstract domain, we say that $P_1 \sqsubseteq_{\mathbb{P}} P_2$ if and only if $\forall \delta_1 \in \gamma_{\mathbb{P}}(P_1).\ \exists \delta_2 \in \gamma_{\mathbb{P}}(P_2).\ \delta_1 \leq \delta_2$. Testing $P_1 \sqsubseteq_{\mathbb{P}} P_2$ mechanically is non-trivial, but is unnecessary in our semantics. Rather, we need to test whether a distribution represents only the zero distribution $0_{\mathbf{Dist}} \stackrel{\text{def}}{=} \lambda\sigma.0$ in order to see that a fixed point for evaluating $\langle\!\langle \mathsf{while}\ B\ \mathsf{do}\ S \rangle\!\rangle\, P$ has been reached. Intuitively, no further iterations of the loop need to be considered once the probability mass flowing into the n^{th} iteration is zero. This condition can be detected as follows:

$$\begin{aligned}&\mathit{iszero}(P) \stackrel{\text{def}}{=} \\ &\quad \left(\mathrm{s}^{\min} = \mathrm{s}^{\max} = 0 \wedge \mathrm{m}^{\min} = 0 \leq \mathrm{m}^{\max}\right) \\ &\vee\ \left(\mathrm{m}^{\min} = \mathrm{m}^{\max} = 0 \wedge \mathrm{s}^{\min} = 0 \leq \mathrm{s}^{\max}\right) \\ &\vee\ \left(\mathit{isempty}(C) \wedge \mathrm{s}^{\min} = 0 \leq \mathrm{s}^{\max} \wedge \mathrm{m}^{\min} = 0 \leq \mathrm{m}^{\max}\right) \\ &\vee\ \left(\mathrm{p}^{\min} = \mathrm{p}^{\max} = 0 \wedge \mathrm{s}^{\min} = 0 \leq \mathrm{s}^{\max} \wedge \mathrm{m}^{\min} = 0 \leq \mathrm{m}^{\max}\right)\end{aligned}$$

If $iszero(P)$ holds, it is the case that $\gamma_{\mathbb{P}}(P) = \{0_{\mathbf{Dist}}\}$. This definition distinguishes $\gamma_{\mathbb{P}}(P) = \emptyset$ (if P has inconsistent constraints) from $\gamma_{\mathbb{P}}(P) = \{0_{\mathbf{Dist}}\}$. Note that having a more conservative definition of $iszero(P)$ (which holds for fewer probabilistic polyhedra P) would simply mean our analysis would terminate less often than it could, with no effect on security.

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 $\mathbb{P}$.

5.3 Abstract Semantics for $\mathbb{P}$

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 separate technical report [36].

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 3.2. 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.

Theorem 6. *For all P, δ, if $\delta \in \gamma_{\mathbb{P}}(P)$ and $\langle\langle S \rangle\rangle\, P$ terminates, then $[\![S]\!]\delta$ terminates and $[\![S]\!]\delta \in \gamma_{\mathbb{P}}(\langle\langle S \rangle\rangle\, P)$.*

When we say $[\![S]\!]\delta$ terminates (or $\langle\langle S \rangle\rangle\, P$ terminates) we mean that only a finite number of loop iterations are required to interpret the statement on a particular distribution (or probabilistic polyhedron). In the concrete semantics, termination can be checked by iterating until the mass of $\delta \wedge B$ (where B is a guard) becomes zero. (Note that $[\![S]\!]\delta$ is always defined, even for infinite loops, as the least fixed-point is always defined, but we need to distinguish terminating from non-terminating loops for security reasons, as per the comment at the end of Section 3.3.) To check termination in the abstract semantics, we check that upper bound on the mass of $P \wedge B$ becomes zero. 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, however, 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 [19, 15]. 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 in some predefined amount of time. 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 a widening operator that soundly accounts for non-termination behavior.

The proof for Theorem 6 is a structural induction on S; the meat of the proof is in the soundness of the various abstract operations. The following sections present these abstract operations and their soundness relative to the concrete operations. The basic structure of all the arguments is the same: the abstract operation over-approximates the concrete one.

5.3.1 Forget

We first describe the abstract forget operator $\mathrm{f}_y(P_1)$, which is used in implementing assignment. Our abstract implementation of the operation must be sound relative to the concrete one, specifically, if $\delta \in \gamma_{\mathbb{P}}(P)$ then $\mathrm{f}_y(\delta) \in \gamma_{\mathbb{P}}(\mathrm{f}_y(P))$.

The concrete forget operation projects away a single dimension:

$$\begin{aligned}\mathrm{f}_x(\delta) &\stackrel{\text{def}}{=} \delta \upharpoonright (\mathit{fv}(\delta) - \{x\}) \\ &= \lambda\sigma_V \in \mathbf{State}_V.\ \sum_{\tau\,:\,\tau\upharpoonright V = \sigma_V} \delta(\tau) \qquad \text{where } V = \mathit{fv}(\delta) - \{x\}\end{aligned}$$

When we forget variable y, we collapse any states that are equivalent up to the value of y into a single state.

To do this soundly, we must find an upper bound $\mathrm{h}_y^{\max}$ and a lower bound $\mathrm{h}_y^{\min}$ on the number of integer points in C_1 that share the value of the remaining dimensions (this may be visualized of as the min and max height of C_1 in the y dimension). More precisely, if $V = \mathit{fv}(C_1) - \{y\}$, then for every $\sigma_V \in \gamma_{\mathbb{C}}(C_1 \upharpoonright V)$ we have $\mathrm{h}_y^{\min} \leq |\{\sigma \in \gamma_{\mathbb{C}}(C_1) : \sigma \upharpoonright V = \sigma_V\}| \leq \mathrm{h}_y^{\max}$. Once these values are obtained, we have that $\mathrm{f}_y(P_1) \stackrel{\text{def}}{=} P_2$ where the following hold of P_2.

$$\begin{array}{lcl|lcl} C_2 & = & \mathrm{f}_y(C_1) & & & \\ \mathrm{p}_2^{\min} & = & \mathrm{p}_1^{\min} \cdot \max\left\{\mathrm{h}_y^{\min} - (\#(C_1) - \mathrm{s}_1^{\min}),\ 1\right\} & & & \\ \mathrm{p}_2^{\max} & = & \mathrm{p}_1^{\max} \cdot \min\left\{\mathrm{h}_y^{\max},\ \mathrm{s}_1^{\max}\right\} & & & \\ \mathrm{s}_2^{\min} & = & \lceil \mathrm{s}_1^{\min} / \mathrm{h}_y^{\max} \rceil & \mathrm{m}_2^{\min} & = & \mathrm{m}_1^{\min} \\ \mathrm{s}_2^{\max} & = & \min\left\{\#(\mathrm{f}_y(C_1)),\ \mathrm{s}_1^{\max}\right\} & \mathrm{m}_2^{\max} & = & \mathrm{m}_1^{\max} \end{array}$$

The new values for the under and over-approximations of the various parameters are derived by reasoning about the situations in which these quantities could be the smallest or the greatest, respectively, over all possible $\delta_2 \in \gamma_{\mathbb{P}}(P_2)$ where $\delta_2 = \mathrm{f}_y(\delta_1)$, $\delta_1 \in \gamma_{\mathbb{P}}(P_1)$. We summarize the reasoning behind the calculations below:

- $\mathrm{p}_2^{\min}$: The minimum probability per support point is derived by considering a point of P_2 that had the least amount of mass of P_1 mapped to it. Let us call this point σ_V and the set of points mapped to it $S = \{\sigma \in \gamma_{\mathbb{C}}(C_1) \ :\ \sigma \upharpoonright V = \sigma_V\}$. S could have as little as $\mathrm{h}_y^{\min}$ points, as per definition of $\mathrm{h}_y^{\min}$ and not all of these points must be mass-carrying. There are at least $\mathrm{s}_1^{\min}$ mass-carrying points in C_1. If we assume that as many as possible of the mass carrying points in the region C_1 are outside of S, it must be that S still contains at least $\mathrm{h}_y^{\min} - (\#(C_1) - \mathrm{s}_1^{\min})$ mass carrying-points, each having probability at least $\mathrm{p}_1^{\min}$.

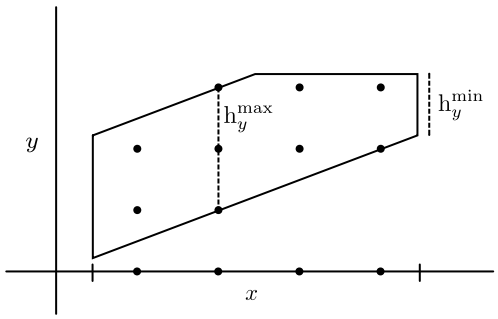


Figure 4: Example of a forget operation in the abstract domain $\mathbb{P}$. In this case, $\mathrm{h}_y^{\min} = 1$ and $\mathrm{h}_y^{\max} = 3$. Note that $\mathrm{h}_y^{\max}$ is precise while $\mathrm{h}_y^{\min}$ is an under-approximation. If $\mathrm{s}_1^{\min} = \mathrm{s}_1^{\max} = 9$ then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

- $\mathrm{p}_2^{\max}$: The maximum number of points of P_1 that get mapped to a single point in P_2 cannot exceed $\mathrm{s}_1^{\max}$, the number of support points in P_1. Likewise it cannot exceed $\mathrm{h}_y^{\max}$ as per definition of $\mathrm{h}_y^{\max}$.
- $\mathrm{s}_2^{\min}$: There cannot be more than $\mathrm{h}_y^{\max}$ support points of P_1 that map to a single point in P_2 and there are at least $\mathrm{s}_1^{\min}$ support points in P_1. If we assume that every single support point of P_2 had the maximum number of points mapped to it, there would still be $\lceil \mathrm{s}_1^{\min} / \mathrm{h}_y^{\max} \rceil$ distinct support points in P_2.
- $\mathrm{s}_2^{\max}$: The maximum number of support points cannot exceed the size of the region defining P_2. It also cannot exceed the number of support points of P_1, even if we assumed there was a one-to-one mapping between the support points of P_1 and support points of P_2.

Figure 4 gives an example of a forget operation and illustrates the quantities $\mathrm{h}_y^{\max}$ and $\mathrm{h}_y^{\min}$. If $C_1 = (B_1, V_1)$, the upper bound $\mathrm{h}_y^{\max}$ can be found by maximizing $y - y'$ subject to the constraints $B_1 \cup B_1[y'/y]$, where y' is a fresh variable and $B_1[y'/y]$ represents the set of constraints obtained by substituting y' for y in B_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 $\mathrm{h}_y^{\min} = 1$. A more precise estimate can be obtained by treating the convex polyhedron as a subset of $\mathbb{R}^n$ and finding the vertex with minimal height along dimension y. Call this distance u. An example of this quantity is labeled $\mathrm{h}_y^{\min}$ in Figure 4. 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 $\lceil u \rceil - 1$. The final under-approximation is then taken to be the larger of 1 and $\lceil u \rceil - 1$. As this method requires us to inspect every vertex of the convex polyhedron and to compute the y height of the polyhedron at that vertex, we can also look for the one upon which

the polyhedron has the greatest height, providing us with the estimate for $\mathrm{h}_y^{\max}$.

Lemma 7. *If* $\delta \in \gamma_{\mathbb{P}}(P)$ *then* $\mathrm{f}_y(\delta) \in \gamma_{\mathbb{P}}(\mathrm{f}_y(P))$.

We can define an abstract version of projection using forget:

Definition 8. Let $\mathrm{f}_{\{x_1,x_2,\ldots,x_n\}}(P) = \mathrm{f}_{\{x_2,\ldots,x_n\}}(\mathrm{f}_{x_1}(P))$. Then $P \restriction V' = \mathrm{f}_{(fv(P)-V')}(P)$.

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

5.3.2 Assignment

The concrete assignment operation is defined so that the probability of a state σ is the accumulated probability mass of all states τ that lead to σ via the assignment:

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

The abstract implementation of this operation strongly depends on the invertibility of the assignment. Intuitively, the set $\{\tau\ :\ \tau\left[x \to [\![E]\!]\tau\right] = \sigma\}$ can be obtained from σ by inverting the assignment, if invertible.[2] Otherwise, the set can be obtained by *forgetting* about the x variable in σ.

Similarly, we have two cases for abstract assignment. If $x\ :=\ E$ is invertible, the result of the assignment $P_1\left[x \to E\right]$ is the probabilistic polyhedron P_2 such that $C_2 = C_1\left[x \to E\right]$ and all other components are unchanged. If the assignment is not invertible, then information about the previous value of x is lost. In this case, we forget x thereby projecting (or "flattening") onto the other dimensions. Then we introduce dimension x back and add a constraint on x that is defined by the assignment. More precisely the process is as follows. Let $P_2 = \mathrm{f}_x(P_1)$ where $C_2 = (B_2, V_2)$. Then $P_1\left[x \to E\right]$ is the probabilistic polyhedron P_3 with $C_3 = (B_2 \cup \{x = E\}, V_2 \cup \{x\})$ and all other components as in P_2.

The test for invertibility itself is simple as our system restricts arithmetic expressions to linear ones. Invertibility relative to a variable x is then equivalent to the presence of a non-zero coefficient given to x in the expression on the right-hand-side of the assignment. For example, $x\ :=\ 42x + 17y$ is invertible but $x\ :=\ 17y$ is not.

Lemma 9. *If* $\delta \in \gamma_{\mathbb{P}}(P)$ *then* $\delta\left[v \to E\right] \in \gamma_{\mathbb{P}}(P\left[v \to E\right])$.

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 would collapse multiple points to one and so could be handled similarly to projection.

[2] An assignment $x\ :=\ E$ is invertible if there exists an *inverse* function f : **State** $\to$ **State** such that $f\left([\![x\ :=\ E]\!]\sigma\right) = \sigma$ for all σ. Note that the f here needs not be expressible as an assignment in our (integer-based) language, and generally would not be as most integers have no integer multiplicative inverses.

5.3.3 Plus

The concrete plus operation adds together the mass of two distributions:

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

The abstract counterpart needs to over-approximate this semantics. Specifically, if $\delta_1 \in \gamma_{\mathbb{P}}(P_1)$ and $\delta_2 \in \gamma_{\mathbb{P}}(P_2)$ then $\delta_1 + \delta_2 \in \gamma_{\mathbb{P}}(P_1 + P_2)$.

The abstract sum of two probabilistic polyhedra can be easily defined if their support regions do not overlap. In such situations, we would define P_3 as below:

$$\begin{array}{lcl} C_3 & = & C_1 \sqcup_{\mathbb{C}} C_2 \\ \mathrm{p}_3^{\min} & = & \min\left\{\mathrm{p}_1^{\min}, \mathrm{p}_2^{\min}\right\} \\ \mathrm{p}_3^{\max} & = & \max\left\{\mathrm{p}_1^{\max}, \mathrm{p}_2^{\max}\right\} \\ \mathrm{s}_3^{\min} & = & \mathrm{s}_1^{\min} + \mathrm{s}_2^{\min} \\ \mathrm{s}_3^{\max} & = & \mathrm{s}_1^{\max} + \mathrm{s}_2^{\max} \\ \mathrm{m}_3^{\min} & = & \mathrm{m}_1^{\min} + \mathrm{m}_2^{\min} \\ \mathrm{m}_3^{\max} & = & \mathrm{m}_1^{\max} + \mathrm{m}_2^{\max} \end{array}$$

If there is overlap between C_1 and C_2, the situation becomes more complex. To soundly compute the effect of plus we need to determine the minimum and maximum number of points in the intersection that may be support points 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 10. 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 $P_1 \ \dot{\frown}\ P_2$, 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* $P_1 \ \dot{\smile}\ P_2$ is the cardinality of the largest possible overlap. Formally, we define these as follows. .

$$P_1 \ \dot{\frown}\ P_2 \stackrel{\text{def}}{=} \max\left\{\mathrm{s}_1^{\min} + \mathrm{s}_2^{\min} - \left(\#(C_1) + \#(C_2) - \#(C_1 \sqcap_{\mathbb{C}} C_2)\right),\ 0\right\}$$
$$P_1 \ \dot{\smile}\ P_2 \stackrel{\text{def}}{=} \min\left\{\mathrm{s}_1^{\max}, \mathrm{s}_2^{\max}, \#(C_1 \sqcap_{\mathbb{C}} C_2)\right\}$$

The pessimistic overlap is derived from the usual inclusion-exclusion principle: $|A \cap B| = |A| + |B| - |A \cup B|$. The optimistic overlap is trivial; it cannot exceed the support size of either distribution or the size of the intersection.

We can now define abstract addition.

Definition 11. If not $iszero(P_1)$ and not $iszero(P_2)$ then $P_1 + P_2$ 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.

$$\begin{array}{lcl}
C_3 & = & C_1 \sqcup_{\mathbb{C}} C_2 \\
\mathrm{p}_3^{\min} & = & \begin{cases} \mathrm{p}_1^{\min} + \mathrm{p}_2^{\min} & \text{if } P_1 \circledcirc P_2 = \#(C_3) \\ \min\left\{\mathrm{p}_1^{\min}, \mathrm{p}_2^{\min}\right\} & \text{otherwise} \end{cases} \\
\mathrm{p}_3^{\max} & = & \begin{cases} \mathrm{p}_1^{\max} + \mathrm{p}_2^{\max} & \text{if } P_1 \circledcirc P_2 > 0 \\ \max\left\{\mathrm{p}_1^{\max}, \mathrm{p}_2^{\max}\right\} & \text{otherwise} \end{cases} \\
\mathrm{s}_3^{\min} & = & \max\left\{\mathrm{s}_1^{\min} + \mathrm{s}_2^{\min} - P_1 \circledcirc P_2,\ 0\right\} \\
\mathrm{s}_3^{\max} & = & \min\left\{\mathrm{s}_1^{\max} + \mathrm{s}_2^{\max} - P_1 \circledcirc P_2,\ \#(C_3)\right\} \\
\mathrm{m}_3^{\min} & = & \mathrm{m}_1^{\min} + \mathrm{m}_2^{\min} \quad \mid \quad \mathrm{m}_3^{\max} = \mathrm{m}_1^{\max} + \mathrm{m}_2^{\max}
\end{array}$$

If $iszero(P_1)$ then we define $P_1 + P_2$ as identical to P_2; if $iszero(P_2)$, the sum is defined as identical to P_1.

Lemma 12. *If $\delta_1 \in \gamma_{\mathbb{P}}(P_1)$ and $\delta_2 \in \gamma_{\mathbb{P}}(P_2)$ then $\delta_1 + \delta_2 \in \gamma_{\mathbb{P}}(P_1 + P_2)$.*

5.3.4 Product

The concrete product operation merges two distributions over distinct variables into a compound distribution over the union of the variables:

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

When evaluating the product $P_3 = P_1 \times P_2$, we assume that the domains of P_1 and P_2 are disjoint, i.e., C_1 and C_2 refer to disjoint sets of variables. If $C_1 = (B_1, V_1)$ and $C_2 = (B_2, V_2)$, then the polyhedron $C_1 \times C_2 \stackrel{\text{def}}{=} (B_1 \cup B_2, V_1 \cup V_2)$ 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 $\sigma \restriction V_2 \in \gamma_{\mathbb{C}}(C_2)$. Determining the remaining components is straightforward since P_1 and P_2 are disjoint.

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

Lemma 13. *For all P_1, P_2 such that $fv(P_1) \cap fv(P_2) = \emptyset$, 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, we extend statements S to include the statement of the form uniform $x\ n_1\ n_2$ whose semantics is to define variable x as having a value uniformly distributed between n_1 and n_2.

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

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 $C_2 = (\{x \geq n_1, x \leq n_2\}, \{x\})$.

We will say that the abstract semantics correspond to the concrete semantics of uniform defined similarly as follows.

$$[\![\mathsf{uniform}\ x\ n_1\ n_2]\!]\delta\ =\ (\delta \upharpoonright \mathit{fv}(\delta) - \{x\}) \times \delta_2$$

where $\delta_2 = (\lambda\sigma.\textbf{if}\ n_1 \leq \sigma(x) \leq n_2\ \textbf{then}\ \frac{1}{n_2 - n_1 + 1}\ \textbf{else}\ 0)$.

The soundness of the abstract semantics follows immediately from the soundness of forget and product.

5.3.5 Conditioning

The concrete conditioning operation restricts a distribution to a region defined by a boolean expression, nullifying any probability mass outside it:

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

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 14. 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 $P_1 \wedge B$ is the probabilistic polyhedron P_2 defined as follows.

$$\begin{array}{ll|l}
\mathrm{p}_2^{\min} & = \mathrm{p}_1^{\min} & \mathrm{s}_2^{\min} = \max\left\{\mathrm{s}_1^{\min} - \overline{n}, 0\right\} \\
\mathrm{p}_2^{\max} & = \mathrm{p}_1^{\max} & \mathrm{s}_2^{\max} = \min\left\{\mathrm{s}_1^{\max}, n\right\} \\
\multicolumn{3}{l}{\mathrm{m}_2^{\min} = \max\left\{\mathrm{p}_2^{\min} \cdot \mathrm{s}_2^{\min},\ \mathrm{m}_1^{\min} - \mathrm{p}_1^{\max} \cdot \min\left\{\mathrm{s}_1^{\max}, \overline{n}\right\}\right\}} \\
\multicolumn{3}{l}{\mathrm{m}_2^{\max} = \min\left\{\mathrm{p}_2^{\max} \cdot \mathrm{s}_2^{\max},\ \mathrm{m}_1^{\max} - \mathrm{p}_1^{\min} \cdot \max\left\{\mathrm{s}_1^{\min} - n, 0\right\}\right\}} \\
\multicolumn{3}{l}{C_2 = \langle\!\langle B \rangle\!\rangle\, C_1}
\end{array}$$

The maximal and minimal probability per point are unchanged, as conditioning simply 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 dually 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 $\overline{n} = C_1 \# \neg B$,

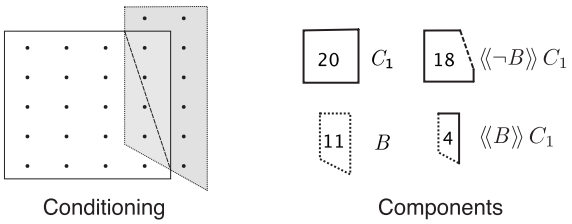


Figure 5: Example of distribution conditioning in the abstract domain $\mathbb{P}$.

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 under-approximation 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 $\max\bigl(\mathrm{s}_1^{\min} - n, 0\bigr)$.

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.

$$\begin{array}{lll} \mathrm{s}_1^{\min} = 19 & \mathrm{p}_1^{\min} = 0.01 & \mathrm{m}_1^{\min} = 0.85 \\ \mathrm{s}_1^{\max} = 20 & \mathrm{p}_1^{\max} = 0.05 & \mathrm{m}_1^{\max} = 0.9 \end{array}$$

Then $n = 4$ and $\overline{n} = 16$. Note that we have set $\overline{n}$ to be the number of points in the non-shaded region of Figure 5. This is more precise than the count given by $\#(\langle\langle B \rangle\rangle\, C)$, which would yield 18. This demonstrates why it is worthwhile to have a separate operation for counting points satisfying a boolean expression. These values of n and $\overline{n}$ give us the following for the first four numeric components of P_2.

$$\begin{array}{ll} \mathrm{s}_2^{\min} = \max(19 - 16, 0) = 3 & \mathrm{p}_2^{\min} = 0.01 \\ \mathrm{s}_2^{\max} = \min(20, 4) = 4 & \mathrm{p}_2^{\max} = 0.05 \end{array}$$

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 $\mathrm{s}_2^{\min/\max}$.

$$\mathrm{m}_2^{\min} = 0.01 \cdot 3 = 0.03 \qquad \mathrm{m}_2^{\max} = 0.05 \cdot 4 = 0.2$$

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

$$\begin{array}{rcl} \mathrm{m}_2^{\min} & = & 0.85 - 0.05 \cdot 16 \;\; = \;\; 0.05 \\ \mathrm{m}_2^{\max} & = & 0.9 - 0.01 \cdot (19 - 4) \;\; = \;\; 0.75 \end{array}$$

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 15. *If $\delta \in \gamma_{\mathbb{P}}(P)$ then $\delta \wedge B \in \gamma_{\mathbb{P}}(P \wedge B)$.*

5.3.6 Scalar Product

The scalar product is straightforward both in the concrete and abstract sense; it just scales the mass per point and total mass:

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

Definition 16. Given a scalar p in $(0, 1]$, we write $p \cdot P_1$ for the probabilistic polyhedron P_2 specified below.

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

If $p = 0$ then $p \cdot P_2$ is defined instead as below:

$$\begin{array}{rcl|rcl} \mathrm{s}_2^{\min} & = & 0 & \mathrm{p}_2^{\min} & = & 0 \\ \mathrm{s}_2^{\max} & = & 0 & \mathrm{p}_2^{\max} & = & 0 \\ \mathrm{m}_2^{\min} & = & 0 & C_2 & = & 0_{\mathcal{C}} \\ \mathrm{m}_2^{\max} & = & 0 & & & \end{array}$$

Here $0_{\mathcal{C}}$ refers to a convex polyhedra (over the same dimensions as C_2) whose concretization is empty.

Lemma 17. *If $\delta_1 \in \gamma_{\mathbb{P}}(P_1)$ then $p \cdot \delta_1 \in \gamma_{\mathbb{P}}(p \cdot P_1)$.*

5.3.7 Normalization

The normalization of a distribution produces a true probability distribution, whose total mass is equal to 1:

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

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.

Definition 18. Whenever $\mathrm{m}_1^{\min} > 0$, we write $\mathrm{normal}(P_1)$ for the probabilistic polyhedron P_2 specified below.

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

When $\mathrm{m}_1^{\min} = 0$, we set $\mathrm{p}_2^{\max} = 1$. Note that if P_1 is the zero distribution then normal(P_1) is not defined.

The normalization operator illustrates the key novelty of our definition of probabilistic polyhedron: to ensure that the overapproximation of a state's probability ($\mathrm{p}^{\max}$) is sound, we must divide by the *underapproximation* of the total probability mass ($\mathrm{m}^{\min}$).

Lemma 19. *If $\delta_1 \in \gamma_{\mathbb{P}}(P_1)$ and normal(δ_1) is defined, then normal$(\delta_1) \in \gamma_{\mathbb{P}}($normal$(P_1))$.*

5.4 Policy Evaluation

Here we show how to implement the threshold test given as Definition 3 using probabilistic polyhedra. To make the definition simpler, let us first introduce a bit of notation.

Notation 20. If P is a probabilistic polyhedron over variables V, and σ is a state over variables $V' \subseteq V$, then $P \wedge \sigma \stackrel{\text{def}}{=} P \wedge B$ where $B = \bigwedge_{x \in V'} x = \sigma(x)$.

Recall that we define $\delta|B$ in the concrete semantics to be normal$(\delta \wedge B)$. The corresponding operation in the abstract semantics is similar: $P|B \stackrel{\text{def}}{=}$ normal$(P \wedge B)$.

Definition 21. Given some probabilistic polyhedron P_1 and statement S, with low security variables L and high security variables H, where $\langle\!\langle S \rangle\!\rangle\, P_1$ terminates, let $P_2 = \langle\!\langle S \rangle\!\rangle\, P_1$ and $P_3 = P_2 \restriction L$. If, for every $\sigma_L \in \gamma_{\mathbb{C}}(C_3)$ with $\neg iszero(P_2 \wedge \sigma_L)$, we have $P_4 = (P_2|\sigma_L) \restriction H$ with $\mathrm{p}_4^{\max} \leq t$, then we write $tsecure_t(S, P_1)$.

The computation of P_3 involves only abstract interpretation and projection, which are computable using the operations defined previously in this section. If we have a small number of outputs (as for the binary outputs considered in our examples), we can enumerate them and check $\neg iszero(P_2 \wedge \sigma_L)$ for each output σ_L. When this holds (that is, the output is feasible), we compute P_4, which again simply involves the abstract operations defined previously. The final threshold check is then performed by comparing $\mathrm{p}_4^{\max}$ to the probability threshold t.

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 $tsecure_t(S, P_1)$, then S is threshold secure for threshold t when evaluated with initial belief δ.*

The proof of this theorem follows from the soundness of the abstraction (Theorem 6), noting the direct parallels of threshold security definitions for distributions (Definitions 3) and probabilistic polyhedra (Definition 21).

5.5 Supporting Other Domains, Including Intervals and Octagons

Our approach to constructing probabilistic polyhedra from normal polyhedra can be adapted to add probabilities any other abstract domain for which the operations defined in Section 5.1 can be implemented. Most of the operations listed there are standard to

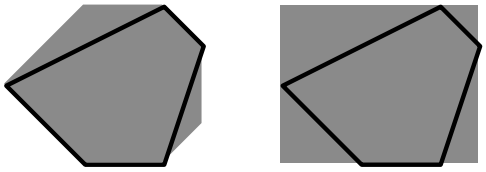


Figure 6: The (over)approximation of a polyhedron using an octagon (left) and an interval (right).

abstract domains in general, except for the size operation and the related expression count. Adopting an abstract domain to our system would therefore only require designing these counting methods for the new domain.

Two domains that are very easy to adapt that are in common use are intervals and octagons. *Intervals* [16], $\mathbb{C}_{\mathbb{I}}$, are convex shapes that can be described as a set of closed intervals, one for each dimension. Alternatively they can be thought of a restricted form of polyhedra in which the constraints I all have the form $a \leq x \leq b$. Operations on intervals are much faster than on polyhedra. Specific to our requirements, counting the integer points inside interval regions and determining their height for the forget operation are both trivial computations.

Octagons [39], $\mathbb{C}_{\mathbb{O}}$, are formed by constraints O that have the form $ax + by \leq c$ where $a, b \in \{-1, 0, 1\}$. In two dimensions these shapes, appropriately, have at most 8 sides. If the number of dimensions is fixed, the number of constraints and the number of vertices of an octagon are bounded. Furthermore, the operations on octagons have lower computational complexity than those for polyhedra, though they are not as efficient as those for intervals.

Any interval or octagon is also a polyhedron. Conversely, one can over-approximate any polyhedron by an interval or octagon. Naturally the smallest over-approximation is of greatest interest. Examples are illustrated in Figure 6. This fact is relevant when computing the various equivalent operations to those listed for polyhedra in Section 5.1: applying the definitions given there on octagons/intervals may not necessarily result in octagons/intervals, and so the result must be further approximated. For example, consider the evaluation operation $\langle\langle B \rangle\rangle\, I$. This must compute a region that contains *at least* the points in I satisfying B. Thus, if a non-octagon/interval is produced, it can simply be over-approximated. Another example is the affine transform operation $I\,[x \to E]$, which should contain *at least* the points $\tau = \sigma\,[x \to E]$ with $\sigma \in \gamma_{\mathbb{C}_{\mathbb{I}}}(I)$, where $\sigma\,[x \to E]$ is a single state transformed by the expression E. In general the operations for simpler domains are much faster than those for more complex domains. Though the imprecision and thus the need to approximate expression evaluation might make it occasionally slower, for our experiments any slowdown is typically overshadowed by the reduced complexity overall.

Thus we can construct abstractions of probability distributions based on these simpler domains instead of polyhedra. The domain of probabilistic intervals $\mathbb{I}$ (octagons $\mathbb{O}$) is defined as in Section 5, except using an interval (octagon) instead of polyhedron

for the region constraint. The abstract semantics described in this section can then be soundly implemented in terms of these simpler shapes in place of polyhedra.

Remark 23. If $\delta \in \gamma_{\mathbb{P}}(P)$ then $\delta \in \gamma_{\mathbb{I}}(I)$ and $\delta \in \gamma_{\mathbb{O}}(O)$, where I and O are identical to P except P has region constrained by C, a convex polyhedron, while I is constrained by interval C_I and O is constrained by octagon C_O, with $\gamma_{\mathbb{C}}(C) \subseteq \gamma_{\mathbb{C}_{\mathbb{I}}}(C_I)$ and $\gamma_{\mathbb{C}}(C) \subseteq \gamma_{\mathbb{C}_{\mathbb{O}}}(C_O)$.

6 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 24. A *probabilistic (polyhedral) set* Δ is a set of probabilistic polyhedra, or $\{P_i\}$ with each P_i over the same variables.[3] We write $\mathcal{P}_n(\mathbb{P})$ for the domain of probabilistic polyhedral powersets composed of no more than n probabilistic polyhedra.

Each probabilistic polyhedron P is interpreted disjunctively: it characterizes one of many possible distributions. The probabilistic polyhedral set is interpreted additively. To define this idea precisely, we first define a lifting of $+$ to sets of distributions. Let D_1, D_2 be two sets of distributions. We then define addition as follows.

$$D_1 + D_2 = \{\delta_1 + \delta_2 \ : \ \delta_1 \in D_1 \wedge \delta_2 \in D_2\}$$

This operation is commutative and associative and thus we can use $\sum$ for summations without ambiguity as to the order of operations. The concretization function for $\mathcal{P}_n(\mathbb{P})$ is then defined as:

$$\gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta) \stackrel{\text{def}}{=} \sum_{P \in \Delta} \gamma_{\mathbb{P}}(P)$$

Following Monniaux's formulation of a finite sums abstraction [41], elements of Δ need not be disjoint. While enforcing disjointness would simplify determining the most probable points for policy evaluation (see Section 6.2), it would necessitate splitting of probabilistic polyhedra when overlaps arise. Repeated splitting of already approximate probabilistic polyhedra decreases their precision and can hurt performance by increasing the number of regions to track during abstract interpretation.

We can characterize the condition of Δ containing only the zero distribution, written $\mathit{iszero}(\Delta)$, via the condition that all of the member probabilistic polyhedra are zero.

$$\mathit{iszero}(\Delta) \stackrel{\text{def}}{=} \bigwedge_{P \in \Delta} \mathit{iszero}(P)$$

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

The semantics for the powerset abstraction we describe in this section is designed to soundly approximate the concrete semantics.

[3] We write $\{X_i\}$ as shorthand for a set of n elements of type X, for some n. We write $\{X_i\}_{i=1}^n$ when the choice of n is important.

Theorem 25. *For all δ, S, Δ, if $\delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta)$ and $\langle\langle S \rangle\rangle\, \Delta$ terminates, then $[\![S]\!]\delta$ terminates and $[\![S]\!]\delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\langle\langle S \rangle\rangle\, \Delta)$.*

The proof for this theorem follows the same form as the corresponding soundness theorem for probabilistic polyhedra (Theorem 6), via soundness of the individual abstract operations in relation to their concrete versions. The full proof of this is given in the companion technical report [36].

To bound the size of the set of probabilistic polyhedra that will arise from the various operations that will follow, we introduce a simplification operation.

Definition 26. The *powerset simplification* transforms a set containing potentially more than n elements into one containing no more than n, for $n \geq 1$. The simplest approach involves repeated use of abstract plus in the base domain $\mathbb{P}$.

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

Lemma 27. $\gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta) \subseteq \gamma_{\mathcal{P}_n(\mathbb{P})}(\lfloor \Delta \rfloor_m)$ *where* $m \leq n$.

Note that the order in which individual probabilistic polyhedra are simplified has no effect on soundness but may impact the precision of the resulting abstraction. We explore the variation in precision due to these choices in Section 7.3.

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 (the first four, below) we simply list the definition; we elaborate on the remaining four.

Forget $\mathrm{f}_y(\Delta) \stackrel{\text{def}}{=} \{\mathrm{f}_y(P) \ : \ P \in \Delta\}$

Project. $\Delta \restriction V \stackrel{\text{def}}{=} \{P \restriction V \ : \ P \in \Delta\}$

Assignment. $\Delta\,[x \to E] \stackrel{\text{def}}{=} \{P[x \to E] \ : \ P \in \Delta\}$

Scalar product. $p \cdot \Delta \stackrel{\text{def}}{=} \{p \cdot P \ : \ P \in \Delta \wedge \neg \mathit{iszero}(p \cdot P)\}$

Conditioning Recall that for probabilistic polyhedra, conditioning $P \wedge B$ is defined in terms of logical expression evaluation for convex polyhedra, $\langle\langle B \rangle\rangle\, C$. This operation returns a convex polyhedron that contains at least the points in C that satisfy the logical expression B. This operation is tight if B does not contain disjuncts. When B does have disjuncts whose union does not define a convex region then the operation will be approximate. Consider Example 2. The condition $age = 20 \vee age = 30 \vee ... \vee age = 60$, were it be approximated using a single convex region, would be equivalent to the condition $age \geq 20 \vee age \leq 60$.

In the powerset domain we keep track of multiple convex regions hence can better approximate the conditioning operation. The approach we take is to convert the logical expression into a *disjoint* disjunctive normal form: $\mathrm{ddnf}\,(B) \stackrel{\text{def}}{=} \{B_1, B_2, \cdots, B_m\}$, such that $\{\sigma \ : \ \sigma \models B\} = \{\sigma \ : \ \sigma \models B_1 \vee \cdots \vee B_m\}$, each disjunct B_i contains no further disjunctions, and $\{\sigma \ : \ \sigma \models B_i \wedge B_j\} = \emptyset$ for all $i \neq j$ (B_i are disjoint).

Conditioning is thus defined as follows:

$$\Delta \wedge B \stackrel{\text{def}}{=} \lfloor \{P \wedge B_i \ : \ P \in \Delta \wedge B_i \in \text{ddnf}\,(B) \wedge \neg \mathit{iszero}(P \wedge B_i)\} \rfloor_n$$

The powerset simplification here reduces the set of probabilistic polyhedra to no more than n. Before the simplification, the number of probabilistic polyhedra could be as large as $|\Delta| \cdot |\text{ddnf}\,(B)|$. The number of disjuncts itself can be exponential in the size of B.

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. $\Delta \times P' \stackrel{\text{def}}{=} \{P \times P' \ : \ P \in \Delta\}$ (where we assume that $\mathit{fv}(\Delta) \cap \mathit{fv}(P') = \emptyset$).

Plus The abstract plus operation involves simplifying the combined contributions from two sets into one bounded set: $\Delta_1 + \Delta_2 \stackrel{\text{def}}{=} \lfloor \Delta_1 \cup \Delta_2 \rfloor_n$, whenever $\neg \mathit{iszero}(\Delta_1)$ and $\neg \mathit{iszero}(\Delta_2)$. Alternatively, if $\mathit{iszero}(\Delta_1)$ (or $\mathit{iszero}(\Delta_2)$) then $\Delta_1 + \Delta_2$ is defined to be identical to Δ_2 (or Δ_1).

The definition of abstract plus given above is technically sound but for an implementation it would make sense to assuming that Δ_1 contains probabilistic polyhedra that are somewhat more related to each other than those in Δ_2. It is preferable to merge regions that close together rather than those further apart. Therefore our implementation performs abstract plus heuristically as follows.

$$\Delta_1 + \Delta_2 = \lfloor \Delta_1 \rfloor_{\lfloor n/2 \rfloor} \cup \lfloor \Delta_2 \rfloor_{n - \lfloor n/2 \rfloor}$$

This may not always be the best grouping of probabilistic polyhedra to merge. There is quite a lot of arbitrary choice that can be made in order to evaluate this heuristic or the base definition of abstract plus without this heuristic.

Normalization Since in the $\mathcal{P}_n\,(\mathbb{P})$ 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 polyhedron set $\Delta = \{P_1, \ldots, P_n\}$ is defined as follows.

$$\mathrm{M}^{\min}(\Delta) \stackrel{\text{def}}{=} \textstyle\sum_{i=1}^{n} \mathrm{m}_i^{\min} \quad \Big| \quad \mathrm{M}^{\max}(\Delta) \stackrel{\text{def}}{=} \textstyle\sum_{i=1}^{n} \mathrm{m}_i^{\max}$$

Definition 28. The normalization a probabilistic polyhedra P_1 relative to a probabilistic polyhedron set Δ, written $\text{normal}_\Delta(P_1)$, is the probabilistic polyhedron P_2 defined as follows whenever $\mathrm{M}^{\min}(\Delta) > 0$.

$$\begin{array}{rcl|rcl} \mathrm{p}_2^{\min} & = & \mathrm{p}_1^{\min}/\mathrm{M}^{\max}(\Delta) & \mathrm{s}_2^{\min} & = & \mathrm{s}_1^{\min} \\ \mathrm{p}_2^{\max} & = & \mathrm{p}_1^{\max}/\mathrm{M}^{\min}(\Delta) & \mathrm{s}_2^{\max} & = & \mathrm{s}_1^{\max} \\ \mathrm{m}_2^{\min} & = & \mathrm{m}_1^{\min}/\mathrm{M}^{\max}(\Delta) & C_2 & = & C_1 \\ \mathrm{m}_2^{\max} & = & \mathrm{m}_1^{\max}/\mathrm{M}^{\min}(\Delta) & & & \end{array}$$

Whenever $\mathrm{M}^{\min}(\Delta) = 0$ the resulting P_2 is defined as above but with $\mathrm{p}_2^{\max} = 1$ and $\mathrm{m}_2^{\max} = 1$.

Normalizing a set of probabilistic polyhedra is then defined as follows

$$\text{normal}(\Delta) \stackrel{\text{def}}{=} \{\text{normal}_\Delta(P) \ : \ P \in \Delta\}$$

6.1.1 Powersets of Intervals and Octagons

Following the probabilistic extensions to the interval and octagon domains described in Section 5.5, we can also define powersets of probabilistic intervals and octagons: $\mathcal{P}_n(\mathbb{I})$ is composed of at most n probabilistic intervals and $\mathcal{P}_n(\mathbb{O})$ is composed of at most n probabilistic octagons. The operations have the same form as those described above.

6.2 Policy Evaluation

Determining the bound on the probability of any state represented by a single probabilistic polyhedron is as 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 simple estimate of the bound can be computed by abstractly adding all the probabilistic polyhedra in the set, and using the $\mathrm{p}^{\max}$ value of the result.

Lemma 29. *If* $\delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta)$ *and* $P_1 = \sum_{P \in \Delta} P$ *then* $\max_\sigma \delta(\sigma) \leq \mathrm{p}_1^{\max}$.

This approach has an unfortunate tendency to increase the max probability bound as one increases the bound on the number of probabilistic polyhedra allowed. A more complicated method, which is used in our implementation, computes a partition of the polyhedra in the set into another set of disjoint polyhedra and determines the maximum probable point among the representatives of each region in the partition. In order to present this method precisely we begin with some definitions.

Definition 30. The *maximum* probability of a state σ according to a probabilistic polyhedron P_1, written $P_1^{\max}(\sigma)$, is as follows.

$$P_1^{\max}(\sigma) \stackrel{\text{def}}{=} \begin{cases} \mathrm{p}_1^{\max} & \text{if } \sigma \in \gamma_{\mathbb{C}}(C_1) \\ 0 & \text{otherwise} \end{cases}$$

Likewise the *maximum* probability of σ according to a probabilistic polyhedron set $\Delta = \{P_i\}$, written $\Delta^{\max}(\sigma)$, is defined as follows.

$$\Delta^{\max}(\sigma) \stackrel{\text{def}}{=} \sum_i P_i^{\max}(\sigma)$$

A mere application of the various definitions allows one to conclude the following.

Remark 31. If $\delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta)$ then for every σ, $\delta(\sigma) \leq \Delta^{\max}(\sigma)$, and therefore $\max_\tau \delta(\tau) \leq \max_\tau \Delta^{\max}(\tau)$.

Notice that in the case of a single probabilistic polyhedron, $P_1^{\max}(\sigma) = P_1^{\max}(\tau)$ for every $\sigma, \tau \in \gamma_{\mathbb{C}}(C_1)$. That is, every supported state has the same maximum probability. On the other hand, this is not the case for sets of probabilistic polyhedra, $\Delta^{\max}(\sigma)$ is not necessarily equal to $\Delta^{\max}(\tau)$, for supported states $\sigma, \tau \in \bigcup_{P_i \in \Delta} \gamma_{\mathbb{C}}(C_i)$, or even for states $\sigma, \tau \in \gamma_{\mathbb{C}}(C_i)$, supported by a single probabilistic polyhedron $P_i \in$

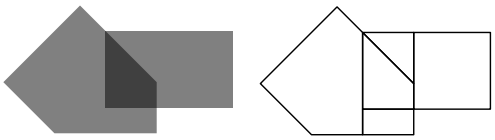


Figure 7: Example of a poly partition of two overlapping convex polyhedra (shaded), resulting in 5 disjoint convex polyhedra (outlined).

Δ. This is the case as there might be one set of probabilistic polyhedra in Δ that supports a state σ, while a different set supports τ.

Taking advantage of the polyhedra domain, we will produce a set of representative points $\{\sigma_j\}_{j=1}^{m}$ with $\max_{j=1}^{m} \Delta^{\max}(\sigma_j) = \max_{\sigma} \Delta^{\max}(\sigma)$. This set will thus let us determine the maximum probability over all points, without having to look at all points. To do this, we first need to define a linear partition.

Definition 32. A *poly partition* of a set of polyhedra $\{P_i\}_{i=1}^{n}$ is another set of polyhedra $\{L_j\}_{j=1}^{m}$, usually of larger size, with the following properties.

1. $\gamma_{\mathbb{C}}(L_i) \cap \gamma_{\mathbb{C}}(L_j) = \emptyset$ for every $i \neq j$.
2. $\cup_{j=1}^{m} \gamma_{\mathbb{C}}(L_j) = \cup_{i=1}^{n} \gamma_{\mathbb{C}}(P_i)$
3. For every i, j, either $\gamma_{\mathbb{C}}(L_i) \subseteq \gamma_{\mathbb{C}}(P_j)$ or $\gamma_{\mathbb{C}}(L_i) \cap \gamma_{\mathbb{C}}(P_j) = \emptyset$.

We call any set $R = \{\sigma_j\}_{j=1}^{m}$ a *representative set* of partition $\mathcal{L} = \{L_j\}_{j=1}^{m}$ when the jth element $\sigma_j \in R$ is in the concretization of the respective element $L_j \in \mathcal{L}$; i.e., $\sigma_j \in \gamma_{\mathbb{C}}(L_j)$.

We can now determine the maximal probability using only representative points, one from each piece of the poly partition.

Lemma 33. $\max_{\sigma \in R} \Delta^{max}(\sigma) = \max_{\sigma} \Delta^{max}(\sigma)$ *where* $\mathcal{L}$ *is a poly partition of* Δ *and* R *is a representative set of* $\mathcal{L}$*.*

Note that the set of representatives R is not unique and the lemma holds for any such set and the maximal state probability is the same, regardless of which set of representatives is used, or even which poly partition is computed. Also note that the process of producing the poly partition would be unnecessary if somehow we kept the regions defined by probabilistic polyhedra in Δ disjoint from each other, as they would already define a poly partition. Doing so would simplify our task here, but would significantly complicate matters in the abstract interpretation of a program, as well as reducing the precision of the final result.

The process of producing a poly partition from a set of polyhedra is achieved via repeated use of the linear partition operation for polyhedra, which splits two polyhedra into disjoint pieces. Our implementation does this in the most naive way possible:

we maintain a bag of polyhedra, splitting any overlapping pairs, until no overlapping regions remain, as in the pseudo-code below.

$$
\begin{array}{l}
\mathit{poly\text{-}partition}(\Phi) \stackrel{\text{def}}{=} \\
\quad \textbf{while } \exists\, C_1, C_2 \in \Phi \;:\; \neg \mathit{isempty}(C_1 \sqcap_{\mathbb{C}} C_2) \\
\quad\quad \Phi \leftarrow (\Phi - \{C_1, C_2\}) \cup (C_1 \swarrow\!\!\!\searrow C_2) \\
\quad \textbf{return } \Phi
\end{array}
$$

We will write $max_{pp}(\Delta)$ for $\max_\sigma \Delta^{\max}(\sigma)$ to make explicit the method with which this value can be computed according to the lemma above.

Notation 34. If Δ is a probabilistic polyhedron set over variables V, and σ is a state over variables $V' \subseteq V$, then $\Delta \wedge \sigma \stackrel{\text{def}}{=} \Delta \wedge B$ where $B = \bigwedge_{x \in V'} x = \sigma(x)$.

Definition 35. Given some probabilistic polyhedron set Δ_1 and statement S where $\langle\!\langle S \rangle\!\rangle\, \Delta_1$ terminates, let $\Delta_2 = \langle\!\langle S \rangle\!\rangle\, \Delta_1$ and $\Delta_3 = \Delta_2 \restriction L = \{P_i'\}$. If for every $\sigma_L \in \gamma_{\mathcal{P}(\mathbb{C})}(\{C_i'\})$ with $\neg \mathit{iszero}(\Delta_2 \wedge \sigma_L)$ we have $\Delta_4 = (\Delta_2 | \sigma_L) \restriction H$ and $max_{pp}(\Delta_4) \leq t$, 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 states that the abstract interpretation just described can be used to soundly determine whether to accept a query.

Theorem 36. *Let δ be an attacker's initial belief. If $\delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta)$ and $\mathit{tsecure}_t(S, \Delta)$, then S is threshold secure for threshold t when evaluated with initial belief δ.*

Note that the process described in computing threshold security involves merging probabilistic polyhedra via the simplification operation (Definition 26); the order in which these polyhedra are combined has no effect on soundness, but could affect precision. We explore the variation possible in the precision due to ordering in Section 7.3. The heuristic used for simplification in the abstract plus operation aims to optimize some of these choices; further optimizations are part of our ongoing work.

7 Experiments

We have implemented an interpreter for the language in Figure 2 based on the probabilistic polyhedra powerset domain as well the simpler probabilistic domains constructed from the interval and octagon base domains. The manipulations of base domain regions are done using the Parma Polyhedra Library [6] (ppl-0.11.2). Counting calculations are done using the LattE [21] tool (LattE macchiato 1.2-mk-0.9.3) in the case of polyhedra and octagons. The trivial counting calculation for the interval domain we implemented ourselves as part of the abstract interpreter, which itself is written in OCaml (3.12.0). While many of the abstract operations distribute over the set of probabilistic regions and thus could be parallelized, our implementation is currently single-threaded.

This section presents an experimental evaluation of our implementation on several benchmark programs. Overall, the use of the octagonal and polyhedral domains results in running times ranging from a few seconds to a few minutes. Compared to

enumeration-based approaches to probabilistic computation, using probabilistic polyhedra improves running times by up to 1–2 orders of magnitude. Intervals do even better, with running times ranging from tens to hundreds of milliseconds, constituting an additional 1–2 orders of magnitude improvement. For our particular experiments, exact precision is reached by all domains if the bound on the number of regions is sufficiently large.

Our experiments were conducted on a Mac Pro with two 2.26 GHz quad-core Xeon processors using 16 GB of RAM and running OS X v10.6.7.

7.1 Benchmark Programs

We applied our implementation to several queries. The timings measure, for each query, the construction of a pre-belief, the probabilistic evaluation of a query, and finally a policy check over all secret variables. We describe each query here, and show the complete source code, pre-belief, and further details in Appendix A.

Birthday We benchmark the birthday queries described in Section 2:

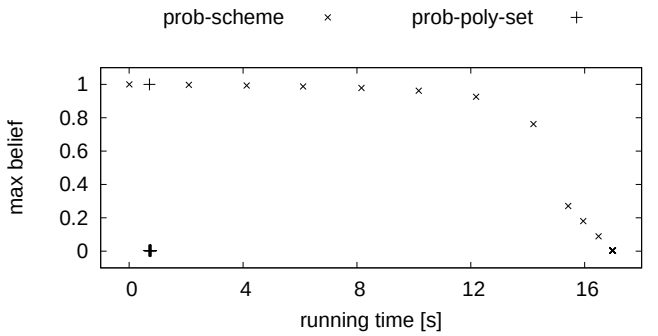
- **bday 1** The first birthday query (Example 1).
- **bday 1+2+special** The sequence of the first two birthday queries (Example 1 and then the same code, but with $today$ increased by 1) followed by the special birthday query (Example 2). Below we refer to this benchmark as the *birthday query sequence*.

We consider *small* and *large* variants of these two queries: the former assumes the birth year ranges from 1956 to 1992, while the latter uses the range 1910 to 2010.

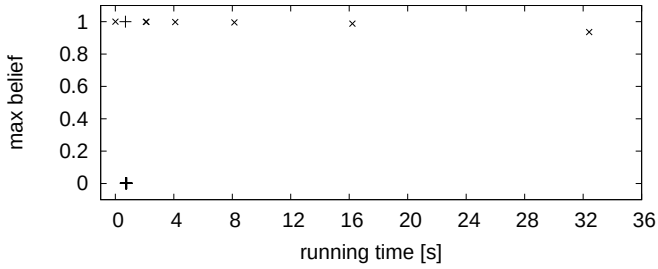
Pizza This query evaluates whether a user might be interested in a local pizza parlor. To do this, the code checks whether the user's location is within a certain square area and whether they match an age or current education criteria most associated with pizza-eating habits (18-28 year old or currently undergrad or above). The modeled secret variables thus include: the user's birth year, the level of school currently being attended, and their address latitude and longitude (scaled by 10^6 and represented as an integer). The last two variables have large magnitudes. The true values used in the benchmark were 39003178 and 76958199 for latitude and longitude, respectively.

Photo This query is a direct encoding of a real targeted advertisement that Facebook includes on their information page [4]. The query itself checks whether the user is female, engaged, and is in a certain age range, indicative of interest in (wedding) photography service. There are three secret variables in this example: gender, relationship status, and birth year.

Travel This query is another adaptation of a Facebook advertisement case study [3], based on a campaign run by a tourism agency. The aim of the query is to determine whether the user lives in one of several relevant countries, speaks English, is over the



(a) birthday query **bday 1 (small)**



(b) birthday query, larger state space - **bday 1 (large)**

Figure 8: Query evaluation comparison

age of 21, and has completed a high level of education. The secret variables thus include four dimensions: the user's country of residence, birth year, highest level of completed education, and primary language.

Is Target Close, Who is Closer These queries were designed to demonstrate the need for relational abstractions. They perform simple (Manhattan) distance calculations over points defined by 2D coordinates. **Is Target Close** checks whether an unknown point is within some distance of a given point and **Who is Closer** determines which of two unknown points is closer to a given point. The nature of these queries is further discussed in Section 7.4 and their full specification is shown in Appendix B.

7.2 Comparison to Enumeration

Figure 8(a) illustrates the result of running the **bday 1 (small)** query using our implementation and one using Probabilistic Scheme [47], which is capable of sound probability estimation after partial enumeration. Each $\times$ plots prob-scheme'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 17 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 one second. The timing measurements are taken to be the medians of 20 runs.

The advantage of our approach is more evident in Figure 8(b) where we use the same program but allow $byear$ to span 1910 to 2010 rather than 1956 to 1992 (this is **bday 1 (large)**). In this case prob-scheme makes little progress even after a minute. Our approach, however, is unaffected by this larger state space and produces the exact maximum belief in less than one second when using only 2 probabilistic polyhedra.

We can push the advantage much further with the use of the simpler interval domain which can compute the exact probabilities in both the smaller and larger birthday examples, using only 2 intervals, in around 0.008 seconds. These benchmarks can be seen in Figure 9, discussed shortly.

7.3 Performance analysis

Figures 9-13 summarize the performance results for all the benchmarks programs and for each of the three base domains. The raw data producing these figures are found in Table 1 in the appendix. The bottom graph of each figure zooms in on the results for the interval domain which can also be seen in the upper graphs.

The timing benchmarks in the figures are based on 20 runs, the median of which is denoted by one of three symbols: a box, a diamond, and a pentagon for the interval, octagon, and polyhedron domains, respectively. The symbols are scaled based on the precision in max probability, relative to exact, the analysis achieves; a tiny dot signifies exact probability and increasing symbol size signifies worsening precision. Note, however, that the sizes of the symbols are *not* proportional to precision. The vertical gray boxes range from the 1st to 3rd quartiles of the samples taken while the vertical lines outside of these boxes represent the full extent of the samples.

We discuss the performance and precision aspects of these results in turn.

Performance. Overall, the use of the octagonal base domain results in slightly improved performance over the polyhedral base domain. The interval domain, however, is much faster than both due to the simpler counting and base domain operations. Though the performance gains from the use of octagons are meager, we note that it is likely they can be greatly improved by implementing a octagon-specialized counting method instead of using the general polyhedron counting tool (LattE).

As the number of base domain regions increases, the running time generally increases, though there are exceptions to this trend. A good example can be seen with intervals in Figure 10. Specifically, when there is no interval set size bound, the analysis takes a less time than with a bound of 40 (and even produces a more precise answer). In such situations the additional computational cost of manipulating a larger number of regions is less than the cost that would have been incurred by having to merge them to maintain some (large) bound.

In the cases of polyhedron and octagon base domains, the running time oddities are due to the difficulty of accurately counting points in complex regions. We measured that, when evaluating the various queries in Figures 9-13, 95% or more of the running

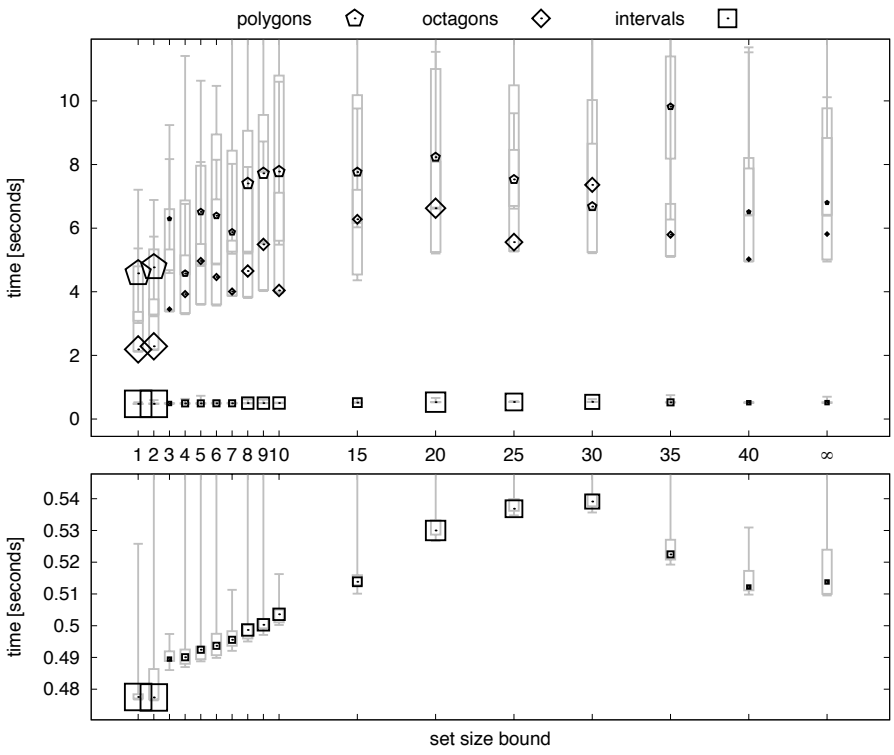


Figure 9: birthday query sequence benchmarks

time is spent in LattE, performing counting. Figure 14 plots the running time of LattE against the number of constraints used to define a polyhedron (we used the polyhedra that arose when evaluating the above queries). Note that the y-axis is a log scale, and as such we can see the running time is super-exponential in the number of constraints. As such, overall running time is sensitive to the complexity of the polyhedra involved, even when they are few in number.

It turns out that when merging to respect the total bound can result in complicated shapes which then, unintuitively, increase the running time. Two very stark examples of this phenomenon are seen for in the pizza and travel queries (Figures 11 and 13 respectively). With a region bound of one, the analysis of both these queries takes much longer than with the bound of two; the large amount of region merging in these instances resulted in a single, but highly complex region. Using octagons in both these instances does not result in complex regions which explains the massive performance improvement. These observations suggest a great deal of performance improvement can be gained by simplifying the polyhedra if they become too complex.

Precision. The figures (and Table 1 in the appendix) generally show the trend that the maximum belief improves (decreases) as the region bound increases, though there are

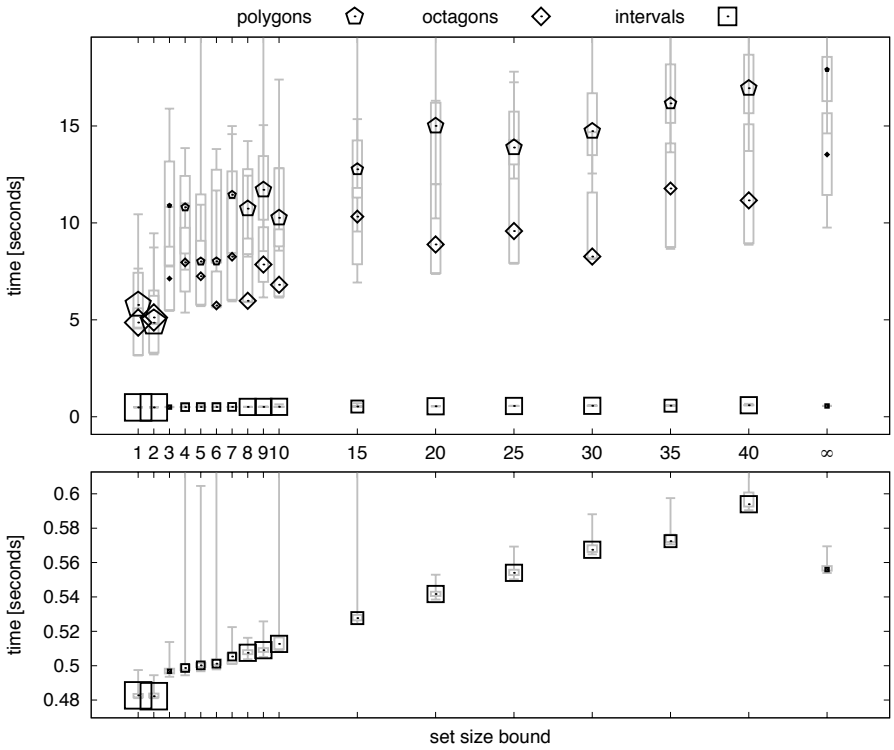


Figure 10: birthday (large) query sequence benchmarks

exceptions. A good example appears in Figure 9 which depicts the performance of the birthday query sequence; with a set size bound of 3, the analysis is able to produce the exact max belief. Allowing 4 probabilistic polyhedra (or intervals, octagons), however, does not produce the exact belief anymore. The same occurs in Figure 10 for the larger state space bday sequence benchmark.

The data also demonstrate that, as expected, the use of polyhedra is more precise then use of octagons which itself is more precise than the use of intervals. For the birthday queries, this difference manifests itself rarely, in particular for the set size bound of 20 and 25 in Figure 9. In the pizza query benchmarks, polyhedra provide a precision advantage over the other two domains when the region set size bound is between 5 and 10. Based on these sample queries, the precision advantage of using polyhedra or octagons over intervals seems insignificant. This is due to a general lack, in these queries, of conditionals that cannot be expressed exactly when using intervals (or octagons) exactly. Section 7.4 shows two queries that demonstrate the precision advantages of polyhedra and octagons more clearly.

Impact of merge order. Another reason for the the lack of a steady improvement of precision as the region bound increases is due to order in which polyhedra are merged.

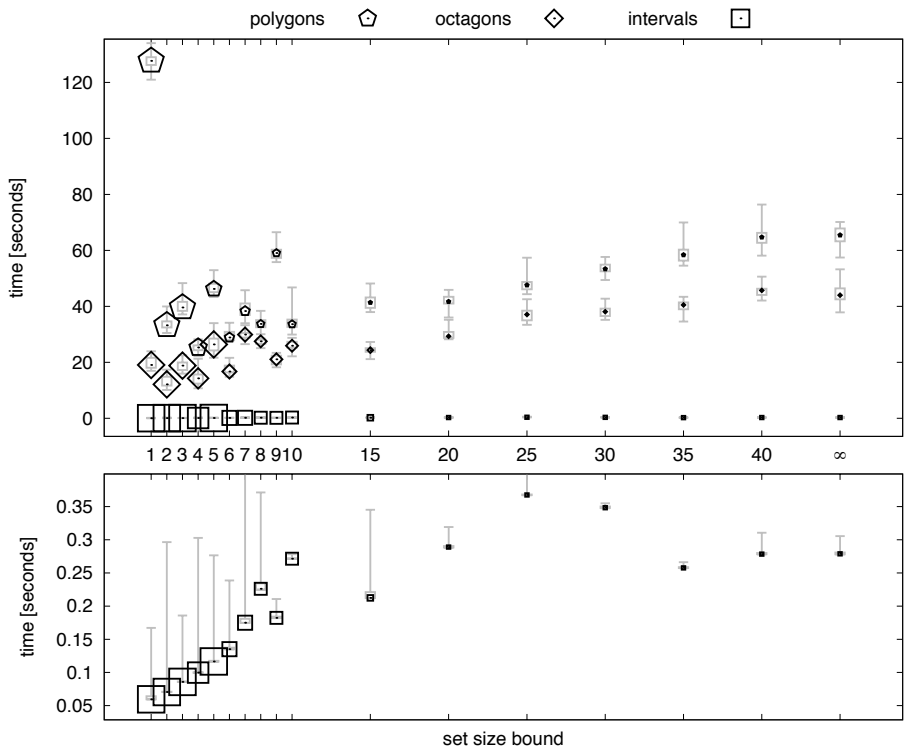


Figure 11: pizza query benchmarks

That is, when simplifying a set of m probabilistic polyhedra to $n < m$ requires that we iteratively merge pairs of polyhedra until the bound is reached. But which pairs should we use? The choice impacts precision. For example, if we have two largely overlapping polyhedra, we would preserve more precision if we merge them rather than merging one of them with some other, non-overlapping one. We used a deterministic strategy for the benchmarks in this section, producing identical precision, though some timing variations. The question is how well might we have done with a better strategy?

To test the precision variation possible due to these arbitrary choices, we analyzed the birthday query sequence, for each interval set size bound, but with 200 different random seeds for randomized merging decisions. The results can be seen in Figure 15. The median max belief is included, as well as the lower and upper quartiles (shaded). Note that we did not use the merging heuristic for the abstract plus operation described in Section 6.1 for these measurements (but we do use it in the results given up to this point).

Naturally there are few arbitrary choices to make when the set size bound is very low, or very high. In the middle, however, there are many possibilities. Turning to the figure, we can see the variation possible is significant except for when the set size bound is at least 36 (at which point there is no merging occurring). For $n \leq 7$ it is

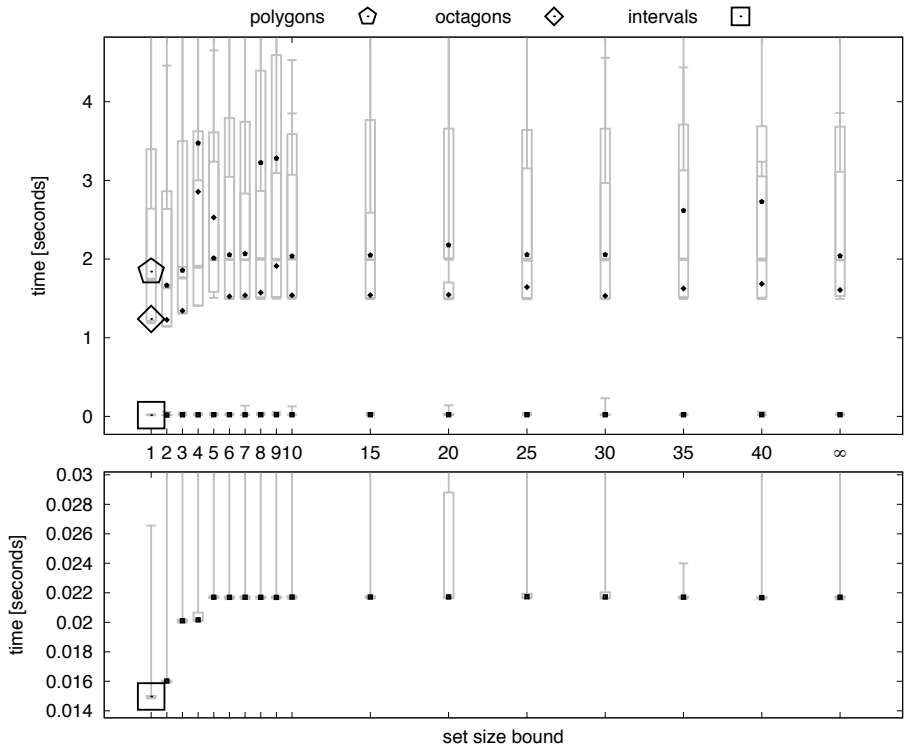


Figure 12: photo query benchmarks

more likely than not to conclude max belief is 1, with 1 being the median. On the other hand, from as little as $n = 2$, it is possible to do much better, even computing the exact belief (lowest sample for $n = 3$). This itself suggests that our heuristic for the merging order in abstract plus is useful, as it managed to result in this exact max belief, against all odds. Also, even though it did not produce exact beliefs for $n > 3$, it did a lot better than the trivial median one would get by performing merging randomly. Nevertheless, the merging order is an aspect of the implementation that has room for improvement, we consider some options in Section 8.

From $n = 8$, the random merging order starts producing non-trivial results, on average. Increasing n further makes it less and less likely to merge poorly; at $n = 30$ for example, no random merge order out of the 200 samples managed to do terribly, all samples had belief below 0.01. Overall, the median max-belief is more or less monotonically improving as n increases as one would expect.

An interesting feature of Figure 15 is the best max-belief achieved for each n (the bottom mark of each column). This quantity seems to be getting worse from $n = 3$ all the way to around $n = 14$ before it starts coming down again. We expect this is due to two counteracting effects:

- Larger powerset size bound allows for more precise representation of distribu-

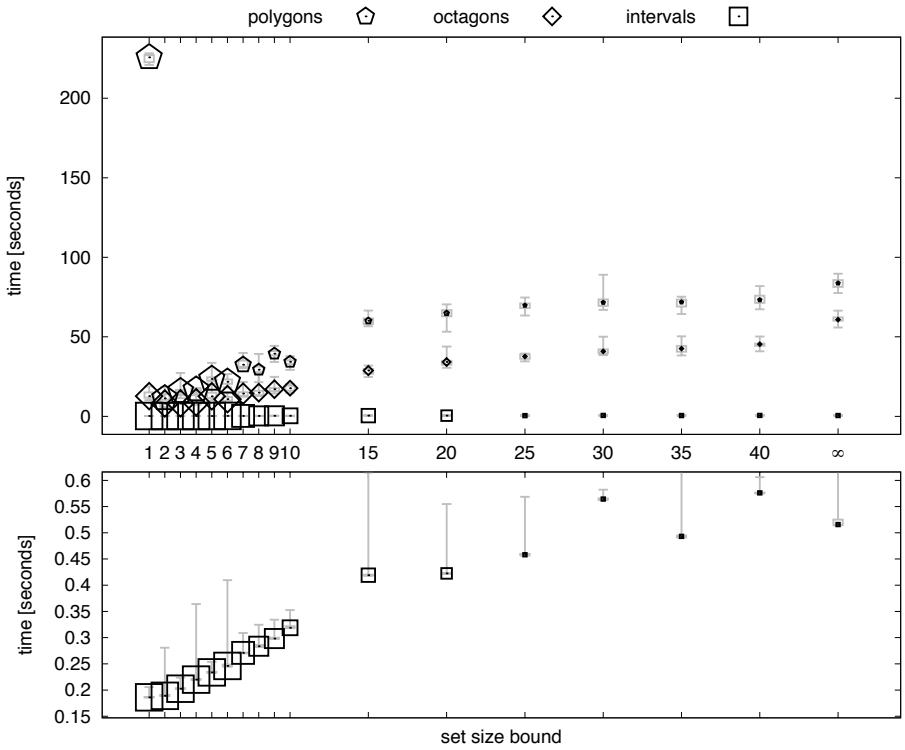


Figure 13: travel query benchmarks

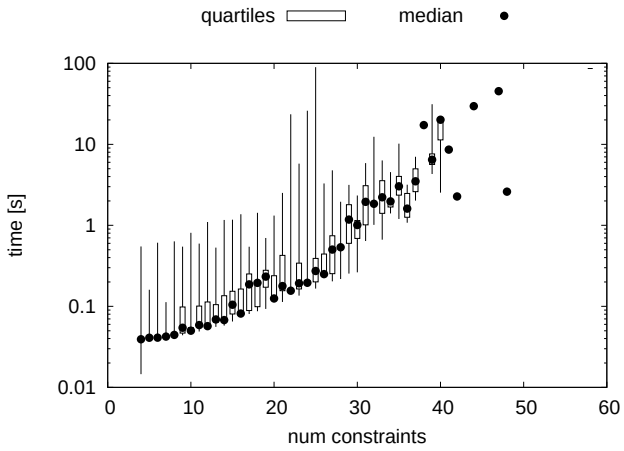


Figure 14: LattE benchmarks

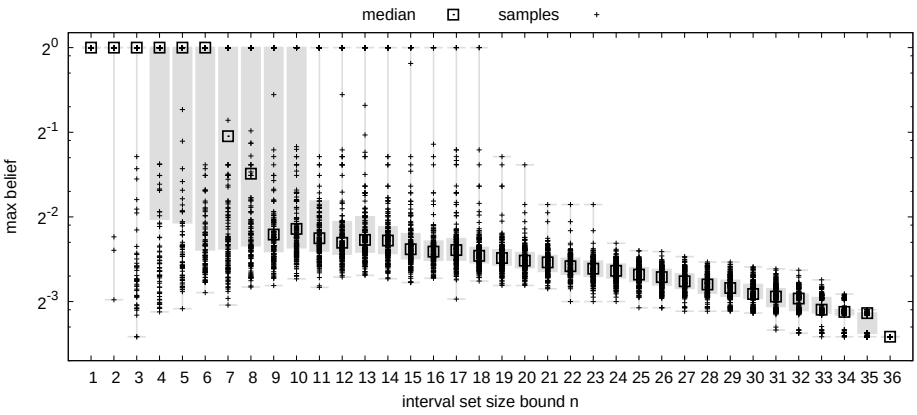


Figure 15: birthday query sequence precision variation

tions, for some merging order.

- It is easier to find a good merging order if there are only a few options. For low values of n, most merging orders are explored in the 200 samples, hence good orders are found.

7.4 Relational queries

The examples presented to this point are handled well by the interval-based abstraction, assuming a sufficient number of intervals are used. The simple interval-based abstraction does not work so well programs that introduce relational constraints on variables. Such constraints can arise due to simple distance computations, which we would expect to be found in many useful queries. We provide examples of such computations here (the full specification of the queries appears in Appendix B).

Consider an initial belief composed of two pairs of 2-dimensional coordinates, specifying the location of two objects: $(x_1, y_1), (x_2, y_2)$. The **Is Target Close** query checks whether the first of the objects is within a specified distance d from a specified coordinate (x, y). Distance is measured using Manhattan distance, that is $|x - x_1| + |y - y_1| \leq d$.

Notice that if the first object is indeed within d of the target coordinate, we learn a relational constrains involving both x_1 and y_1, the coordinates of the first object:

$$\begin{aligned} x_1 + y_1 &\leq d + x + y \wedge \\ x_1 - y_1 &\leq d + x - y \wedge \\ -x_1 + y_1 &\leq d - x + y \wedge \\ -x_1 - y_1 &\leq d - x - y \end{aligned}$$

Intervals are incapable of exact representation of relational constraints like $x_1 + y_1 \leq C$. Octagons, however, are a suitable representation. Thus using our implementation, the **Is Target Close** query fails (that is, overapproximates the probability of all

secret values to be 1) if interval base domain is used, regardless of the bound on number of intervals used, but is exactly handled using only a few octagons.

The next query, **Who Is Closer**, determines which of the two given objects is closer to the target: $|x - x_1| + |y - y_1| \leq |x - x_2| + |y - y_2|$. The truth of this equation implies the disjunction of 4 constraints. One among them is the below.

$$\begin{aligned} x_1 + y_1 + x_2 + y_2 &\leq 2 * x + 2 * y \wedge \\ x_1 - y_1 + x_2 + y_2 &\leq 2 * x \wedge \\ -x_1 + y_1 + x_2 + y_2 &\leq 2 * y \wedge \\ -x_1 - y_1 + x_2 + y_2 &\leq 0 \end{aligned}$$

While x and y can be treated as constants, such a constraint still involves four secret dimensions, x_1, y_1, x_2, y_2 and hence cannot be represented exactly using an octagon, which can express relational constraints of at most two dimensions. For this reason, our implementation fails when octagons are used (no matter their number), whereas the polyhedra-based domain performs exact analysis with just a few polyhedra.

It is important to mention that our implementation does not split regions unless performing a conditioning operation. It might be beneficial in some cases to split an interval into pieces so that their union is capable of better approximating relational constraints. We are considering such options for future work but several aspects of our implementation need to be improved to take advantage of this idea.

8 Discussion

This section considers the some of the tradeoffs, challenges, design alternatives, and possibilities for future work on knowledge-based security policies and probabilistic polyhedra.

8.1 Knowledge-based policies

Employing knowledge-based policies successfully requires maintaining a reasonable estimate of queriers' beliefs. Two difficulties that arise in doing so are (1) establishing the initial belief, and (2) accounting for collusion among queriers. Here we discuss these two issues, and suggest further applications of knowledge-based policies.

Initial belief. For P_1 to enforce a knowledge-based policy on queries by P_2 requires that P_1 estimate P_2's belief about the possible valuations of P_1's secret data. When P_2 has no particular knowledge of P_1's secrets, statistical demographic data can be used; e.g., the US Census and other public sources could be used for personal information. When P_2 might have inside knowledge, P_1 must expend more effort to account for it. For example, in a military setting, estimating a coalition partner's estimate of one's resources could be derived from shared information, and from the likelihood of P_2 having acquired illicit information. Our benchmarks largely considered personal, private information, e.g., gender, age, level of education, etc. This information can be

drawn from public sources (e.g., Facebook demographics [1]). However, the distributions we experimented with were oversimplifications of the actual, reported distributions: they were largely simple, uniform distributions. At present, the syntax of our query language itself only allows specification of a single distribution. Of course, we could permit users to more directly specify distributions of interest in a separate syntax. A more interesting alternative is to define the initial distribution using a probabilistic program employing conditioning based on observations, using the same mechanisms already developed to determining whether to answer queries. This is done in languages like IBAL [45] and Fun [11] and could be adapted to our setting.

If the initial belief estimate is (very) inaccurate, then P_1 risks releasing information to P_2 contrary to his intended policy. To "hedge his bets" against this possibility, he might choose to maintain a set of possible beliefs Δ, rather than a single belief δ, and only release if the threshold was satisfied for every $\delta \in \Delta$. We return to this idea in Section 9.3. In some sense we already do maintain multiple beliefs, as our implementation models/abstracts powersets of probabilistic polyhedra, rather than single polyhedra, for performance reasons. We leave to future work the exploration of the practical implementation of this idea.

Collusion. Assuming that P maintains separate beliefs and thresholds for distinct queriers Q_1 and Q_2, the possibility of collusion arises. Following our motivating example in the Introduction, P's privacy would be thwarted if he shared only his birth day with Q_1 and only his birth year with Q_2 but then Q_1 and Q_2 shared their information. This problem is not unique to our work; the same problem would arise if P used an access control policy to protect birth day and birth year in the same way — the policy will be thwarted if Q_1 and Q_2 pool their information.

A simple approach to preventing this would be to model adversary knowledge globally, effectively assuming that all queriers share their query results; doing so would prevent either Q_1's or Q_2's query (whichever was last). This approach is akin to having a global privacy budget in differential privacy (see Section 9.3) or a single, global access control policy and would obviously harm utility. Moreover, because our query language includes probabilistic if statements, using a global belief introduces an additional, subtle complication: improbable results to non-deterministic queries (like Example 2) can make an adversary more uncertain of the secret value than before seeing the query's output. If it happens that the agents purported to be colluding are in fact not colluding then a global belief might under-approximate a non-colluding adversary's true level of knowledge.

One possible compromise would be to consider both the potential of collusion and non-collusion by tracking a global belief *and* a set of individual beliefs. When considering a query, a rejection would be issued if either belief fails the policy check.

It is important to note that despite the possibility of a decrease certainty due to probabilistic queries, rational adversaries, interested in maximizing their expectation of guessing the secret value, will take all query outputs into account; unlikely query outcomes are detrimental to certainty, but in expectation, potentially deceiving queries increase chances of guessing.

Further applications We have used the goal of decentralizing social networking applications as a motivator for our technique. But knowledge-based policies have other applications as well. Here are four examples.

The first application is *secure multiparty computation* (SMC) [57]. Such computations allow a set of mutually distrusting parties to compute a function f of their private inputs while revealing nothing about their inputs beyond what is implied by the result. Depending on f, however, the result itself may reveal more information than parties are comfortable with. Knowledge-based policies can generalized to this setting: each party X can assess whether the other parties Y_i could have secret values such that f's result would exceed a knowledge threshold about X's secret. We have explored two methods for implementing the generalized knowledge threshold check; details are elsewhere [34].

Another application is to protecting user browsing history. With the advent of "do not track" guidelines that forbid storing cookies to track users' browsing habits across sites [23], service providers have turned to *fingerprinting*, which aims to identify a user based on available browser characteristics [10]. We can protect these attributes with knowledge-based policies, and enforce them by analyzing the javascript code on browsed pages. The flexibility of knowledge-based policies is useful: with access control we would have to choose, in advance, which attributes to reveal, but with knowledge-based policies we can set a threshold on the entire tuple of the most sensitive attributes and a web page can have access to whatever (legal) subset of information it likes, for a better user experience.

A third application is to protect sensing capabilities. In particular, we can treat sensor readings as samples from a random process parameterized by confidential characteristics of the sensor. Each reading provides information about these parameters, in addition to information from the reading itself. For example, suppose we want to share mobility traces for traffic planning, but want to protect individuals' privacy. We can view each trace as a series of samples from a random process that determines the individual's location based on periodic factors, like previous location, time of day, day of week, etc. [51]. We can define the sampling function in our simple language, involving probabilistic choices over the hidden parameters. Belief tracking can be used to narrow down the set of possible parameters to the model that could have produced the observed traces; if the observer's certainty about these parameters exceeds the threshold, then the trace elements are not revealed. Note that trace obfuscation techniques are easily accounted for—they can simply be composed with the sampling function and reasoned about together.

Finally, we observe that we can simply track the *amount* of released information due to an interaction as a degenerate case of enforcing knowledge-based policies. In particular, we can set the pre-belief over some sensitive information to be the uniform distribution, and set the threshold to be 1. In this case, we will always answer any query, and at any time we can compute the entropy of the current belief estimate to calculate the (maximum) number of bits leaked. This information may be used to evaluate the susceptibility to attack by gauging the confidence an attacker might have in their belief about secret information. It can be used to gauge what aspects of secret information were of interest to the attacker, giving hints to as their motive, or maybe even differentiating an honest querier from a malicious one. Tracking information in

this manner is less expensive than enforcing threshold policies directly, since not all possible outputs need to be considered, and carries no risk of a mis-estimate of the pre-belief: the number of reported leaked bits will be conservatively high.

8.2 Improving the Performance of Probabilistic Polyhedra

While the performance of probabilistic polyhedra compares favorably to alternative approaches, it can nevertheless be improved; this will be important for applying it to the deployment scenarios listed above. Here we present several ideas for improving the implementation that we hope to explore in future work.

Handling nominal values. Our probabilistic domains are based on polyhedra, octagons, and intervals, which are best for analyzing programs with *numeric* variables, that contain linear conditionals. Most of the variables in the benchmark programs were of the numeric variety. However, some were *nominal*, e.g., the variable encoding a user's language in the travel query, and these are unordered. Some variables are nominal but partially ordered, like education level in the pizza query. While we can encode nominal values as integers, they may be better handled via other means, perhaps even via naïve enumeration. Handling of large quantities of nominal values could be performed symbolically using some of the tools used by other probabilistic languages: [12], graphical models [38], or factor graphs [11, 45]. Ideally abstract domains like used in our system and ones better suited for nominal values, could be integrated (i.e. via reduced product [18]) to effectively process programs that contain both types of variables.

Region splitting. As the performance experiments in the previous section show, intervals can be far more efficient than polyhedra. While a single interval may be more imprecise than a single polyhedron, an interesting idea is consider *splitting* a polyhedron into many intervals, aiming for the best of both worlds. The simplest means of implementing this idea is to modify the handling of the uniform statement for the powerset domains to result not in one, but several intervals. Though a single interval is sufficient to exactly represent distributions produced by uniform, it would be insufficient if, later, the program introduces relations not representable by intervals.

The challenge is to find the right tradeoff—increasing the number of intervals will slowly degrade performance and may hurt precision if we use an unfortunate merging order at join points, as seen in Figure 15. Heuristically picking the right merge order is a known challenge in abstract interpretation-based analyses.

Querier belief tracking. Recall a vital property of our knowledge-based policy enforcement is *simulatability*: the adversary has (or is allowed to have) all the information necessary to decide the policy that governs its access, without interacting with the data owner. As such, the computation of policy enforcement could, in theory, by done by the querier. Naturally, they should not be trusted in this regard completely. One general direction is to imagine the querier performing the entire computation, and providing proof of the outcome, via something like proof-carrying code [43]. Alternatively, the

querier could provide hints to the data owner to improve the speed of his computation. For example, the querier could determine the optimal choices for merge order, and send a digest of these choices.

9 Related work

We consider four areas of work related to ours: systems aimed at protecting access to users' private data; methods for quantifying information flow from general-purpose programs; methods for privacy-preserving computation, most notably *differential privacy*; and finally approaches to performing general-purpose probabilistic computation.

9.1 Privacy enhancing network services

Several recent proposals have considered alternative service architectures for better ensuring the privacy of individual participants. These systems tend to enforce access control policies. For example, PrPl [50] is a decentralized social networking infrastructure aimed to permit participants to participate in social networking without losing ownership of their data. The system uses *Personal-Cloud Butler* services to store data and enforce access control policies. Persona [8] uses a centralized Facebook-style service, but users can store personal data on distributed storage servers that use attribute-based encryption. Access to data is granted to those parties who have the necessary attribute keys. XBook [52] mediates social network data accesses by third-party application extensions. Privad [28] is a privacy-preserving advertising platform that, like our running examples, runs the ad selection algorithm over the user's data on the user's platform. Privad *presumes* that outputs of ad selection reveal little about the inputs.

Knowledge-based security policies generalize access control policies: if we maintain a belief estimate for each principal P, then the equivalent of granting P access to data d is to just set P's knowledge threshold for d to 1; for principals R who should not have access, the threshold for d is 0. Knowledge-based policies afford greater flexibility by allowing *partial* access, i.e., when a threshold less than 1. Moreover, as mentioned in the introduction, we can set a policy on multiple data items, and thus grant more access to one item than another (e.g., birth day and/or year) as long as knowledge of the aggregate is controlled.

9.2 Quantified information flow

Others have considered how an adversary's knowledge of private data might be informed by a program's output. Clark, Hunt, and Malacaria [13] define a static analysis that bounds the secret information a straight-line program can leak in terms of equivalence relations between the inputs and outputs. Backes et al. [7] automate the synthesis of such equivalence relations and quantify leakage by computing the exact size of equivalence classes. Köpf and Rybalchenko [32] extend this approach, improving its scalability by using sampling to identify equivalence classes and using under- and over-approximation to obtain bounds on their size. Mu and Clark [42] present a sim-

ilar analysis that uses over-approximation only. In all cases, the inferred equivalence classes can be used to compute entropy-based metrics of information leakage.

We differ from this work in two main ways. First, we implement a different security criterion. The most closely related metric is *conditional vulnerability* V as proposed by Smith [53], which can be defined using our notation as follows:

Definition 37. Let $\delta' = [\![S]\!]\delta$, where δ is the model of the querier's initial belief.

Then query S is *vulnerability threshold secure* iff for

$$V = \sum_{\sigma_L \in \mathit{support}(\delta' \restriction L)} (\delta' \restriction L)(\sigma_L) \cdot \max_{\sigma_H \in \mathbf{State}_H} (\delta' | \sigma_L \restriction H)(\sigma_H)$$

we have $V \leq t$ for some threshold t.

The above definition is an *expectation* over all possible outputs σ_L, so unlikely outputs have less influence. Our notion of threshold security (Definition 3), in contrast, is equivalent to a bound on the following quantity:

$$V^* = \max_{\sigma_L \in \mathit{support}(\delta' \restriction L)} \max_{\sigma_H \in \mathbf{State}_H} (\delta' | \sigma_L \restriction H)(\sigma_H)$$

This notion of security is strictly stronger as it considers each output individually: if *any* output, however unlikely, would increase knowledge beyond the threshold, the query would be rejected. For example, recall the query from Example 1 where the secret data $bday$ is (assumed by the querier to be) uniformly distributed; call this query S_1. According to Definition 37, the minimum acceptable threshold for which the query would be considered safe is $V = \frac{7}{365} * \frac{1}{7} + \frac{358}{365} * \frac{1}{358} = 2/365 \approx 0.005$, whereas according to Definition 3, the minimum threshold is $V^* = 1/7 \approx 0.143$.

The idea of strengthening of an entropy measure by eliminating the expectation has been briefly considered by Köpf and Basin [31]. In their work this measure is proposed as a stronger alternative, a choice ultimately dependent on the application. In our case, however, the worst-case measure is absolutely necessary in order to prevent the leakage of information when rejecting a query.

The other distinguishing feature of our approach is that we keep an on-line model of adversary knowledge according to prior, actual query results. Once a query is answered, the alternative possible outputs of this query no longer matter. To see the benefit of this query-by-query approach, consider performing query S_1 followed by a query S_2 which uses the same code as S_1 (from Example 1) but has $today = 265$. With our system and $bday = 270$ the answer to S_1 is False and with the revised belief the query S_2 will be accepted as below threshold $t_d = 0.2$. If instead we had to model this pair of queries statically they would be rejected because (under the assumption of uniformity) the pair of outputs True,True is possible and implies $bday \in \{265, 266\}$ which would require $t_d \geq 0.5$. Our approach also inherits from the belief-based approach the ability to model a querier who is misinformed or incorrect, which can arise following the result of a probabilistic query or because of a change to the secret data between queries [14]. We note these advantages come at the cost of maintaining on-line belief models.

Our proposed abstract domains are useful beyond the application of belief-based threshold security; e.g., they could be used to model uncertainty off-line (as in the

above work) rather than beliefs on-line, with the advantage that they are not limited to uniform distributions (as required by [7, 32]).

McCamant and Ernst's FLOWCHECK tool [37] measures the information released by a particular execution. However, it measures information release in terms of *channel capacity*, rather than remaining uncertainty which is more appropriate for our setting. For example, FLOWCHECK would report a query that tries to guess a user's birthday leaks one bit regardless of whether the guess was successful, whereas the belief-based model (and the other models mentioned above) would consider a failing guess to convey very little information (much less than a bit), and a successful guess conveying quite a lot (much more than a bit).

9.3 Differential privacy

A recent thread of research has aimed to enforce the privacy of database queries. Most notably, Dwork and colleagues have proposed *differential privacy* [24]: a differentially private query Q over a database of individuals' records is a randomized function that produces roughly the same answer whether a particular individual's data is in the database or not. An appealing feature of this definition is that the querier's knowledge is not considered directly; rather, we are merely assured that Q's answer will not differ by much whether a particular individual is in the database or not. On the other hand, differentially private databases require that individuals trust the database curator, a situation we would prefer to avoid, as motivated in the introduction.

We can compare differential privacy to our notion of threshold security by recasting the differential privacy definition into the form of an *adversarial privacy* definition, which was formulated by Rastogi and Suciu to compare their own privacy definition on databases to differential privacy [48]. Rastogi and Suciu's definition is in terms of the presence or absence of a record in a database, whereas our notion is defined on secret states over variables in H. To bridge this gap suppose that, without loss of generality, variables $x \in H$ range over $\{0, 1\}$, and we say a variable x is "in a state" σ when $\sigma(x) = 1$. (We can always encode an integer i as a sequence of bits.) Define $\delta(\{x_1, ..., x_n\})$ to be the probability of all states σ such that $\sigma(x_i) = 1$ for all $1 \leq i \leq n$; i.e.,

$$\delta(\{x_1, ..., x_n\}) \stackrel{\text{def}}{=} \sum_{\sigma | \sigma(x_1)=1 \wedge ... \wedge \sigma(x_n)=1} \delta(\sigma)$$

Then we can state adversarial privacy as follows.

Definition 38 (Adversarial Privacy)**.** Given a query statement S, let $O \subseteq \mathbf{State}_{L \cup H}$ be the set of all possible output states of S (over some set of secret variables H and public variables L), and $\Delta_H \subseteq \mathbf{Dist}_H$ be a set of distributions over secret states. We say that program S is *ε-adversarially private* w.r.t. Δ_H iff for all adversary beliefs $\delta_H \in \Delta_H$, all secret variables $x \in H$, and all possible outputs $\sigma_o \in O$, that $\delta_H(\{x\}) \leq e^{\varepsilon}\, \delta'_H(\{x\})$. Here, $\delta'_H = (\llbracket S \rrbracket(\delta_H \times \dot{\sigma}_L))|\sigma_o \upharpoonright H$; that is, it is the revised adversary belief after seeing that public output state of S is σ_o.

This definition says that query S is acceptable when the output of S increases only by a small multiplicative factor an adversary's certainty that some variable x is in the

secret input state, for all adversaries whose prebeliefs are characterized by Δ_H. Rastogi and Suciu show that defining Δ_H to be the class of *planar, total sub-modular* (or $PTLM$) distributions renders an adversarial privacy definition equivalent to differential privacy. Among other conditions, $PTLM$ distributions require fixed-sized databases (which comes by construction in our setting), and require the probability of one tuple's presence to not be positively correlated with another's. We can transfer this latter condition to our setting by requiring that for all $\delta \in \Delta_H$, and all $x_1, x_2 \in H$, we have $\delta(\{x_1\})\delta(\{x_2\}) \leq \delta(\{x_1, x_2\})$.

Comparing Definition 38 to Definition 3 we can see that the former makes a *relative* assertion about the increased certainty of any one of a *set* of possible adversarial beliefs, while the latter makes an *absolute* assertion about the certainty of a *single* adversarial belief. The absolute threshold of our definition is appealing, as is the more flexible prebelief, which allows positive correlations among state variables. On the other hand, our definition assumes we have correctly modeled the adversary's belief. While this is possible in some cases, e.g., when demographic information is available, differential privacy is appealing in that it is robust a much larger number of adversaries, i.e., all those whose prebeliefs are in $PTLM$.

Unfortunately, this strong privacy guarantee seems to come at strong cost to utility. For example, deterministic queries are effectively precluded, eliminating some possibly useful applications. For the application of social networks, Machanavajjhala et al. [33] have shown that good private social recommendations are feasible only for a small subset of the users in the social network or for a lenient setting of privacy parameters. We can see the utility loss in our motivating scenario as well. Consider the birthday query from Example 1. Bob's birthday being/not being in the query range influences the output of the query only by 1 (assuming yes/no is 1/0). One could add an appropriate amount of (Laplacian) noise to the query answer to hide what the true answer was and make the query differentially private. However, this noise would be so large compared to the original range $\{0, 1\}$ that the query becomes essentially useless—the user would be receiving a birthday announcement most days.[4]

By contrast, our approach permits answering (deterministic or randomized) queries exactly if the release of information is below the threshold. Moreover, by tracking beliefs between queries, there is no artificial limit on the number of queries since we can track the released information exactly. Differential privacy imposes an artificial limit on the number of queries because the post-query, revised beliefs of the adversary are not computed. Rather, each query conservatively adds up to ε to an adversary's certainty about a tuple, and since the precise release is not computed, we must assume the worst. Eventually, performing further queries will pose too great of a risk.

As mentioned in the previous section, we could gain some of the privacy advantages of differential privacy by modeling a (larger) set of beliefs rather than a single one. Our abstractions $\mathbb{P}$ and $\mathcal{P}_n(\mathbb{P})$ already model sets of distributions, rather than a single distribution, so it remains interesting future work to exploit this representation toward increasing privacy.

[4]By our calculations, with privacy parameter $\varepsilon = 0.1$ recommended by Dwork [24], the probability the query returns the correct result is approximately 0.5249.

9.4 Probabilistic programming

The core of our methodology relies on probabilistic computation. A variety of tools exist for specifying random processes as computer programs and performing inference on them.

Implementations based on partial sampling [26, 44] or full enumeration [47] of the state space are unsuitable in our setting. Such tools are either too inefficient or too imprecise. Works based on smarter representations of probability distributions are promising alternatives. Projects based on algebraic decision diagrams [12], graphical models [38], and factor graphs [11, 45] translate programs into convenient structures and take advantage of efficient algorithms for their manipulation or inference.

Our implementation for probabilistic computation and inference differs from existing works in two main ways. Firstly, we are capable of approximation and hence can trade off precision for performance, while maintaining soundness in terms of a strong security policy. The second difference is the nature of our representation of probability distributions. Our work is based on numerical abstractions: intervals, octagons, and polyhedra. These abstractions are especially well suited for analysis of imperative programs with numeric variables and linear conditionals. Other probabilistic languages might serve as better choices when nominal, rather than numeric, variables are used in queries. The comparative study of the power and effectiveness of various representations in probabilistic computation is a topic of our ongoing research.

We are not the first to propose probabilistic abstract interpretation. Monniaux [41] gives an abstract interpretation for probabilistic programs based on over-approximating probabilities of points. Di Pierro describes abstract interpretation for probabilistic lambda calculus [22]. Smith [54] describes probabilistic abstract interpretation for verification of quantitative program properties. Cousot [20] unifies these and other probabilistic program analysis tools. However, these do not deal with sound distribution conditioning, the unique element of our approach, which is crucial for belief-based information flow analysis.

10 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 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 domains of *powersets of probabilistic polyhedra, octagons, and intervals*. Compared to typical approaches to implementing belief revision, our implementation using this domain is more efficient and scales better.

Acknowledgments The authors are grateful to Jeff Foster, Nataliya Guts, the CSF'11 anonymous referees, and especially Boris Köpf for their helpful comments on drafts of this paper.

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorised to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon. Support was also provided by National Science Foundation grants 0905419 and 0910530.

References

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

[2] Statement of rights and responsibilities. `http://www.facebook.com/terms.php`, October 2010.

[3] Facebook ads: A guide to targeting and reporting. `http://www.openforum.com/articles/facebook-ads-a-guide-to-targeting-and-reporting-adele-cooper`, 2011.

[4] Facebook ads: Case studies. `http://www.facebook.com/advertising/?campaign_id=402047449186`, 2011.

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

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

[7] Michael Backes, Boris Köpf, and Andrey Rybalchenko. Automatic discovery and quantification of information leaks. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2009.

[8] Randy Baden, Adam Bender, Neil Spring, Bobby Bhattacharjee, and Daniel Starin. Persona: an online social network with user-defined privacy. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2009.

[9] Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. Widening operators for powerset domains. *International Journal on Software Tools for Technology Transfer*, 8(4):449–466, 2006.

[10] Károly Boda, Ádám Máté Földes, Gábor György Gulyás, and Sándor Imre. User tracking on the web via cross-browser fingerprinting. In *Proceedings of the Nordic Conference in Secure IT Systems (NordSec)*, 2011.

[11] Johannes Borgström, Andrew D. Gordon, Michael Greenberg, James Margetson, and Jurgen Van Gael. Measure transformer semantics for bayesian machine learning. In *Proceedings of the European Symposium on Programming (ESOP)*, 2011.

[12] Guillaume Claret, Sriram K. Rajamani, Aditya V. Nori, Andrew D. Gordon, and Johannes Borgstroem. Bayesian inference for probabilistic programs via symbolic execution. Technical Report MSR-TR-2012-86, Microsoft Research, 2012.

[13] David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantitative information flow, relations and polymorphic types. *Journal of Logic and Computation*, 15:181–199, 2005.

[14] Michael R. Clarkson, Andrew C. Myers, and Fred B. Schneider. Quantifying information flow with beliefs. *Journal of Computer Security*, 17(5):655–701, 2009.

[15] Agostino Cortesi. Widening operators for abstract interpretation. In *Proceedings of the Sixth IEEE International Conference on Software Engineering and Formal Methods*, pages 31–40, November 2008.

[16] Patrick Cousot and Radhia Cousot. Static determination of dynamic properties of programs. In *Proceedings of the Second International Symposium on Programming*, 1976.

[17] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the ACM SIGPLAN Conference on Principles of Programming Languages (POPL)*, 1977.

[18] Patrick Cousot and Radhia Cousot. Systematic design of program analysis frameworks. In *Proceedings of the Sixth International Symposium on Programming*, 1979.

[19] Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the ACM SIGPLAN Conference on Principles of Programming Languages (POPL)*, 1978.

[20] Patrick Cousot and Michael Monerau. Probabilistic abstract interpretation. In *Proceedings of the European Symposium on Programming (ESOP)*, 2012.

[21] Jesus A. De Loera, David Haws, Raymond Hemmecke, Peter Huggins, Jeremy Tauzer, and Ruriko Yoshida. Latte. `http://www.math.ucdavis.edu/latte`, 2008.

[22] Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky. Probabilistic lambda-calculus and quantitative program analysis. *Journal of Logic and Computation*, 15(2):159–179, 2005.

[23] Do not track. `https://www.eff.org/issues/do-not-track`, 2012.

[24] Cynthia Dwork. A firm foundation for private data analysis. *Communications of the ACM*, 54(1):86–95, 2011.

[25] Philippe Golle. Revisiting the uniqueness of simple demographics in the us population. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES)*, 2006.

[26] Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. Church: a language for generative models. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2008.

[27] Saikat Guha, Bin Cheng, and Paul Francis. Challenges in measuring online advertising systems. In *Proceedings of the Internet Measurement Conference (IMC)*, 2010.

[28] Saikat Guha, Bin Cheng, and Paul Francis. Privad: Practical privacy in online advertising. In *Proceedings of the Symposium on Networked Systems Design and Implementation (NSDI)*, March 2011.

[29] Krishnaram Kenthapadi, Nina Mishra, and Kobbi Nissim. Simulatable auditing. In *Proceedings of the ACM SIGMOD Symposium on Principles of Database Systems (PODS)*, 2005.

[30] Oleg Kiselyov and Chung-Chieh Shan. Embedded probabilistic programming. In *Proceedings of the Working Conference on Domain Specific Languages (DSL)*, 2009.

[31] Boris Köpf and David Basin. An Information-Theoretic Model for Adaptive Side-Channel Attacks. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2007.

[32] Boris Köpf and Andrey Rybalchenko. Approximation and randomization for quantitative information-flow analysis. In *Proceedings of the IEEE Computer Security Foundations Symposium (CSF)*, 2010.

[33] Ashwin Machanavajjhala, Aleksandra Korolova, and Atish Das Sarma. Personalized social recommendations: accurate or private. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 2011.

[34] Piotr Mardziel, Michael Hicks, Jonathan Katz, and Mudhakar Srivatsa. Knowledge-oriented secure multiparty computation. In *Proceedings of the ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS)*, 2012.

[35] Piotr Mardziel, Stephen Magill, Michael Hicks, and Mudhakar Srivatsa. Dynamic enforcement of knowledge-based security policies. In *Proceedings of the IEEE Computer Security Foundations Symposium (CSF)*, 2011.

[36] Piotr Mardziel, Stephen Magill, Michael Hicks, and Mudhakar Srivatsa. Dynamic enforcement of knowledge-based security policies. Technical Report CS-TR-4978, University of Maryland Department of Computer Science, 2011.

[37] Stephen McCamant and Michael D. Ernst. Quantitative information flow as network flow capacity. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2008.

[38] Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L. Ong, and Andrey Kolobov. Blog: Probabilistic models with unknown objects. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2005.

[39] Antoine Miné. The octagon abstract domain. In *Proceedings of the Working Conference on Reverse Engineering (WCRE)*, 2001.

[40] David Monniaux. Abstract interpretation of probabilistic semantics. In *Proceedings of the Static Analysis Symposium (SAS)*, 2000.

[41] David Monniaux. *Analyse de programmes probabilistes par interprétation abstraite*. Thèse de doctorat, Université Paris IX Dauphine, 2001.

[42] Chunyan Mu and David Clark. An interval-based abstraction for quantifying information flow. *Electronic Notes in Theoretical Computer Science*, 253(3):119–141, 2009.

[43] George C. Necula. Proof-carrying code. In *Proceedings of the ACM SIGPLAN Conference on Principles of Programming Languages (POPL)*, pages 106–119, 1997.

[44] Sungwoo Park, Frank Pfenning, and Sebastian Thrun. A probabilistic language based on sampling functions. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 31(1):4:1–4:46, 2008.

[45] Avi Pfeffer. The design and implementation of IBAL: A general-purpose probabilistic language. In Lise Getoor and Benjamin Taskar, editors, *Statistical Relational Learning*. MIT Press, 2007.

[46] Corneliu Popeea and Wei-ngan Chin. Inferring disjunctive postconditions. In *Proceedings of the Asian Computing Science Conference (ASIAN)*, 2006.

[47] Alexey Radul. Report on the probabilistic language Scheme. In *Proceedings of the Dynamic Languages Symposium (DLS)*, 2007.

[48] Vibhor Rastogi, Michael Hay, Gerome Miklau, and Dan Suciu. Relationship privacy: output perturbation for queries with joins. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, 2009.

[49] Alfredo Rial and George Danezis. Privacy-preserving smart metering. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES)*, 2011.

[50] Seok-Won Seong, Jiwon Seo, Matthew Nasielski, Debangsu Sengupta, Sudheendra Hangal, Seng Keat Teh, Ruven Chu, Ben Dodson, and Monica S. Lam. PrPl: a decentralized social networking infrastructure. In *Proceedings of the Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, 2010. Invited Paper.

[51] Reza Shokri, George Theodorakopoulos, Jean-Yves Le Boudec, and Jean-Pierre Hubaux. Quantifying Location Privacy. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 247–262, 2011.

[52] Kapil Singh, Sumeer Bhola, and Wenke Lee. xBook: Redesigning privacy control in social networking platforms. In *Proceedings of the USENIX Security Symposium*, 2009.

[53] Geoffrey Smith. On the foundations of quantitative information flow. In *Proceedings of the Conference on Foundations of Software Science and Computation Structures (FoSSaCS)*, 2009.

[54] Michael J. A. Smith. Probabilistic abstract interpretation of imperative programs using truncated normal distributions. *Electronic Notes in Theoretical Computer Science*, 220(3):43–59, 2008.

[55] Latanya Sweeney. Simple demographics often identify people uniquely. Technical Report LIDAP-WP4, Carnegie Mellon University, School of Computer Science, Data Privacy Laboratory, 2000.

[56] David Worthington. Mypace user data for sale. *PC World on-line*, March 2010. `http://www.pcworld.com/article/191716/myspace_user_data_for_sale.html`.

[57] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Proceedings of the Annual Symposium on Foundations of Computer Science*, pages 162–167, 1986.

A Example queries

We provide here the queries and prebeliefs we used for the experiments in Section 7. The queries are described as functions from some set of inputs to some set of outputs. The exact syntax is as follows.

$$\begin{array}{l}\mathsf{querydef}\ queryname\ in_1 \cdots in_n \to out_1 \cdots out_m\ : \\ \quad querybody\end{array}$$

Query definitions that follow sometimes include pre-processing statements of the form:

$$\#\mathsf{define}\ x\ =\ exp$$

Such statements result in any occurrence of a variable x being replaced by the expression exp. This is used for convenience to refer to common expressions without actually requiring the analysis to track additional variables/dimensions.

To specify a query invocation we use the following syntax.

$$\begin{array}{l}\mathsf{query}\ querymame\ : \\ \quad in_1\ :=\ val_1; \\ \quad \ldots \\ \quad in_n\ :=\ val_n\end{array}$$

Each experiment must also specify the values of the secrets being queried, and the querier's prebelief. Each specification is a merely a program that sets the values of these variables. For the actual secret values this program begins with the declaration secret; the resulting state of executing program is taken to be the secret state. The program to set the prebelief begins belief and has the same format; note that this program will use pif or uniform $x\ n_1\ n_2$ to give secrets different possible values with different probabilities.

We now give the content of the queries used in the experiments.

A.1 Birthday

For the small stateset size birthday experiments we used the following secret and prebelief.

$$\begin{array}{l}\mathsf{secret}: \\ \quad s\_bday\ :=\ 270\ ; \\ \quad s\_byear\ :=\ 1980\end{array}$$

$$\begin{array}{l}\mathsf{belief}: \\ \mathsf{uniform}\ s\_bday\ 0\ 364\ ; \\ \mathsf{uniform}\ s\_byear\ 1956\ 1992\end{array}$$

The two queries used were as follows.

$$\begin{array}{l}\mathsf{querydef}\ bday\ :\ c\_day\ \rightarrow\ out \\ \mathsf{if}\ s\_bday \geq c\_day \wedge c\_day + 7 > s\_bday\ \mathsf{then} \\ \quad out\ :=\ 1 \\ \mathsf{else} \\ \quad out\ :=\ 0\end{array}$$

$$\begin{array}{l}\mathsf{querydef}\ spec\ :\ c\_year\ \rightarrow\ out \\ \#\mathsf{define}\ age\ =\ c\_year - s\_byear \\ \mathsf{if}\ age = 10\ \vee\ age = 20\ \vee\ age = 30\ \vee\ age = 40 \\ \quad \vee\ age = 50\ \vee\ age = 60\ \vee\ age = 70\ \vee\ age = 80 \\ \quad \vee\ age = 90\ \vee\ age = 100\ \mathsf{then} \\ \quad out\ :=\ 1 \\ \mathsf{else} \\ \quad out\ :=\ 0\ ; \\ \mathsf{pif}\ 1/10\ \mathsf{then} \\ \quad out\ :=\ 1\end{array}$$

The statistics described in comparison to the enumeration approach (Section 7.2) include the time spent processing this initial setup as well as time processing one birthday query. Figure 10 benchmarks two bday queries followed by a spec year query.

- A single bday query alone.

$$\begin{array}{c}\mathsf{query}\ bday\ :\\ c\_day\ :=\ 260\end{array}$$

- Two bday queries followed by a spec query.

$$\begin{array}{c}\mathsf{query}\ bday\ :\\ c\_day\ :=\ 260\\ \mathsf{query}\ bday\ :\\ c\_day\ :=\ 261\\ \mathsf{query}\ spec\ :\\ c\_year\ :=\ 2011\end{array}$$

A.2 Birthday (large)

For the larger statespace birthday example we used the following secret and prebelief generators.

$$\begin{array}{c}\mathsf{secret}:\\ s\_bday\ :=\ 270\ ;\\ s\_byear\ :=\ 1980\\ \mathsf{belief}:\\ \mathsf{uniform}\ s\_bday\ 0\ 364\ ;\\ \mathsf{uniform}\ s\_byear\ 1910\ 2010\end{array}$$

The queries used were identical to the ones for the smaller statespace birthday example. For our benchmarks we analyzed the vulnerability of the pair of secrets s_bday, s_byear.

A.3 Pizza

The pizza example is slightly more complicated, especially in the construction of the prebelief. This example models a targeted Facebook advertisement for a local pizza shop. There are four relevant secret values. The level of school currently being attended by the Facebook user is given by `s_in_school_type`, which is an integer ranging from 0 (not in school) to 6 (Ph.D. program). Birth year is as before and `s_address_lat` and `s_address_long` give the latitude and longitude of the user's home address (represented as decimal degrees scaled by a factor of 10^6 and converted to an integer).

The initial belief models the fact that each subsequent level of education is less likely and also captures the correlation between current educational level and age. For example, a user is given an approximately 0.05 chance of currently being an undergraduate in college, and college attendees are assumed to be born no later than 1985 (whereas elementary school students may be born as late as 2002).

```
secret :
    s_in_school_type := 4 ;
    s_birth_year := 1983 ;
    s_address_lat := 39003178 ;
    s_address_long := −76958199

belief :
pif 4/24 then
  uniform s_in_school_type 1 1 ;
  uniform s_birth_year 1998 2002
else
  pif 3/19 then
    uniform s_in_school_type 2 2 ;
    uniform s_birth_year 1990 1998
  else
    pif 2/15 then
      uniform s_in_school_type 3 3 ;
      uniform s_birth_year 1985 1992
    else
      pif 1/12 then
        uniform s_in_school_type 4 4 ;
        uniform s_birth_year 1980 1985
      else
        uniform s_in_school_type 0 0 ;
        uniform s_birth_year 1900 1985 ;
uniform s_address_lat 38867884 39103178 ;
uniform s_address_long −77058199 − 76825926
```

The query itself targets the pizza advertisement at users who are either in college or aged 18 to 28, while living close to the pizza shop (within a square region that is 2.5 miles on each side and centered on the pizza shop). If this condition is satisfied, then the query returns 1, indicating that the ad should be displayed. The full text of the query is given below.

```
querydef pizza : → out
```

```
#define age = 2010 − s_birth_year
#define lr_lat = 38967884
#define ul_lat = 39003178
#define lr_long = −76958199
#define ul_long = −76925926
if s_in_school_type ≥ 4 then
  in_school := 1
else
  in_school := 0 ;
if age ≥ 18 ∧ age ≤ 28 then
  age_criteria := 1
else
  age_criteria := 0 ;
if s_address_lat ≤ ul_lat
  ∧ s_address_lat ≥ lr_lat
  ∧ s_address_long ≥ lr_long
  ∧ s_address_long ≤ ul_long
then
  in_box := 1
else
  in_box := 0 ;
if (in_school = 1 ∨ age_criteria = 1)
  ∧ in_box = 1 then
  out := 1
else
  out := 0
```

A.4 Photo

The photo query is a direct encoding of a case study that Facebook includes on their advertising information page [4]. The advertisement was for CM Photographics, and targets offers for wedding photography packages at women between the ages of 24 and 30 who list in their profiles that they are engaged. The secret state consists of birth year, as before, gender (0 indicates male, 1 indicates female), and "relationship status," which can take on a value from 0 to 9. Each of these relationship status values indicates one of the status choices permitted by the Facebook software. The example below involves only four of these values, which are given below.

0 No answer

1 Single

2 In a relationship

3 Engaged

The secret state and prebelief are as follows.

secret :

```
  s_birth_year := 1983 ;
  s_gender := 0 ;
  s_relationship_status := 0

belief :

uniform s_birth_year 1900 2010 ;
uniform s_gender 0 1 ;
uniform s_relationship_status 0 3
```

The query itself is the following.

```
querydef cm_advert : → out

#define age = 2010 − s_birth_year
if age ≥ 24 ∧ age ≤ 30 then
  age_sat := 1
else
  age_sat := 0 ;
if s_gender = 1
 ∧ s_relationship_status = 3
 ∧ age_sat = 1 then
  out := 1
else
  out := 0
```

A.5 Travel

This example is another Facebook advertising case study [3]. It is based on an ad campaign run by Britain's national tourism agency, VisitBritain. The campaign targeted English-speaking Facebook users currently residing in countries with strong ties to the United Kingdom. They further filtered by showing the advertisement only to college graduates who were at least 21 years of age.

We modeled this using four secret values: country, birth year, highest completed education level, and primary language. As with other categorical data, we represent language and country using an enumeration. We ranked countries by number of Facebook users as reported by socialbakers.com. This resulted in the US being country number 1 and the UK being country 3. To populate the list of countries with "strong connections" to the UK, we took a list of former British colonies. For the language attribute, we consider a 50-element enumeration where 0 indicates "no answer" and 1 indicates "English" (other values appear in the prebelief but are not used in the query).

```
secret :

country := 1 ;
birth_year := 1983 ;
completed_school_type := 4 ;
language := 5

belief :
```

```
  uniform country 1 200 ;
  uniform birth_year 1900 2011 ;
  uniform language 1 50 ;
  uniform completed_school_type 0 5

      querydef travel : → out

#define age = 2010 − birth_year
if country = 1 ∨ country = 3
  ∨ country = 8 ∨ country = 10
  ∨ country = 18 then
  main_country := 1
else
  main_country := 0 ;
if country = 169 ∨ country = 197
  ∨ country = 194 ∨ country = 170
  ∨ country = 206 ∨ country = 183
  ∨ country = 188 then
  island := 1
else
  island := 0 ;
if language = 1
  ∧ (main_country = 1 ∨ island = 1)
  ∧ age ≥ 21
  ∧ completed_school_type ≥ 4 then
  out := 1
else
  out := 0
```

B Relational Queries

The two queries below, is_target_close and who_is_closer introduce relations between variables after revision, even though the example initial belief had no such relations. The initial belief stipulates the location of 2 objects is somewhere within a rectangular region. The is_target_close query determines if a given location is within $dist$ of the first object, measured using Manhattan distance. This query introduces relations between only 2 variables (latitude and longitude) which can be exactly represented using octagons but cannot using intervals.

The who_is_closer query performs a similar computation, but instead determines which of the two objects in the initial belief is closer to the new target location. The post belief can be handled by use of polyhedra, but not octagons, as it introduces relationships between more than 2 variables (latitudes and longitudes of 2 different objects).

```
belief :

uniform loc_lat1 29267245 36332852 ;
uniform loc_long1 41483216 46405563 ;
uniform loc_lat2 29267245 36332852 ;
uniform loc_long2 41483216 46405563
```

querydef is_target_close : $target\_location\_lat\ target\_location\_long\ dist \rightarrow is\_close$

$is\_close := 0$;
$dist\_lat := loc\_lat1 - target\_location\_lat$;
$dist\_long := loc\_long1 - target\_location\_long$;
if $dist\_lat < 0$ then
  $dist\_lat := -1 \times dist\_lat$;
if $dist\_long < 0$ then
  $dist\_long := -1 \times dist\_long$;
if $dist\_lat + dist\_long \leq dist$ then
  $is\_close := 1$

querydef who_is_closer : $target\_location\_lat\ target\_location\_long \rightarrow who\_closer$

$diff\_lat1 := loc\_lat1 - target\_location\_lat$;
$diff\_long1 := loc\_long1 - target\_location\_long$;
$diff\_lat2 := loc\_lat2 - target\_location\_lat$;
$diff\_long2 := loc\_long2 - target\_location\_long$;
if $diff\_lat1 < 0$ then
  $diff\_lat1 := -1 \times diff\_lat1$;
if $diff\_long1 < 0$ then
  $diff\_long1 := -1 \times diff\_long1$;
if $diff\_lat2 < 0$ then
  $diff\_lat2 := -1 \times diff\_lat2$;
if $diff\_long2 < 0$ then
  $diff\_long2 := -1 \times diff\_long2$;
$dist1 := diff\_long1 + diff\_lat1$;
$dist2 := diff\_long2 + diff\_lat2$;
if $dist1 \leq dist2$ then
  $who\_closer := 0$
else
  $who\_closer := 1$

C Benchmark Results

Table 1 tabulates performance results for all of the benchmark programs, for each possible base domain (intervals, octagons, and polyhedra labeled $\Box$, $\Diamond$, and $\pentagon$ respectively). Each column is the maximum size of the permitted powerset, whereas each grouping of rows contains, respectively, the wall clock time in seconds (median of 20 runs), the running time's semi-interquartile range (SIQR) with the number of outliers in parentheses (which are defined to be the points $3 \times$ SIQR below the first quartile or above the third), and the max belief computed (smaller being more accurate).

	1	2	3	4	5	6	7	8	9	10	15	20	25	30	35	40	∞
bday 1																	
□	0.008	0.008	0.009	0.009	0.009	0.009	0.009	0.009	0.009	0.009	0.009	0.009	0.009	0.009	0.009	0.009	0.009
	0.000(3)	0.004(3)	0.000(1)	0.000(3)	0.000(2)	0.002(4)	0.000(3)	0.000(3)	0.000(2)	0.000(2)	0.000(2)	0.000(2)	0.000(3)	0.000(3)	0.002(4)	0.000(4)	0.006(3)
	1	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039
◇	0.484	0.443	0.530	0.524	0.529	0.524	0.538	0.533	0.537	0.532	0.524	0.530	0.539	0.528	0.540	0.529	0.537
	0.011(3)	0.012(3)	0.018(2)	0.020(3)	0.015(3)	0.012(3)	0.073(4)	0.024(4)	0.325(2)	0.036(3)	0.010(2)	0.164(3)	0.035(3)	0.007(2)	0.021(3)	0.010(2)	0.014(1)
	1	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039
⬠	0.713	0.688	0.744	0.745	0.740	0.744	0.734	0.737	0.747	0.732	0.771	0.745	0.750	0.736	0.760	0.742	0.744
	0.487(0)	0.020(2)	0.016(4)	0.022(4)	0.549(0)	0.605(0)	0.015(2)	0.045(2)	0.059(4)	0.039(4)	0.480(0)	0.054(4)	0.203(3)	0.022(4)	0.791(0)	0.322(3)	0.026(3)
	1	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039	0.0039
bday 1+2+special																	
□	0.478	0.477	0.489	0.490	0.492	0.494	0.496	0.499	0.500	0.504	0.514	0.530	0.537	0.539	0.522	0.512	0.514
	0.001(3)	0.005(3)	0.002(1)	0.002(1)	0.002(1)	0.003(3)	0.002(2)	0.002(3)	0.001(2)	0.002(2)	0.002(2)	0.002(1)	0.002(1)	0.002(3)	0.003(2)	0.003(2)	0.007(2)
	1	1	3.84e-4	4.22e-4	4.22e-4	4.22e-4	4.22e-4	8.06e-4	8.06e-4	8.06e-4	4.60e-4	0.0013	0.0011	9.82e-4	4.22e-4	3.84e-4	3.84e-4
◇	2.189	2.286	3.453	3.927	4.969	4.463	4.007	4.652	5.489	4.043	6.278	6.625	5.559	7.361	5.797	5.025	5.812
	0.618(1)	0.787(0)	0.964(0)	0.906(0)	0.941(0)	1.650(0)	0.861(0)	1.701(0)	1.856(0)	1.552(0)	1.331(0)	1.420(0)	1.588(0)	1.702(0)	0.818(3)	1.452(0)	1.909(0)
	1	1	3.84e-4	4.22e-4	4.22e-4	4.22e-4	4.22e-4	8.06e-4	8.06e-4	8.06e-4	4.60e-4	0.0013	0.0011	9.82e-4	4.22e-4	3.84e-4	3.84e-4
⬠	4.578	4.767	6.292	4.575	6.510	6.391	5.875	7.404	7.722	7.777	7.763	8.233	7.532	6.681	9.821	6.512	6.800
	0.856(0)	1.024(0)	0.958(0)	1.160(1)	1.549(0)	2.029(0)	1.587(0)	1.899(0)	1.987(0)	2.595(0)	2.010(0)	2.177(0)	1.896(0)	1.705(0)	1.605(0)	0.890(1)	1.674(0)
	1	1	3.84e-4	4.22e-4	4.22e-4	4.22e-4	4.22e-4	8.06e-4	8.06e-4	8.06e-4	4.60e-4	4.60e-4	4.60e-4	4.60e-4	4.22e-4	3.84e-4	3.84e-4
bday large 1																	
□	0.008	0.008	0.009	0.009	0.009	0.009	0.009	0.009	0.009	0.009	0.009	0.009	0.009	0.009	0.009	0.009	0.009
	0.000(2)	0.000(1)	0.000(1)	0.000(3)	0.000(3)	0.000(3)	0.000(4)	0.000(2)	0.000(2)	0.000(4)	0.000(3)	0.000(1)	0.000(2)	0.000(4)	0.000(2)	0.000(4)	0.000(2)
	1	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014
◇	0.500	0.470	0.552	0.546	0.533	0.551	0.535	0.536	0.541	0.532	0.535	0.541	0.535	0.538	0.531	0.533	0.550
	0.035(4)	0.273(1)	0.549(0)	0.510(1)	0.037(4)	0.466(0)	0.041(4)	0.041(3)	0.258(2)	0.038(2)	0.065(4)	0.063(4)	0.019(3)	0.021(3)	0.020(2)	0.016(1)	0.066(4)
	1	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014
⬠	0.693	0.699	0.747	0.757	0.736	0.733	0.739	0.735	0.754	0.791	0.742	0.757	0.756	0.740	0.742	0.734	0.728
	0.027(4)	0.024(1)	0.397(2)	0.307(2)	0.023(4)	0.025(4)	0.057(4)	0.015(3)	0.513(0)	0.658(0)	0.017(3)	0.283(3)	0.034(3)	0.263(4)	0.494(0)	0.027(3)	0.013(1)
	1	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014
bday large 1+2+special																	
□	0.483	0.482	0.497	0.499	0.500	0.501	0.505	0.508	0.509	0.513	0.528	0.542	0.554	0.567	0.572	0.594	0.556
	0.001(2)	0.001(1)	0.001(1)	0.002(2)	0.001(2)	0.002(2)	0.003(1)	0.001(2)	0.001(2)	0.003(4)	0.002(2)	0.001(1)	0.002(2)	0.002(3)	0.002(1)	0.004(1)	0.001(4)
	1	1	1.47e-4	1.61e-4	1.61e-4	1.61e-4	1.61e-4	3.08e-4	3.08e-4	3.08e-4	1.76e-4	3.08e-4	3.08e-4	3.08e-4	1.76e-4	3.08e-4	1.47e-4
◇	4.860	5.122	7.126	7.966	7.250	5.742	8.254	5.980	7.855	6.810	10.32	8.887	9.578	8.262	11.77	11.16	13.52
	1.207(0)	1.464(0)	1.636(0)	1.643(0)	1.646(0)	0.904(3)	2.613(0)	1.626(0)	1.409(1)	1.729(0)	1.968(0)	2.293(0)	3.165(0)	1.696(0)	2.673(0)	3.067(0)	2.109(2)
	1	1	1.47e-4	1.61e-4	1.61e-4	1.61e-4	1.61e-4	3.08e-4	3.08e-4	3.08e-4	1.76e-4	3.08e-4	3.08e-4	3.08e-4	1.76e-4	3.08e-4	1.47e-4
⬠	5.769	4.850	10.90	10.81	8.010	8.017	11.45	10.73	11.71	10.26	12.76	15.01	13.89	14.73	16.16	16.95	17.91
	1.414(0)	0.841(2)	2.680(0)	2.000(0)	1.766(1)	2.388(0)	2.102(0)	2.191(0)	1.642(1)	2.011(0)	1.474(1)	2.102(1)	1.359(0)	1.598(1)	1.509(1)	1.509(2)	1.140(0)
	1	1	1.47e-4	1.61e-4	1.61e-4	1.61e-4	1.61e-4	3.08e-4	3.08e-4	3.08e-4	1.76e-4	3.08e-4	3.08e-4	3.08e-4	1.76e-4	3.08e-4	1.47e-4
pizza																	
□	0.060	0.071	0.086	0.100	0.117	0.135	0.175	0.226	0.182	0.272	0.212	0.289	0.368	0.348	0.258	0.278	0.279
	0.002(4)	0.000(3)	0.000(3)	0.000(3)	0.001(3)	0.001(4)	0.003(4)	0.005(3)	0.001(3)	0.000(3)	0.005(3)	0.001(3)	0.001(2)	0.001(1)	0.001(1)	0.001(1)	0.001(3)
	1	1	1	1.63e-9	1	4.60e-10	4.60e-10	2.14e-10	2.14e-10	2.14e-10	1.08e-10	6.00e-11	6.00e-11	6.00e-11	6.00e-11	6.00e-11	6.00e-11
◇	19.07	12.15	18.83	14.29	26.41	16.72	29.92	27.49	21.08	25.93	24.36	29.33	37.08	38.09	40.51	45.69	43.98
	1.780(0)	1.573(0)	1.331(0)	1.640(1)	2.097(0)	1.233(0)	0.808(1)	0.731(0)	1.553(0)	1.044(0)	0.654(2)	1.049(1)	1.719(0)	1.311(0)	1.211(1)	1.098(2)	1.962(1)
	1	1	1	1.63e-9	1	4.60e-10	4.60e-10	2.14e-10	2.14e-10	2.14e-10	1.08e-10	6.00e-11	6.00e-11	6.00e-11	6.00e-11	6.00e-11	6.00e-11
⬠	127.7	33.24	39.59	25.30	46.22	28.87	38.34	33.75	59.07	33.64	41.38	41.72	47.59	53.36	58.39	64.71	65.47
	1.392(3)	1.100(1)	1.596(1)	0.873(0)	1.486(1)	1.474(0)	2.058(0)	1.271(1)	1.461(2)	1.308(1)	1.853(0)	1.204(1)	1.303(2)	1.195(0)	1.957(1)	1.774(2)	2.291(0)
	1	1	1	8.66e-10	4.95e-10	1.50e-10	1.50e-10	1.37e-10	1.37e-10	1.37e-10	6.00e-11	6.00e-11	6.00e-11	6.00e-11	6.00e-11	6.00e-11	6.00e-11
photo																	
□	0.015	0.016	0.020	0.020	0.022	0.022	0.022	0.022	0.022	0.022	0.022	0.022	0.022	0.022	0.022	0.022	0.022
	0.000(3)	0.000(2)	0.000(2)	0.000(4)	0.000(3)	0.000(1)	0.000(4)	0.000(3)	0.000(3)	0.000(4)	0.000(2)	0.004(3)	0.000(3)	0.000(4)	0.000(1)	0.000(2)	0.000(3)
	1	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429
◇	1.238	1.228	1.343	2.855	2.529	1.526	1.539	1.573	1.911	1.540	1.541	1.545	1.643	1.533	1.626	1.685	1.606
	0.712(1)	0.745(0)	0.290(3)	0.795(2)	0.828(0)	0.772(1)	0.660(1)	0.675(1)	0.786(1)	0.776(0)	0.541(1)	0.100(4)	0.824(1)	0.733(0)	0.805(1)	0.771(0)	0.789(1)
	1	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429
⬠	1.840	1.666	1.856	3.473	2.011	2.054	2.068	3.226	3.281	2.038	2.048	2.178	2.055	2.056	2.617	2.729	2.041
	0.824(0)	0.607(1)	0.866(0)	0.855(1)	0.805(0)	0.898(0)	0.874(0)	1.191(0)	1.294(0)	0.792(0)	0.885(1)	0.823(0)	0.819(1)	0.826(2)	0.851(0)	0.841(0)	0.845(0)
	1	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429	0.1429
travel																	
□	0.186	0.189	0.203	0.220	0.234	0.246	0.271	0.284	0.298	0.319	0.419	0.422	0.458	0.564	0.493	0.576	0.516
	0.000(3)	0.001(3)	0.000(3)	0.000(3)	0.001(2)	0.002(4)	0.001(2)	0.001(4)	0.001(1)	0.002(3)	0.001(3)	0.001(2)	0.001(3)	0.001(2)	0.002(2)	0.001(3)	0.005(4)
	1	1	1	1	1	1	0.0278	0.0139	0.0139	0.0042	0.0037	0.0025	5.05e-4	5.05e-4	5.05e-4	5.05e-4	5.05e-4
◇	12.72	8.111	8.142	8.849	12.58	10.79	14.31	14.90	17.06	17.73	28.80	34.17	37.61	40.87	42.59	45.33	60.81
	1.688(0)	2.268(0)	1.720(0)	1.169(0)	1.668(2)	1.191(0)	1.151(1)	1.408(1)	0.944(2)	1.544(0)	1.498(0)	1.217(1)	1.391(0)	1.240(1)	1.814(1)	0.735(2)	1.003(2)
	1	1	1	1	1	1	0.0167	0.0083	0.0083	0.0042	0.0025	0.0015	5.05e-4	5.05e-4	5.05e-4	5.05e-4	5.05e-4
⬠	225.7	11.31	15.74	16.79	23.48	21.73	32.47	29.31	39.40	34.25	60.11	64.85	69.79	71.61	71.81	73.26	83.74
	2.232(0)	2.884(0)	0.869(2)	0.936(0)	1.017(2)	1.373(0)	1.950(0)	1.750(2)	1.348(1)	1.197(1)	1.755(0)	1.961(1)	1.441(1)	2.037(1)	2.169(0)	2.252(0)	2.150(0)
	1	1	1	1	1	1	0.0056	0.0028	0.0028	0.0028	0.0012	0.0010	5.05e-4	5.05e-4	5.05e-4	5.05e-4	5.05e-4

Table 1: Query evaluation benchmarks