Abstract

Title of dissertation: Modeling, Quantifying, and Limiting Adversary Knowledge

Piotr Mardziel, Doctor of Philosophy, 2015

Dissertation directed by: Professor Michael Hicks
Department of Computer Science

Users participating in online services are required to relinquish control over potentially sensitive personal information, exposing them to intentional or unintentional miss-use of said information by the service providers. Users wishing to avoid this must either abstain from often extremely useful services, or provide false information which is usually contrary to the terms of service they must abide by. An attractive middle-ground alternative is to maintain control in the hands of the users and provide a mechanism with which information that is necessary for useful services can be queried. Users need not trust any external party in the management of their information but are now faced with the problem of judging when queries by service providers should be answered or when they should be refused due to revealing too much sensitive information.

Judging query safety is difficult. Two queries may be benign in isolation but might reveal more than a user is comfortable with in combination. Additionally malicious adversaries who wish to learn more than allowed might query in a manner that attempts to hide the flows of sensitive information. Finally, users cannot rely on human inspection of queries due to its volume and the general lack of expertise.

This thesis tackles the automation of query judgment, giving the self-reliant user a means with which to discern benign queries from dangerous or exploitive ones. The approach is based on explicit modeling and tracking of the knowledge of adversaries as they learn about a user through the queries they are allowed to observe. The approach quantifies the absolute risk a user is exposed, taking into account all the information that has been revealed already when determining to answer a query. Proposed techniques for approximate but sound probabilistic inference are used to tackle the tractability of the approach, letting the user tradeoff utility (in terms of the queries judged safe) and efficiency (in terms of the expense of knowledge tracking), while maintaining the guarantee that risk to the user is never underestimated. We apply the approach to settings where user data changes over time and settings where multiple users wish to pool their data to perform useful collaborative computations without revealing too much information.

By addressing one of the major obstacles preventing the viability of personal information control, this work brings the attractive proposition closer to reality.

Modeling, Quantifying, and Limiting Adversary Knowledge

by

Piotr Mardziel

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2015

Advisory Committee:

Professor Michael Hicks, Chair
Professor Prakash Narayan
Professor Jonathan Katz
Professor David van Horn
Michael R. Clarkson
Professor Boris Köpf

Acknowledgments

First and foremost I am immensely thankful for the opportunity to have studied under the direction of my advisor, Michael Hicks, who despite shouldering innumerable other responsibilities provided consistent and invaluable guidance throughout my graduate career. I'm thankful to the members of my defense committee, particularly Michael Clarkson and Boris Köpf who not only provided valuable feedback during the writing of this thesis but also authored the works that inspired it. I'm grateful to my excellent collaborators, many of whom have contributed to the research that serves as the basis of this thesis. In alphabetical order, they are: Mário Alvim, Adam Bender, Carlos Cid, Michael Clarkson, Matthew Hammer, Michael Hicks, Jonathan Katz, Arman Khouzani, Dave Levin, Stephen Magill, James Parker, Kasturi Raghavan, Aseem Rastogi, and Mudhakar Srivatsa. I would also like to thank all the members, past and present, of the group for Programming Languages at University of Maryland (PLUM) who provided a fun and stimulating research environment (and occasionally tech support). I'm fortunate to have had the opportunity to be taught, advised, and inspired earlier during my undergraduate years at the Worcester Polytechnic Institute by professors Daniel Dougherty, Kathi Fisler, Carolina Ruiz, and Stanley Selkow. Finally, I'm grateful to my family, to my friends, and to my dog Mika, who is a good dog.

Research was sponsored by US Army Research laboratory and the UK 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 authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the U.S. Government, the UK Ministry of Defense, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

Table of Contents

List of Figures

List of Tables

Chapter 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 [3]) to implement services like birthday announcements and horoscopes, while Facebook selects ads based on age, gender, and even sexual preference [49].

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 [5]. Most popular social networking sites include such provisions in their user agreement, including Twitter, LinkedIn, Pinterest, and Google Plus.

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 [12, 96].

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 [102] and 63% in the 2000 census [46]. So the user may be content to reveal any one of these bits of information as long as a querier does not attain a *sufficient amount* of information about the combination of three to identify the user. The user, in effect, can tradeoff utility of the application of which the query is a part of, and the privacy risks involved in the query's output. [1]

When it comes to determining query safety, there are several questions open.

[1] Note that it is safer for the user to assume that the application is adversarial, actively trying to break the privacy of the data the user is concerned about, even though in many, if not most, situations the application is actually benign. For this reason, we will usually refer to the party making the query as the *adversary* and the private information they are seeking as the *secret*.

- **Question 1**: How does one *measure* information?
- **Question 2**: How does the adversary obtain information?
- **Question 3**: How can a user make sure that the adversary does not attain too much information?

Partial answers to some of these questions can be found in the literature on *quantitative information flow* which is focused on measuring the amount of secret information that "leaks" through a system's (that is, a query's) observable behavior (its output) during its execution. Leakage involves (usually) information theoretic answers to Question 1. This thesis takes an attack-oriented view on how to quantify information; the more utility the attack gains from exploiting the information, the larger the measure of information. This notion then plays into the handling of Question 2: the adversary, being adversarial, behaves so as to maximize his success. The exact behavior and the answer to Question 3 varies with the complexity of the information sharing setting. At the core of each of these questions, however, is **adversary knowledge**.

Our thesis is that *query safety can be decided based on explicit modeling of adversary knowledge*. Knowledge models are the focal point of the answers to our three questions: 1) information or privacy risk can be defined as a property of adversary knowledge, 2) adversary learning is the process of inference with their knowledge as the prior, and 3) limiting information or risk can be done by refusing to share if the posterior's risk would exceed a threshold. We note that this thesis is an initial exploration and more work beyond it is necessary to make it fully practical.

1.1 Organization of Thesis

In this thesis we will explore models of adversary knowledge and the three questions above in three related settings. First is the *dynamic enforcement setting* where a user is tasked with judging query safety without necessarily knowing the queries that will be asked in the future. Second, we will look at *predictive setting* in which the user at time 0 wants to predict risks of answering queries from now up to time n without knowing what exactly the adversary will ask nor fixing what his own secret value will be throughout this process (it may change over time). Finally we look at the *symmetric setting* in which two mutually-adversarial users scrutinize queries that will reveal information about each other's secrets.

This thesis is organized roughly into parts corresponding to the three settings. In the remainder of this introduction, we will briefly summarize the settings. In preparation we discuss background material related to quantitative information flow in Section 2.1 which serve as the inspiration and partial solution to the questions posed in this thesis. We also discuss probabilistic programming in Section 2.2 which will be used as the tool for formal specification of various models we present as well as the approach to their implementation. The three settings follow: dynamic setting in Chapter 3, predictive setting in Chapter 4, and symmetric setting in Chapter 5. We

then present our work on the probabilistic programming and the computational aspects of the models and techniques therein in Chapter 6. That work has applicability to probabilistic programming in general and hence is described in its own chapter. We conclude in Chapter 7.

1.2 Dynamic enforcement setting

Part of the difficulty in judging whether a query should be executed or not is that the user might not know what might be asked in the future. Not knowing what will be asked is in fact central to the advantage online query interface has over obfuscation and release of data for offline access: the user does not have to figure out what about their data will be important to each party and different parties can access different information.

For this setting we describe the design 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 1960 and 1996 [6]. 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.

The approach to Questions 1 and 2 here is to model adversary knowledge as probability distributions and then use them to quantify how effective the adversary would be at *guessing* the secret or part of it. Variations of guessability [100] as means of measuring information will be used throughout the thesis though we will leave the exact nature of exploitability (guessing or otherwise) abstract where possible. In all of our settings the answer to Question 2 starts with modeling adversaries as Bayesian agents having prior beliefs. In the next setting the behavior of the adversary will be greatly expanded to include more than just Bayesian reasoning. Lastly, the notion of *knowledge threshold* (or rather guessability threshold) serves as part of the answer to Question 3.

The main contribution of this work [70–72] is the definition of *knowledge-based security policies* (KBSP) to limit adversary knowledge, and a method of enforcing the policies in a manner that does not, itself, leak additional information about the secret. In Chapter 6 we demonstrate how to implement KBSP enforcement using an approximate but policy-sound method of probabilistic computation.

1.3 Predictive setting

In contrast to the dynamic setting, in the predictive setting the task for the user is almost entirely speculative. The user in this setting wants to predict the risks involved at future time n while not knowing exactly what will happen from now until then. What if the secret can change? What if the adversary gets smart about what they query and how they exploit the secret?

Some prior work on QIF has considered this setting. We consider models involving richer query contexts to handle the above-mentioned uncertainties: 1) secrets can change over time, 2) adversary can influence queries by providing inputs adaptively, and 3) adversary can decide when to exploit their acquired knowledge. The more complex setting makes Question 2 more nuanced. In general, the adversaries will be assumed to be not just Bayesian rational agents but also ones that pick their actions optimally from among the choices provided them by the setting.

The work [66–68] draws several conclusions about both the behaviors of the adversary and the effects of changing secrets:

- Frequent change of a secret can increase leakage, even though intuition might initially suggest that frequent changes should decrease it. The increase occurs when there is an underlying order that can be inferred and used to guess future (or past) secrets.
- *Wait-adaptive* adversaries, ones which can decide when to exploit the secret based on what they observe, can derive significantly more gain than adversaries who cannot adaptively choose when to attack. So ignoring the adversary's adaptivity (as in prior work on static secrets) might lead one to conclude secrets are safe when they really are not.
- A wait-adaptive adversary's expected gain increases monotonically with time, whereas a non-adaptive adversary's gain might not.
- Adversaries that are *low adaptive*, meaning they are capable of influencing their observations by providing low-security inputs (or selecting their query from among some fixed set), can learn exponentially more information than adversaries who cannot provide inputs.

1.4 Symmetric setting

What if both the user and the adversary are simultaneously holders of secrets and queriers of (each other's) secrets? In the symmetric setting we consider how to determine query safety in this situation. The motivation for this setting comes from work on secure multi-party computation (SMC) [43, 44, 59, 105] that describes a technique in which multiple parties can compute a function f (a query) of their joint inputs s_i without revealing those inputs, all the while without relying on any trusted third party.

Most work on SMC provides an answer to the question of *how* to compute f, but does not address the complementary question of *when* it is "safe" to compute f in the first place, i.e., when the output of f may reveal more information than parties are comfortable with. The latter question is important: the information implied by a query's result depends both on the parties' inputs and their prior knowledge. As an example of the former, suppose two parties want to compute the "less than or equal" function, $f(s_1, s_2) \stackrel{\text{def}}{=} s_1 \leq s_2$ with variables ranging in $\{1, \ldots, 10\}$. This function could reveal a lot about s_1 to P_2. If $s_2 = 1$ and $f(s_1, s_2)$ returns *true*, then P_2 learns that s_1 can only be the value 1. However, if $s_2 = 5$ then regardless of the output of the function, P_2 only learns that s_1 is one of 5 possibilities, a lower level of knowledge than in the first case.

On the other hand we may deem a pair of queries acceptable in isolation, but allowing their composition would be too revealing. For example, suppose the parties also want to compute "greater than or equal", $f_2(s_1, s_2) \stackrel{\text{def}}{=} s_1 \geq s_2$. When $s_2 = 5$, either query in isolation narrows the values of s_1 to a set of at least 4 possibilities from P_2's perspective. But if f_1 and f_2 both return *true*, P_2 can infer $s_1 = s_2 = 5$.

To address the problem of query safety in the symmetric setting, we use the ideas and techniques developed for the dynamic enforcement setting (asymmetric setting) and generalize them to SMC.

This work [69] has two main contributions:

- We describe a pair of techniques for evaluating the information security implications of SMC queries. This problem is significantly harder than in the asymmetric case as now a party P cannot be sure what the other parties learn due to P not knowing the values of the other parties' secrets.

- Similarly to the dynamic setting, we describe how to implement a knowledge-limiting threshold policy in a manner that does not leak additional information.

1.5 Probabilistic programming

As we have already noted, we will view adversary knowledge as probability distributions that are revised using probabilistic inference throughout the three settings. To describe the knowledge and the inference process as well as the queries involved we will make great use of *probabilistic programming* [48].

In general, probabilistic programming is about viewing programs as conditional distributions and taking advantage of this higher-level view. A program that takes in input X and produces output Y can be viewed as a conditional distribution of Y given X. This is most interesting when the program has some randomness in its definition but can also be useful when the input X is itself probabilistic (as is the case of adversary knowledge about the secret). The program viewed this way can then be "inverted" to perform probabilistic inference by producing the distribution on X given the observation that Y has some value y. This exactly captures the process of adversary learning something about the secret X (input) given the query output Y.

Throughout the thesis we will also favor specification of probabilistic processes and distributions via programs for their conciseness (as compared to purely mathematical formulations) whenever possible.

We also use probabilistic programming and inference as a means of implementing the enforcement mechanisms and predictive models presented in the thesis. Probabilistic inference, however, is intractable in general (even undecidable [7]). Most work that addresses this problem relies on approximation which has undesirable implications for security applications where soundness is desired.

In the last main chapter of this thesis we describe an implementation of probabilistic programming based on abstract interpretation [28] that is capable of approximate inference, but is sound relative to our knowledge-limiting 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.

In that work [70–72] we make two main contributions:

- Design and implementation of approximate but sound probabilistic programming with inference by extending the abstract domains of polyhedra, octagons, and intervals with under- and over-approximations of probability. Both are necessary for sound estimation of probability after probabilistic inference (conditioning).

- Experimental evaluation of the approach of implementation based on the extended abstract domains of varying complexity that allow for tradeoff between precision and efficiency.

Chapter 2

Background

In this section we present several background underlying this thesis. First, we describe the area of quantified information flow which addresses problems similar to those posed by this work. Second, we introduce the idea of probabilistic programming which we use as a modeling language and computational tool. Finally, we present some background on secure multi-party computation which is the substrate underlying the symmetric setting.

2.1 Quantitative Information Flow

Consider a password checker, which grants or forbids access to a user based on whether the password supplied matches the one stored by the system. The password checker must leak secret information, because it must reveal whether the supplied password is correct. Quantifying the amount of information leaked is useful for understanding the security of the password checker—and of other systems that, by design or by technological constraints, must leak information.

The classic model for QIF, pioneered by Denning [34], represents a system, or in our case, a query as an information-theoretic channel:

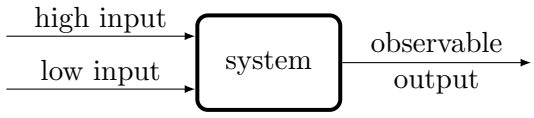


Figure 2.1: System modeled as an information theoretic channel.

A *channel* is a probabilistic function [1] that maps a high security (i.e., secret) input and a low security (typically adversary controlled) input to an observable output. The adversary is assumed to have some prior notion about the distribution of high inputs. By providing low input and observing the output, the adversary revises their view about the about the high input. If the prior and posterior beliefs can be formulated in terms of some notion of uncertainty, then the change in the adversary's uncertainty is the amount of information leakage due to the channel:

$$\textit{leakage} = \textit{initial uncertainty} - \textit{revised uncertainty}.$$

More formally, let X_h, X_ℓ and X_o be random variables representing the distribution of high inputs, low inputs, and observables, respectively. Given a function $F(X)$

[1] You can think of such a function as returning a probability distribution instead of a concrete value.

of the uncertainty of X, leakage is then:

$$F(X_h) - F(X_h \mid X_\ell = \ell, X_o), \tag{0.1}$$

where ℓ is the low input chosen by the adversary, $F(X_h)$ is the adversary's initial uncertainty about the high input, and $F(X_h \mid X_\ell = \ell, X_o)$ is the revised uncertainty. As is standard, $F(X_h \mid X_\ell = \ell, X_o)$ is defined to be $\mathbb{E}_{o \leftarrow X_o}\left[F(X_h \mid X_o = o, X_\ell = \ell)\right]$, where $\mathbb{E}_{x \leftarrow X}\left[f(x)\right]$ denotes $\sum_x \Pr\left(X = x\right) \cdot f(x)$. The definition of $F(X_h \mid X_\ell = \ell, X_o)$ hides the relationship between the inputs (low and high) and the output of a channel. This can be made explicit by writing it alternatively as $\mathbb{E}_{h \leftarrow X_h}\left[\mathbb{E}_{o \leftarrow X_{C(h,\ell)}}\left[F(X_h \mid X_{C(h,\ell)} = o, X_\ell = \ell)\right]\right]$, where $X_{C(h,\ell)}$ is the random variable describing the distribution of channel outputs when h and ℓ are its inputs. Written in this manner it is more clear that information leakage as defined per Equation 0.1 is an *expectation over the high values* (and potentially the non-determinism of the channel). This point will be important to the applicability of the definition as we will discuss in Section 2.1.2.

Various instantiations of F have been proposed, including Shannon entropy [10, 20, 23, 62, 64, 82, 83], guessing entropy [63, 73], marginal guesswork [90], and (Bayes) vulnerability [19, 100].

Some of these metrics have interpretations that connect them in some way to a measurement of adversary success in some form of exploitation. In this sense the function F can be thought of as measuring the inverse of risk in that the larger the uncertainty, the lower the risk to the user. The exact meaning of risk in terms of the probability or magnitude of bad events varies. For example, *vulnerability* [100] measures the expected probability that an optimal adversary will guess the secret value in one try. The g-leakage framework [10] uses *gain functions* to generalize the interpretation of many metrics by describing them as the expected payoff attributed to the adversary making some optimal action or exploit. In this thesis we use a similar approach for quantifying uncertainty; instead of information-theoretic quantities we will view it as the payoff an optimal adversary receives.

2.1.1 Example: Birthday query

Consider the problem of answering queries about a user's demographic information similar to that of Section 1.2. Specifically let us assume that a user drawn randomly from a population has an associated triple $(\text{byear}, \text{bday}, \text{gndr})$ with each element distributed independently according to the following uniform distributions:

$$\begin{aligned}
\Pr\left(X_{\text{byear}} = y\right) &= \frac{1}{37} && \text{for } y \in \mathit{year} \stackrel{\text{def}}{=} \{1960, \cdots, 1996\} \\
\Pr\left(X_{\text{bday}} = d\right) &= \frac{1}{365} && \text{for } d \in \mathit{day} \stackrel{\text{def}}{=} \{0, \cdots, 364\} \\
\Pr\left(X_{\text{gndr}} = g\right) &= \frac{1}{2} && \text{for } g \in \mathit{gender} \stackrel{\text{def}}{=} \{\text{Male}, \text{Female}\}
\end{aligned}$$

The joint distribution of the triple is therefore uniform with probability $\frac{1}{37 \cdot 365 \cdot 2}$ for triples in the ranges above. Let demo refer to the triple $(\text{birthyear}, \text{birthday}, \textit{gender})$, then the prior or initial uncertainty in X_{demo} or $F(X_{\text{demo}})$ varies depending on the choice of F. For example, we have:

- The (Shannon) entropy of X_{demo} is approximately 14.721. Entropy can be interpreted as the expected number of (optimally designed) yes/no questions that need to be answered about demo in order to determine it's value.
- The guessing entropy of X_{demo} is 1.3506×10^4 and has the interpretation as the expected number of guesses an optimal adversary would need to make before they correctly guess the demographics.
- The vulnerability of X_{demo} is 3.7×10^{-5} and is the probability that an optimal adversary will guess a user's demographics correctly in one try.

Now consider the following query, which checks whether the given user's birthday occurs in the next week:

$$\text{bday\_query} : \textit{day} \times (\textit{year} \times \textit{day} \times \textit{gender}) \to \textit{bool}$$

```
let bday_query: today (byear, bday, gndr) ↦
  if today < 365 − 7 then (bday >= today and bday + 7 < today)
  else (bday >= today or bday < (today+7) mod 365)
```

This query constitutes a channel through which information about the demographic flows, though just the bday in this case. To compute the revised uncertainty we need to consider both possible outputs of this query. In the first case, the query returns **true** and does so with probability $\frac{7}{365}$ on a randomly sampled user. In the other case it returns **false** with probability $\frac{358}{365}$. Let us say that $\text{today} = 100$, then the two possible revised distributions of X_{bday} are:

$$\Pr\left(X_{\text{bday}} = d \mid X_o = \textbf{true}, \cdots\right) = \frac{1}{7} \qquad \text{for } d \in \{100, \cdots, 106\}$$

$$\Pr\left(X_{\text{bday}} = d \mid X_o = \textbf{false}, \cdots\right) = \frac{1}{358} \qquad \text{for } d \in \{0, \cdots, 99, 107, \cdots, 364\}$$

Note that we omitted the condition $X_{\text{today}} = 100$ for brevity. The distributions over the whole demographic, X_{demo}, are the above, except additionally weighted by $\frac{1}{37*2}$ (the birthyear and *gender* are independent of birthday). The expected revised measures of uncertainty average the uncertainty in these two by their respective probabilities of occurrence (as indicated by probabilities of X_o).

- The revised (or "conditional") entropy of $X_{\text{demo}} \mid \left(X_{\text{today}} = 100, X_o\right)$ is 14.584 as compared to 14.721 in the prior. The query thus reduces the expected number of optimally chosen yes/no questions required to determine the demographic by approximately 0.137.

- The revised guessing entropy of $X_{\text{demo}} \mid \left(X_{\text{today}} = 100, X_o\right)$ is 1.2997×10^4 as compared to 1.3506×10^4 in the prior. Thus the query reduces the number of expected guesses by approximately 509.
- The revised vulnerability of $X_{\text{demo}} \mid \left(X_{\text{today}} = 100, X_o\right)$ is 7.40×10^{-5}, compared to 3.70×10^{-5} in the prior. Note that vulnerability increases with observations. For this reason the same idea is often expressed as the related *min-entropy* [2] to better align with the notion of uncertainty. In that sense, the query leaks exactly 1 bit of information.

2.1.2 Static vs. Dynamic

The definition of information flow in Equation 0.1 above inherently makes assumptions about adversary belief and the distribution of secrets that dictate nuances in its interpretation and restrict its applicability. In particular the prior distribution of secrets, represented via the random variable X_h, serves two functions at once: 1) it is the prior belief of the adversary, 2) it is the distribution of secrets over which the flow is averaged. In other words, adversary belief is conflated with the true distribution over the secrets.

The very notion that there is a true distribution of secrets means that the equation has a *static* interpretation, one that is independent of any particular value of the secret. Such an interpretation is useful to calculating the expected risks inherent in the channel without knowing what the secret value is. On the other hand, this interpretation does not say anything about the risk for the user that has a particular secret value. If we assume X_h represents the distributions of secrets among some population of users then the leakage quantity in Equation 0.1 would measure risks in expectation over the population. It would not say anything, however, about the risks to a particular user with a particular secret.

This problem can be seen in the example of Section 2.1.1. There we see how two outcomes of a query are averaged in the computation of the revised uncertainty measure. For any individual, only one of those two outcomes will occur and if the query returns **true** it reveals a lot more information than if it returns **false**. It is much more likely to leak less but this is no comfort to individuals for whom it leaks more.

In contrast to the static case, a *dynamic* variation of the notion of leakage can be adapted from the static equation by assuming that there is a true secret value h that is consistent with the prior belief of the adversary (that is, the adversary considers h possible). Likewise the distribution of channel outputs represented by the random variable X_o is exactly the distribution of channel outputs when h is the high input. The interpretation of the dynamic variation is applicable to users as individuals instead of as members of a population. This is the approach taken in the experimental protocol of Clarkson et al. [24] and which we build upon in this thesis to address the dynamic enforcement setting.

The difference between dynamic and static notions of uncertainty can be quite stark. Consider, for example, the flow calculations in the birthday query example

[2] Vulnerability of v is equivalent to min-entropy of $\lg(1/v)$.

in Section 2.1.1. Specifically, let us take a closer look at the static vulnerability calculation. It averages two outcomes: the two different outputs of bday_query:

$$\begin{aligned} F\left(X_{\text{demo}} \mid X_o, \cdots\right) = &\Pr\left(X_o = \textbf{true}\right) \cdot F\left(X_{\text{demo}} \mid X_o = \textbf{true}, \cdots\right) + \\ &\Pr\left(X_o = \textbf{false}\right) \cdot F\left(X_{\text{demo}} \mid X_o = \textbf{false}, \cdots\right) \\ \approx &\Pr\left(X_o = \textbf{true}\right) \cdot 1.9305 \times 10^{-3} + \\ &\Pr\left(X_o = \textbf{false}\right) \cdot 0.3775 \times 10^{-3} \\ \approx &\, 7.4047 \times 10^{-5} \end{aligned}$$

On average we see that an adversary has a fairly low chance of guessing the demographic. For users which the query returns **true**, however, the risk is over 26 times higher than the average. Though the probability of a random user being one of those is low, this is not a useful fact from their perspective.

2.1.3 Example: Second birthday query

Consider now another extending the previous example of Section 2.1.1 with another query asked after the first. In particular consider the same bday_query but with today equal to 101 instead of 100. What is the flow due to the second query? The notion of flow is easily extended by comparing the measures of uncertainty after the first query to that after the second:

$$F(X_H \mid X_{\text{today}_1} = 100, X_{o_1}) - F(X_H \mid X_{\text{today}_1} = 100, X_{\text{today}_2} = 101, X_{o_1}, X_{o_2})$$

We use today_1 and today_2 to differentiate the two instances of query input. We do so similarly for the two outputs o_1 and o_2. There are now 4 possible outcomes of the two queries: the 4 combinations of **true** and **false**:

$$\begin{aligned} \Pr\left(X_{\text{bday}} = d \mid X_{o_1} = \textbf{true}, X_{o_2} = \textbf{true}, \cdots\right) &= \frac{1}{6} && \text{for } d \in \{101, \cdots, 106\} \\ \Pr\left(X_{\text{bday}} = d \mid X_{o_1} = \textbf{true}, X_{o_2} = \textbf{false}, \cdots\right) &= 1 && \text{for } d = 100 \\ \Pr\left(X_{\text{bday}} = d \mid X_{o_1} = \textbf{false}, X_{o_2} = \textbf{true}, \cdots\right) &= 1 && \text{for } d = 107 \\ \Pr\left(X_{\text{bday}} = d \mid X_{o_1} = \textbf{false}, X_{o_2} = \textbf{false}, \cdots\right) &= \frac{1}{357} && \text{for } d \in \{0, \cdots, 99, 107, \cdots, 364\} \end{aligned}$$

Thus the expectation of the information release due to the second query (after the first) can be calculated as the expectation of uncertainties in those four cases (adding the birthyear and *gender* factors) weighted by their respective probabilities, which are $\frac{6}{365}$, $\frac{1}{365}$, $\frac{1}{365}$, and $\frac{357}{365}$, respectively.

- The revised (or "conditional") entropy of $X_{\text{demo}} \mid X_{o_1}, X_{o_2}, \cdots$ is 14.546 as compared to 14.584 after the first query and 14.721 initially. Thus the expected number of optimally chosen yes/no questions required to determine the demographic is reduced by 0.038 due to the second query, as compared to 0.137 due to the first query.

- The revised guessing entropy of $X_{\text{demo}} \mid X_{o_1}, X_{o_2}, \cdots$ is 1.2924×10^4 as compared to 1.2997×10^4 after the first query and 1.3506×10^4 initially. Thus the expected number of guesses has further reduced by another 73 guesses after it has already reduced by 509 due to the first query.

- The revised vulnerability of $X_{\text{demo}} \mid X_{o_1}, X_{o_2}, \cdots$ is 1.48×10^{-4} as compared to 7.40×10^{-5} after the first query and 3.70×10^{-5} initially. In terms of min-entropy the second query leaked a additional 1 bit as has the first query before it.

The difference between static and dynamic measurements of uncertainty is even greater in the case of the two queries. In particular the vulnerability for a user for whom the two queries return different answers is 1.35×10^{-2} as compared to 0.00148×10^{-2} in the average. For these unfortunate users the queries fully reveal their birthday and their risk is over 91 times larger than expected.

2.1.4 Relative vs. Absolute

The definition of information flow in Equation 0.1 is *relative*. It relates the risks in a level of adversary knowledge after observing the output of a query to the risk before the observation. The relative measure is useful for a variety of reasons:

- It can compare one query to another on the basis of the risks associated with each.

- It measures the dangers of query in isolation of much of the details of what has been revealed before.

- Measures of relative flow are sometimes easy to bound. For example, relative vulnerability can be bounded knowing only how many possible outputs a query has [57]. This can be seen in examples in Sections 2.1.1,2.1.3: the increasing vulnerability there would be identical for any deterministic query with 2 possible outputs, or any two queries with 4 possible output combinations.

On the other hand, relative measurement is not well suited in addressing the challenges posed in this thesis. The goal of the thesis is to limit risk in information release. It is not sufficient to merely compare queries relative to each other in isolation of the knowledge already revealed; we need to reason about the absolute level of knowledge, and therefore absolute risk, accrued by queries in aggregate. A query which reveals a lot in the relative sense but does so to within a reasonable risk threshold is not as big of a concern as a query which reveals a little but does so beyond the threshold. Without tracking knowledge the correct distinction between those two cases cannot be made.

2.2 Probabilistic Programming

Probabilistic programming (PP) is the idea of viewing programs as (conditional) probability distributions and taking advantage of this view to perform probabilistic inference, in effect inverting the semantics of a program to reason about what its outputs imply about its inputs.

Writing programs with non-deterministic elements is much easier than reasoning about the eventual distributions induced by said programs. Likewise, inference is a conceptually and computationally more difficult task than description of a probabilistic process itself. A big part of the motivation for probabilistic programming is thus the goal of making probabilistic inference more accessible to non-experts [48]. For the purposes of this thesis, probabilistic programming will serve both as a conceptual modeling tool to describe various information release scenarios, but also as the computational tool for experimentation and for implementation of security policies.

As a simple example, consider the birthday examples of Sections 2.1.1 and 2.1.3. We can express the prior uniform demographic information as a function sample_user which samples the three demographic attributes uniformly from given ranges:

$$\text{sample\_user} : \mathit{unit} \to \mathbb{D}\,(\mathit{year} \times \mathit{day} \times \mathit{gender})$$

```
let sample_user () ↦
  let byear ← uniform {1960, 1961, ···, 1996} in
  let bday ← uniform {0, 1, ···, 364} in
  let gndr ← uniform {Male, Female} in
    (byear, bday, gndr)
```

The expression uniform_in S uniformly samples an element from a set S. Running sample_user many times we would find that the distribution of its output approaches that of the random variable X_{demo} in Section 2.1.1 and that in the limit the two are equal. The core of probabilistic programming is the equating of the expression sample_user (), which produces a value non-deterministically, with the distribution of those values.

The process of adversary learning about the demographic by observing the output of birthday_query can be captured using the following program:

```
let demo ← sample_user () in
let o ← birthday_query 100 demo in
condition o = true in
  demo
```

The **condition** P **in** e expression (sometimes alternatively called "observe" or "assert") serves to probabilistically condition the distribution of program variables up that point and evaluates the expression e on a distribution in which P holds. Conceptually we can view conditioning as throwing away samples for which the predicate fails and only considering the distribution of the ones that remain. The end result of this program is distributed as the random variable $X_{\text{demo}} \mid \big(X_{\text{today}} = 100, X_o = \textbf{true}\big)$.

The same idea applies to multiple observations as in the same of two birthday queries:

```
let demo ← sample_user () in
let o1 ← bday_query 100 demo in
let o2 ← bday_query 101 demo in
condition o1 = true in
condition o2 = true in
  demo
```

For most of this thesis we will use pseudo-code of this form and ignore how to actually produce distributions from these programs. In Section 6 we will formally define a simpler language and its probabilistic semantics as well as address the issue of computational tractability of inference. Below we briefly summarize some notation related to probabilistic programs and probability in general that we will make use throughout the thesis.

Notation 1 (Probability and Programs)**.**

- We write $f : \mathcal{A} \to \mathbb{D}(\mathcal{B})$ to designate a *probabilistic* function of type $\mathcal{A} \to \mathcal{B}$, i.e., a function that takes an element of $\mathcal{A}$ as input and probabilistically returns an element of $\mathcal{B}$ (so $\mathbb{D}$ should be read "distribution over").
- Given a program expression exp that evaluates to a distribution, we write X_{exp} as the random variable distributed according to the possible values of exp and write $\Pr\left(X_{\text{exp}} = \text{v}\right)$ for the probability that the expression's value is equal to v.
- Given a random variable X, we will write $x \leftarrow X$ as the process of sampling a value from the distribution $\Pr(X)$ and write $x \in X$ to designate any value x that has non-zero probability according to $\Pr(X)$. We will sometimes write $\text{support}(X)$ as the set of non-zero probability values of X when the dual-use of X as a set and a random variable would be otherwise confusing.

2.3 Secure Multiparty Computation

Though a completely formal treatment of the security provided by SMC [43,44,59,105] is beyond the scope of this thesis we provide here a brief background on the topic. Throughout this chapter we assume that all parties are *semi-honest*. This means that they run any specified protocol exactly as prescribed, but may try to infer information about other parties' inputs based on their *view* of the protocol execution. (A party's view consists of its local state, along with all messages that it sent or received.) We also assume that parties do not collude. SMC can be extended to malicious parties who behave arbitrarily, as well as to handle collusion, but these complicate the treatment and are tangential to our main thrust.

The symmetric sharing setting considers a scenario where N mutually distrusting parties $P_1, \ldots, P_N$ wish to compute some query $f(s_1, \ldots, s_N)$ of their respective inputs, while ensuring *privacy* of their inputs to the extent possible. In an ideal scenario, the parties would all have access to a trusted entity P_T who would compute the function on their behalf. That is, each party P_i would simply send its input s_i to P_T,

who would in turn evaluate $(out_1, \ldots, out_n) = f(s_1, \ldots, s_n)$ and return the result out_i to party P_i. We write $out = f(...)$ if the same output is sent to all participants. If f is a probabilistic function, then P_T evaluates it using uniform random choices.

Fix some distributed protocol Π that computes f. (This just means that when the parties run the protocol using their inputs $s_1, \ldots, s_N$, the protocol terminates with each party holding output out_i.) We say that Π is *secure* if it simulates the ideal computation of f described above (where a trusted entity is available). Specifically, an execution of Π should reveal no information beyond what is revealed in the ideal computation. [3] This is formally defined by requiring that any party in the ideal world can sample from a distribution that is "equivalent" to the distribution of that party's view in a real-world execution of Π. Since any party P_i in the ideal world knows only its own input s_i and the output out_i that it received from P_T, this implies that Π achieves the level of privacy desired. We stress that not only is no information (beyond the output) about any *single* party's input is revealed, but also no *joint* information about several parties' inputs is revealed either (just as in the ideal world).

The cryptographic literature considers several notions of what it means for two distributions D, D' to be "equivalent". The simplest notion is to require D, D' to be *identical*. If this is the case for the distributions described above, then Π is said to achieve *perfect* security. Alternately, we may require that D, D' be indistinguishable by computationally bounded algorithms. (We omit a formal definition, though remark that this notion of indistinguishability is pervasive in all of cryptography, beyond SMC.) In this case, we say that Π achieves *computational* security. Perfect security is achievable for $N \geq 3$, whereas only computational security is possible for $N = 2$.

In Section 5 we will look at how the parties can scrutinize the queries they execute securely to make sure that what is revealed by their outputs (and policy decisions) is sufficiently safe. The approach will follow the general techniques used in the previous settings though adjusted to take into account correlations among multiple parties and the inability of parties to hold representations of each other's knowledge.

[3] Readers who are familiar with SMC may note that this definition is slightly simpler than usual. The reason is that we are considering semi-honest security, and in this paragraph assume a deterministic function for simplicity. We are also glossing over various technical subtleties that are inessential to get the main point across.

Chapter 3

Dynamic enforcement

In this Chapter we describe means with which an *individual* user can enforce dynamically limits in adversary knowledge about *their* secret. The setting here is dynamic in two senses. First dynamic is in contrast with static measurements of expected risk which we noted in Section 2.1.2 is not necessarily applicable to limiting risk for an individual. Second, the enforcement mechanism applies to streams of queries, where at any time, some queries have been considered so far, and unknown queries will need be handled in the future.

We begin with an elaboration of our running birthday example in Section 3.1. We then proceed to the details of knowledge tracking using beliefs in Section 3.2, the measurement of risk using beliefs in Section 3.3, and complete the approach with the mechanism for limiting risk in Section 3.4. We discuss the approach and related work in Section 3.5.

This chapter is based on the work originally published in the 2011 Computer Security Foundations Symposium [70], a 2013 issue of the Journal of Computer Security [72], and a University of Maryland technical report released in 2011 [71]. The work was done in collaboration with Michael Hicks, Stephen Magill, and Mudhakar Srivatsa.

3.1 Example: Dynamic birthday queries

To motivate the setting concretely, recall the bday_query queries of Section 2.1.1, which we replicate below:

$$\text{bday\_query} : \mathit{day} \times (\mathit{year} \times \mathit{day} \times \mathit{gender}) \to \mathit{bool}$$

```
let bday_query: today (byear, bday, gndr) ↦
  if today < 365 − 7 then (bday >= today and bday + 7 < today)
  else (bday >= today or bday < (today+7) mod 365)
```

We have looked at this query as it reveals information about users drawn from a population described by the $\Pr(X_{\text{demo}})$ composed of the following three independent uniform distributions:

$$\Pr\left(X_{\text{byear}} = y\right) = \frac{1}{37} \qquad \text{for } y \in \mathit{year} = \{1960, \cdots, 1996\}$$

$$\Pr\left(X_{\text{bday}} = d\right) = \frac{1}{365} \qquad \text{for } d \in \mathit{day} = \{0, \cdots, 364\}$$

$$\Pr\left(X_{\text{gndr}} = g\right) = \frac{1}{2} \qquad \text{for } g \in \mathit{gender} = \{\text{Male}, \text{Female}\}$$

The above distribution also served to model the prior knowledge an adversary has about the population. In contrast to the analysis in Chapter 2, we are now concerned with a single individual from this population who has a particular demographic. We assume that the real values of the demographic are consistent with the adversary knowledge in that the real values could have been drawn from the assumed distribution. We will call the real values S_{byear}, S_{bday}, S_{gndr} or S_{demo} in combination. Thus $S_{\text{demo}} \in \text{support}\,(X_{\text{demo}})$.

Let us consider what happens with adversary knowledge for a user whose birthday is 101, after the adversary learns the output of the query when today is 100. Recall in Section 2.1.1 we showed that there are two possible states of adversary knowledge in the posterior, after learning the output of the query. Which one is reached depends on the output. For a user with $S_{\text{bday}} = 101$, the query can only return in the affirmative. Hence after learning this, the adversary's belief about the user's birthday becomes:

$$\Pr\left(X_{\text{bday}} = d \mid X_o = \textbf{true}, \cdots\right) = \frac{1}{7} \qquad \text{for } d \in \{100, \cdots, 106\}$$

This means that the adversary has now narrowed down the possible values of birthday to one in seven possibilities, each with equal probability. Let us use the probability of the most probable secret value as the measure of risk. In this case it is $\frac{1}{7}$. Now, if the user is content with this level of risk then they would be OK with letting the adversary learn the answer to the query. Let us assume that the user is comfortable with the measure of risk being no more than $\frac{364}{365}$. That is, they do not want their birthday to be known exactly, but anything less that is deemed safe. The user could thus use the speculation of the outcome of answering as a means of judging the risk involved. We will later see how to handle situations where the query includes probabilistic choice (randomness) which the user must take into account during this speculation.

Let us say that the user did answer the query. He or she can then update the estimated knowledge of the adversary to the random variable X_{bday} conditioned on $X_o = \textbf{true}, X_{\text{today}} = 100$. This *revised* belief can then serve as the prior when speculating as to the outcomes of a future query. Let us consider then a second query with today $= 101$. This query can again return only true and revise the belief of the adversary to the following:

$$\Pr\left(X_{\text{birthday}} = d \mid X_{o_2} = \textbf{true}, X_{o_1} = \textbf{true}, \cdots\right) = \frac{1}{6} \qquad \text{for } d \in \{101, \cdots, 106\}$$

Notice that were this query to be answered, the measure of risk increases to $\frac{1}{6}$, which is still below the comfort. It would seem that the user should reveal the answer to this query. Consider, though, what would be the situation if the user's birthday was 100 instead of 101. The first query's outcome would be unchanged, but the second query would return false and revise the belief of the adversary to:

$$\Pr\left(X_{\text{birthday}} = d \mid X_{o_2} = \textbf{false}, X_{o_1} = \textbf{true}, \cdots\right) = 1 \qquad \text{for } d \in \{100\}$$

The risk in this belief reaches the threshold at which the user is no longer comfortable which would mean that the user should not answer the query. If we assume that the adversary knows the process by which the user decides to answer the queries, however, we see that there is a problem. The birthday value of 100 is the only possibility that would result in the second query revealing too much information. Therefore, if the adversary is told that the second birthday query will not be answered after learning the first's output was true, they can infer that $S_{\text{birthday}} = 100$.

The root of the problem here is that since the user uses their secret value in the determination of whether to answer a query, there is danger that this secret leaks information through policy decisions. The problem of the refusal to answer as potentially leaking information is well known. The work of Kenthapadi et al., in particular, addresses the problem in a setting similar to ours [53]. In general, a simple solution to the problem is to make sure policy decisions do not depend on the true secret value. A slight refinement of this approach is to make sure policy decisions are *simulatable*, in that the adversary has all the knowledge necessary in order to determine the policy decision without asking the user for it.

In our setting, the approach to overcome the problem of leaky policies is to broaden the range of possibilities a user speculates about when deciding to answer a query, from just those that are consistent with their secret value, to those that are consistent with all possible secret values according to the knowledge of the adversary. The decision procedure can be made simulatable in this case as the adversary is already assumed to have the knowledge we attribute to them.

Note that this approach is partially static and partially dynamic. On one hand the policy decision does not depend on the true value of the secret. On the other hand, it only considers values consistent with adversary knowledge which does depend on what has been revealed about the true secret in the past. In this sense the approach can be described as modeling adversary knowledge dynamically, but performing enforcement statically.

In terms of our running example, when a user is considering the second birthday query with $\text{today} = 101$, they must consider what would happen to the adversary knowledge under any of the 6 secret values they adversary considers possible. For most of them (101 through 106), the result will be sufficiently comfortable to the user. For one of them (100), however, the query would reveal the secret birthday exactly. Thus the user must reject this second query even if their birthday is not 100.

This mode of enforcement might seem overly restrictive to the point that no additional birthday queries could be answered after the first. To the contrary, if the second birthday query had $\text{today} = 102$ instead, then no matter the outcome, the adversary will get no better idea of the birthday than one in two possibilities. In that case, this alternate second query would be allowed.

In the rest of this chapter we formalize the policy and enforcement mechanism we just informally described. To this end we start with *knowledge tracking* in Section 3.2. We then describe *risk assessment* and the policy definitions aimed at limiting risk in Section 3.3. We conclude with *simulatable enforcement* of these policies in Section 3.4.

3.2 Knowledge tracking

In this section we describe how to track adversary knowledge using probabilistic programs. This is done essentially as in the experimental protocol of Clarkson et al. [24] but with simplified notation.

- Let S be a set of secret values and let $\text{prior} : \mathbb{D}(S)$ be a distribution over the set S that is the prior belief of an adversary about the true secret. The function sample_user of Section 2.2 serves to produce a prior belief:

```
let sample_user () ↦
  let byear ← uniform {1960, 1961, ···, 1996} in
  let bday ← uniform {0, 1, ···, 364} in
  let gndr ← uniform {Male, Female} in
    (byear, bday, gndr)
```

- Let $s \in S$ be the true secret value and assume that s is consistent with the prior: $s \in X_{\text{prior}}$.

- Let q be a potentially non-deterministic query with $q : S \to \mathbb{D}(O)$ for some set of outputs O. We model queries here as having only the secret as input so the differing birthday queries in our running example can be modeled as single-input queries with the today argument already provided.

The essence of knowledge tracking is encapsulated in the function posterior below which is effectively an expression of Bayesian inference in the form of a probabilistic program:

$$\text{posterior} : \mathbb{D}(S) \times (S \to \mathbb{D}(O)) \times O \to \mathbb{D}(S)$$

```
1 let posterior: prior q o ↦
2   let o' ← q prior in
3   condition o' = o in
4     prior (* now the revised posterior *)
```

The output of the program is the subject of the following core lemma.

Lemma 2 (Experimental protocol of Clarkson et al. [24])**.** *Assume that the adversary believes that* o *is sampled from (*q s*) then the program (*posterior prior q s*) produces the adversary's posterior belief about the secret after learning the output of* q *was* o*.*

The assumption in the Lemma as to the origin of the output o is critical; the adversary will only make this belief revision if he thinks that the channel through which he is observing the secret is indeed the query q executed faithfully by the user. On the flip side, if a user wishes to accurately model adversary belief, he must convince the querier to trust that the outputs are faithfully produced. This is one among a few important assumptions that underlie this section and this thesis. They are important and recurring enough to warrant a definition.

Definition 3 (**Standard interaction assumptions**)**.**

1. A belief attributed to an adversary is indeed their actual belief.

2. Adversaries are perfect Bayesian reasoning agents. In this Chapter this just means that the adversaries guess the secret in an optimal manner given their observations. In the next Chapter, the adversaries will also have choices to make before the exploitation of the secret.

3. Users respond to queries in an honest manner, in that if they answer, the answer was produced on running the query on the actual secret.

4. Adversaries know exactly the process with which a user responds to their queries, whether they are rejected or not.

5. The mechanisms we model are the sole avenues with which the adversary learns information about the user. That is, we will not consider adversaries learning about the secret using means other than the query interface we describe. Additionally we ignore the possibility of side channels, e.g. timing.

We will make these assumptions implicitly throughout this thesis. We further discuss the first of these assumptions in Section 3.5.

Notation 4 (Strings)**.**

- ϵ is the empty string.
- Given alphabet $\mathcal{S}$, strings are given type $\mathcal{S}^* \stackrel{\text{def}}{=} \{\epsilon\} \cup \mathcal{S} \cup \mathcal{S}^2 \cdots$. We will use capital letters $(L, H, O, \cdots)$ to refer to strings. We will sometimes write S^t to designate an element of $\mathcal{S}^*$ that is t elements long and a_t to refer to the t^{th} element in some string S.
- Given $S \in \mathcal{S}^t$ and $a \in \mathcal{S}$, the concatenation of S with a and the reverse are written $S{\cdot}a \in \mathcal{S}^{t+1}$ and $a{\cdot}S \in \mathcal{S}^{t+1}$, respectively.

Lemma 2 can be applied recursively to model the knowledge of an adversary over time: the posterior belief can be used as the prior for subsequent queries. If we have a sequence of queries and outputs then the revised posterior belief of the adversary after learning outputs of all of them is produced by the following program:

$$\text{posterior\_sequence} : \mathbb{D}\,(S) \times (S \to \mathbb{D}\,(O))^* \times O^* \to \mathbb{D}\,(S)$$

```
let posterior_sequence: prior (q·ϵ) (o·ϵ) ↦
  posterior prior q o
let posterior_sequence: prior (q·Q) (o·O) ↦
  posterior_sequence (posterior prior q o) Q O
```

3.3 Risk assessment

Knowledge tracking computes the adversary's belief about the user's secret after learning the output of a sequence of queries. This distribution, which we will call $\text{belief} \in \mathbb{D}(S)$, can be measured for the risk it represents to the user. We have already seen several options for this measurement in Section 2.1. In this work we base risk on the notion of *vulnerability* which aims to quantify the probability that the adversary will guess the secret in one try. Vulnerability is an attractive option because of its direct connection to a threat scenario (adversary guessing).

In our dynamic setting, the guessing interpretation of vulnerability will not hold. Even though in our setting the true secret is known, we cannot use it in calculating risk. Were such a quantity used for policy enforcement, it could leak information about the secret. Thus our measurement will be based purely on the tracked adversary belief, independent of the true secret value. Due to this, our measurement of risk will correspond to what the adversary *believes* are his chances of guessing the secret in one try, even though we, knowing what the true secret is, might know better. For example, if the adversary thinks the most likely secret is 42 and has probability 0.5, then he believes he has 50% chance of guessing it correctly in one try. The user, on the other hand, knows that the actual probability is either 0 if the secret is not 42 or 1 if it is. We argue that this alternate interpretation is still attractive as bounding the risk from the adversary's perspective, we are essentially making sure the adversary never becomes confident enough to exploit their knowledge.

Definition 5 (Vulnerability)**.** *Vulnerability* of a random variable, written $V(X)$ is defined as $V(X) \stackrel{\text{def}}{=} \max_x \Pr(X = x)$. Vulnerability is the probability with which the adversary believes he can guess the secret in one try.

In our running example we stipulated that only a part of the secret is sensitive, the bday field of the demographic. In general we will assume that there is some sensitive property of a secret that needs to be protected, not necessarily a subset of the fields in a secret. We model this by introducing a function $\text{target} : S \to T$ to some set of property values that is of interest to assessing risk. In our running example target would merely select the bday field:

```
let target: (byear, bday, gndr) ↦ bday
```

A vulnerability threshold along with the target property to be protected to a degree specified by the threshold constitute a policy:

Definition 6 (Knowledge-Threshold Security)**.** Let $\text{target} : S \to T$ be a deterministic function describing property about the secret that is sensitive. Let $h \in [0, 1]$ be a vulnerability threshold. A belief belief satisfies $(target, h)$ *knowledge-threshold security* as long as $V\left(X_{(\text{target belief})}\right) \leq h$.

In the next section we will show how the user can enforce knowledge-threshold security when deciding which query to answer.

3.4 Knowledge Threshold enforcement

Though Lemma 2 requires that queries be evaluated honestly on the true secret and the outputs provided unmodified, the user can still speculate as to what the knowledge of the adversary would be for any possible output of a query, not necessarily the output that he samples when running it. This has the benefit of being independent of the true secret value (conditional on the knowledge already contained in the adversary belief). Because of this, it can be used for policy enforcement without leaking additional information about the secret. Speculation over all outputs is the function of the following definition.

Definition 7 (Worst-case Posterior Vulnerability)**.** The *worst-case posterior target vulnerability* given a prior belief and query q is the maximal vulnerability achievable in all possible posterior beliefs that can result in observing the output of the query. It is defined as below:

$$V^{max}(\text{belief}, \text{q}) \stackrel{\text{def}}{=} \max_{o \in X_{(\text{q belief})}} V\left(X_{(\text{posterior belief q o})}\right)$$

This worst-case measure can then be used to bound risk in revealing query outputs. Let h be a vulnerability threshold and belief be a prior belief with $V(X_{(\text{target belief})}) < h$. That is, the belief does not already exceed the risk threshold. The following program defines the query interface that is the goal of this Chapter.

```
1 | let query: s belief q h ↦
2 |   if V^max(belief, q) ≤ h then
3 |     let o ← q s in
4 |     let post ← posterior belief o in
5 |       (post, Accept o)
6 |   else
7 |     (belief, Reject)
```

The function produces a revised adversary belief along with a query result to be output to the adversary (or a rejection).

Theorem 8. *Under the standard interaction assumptions, the query mechanism above preserves* $(target, h)$*-security. That is, if an adversary has belief belief that is* $(target, h)$*-secure, then their posterior belief after learning the (second) output of query, will still be* $(target, h)$*-secure.*

Proof. (sketch) In the case that query returns Reject then the belief of the adversary does not change as the condition on Line 2 does not involve the secret. Hence the unchanged belief is still $(target, h)$-secure.

In the other case query returns an output o. By Lemma 2, we know that the adversary's belief becomes posterior belief o. As the conditional on Line 2 passed, it must have been that the target vulnerability of posterior belief o' for every possible o' was below h, including the posterior for o. Thus posterior belief o, the adversary's new belief is $(target, h)$-secure. □

3.5 Discussion

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

3.5.1 Initial belief

For P to enforce a knowledge-based policy on queries by P_1 requires that P estimate P_1's belief about the possible valuations of P's secret data. When P_1 has no particular knowledge of P'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_1 might have inside knowledge, P 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_1 having acquired illicit information.

Our examples and benchmarks in Section 6 largely consider personal, private information, e.g., gender, age, level of education, etc. This information can be drawn from public sources (e.g., Facebook demographics [6]). However, the distributions we experiment with are oversimplifications of the actual, reported distributions: they are largely simple, uniform distributions. This does not preclude more complex distributions in the prior. Due to computational issues which we will see in Section 6 it might become too expensive to be perfectly accurate in this regard.

If the initial belief estimate is (very) inaccurate, then P risks releasing information to P_1 contrary to his intended policy. To "hedge his bets" against this possibility, he might choose to maintain a set of possible beliefs $\text{Beliefs} = \{\text{belief}_i\}_i$, rather than a single belief belief, and only release if the threshold was satisfied for every $\text{belief} \in \text{Beliefs}$. We return to this idea in when we discuss differential privacy in Section 3.6 below.

3.5.2 Collusion

Assuming that P maintains separate beliefs and thresholds for distinct adversaries P_1 and P_2, the possibility of collusion arises. Following the example in Section 3.1, P's privacy would be thwarted if he shared the output of the first birthday query with P_1 and the output of the second with P_2 but then P_1 and P_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 two pieces of his data, each visible to one of the queriers but which, when brought together, violated privacy.

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 P_1's or P_2's query (whichever was last). This approach is akin to having

a global privacy budget in differential privacy (see Section 3.6) or a single, global access control policy and would obviously harm utility. Moreover, the confidence of an agent can actually decrease as a result of observing outputs of probabilistic queries [1]. This can occur when the agent is initially confident in something that is not true, or as a result of improbable outputs of probabilistic queries even when the agent is not mistakenly confident. Due to this non-monotonicity in knowledge, if agents purported to be colluding are in fact not colluding then a global belief might under-approximate a non-colluding agent'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 decreased certainty due to queries, rational adversaries, interested in maximizing their chances of guessing the secret value, will take all outputs into account, even unlikely outcomes of probabilistic queries. Though they might be detrimental to certainty, in expectation, all queries increase chances of guessing the secret correctly. This point is discussed further in the work of Clarkson et al., specifically the Section on Accuracy, Uncertainty, and Misinformation [24].

3.5.3 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 [85]. 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.

3.5.4 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) [105]. We will cover this application in Section 5 later in this thesis. 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

[1] An example of a probabilistic query following the demographic scenario is the query spec_year of Section 6.2.

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.

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 [37], service providers have turned to *fingerprinting*, which aims to identify a user based on available browser characteristics [17]. 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. The work of Besson et al. [16] proposes an approach similar to this. Though they use the same security criterion of vulnerability thresholds, their technique adds noise to fingerprinting programs in order to achieve security. Their work applies to individual fingerprinting programs that are given and analyzed ahead of time, hence that work does not have to take into account the dynamics and issues of policies leaking information.

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. [97]. 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 prebelief 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 misestimate of the prebelief: the number of reported leaked bits will be conservatively high.

3.6 Related work

We consider four areas of work related to work presented in this Chapter: 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.

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 [96] 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 [12] 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 [99] mediates social network data accesses by third-party application extensions. Privad [50] 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. A recent survey [88] lists 17 different approaches for implementing decentralized social networks with various forms of access control.

Knowledge-based security policies generalize access control policies: if we maintain a belief estimate for each principal P_i, then the equivalent of granting P_i access to data d is to just set P_i's knowledge threshold for target d to 1; for principals R_i 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.

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 [23] 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. [11] automate the synthesis of such equivalence relations and quantify leakage by computing the exact size of equivalence classes. Köpf and Rybalchenko [56] 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 [84] present a similar 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 security criterion that is applicable to the dynamic setting as opposed to the static one. Recall the *conditional vulnerability* metric we discussed in Section 2.1. It can be defined using our notation as follows.

Definition 9. A query q is *(static) vulnerability threshold secure* iff for

$$V(\,\text{belief}\,,\text{q}) \stackrel{\text{def}}{=} \mathop{\mathbb{E}}_{\text{o}\leftarrow X_{\text{q belief}}} \left[V(X_{\text{posterior belief q o}})\right]$$

we have $V(\,\text{belief}\,,\text{q}) \leq h$ for some threshold h.

The above definition is an *expectation* over all possible query outputs, so unlikely outputs have less influence. Our notation of threshold security (Definition 7) replaces the expectation over $\text{o} \leftarrow X_{(\text{q belief})}$ with maximization over $\text{o} \in \text{support}\left(X_{(\text{q belief})}\right)$.

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. As we have demonstrated in Section 2.1.2, the static notion can greatly underestimate the risk for under-represented individuals.

The idea of strengthening of an entropy measure by eliminating the expectation has been briefly considered by Köpf and Basin [55]. 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 q_1 followed by a query q_2 which use the same code as bday_query but with today $= 200$ in the first case and today $= 205$ in the second. With our system and birthday $= 100$ the answer to q_1 is **false** and with the revised belief the query q_2 will be accepted for any threshold $h \geq \frac{1}{7}$. 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 birthday $\in \{205, 206\}$ which would require threshold $h \geq \frac{1}{2}$. 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 [24]. We note these advantages come at the cost of maintaining on-line belief models.

McCamant and Ernst's FlowCheck tool [75] 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).

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* [39]: 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 adversary'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 [95]. 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 the values of a secret. To bridge this gap we can suppose that the secret in our case is a subset of H, which is the set of "records". So here S, the set of possible secret values, becomes $\mathcal{P}(H)$. Then we can state adversarial privacy as follows.

Definition 10 (Adversarial Privacy)**.** Given a query q, let O be the set of all possible outputs of q and Beliefs be a set of potential adversary beliefs. We say that program q is ε-*adversarially private* w.r.t. Beliefs iff for all belief $\in$ Beliefs, all elements $x \in H$, and all possible outputs $o \in O$, that $\Pr\left(x \in X_{\text{posterior prior q o}}\right) \leq e^{\varepsilon} \cdot \Pr\left(x \in X_{\text{prior}}\right)$.

This definition says that query q is acceptable when the output of q increases only by a small multiplicative factor an adversary's certainty that some element x is in the secret, for all adversaries whose prior beliefs are characterized by Beliefs. Rastogi and Suciu show that defining Beliefs 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 and require the probability of one element's presence to not be positively correlated with another's.

Comparing Definition 10 to Definition 6 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 prior belief, 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 against a much larger number of adversaries, i.e., all those whose prebeliefs are in $PTLM$.

Clarkson and Schenider [25] also demonstrate the equivalence of differential privacy and a notion of information flow. They show that an output of an ε-differentially

private program cannot reveal more than $\varepsilon \cdot \log_2 e$ bits (as measured by mutual information) of information about any element that is not in the input database. Barthe and Köpf [14] also make connections between differential privacy and relative information leakage based on min-entropy. Alvim et al. [9] show a comparison with min mutual information.

The strong privacy guarantees based on differential privacy seem to come at a correspondingly 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. [61] 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 bday_query from our running example. The user's birthday being/not being in the query range influences the output of the query only by 1 (assuming **true**/**false** are akin to 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. [2]

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 limit on the number of queries since we can track the released information exactly. Even if the threshold is reached, queries can still be answered as long as they do not reveal information beyond what has already been revealed. Differential privacy imposes a 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.

[2] By our calculations, with privacy parameter $\varepsilon = 0.1$ recommended by Dwork [39], the probability the query returns the correct result is approximately 0.5249.

Chapter 4

Predicting Risk with Evolving Secrets

Quantitative information-flow models [20, 23, 24, 83, 100] and analyses [11, 56, 70, 84] typically assume that secret information is *unchanging*. Likewise, in the previous chapter, we have assumed that secrets do not change. But real-world secrets evolve over time. Passwords, for example, should be changed periodically. Cryptographic keys have periods after which they must be retired. Memory offsets in address space randomization techniques are periodically regenerated. Medical diagnoses evolve, military convoys move, and mobile phones travel with their owners. Leaking the current value of these secrets is undesirable. But if information leaks about how these secrets change, adversaries might also be able to predict future secrets or infer past secrets. For example, an adversary who learns how people choose their passwords might have an advantage in guessing future passwords. Similarly, an adversary who learns a trajectory can infer future locations. So it is not just the current value of a secret that matters, but also how the secret changes. Methods for quantifying leakage and protecting secrets should, therefore, account for these *dynamics*.

In Section 4.1 we propose a model for measuring risk with dynamic secrets. To capture the dynamics of secrets, our model uses *strategy functions* [104] to generate new secret values based on the history of past secrets and query outputs. For example, a strategy function might yield the GPS coordinates of a high-security user as a function of time, and of the path the user has taken so far. Because of this the risk incurred by answering queries cannot be attributed solely to the belief of the adversary concerning the secret value right after the query. The secret might evolve from that point to increase (or decrease) the eventual risk even if no further interaction between the user and the adversary takes place. This work is thus aims to *predict* the risk over time.

Due to this predictive setting we must also model the behavior of the adversaries in their query choice and their exploitation of revealed information. Our model includes *wait-adaptive adversaries*, who can observe execution of a system, waiting until a point in time at which it appears profitable to attack. For example, an attacker might delay attacking until collecting enough observations of a GPS location to reach a high confidence level about that location. Or an attacker might passively observe application outputs to determine memory layout, and once determined, inject shell code that accesses some secret.

Modeling adversary behavior requires us to model their goals and introduces the possibility of their goals being distinct from the goals of the user (limiting their notion of risk). Most approaches to QIF consider leakage only from the adversary's point of view, whereas the goals and concerns of the *defender*—i.e., the user interested in protecting information—are overlooked. While in many cases the adversary's gain is directly and inversely related to the defender's loss, this is not always the case. Out model allows us to explore this point of view, that is, that the actual leakage

of information due to queries is linked to the *defender's loss of secrecy* and not necessarily the *adversary's gain of information*.

In Section 4.2 we propose an information-theoretic metric for quantifying defender loss and adversary gain with dynamic secrets. Our metric can be used to quantify loss of the current value of the secret, of a secret at a particular point in time, of the history of secrets, or even of the strategy function that produces the secrets. We show how to construct an optimal wait-adaptive adversary with respect to the metric, and how to quantify that adversary's expected *gain* and defender's expected *loss*, as determined by a scenario-specific *gain* and *loss* functions. In Section 4.2.2 we show how our metric generalizes previous metrics for quantifying leakage of static secrets, including vulnerability [100], guessing entropy [55], and g-vulnerability [10].

In Section 4.3 we put our model and metric to use by implementing them in a probabilistic programming language [48] and conducting a series of experiments. Several conclusions can be drawn from these experiments:

- Frequent change of a secret can increase gain, even though intuition might initially suggest that frequent changes should decrease it. The increase occurs when there is an underlying order that can be inferred and used to guess future (or past) secrets.
- Wait-adaptive adversaries can derive significantly more gain than adversaries who cannot adaptively choose when to attack. So ignoring the adversary's adaptivity (as in prior work on static secrets) might lead one to conclude secrets are safe when they really are not.
- A wait-adaptive adversary's expected gain increases monotonically with time, whereas a non-adaptive adversary's gain might not.
- Adversaries that are *low adaptive*, meaning they are capable of influencing their observations by providing low-security inputs, can learn exponentially more information than adversaries who cannot provide inputs.
- Both adversary gain and defender loss are necessary to quantify vulnerability of the secret and that if the distinction is ignored, incorrect conclusions can follow.
- Sound overestimation of adversary gain can lead to unsound underestimation of defender loss (once again motivating the need to distinguish the two).

We conclude the chapter in Section 4.4 with related work.

The work presented in this chapter was originally published in the 2014 Symposium on Security and Privacy [66], the 2014 Workshop on Foundations of Computer Security [67], and a University of Maryland technical report released in 2014 [68]. The work was done in collaboration with Mário Alvim, Michael Clarkson, and Michael Hicks.

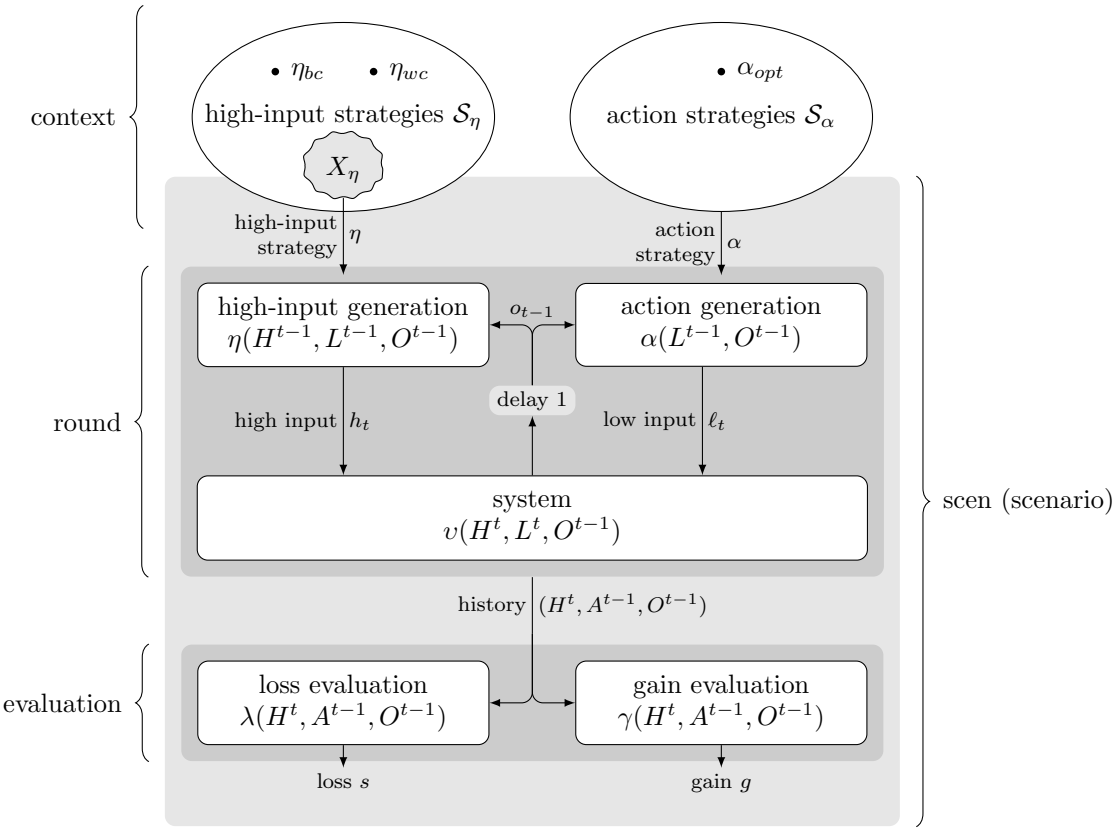


Figure 4.1: Model with changing secrets.

4.1 Model for dynamic secrets

The model we present greatly expands the simple channel model of Figure 2.1 in order to handle the additional nuances of the setting. Most importantly, the model captures an iterative process instead of a one-shot channel execution. Though the simple channel model could be applied iteratively to model the execution of multiple queries, our need to take into account what exactly happens in between the iterations requires us to capture the process of iteration within the model itself. In addition to iteration, we have the following features:

Dynamic secrets Typical information flow models assume the defender's secret is fixed across adversarial observations. Our model permits the defender to replace the existing secret with a new secret without resetting the measure of leakage; thus from the adversary's point of view, the secret is a *moving target*.

Interactivity To model a powerful adversary, our model allows *feedback* from outputs to inputs. That is, the adversary can use knowledge gained from one obser-

vation to choose a subsequent input to feed into the system. We call an adversary with this *low adaptive*.

Input vs. attack Our model distinguishes inputs that are attacks from those that are not. For example, an adversary might navigate through a website before uploading a maliciously crafted string to launch a SQL injection attack; the navigation inputs themselves are not attacks. This distinction is important when quantifying information flow: the leakage at one point is taken with respect to the expectation of a future attack.

Delayed attack Just as adversaries are permitted to decide *what* input to provide, they are also permitted to decide *when* is the best time to attack, and this decision process will be considered when quantifying leakage. An adversary with this ability will be called *wait adaptive*.

The model is depicted in Figure 4.1, and consists of roughly three parts: the *context*, at the top defines the beliefs of the defender and adversary about the secret and its possible evolutions; the *round*, in the middle defines the iterative process of channel execution and secret evolution; and the *evaluation*, at the bottom captures exploitation which results in a measure of adversary gain and defender loss. The latter two phases together are termed the *scenario.* As a whole, the model essentially defines a two-player game with imperfect information and non-zero-sum payoffs. We will limit our analysis to situations where the defender's strategy or the adversary's strategy is fixed. This will let us define metrics by optimizing a party's actions without the need for costly equilibrium analyses that would otherwise be necessary.

We describe each part of our model in turn. The terminology that we use throughout the model is tabulated in Table 4.1.

4.1.1 Iterated rounds

The model depicted in Figure 4.1 characterizes an interaction between two participants, an adversary and a defender. The elements of the classic model from Figure 2.1 can be seen in the middle part of the figure, the *round*, which is iterated over a series of rounds (hence its name). Here, the defender (the box labeled *high-input generation*) starts by picking an initial secret (high) value out of a set $\mathcal{H}$, and may choose a different secret in subsequent rounds t (the secret value h is indexed by the round t). Likewise, in each round the adversary (the box labeled *action generation*) produces a low input drawn from a set $\mathcal{L}$. These inputs are fed into the system v which produces an observable, visible to both participants, that is drawn from set $\mathcal{O}$. The system may consider past observations as well when producing its result; it has type $\mathcal{H}^* \times \mathcal{L}^* \times \mathcal{O}^* \to \mathbb{D}(\mathcal{O})$. At some predetermined time T, or before then if it is in his interest, the adversary picks an exploit from a set $\mathcal{E}$, which is then evaluated, along with the history of events so far, to determine the gain of the adversary and the loss of the defender. We discuss the evaluation phase at the end of this section.

Notation	Explanation
$\mathcal{H}$	Finite set of high inputs
$\mathcal{L}$	Finite set of low inputs
$\mathcal{E}$	Finite set of exploits
history	The set $\mathcal{H}^* \times \mathcal{L}^* \times \mathcal{O}^*$
pubhistory	The set $\mathcal{L}^* \times \mathcal{O}^*$
$\mathcal{A}$	The set of low inputs and exploits, $\mathcal{L} \cup \mathcal{E}$ (also called actions)
$\mathcal{O}$	Finite set of observables
υ	The system function, of type $\mathit{history} \to \mathbb{D}(\mathcal{O})$
η	A high-input strategy, of type $\mathit{history} \to \mathbb{D}(\mathcal{H})$
α	An adversary's action strategy, of type $\mathit{pubhistory} \to \mathbb{D}(\mathcal{A})$
$\mathcal{S}_\eta$	The set of possible high-input strategies
$\Pr(X_\eta)$	The distribution of possible high-input strategies for a given scenario
$\mathcal{S}_\alpha$	The set of possible action strategies
λ, γ	Loss, and gain, functions, respectively, each with type $\mathit{history} \to \mathbb{D}(\mathbb{R})$
$T \in \mathbb{Z}^+$	The latest possible time to attack in a given scenario

Table 4.1: Elements of the model with changing secrets.

The history of an execution can be split into a *secret history* (high values) and a *public history* (low values and observables). The defender's view of an execution consists of both the secret and public history, whereas the adversary's view is restricted to the latter. The choices made in each round by the defender and adversary are specified in terms of *strategy functions*, which depend on their views of the execution. The defender's high-input strategy is a function $\eta : \mathit{history} \to \mathbb{D}(\mathcal{H})$ that produces the next high value given the history so far. For the adversary, the strategy is a function $\alpha : \mathit{pubhistory} \to \mathbb{D}(\mathcal{L} \cup \mathcal{E})$ that produces a low input or an exploit given the public history of the execution. These strategies are determined by the context in a manner described in the next section.

Next we characterize the scenario precisely. A single round of interactions is described by the round function (where we write $\mathcal{A}$ to mean the set $\mathcal{L} \cup \mathcal{E}$):

$$\mathrm{round}_{\eta,\alpha} : \mathit{history} \to \mathbb{D}(\mathit{history})$$

```
1 | let round_{η,α} : (H, A·e, O)↦(H, A·e, O)
2 | let round_{η,α} : (H, A, O)↦ // (A = ϵ or A ∈ L*)
3 |   let a←α(A, O) in
4 |   if a ∈ L then
5 |     let o←υ(H, A·a, O) in
6 |     let h←η(H, A·a, O·o) in
7 |       (H·h, A·a, O·o)
8 |   else // a ∈ E
9 |     (H, A·a, O)
```

The subscripts on the function identify the two strategy functions, and the proper arguments characterize the history of high inputs, actions (low inputs or exploits),

and observations made so far. Lines 2–9 consider the case that the last action of the adversary was a low input ℓ. In this case, the scenario computes next action of the adversary (line 3) and, if it is a low input (line 4), produces the next observation (line 5) and the subsequent high value using the defender's strategy (line 6). Each of these choices is appended to the existing history (line 7). If the adversary instead produces an exploit (line 8), then just this exploit is added to the history (line 9). An exploit is considered the final event of an interaction, and as such line 1 leaves the history untouched.

4.1.2 Context

The context describes the possible strategy functions used in a scenario. It includes a set of high-input strategies $\mathcal{S}_\eta$ and a distribution X_η on these strategies, which encodes an adversary's *prior knowledge* of the possible strategies the defender will use. This distribution helps establish the adversary's initial uncertainty about the defender's secret.

The context also includes a set of adversary strategies $\mathcal{S}_\alpha$ which enumerates the space of behaviors permitted to the adversary. While the defender's strategy can be any one drawn from X_η, our definition of information leakage assumes that adversaries are optimal and pick from the set $\mathcal{S}_\alpha$ only strategies maximizing expected gain. Later, we will define the set $\mathcal{S}_\alpha$ to restrict the adversary in several ways and in so doing show how to define standard QIF metrics.

4.1.3 Full scenario and evaluation

The impact of an exploit is evaluated by two functions, as shown at the bottom of the figure. The *gain function* $\gamma : \mathit{history} \to \mathbb{D}(\mathbb{R})$ determines the gain to the adversary; the *loss function* $\lambda : \mathit{history} \to \mathbb{D}(\mathbb{R})$ determines the defender's loss. Notice that gain functions are defined over the *history* of each of the relevant values, rather than, say, the most recent ones. This is because we are quantifying leakage about moving-target secrets, so we must discriminate current and past high values.

To use these functions to compute the information leakage, we can evaluate the scenario up to (at most) some time T and then compute the gain, and loss, of the final outcome. This is done according to the scen function.

$$\mathrm{scen}_{\gamma,\lambda} : \mathbb{Z} \times \mathcal{S}_\eta \times \mathcal{S}_\alpha \to \mathbb{D}(\mathbb{R} \times \mathbb{R})$$

```
1 | let scen_{γ,λ} : (T, η, α) ↦
2 |   let h_0 ← η(ϵ, ϵ, ϵ) in
3 |   let (H, A, O) ← round^T_{η,α}(h_0, ϵ, ϵ) in
4 |   let s ← λ(H, A, O) in
5 |   let g ← γ(H, A, O) in
6 |     (s, g)
```

Evaluation starts with computing the initial high value h_0 (line 2). It then computes T rounds using round (line 3): on the first iteration it uses arguments $(h_0, \epsilon, \epsilon)$,

which produces some output (H^2, A^1, O^1) that is used as input to the next call to round, whose output is passed to the next call, and so on; this happens T times with (H, A, O) as the final outcome. The loss s and gain g are computed on lines 4 and 5, respectively.

In the remainder of this chapter, when $(s, g) = \mathbb{E}\left(\text{scen}_{\gamma,\lambda}(\cdots)\right)$ we sometimes write $\text{loss}_{\gamma,\lambda}(\cdots)$ and $\text{gain}_{\gamma,\lambda}(\cdots)$ to refer to the expected loss s and the expected gain g, respectively. [1]

4.2 Metrics

The evaluation component of the model from the previous section produces expected values of loss and gain for given defender and adversary strategies. In this section we use these values to define metrics of the information flow of system executions, and we show how to compute these metrics. Our definitions generalizes several popular metrics from the literature which we demonstrate in Section 4.2.2.

4.2.1 Defining the metrics

The amount of information leaked by a system depends on actions performed by both adversary and defender. Our metrics are defined assuming that each party has independent interests, and that each has some knowledge about the other's actions.

In particular, we assume that the adversary knows that the defender draws a high-input strategy according to X_η. Hence an optimal adversary picks an action strategy $\alpha_{opt} \in \mathcal{S}_\alpha$ that would maximize their expected gain. The adversary's *expected gain* $\mathbb{G}$ is defined, then, as the expected gain when using an optimal strategy α_{opt}.

Given that the defender draws a high-input strategy from X_η, the *expected loss* $\mathbb{S}$ is defined as the expectation of the loss evaluation function, assuming the adversary's strategy is their optimal one, α_{opt}.

Figure 4.2 depicts an example of spaces of strategies, together with the gain and loss they induce. In both graphs, the x-axis represents defender's loss and the y-axis represents adversary gain.

The top graph depicts the set of possible adversary strategies. The maximum expected gain, labeled $\mathbb{G}$, is achieved by an optimal strategy α_{opt}, and this strategy induces an expected loss, labeled $\mathbb{S}$, for the defender. This loss, however, is not necessarily the maximum (or minimum) possible.

Worst- and best-case defender loss are depicted in the bottom graph. They are defined, respectively, as the maximum and minimum expected loss over the set $\mathcal{S}_\eta$ of high strategies when the adversary follows the optimal strategy α_{opt}.

Whereas $\mathbb{S}$ represents the expected loss over all defenders in $\mathcal{S}_\eta$, worst-case loss $\mathbb{S}^\wedge$ and best-case loss $\mathbb{S}^\vee$ are absolute bounds that hold for all defenders.

Before formally introducing metrics to capture the interplay between defender's loss and adversary's gain, we will introduce the following notation.

$\text{gain}_{\gamma,\lambda}(T, \eta{\leftarrow}X_\eta, \alpha) \stackrel{\text{def}}{=} \mathbb{E}_{\eta\leftarrow X_\eta}\left[\text{gain}_{\gamma,\lambda}(T, \eta, \alpha)\right]$, and

[1] Multivariate expectation $\mathbb{E}(x, y)$ is defined as $(\mathbb{E}(x), \mathbb{E}(y))$.

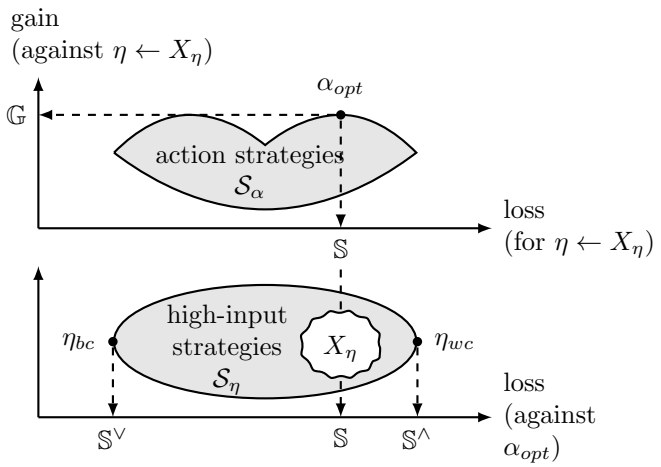


Figure 4.2: Metrics of risk among defender and adversary strategies
Optimal expected gain (above), and (worst case, best-case, expected) loss (below).

$$\text{loss}_{\gamma,\lambda}(T, \eta{\leftarrow}X_\eta, \alpha) \stackrel{\text{def}}{=} \mathbb{E}_{\eta \leftarrow X_\eta}\left[\text{loss}_{\gamma,\lambda}(T, \eta, \alpha)\right],$$

The expectation is taken over the probability $\Pr(X_\eta)$ and the various probabilities in the other model parameters.

Definition 11 (Metrics of information)**.**

1. An adversary's *(optimal) expected gain* against prior $\Pr(X_\eta)$ is the maximal expected gain over the set $\mathcal{S}_\alpha$ of adversary strategies:

$$\begin{aligned}\mathbb{G}^T_\gamma\left(X_\eta, \mathcal{S}_\alpha, \upsilon\right) &\stackrel{\text{def}}{=} \max_{\alpha \in \mathcal{S}_\alpha} \text{gain}_{\gamma,\lambda}(T, \eta \leftarrow X_\eta, \alpha) \\ &= \text{gain}_{\gamma,\lambda}(T, \eta \leftarrow X_\eta, \alpha_{opt})\end{aligned}$$

2. The defender's *expected loss* against the optimal adversary α_{opt} is defined as: [2]

$$\mathbb{S}^T_{\gamma,\lambda}\left(X_\eta, \mathcal{S}_\alpha, \upsilon\right) \stackrel{\text{def}}{=} \text{loss}_{\gamma,\lambda}(T, \eta \leftarrow X_\eta, \alpha_{opt})$$

3. The defender's *worst case* (*best case*) *loss* is the maximal (minimal) expected loss against the optimal adversary α_{opt}:

$$\begin{aligned}\mathbb{S}^{T\wedge}_{\gamma,\lambda}\left(X_\eta, \mathcal{S}_\eta, \mathcal{S}_\alpha, \upsilon\right) &\stackrel{\text{def}}{=} \max_{\eta \in \mathcal{S}_\eta} \text{loss}_{\gamma,\lambda}(T, \eta, \alpha_{opt}) \\ &= \text{loss}_{\gamma,\lambda}(T, \eta_{wc}, \alpha_{opt})\end{aligned}$$

[2] We assume that if there are multiple optimal adversary strategies, α_{opt} is one which maximizes defender loss among them (that is, uses loss as a tie-breaker in the optimization).

$$\mathbb{S}^{T\vee}_{\gamma,\lambda}(X_\eta, \mathcal{S}_\eta, \mathcal{S}_\alpha, \upsilon) \stackrel{\text{def}}{=} \min_{\eta \in \mathcal{S}_\eta} \text{loss}_{\gamma,\lambda}(T, \eta, \alpha_{opt})$$
$$= \text{loss}_{\gamma,\lambda}(T, \eta_{bc}, \alpha_{opt})$$

(So, η_{wc} and η_{bc} are the strategies that, respectively, maximize and minimize loss against an optimal adversary.)

4.2.2 Expressing existing metrics

The model for predicting risk described in Chapter 4 extends various models for quantifying information flow. Expressing more limited models is useful to ground the extensions in already familiar frames. We summarize in this section how to express vulnerability, g-vulnerability, and guessing entropy in our model with varying limits on adversary power.

Low-adaptivity The ability of an adversary to influence the system is modeled using low inputs. This ability can be restricted to model situations without adaptive adversaries by setting $\mathcal{L} = \emptyset$ or alternatively defining adversary strategies such that they ignore observations: $\alpha : (L, O) \mapsto f(L)$. In either way, the adversary lose any ability to adaptively influence the observation process.

Wait-adaptivity An import element of adversaries introduced in [66] is their ability to adaptively wait for the right time to attack (exploit). This is modeled by an adversary strategy producing an exploit instead of a low input. This power can be restricted by defining adversary strategies so that they only output an exploit at time T and no other time.

Dynamics Our model allows the secret to change over time based on the history up to that point. We can restrict the model by defining high-input strategies in a manner that lets them pick only the initial high value, but keeps it constant thereafter.

We will use the term *static model* to refer to scenarios in which the adversary and model is restricted as noted above. We then define existing metrics using a static variant of our model.

Vulnerability

The notion of vulnerability [100] corresponds to an equality gain function (i.e., a guess).

let gain_vul$:(H{\cdot}h, A{\cdot}e, O){:}\rightarrow$
  if $h = e$ **then** 1 **else** 0

The goal of the attacker assumed in vulnerability is evident from gain_vul; they are directly guessing the secret, and they only have one chance to do it.

Theorem 12. *In a static model, the vulnerability (written* $\mathbb{V}$*) of the secret conditioned on the observations is equivalent to dynamic gain using the* gain_vul *gain function.*

$$\mathbb{G}^{T}_{gain\_vul}(\cdots) = \mathbb{V}(X_h \mid X_O)$$

In the above we use X_h and X_O to designate the random variables representing the initial/last/only high value and the string of observations, respectively. We also omit the loss function as it does not influence gain.

g-vulnerability

Generalized gain [10] functions can be used to evaluate metrics in a more fine-grained manner, leading to a metric called g-vulnerability. This metric can also be expressed in terms of the static model. Let gainfunc be a generalized gain function, taking in the secret and an adversary exploit and returning a real number between 0 and 1, then we have:

```
let gain_gen_gain:(H·h, A·e, O)↦
  let g←gainfunc h e in g
```

The difference between expected gain and g-vulnerability are non-existent in the static model. The gain of a system corresponds exactly to g-vulnerability of gainfunc, written $\mathbb{V}_{\text{gainfunc}}(\cdots)$.

Theorem 13. *In a static model the g-vulnerability of* gainfunc *is equivalent to dynamic gain using* gain_gen_gain *gain function.*

$$\mathbb{G}^{T}_{gain\_gen\_gain}(\cdots) = \mathbb{V}_{gainfunc}(X_h \mid X_O)$$

Guessing-entropy

Guessing entropy [73], characterizing the expected number of guesses an optimal adversary will need in order to guess the secret, can also be expressed in terms of the static model. We let exploits be strings of highs (really permutations of all highs), $\mathcal{E} \stackrel{\text{def}}{=} \mathcal{H}^{|\mathcal{H}|}$. The exploit permutation corresponds to an order in which the secret is to be guessed. We then define expected gain to be proportional to how early in that string of guesses the secret appears.

```
let pos_of:(h, H)↦
  // compute the position of h in H //
  // assuming strings are 1−indexed //

let gain_guess_ent:(H·h, A·e, O)↦
  let pos←pos_of(h, e) in −pos
```

Note that we negate the gain as an adversary would optimize for the minimum number of guesses, not the maximum. Guessing entropy, written $\mathbb{G}$, is related to dynamic gain as follows.

Theorem 14. *In a static model, guessing entropy is equivalent to (the negation of) dynamic gain using the* gain_guess_ent *gain function.*

$$-\mathbb{G}^{T}_{gain\_guess\_ent}(\cdots) = \mathbb{G}(X_h \mid X_O)$$

4.2.3 Computing optimal adversary gain and associated defender loss

If the sets of possible strategies are large, enumerating them in order to calculate the given metrics could be too costly. Here we consider the following alternative approach.

We assume that the set of adversary strategies $\mathcal{S}_\alpha$ has a basis of deterministic strategies, which ensures that the search for optimal action strategies can be simplified. We then define a procedure P that predicts the adversary's expected gain and the associated defender's loss. This procedure builds an action strategy (simplified by our assumption) that always picks an action maximizing gain according to the adversary's current belief about the state of the system. This belief is constructed by the procedure infer which describes how the adversary updates their knowledge about the high values in an execution, given their view of the system (the public history of adversary actions and observables). This procedure produces a distribution over high values conditioned on the adversary's view. We describe each of these steps in more detail in the rest of this section.

Structure on $\mathcal{S}_\alpha$ We assume the set $\mathcal{S}_\alpha$ has a *basis* composed of deterministic strategies, meaning that whenever an action can be generated by a probabilistic strategy in $\mathcal{S}_\alpha$, there is also a deterministic strategy in $\mathcal{S}_\alpha$ that can generate the same action. We represent this basis in compact form as a *possibilistic strategy* $\alpha_S : \mathit{pubhistory} \to \mathcal{P}(\mathcal{A})$, which returns the subset of $\mathcal{A}$ consisting of exactly all actions allowed by some strategy in $\mathcal{S}_\alpha$. The set of all deterministic strategies $\mathrm{gen}(\alpha_S)$ generated by the possibilistic strategy α_S is given by

$$\begin{aligned}\mathrm{gen}(\alpha_S) \stackrel{\text{def}}{=} \{\alpha \in \mathit{pubhistory} \to \mathbb{D}(\mathcal{A}) :\ & \\ & \alpha \text{ is deterministic,} \\ & \alpha \text{ is consistent with } \alpha_S\},\end{aligned}$$

where we say strategy α is *consistent* with possibilistic strategy α_S iff $\mathrm{support}(\alpha(L,O)) \subseteq \alpha_S(L,O)$ for every L, O.

Assuming that $\mathrm{gen}(\alpha_S) \subseteq \mathcal{S}_\alpha$, we can greatly reduce the computational cost of finding an optimal strategy, as the best strategy among all of those in $\mathcal{S}_\alpha$ can be found using the possibilistic strategy as the underlying search space. [3]

[3] This is because in games in which an adversary is optimized against a fixed defender, as is the case here, there is always an optimal deterministic strategy [66], assuming it can choose from among the options available to the probabilistic strategies. Effectively, in general probabilistic strategies can be ignored as long as the space of deterministic ones is general enough (which is the case, for instance, when deterministic strategies are *corner points* for the space of all strategies).

Adversary inference By interacting with the system and observing the public history of an execution, the adversary learns about the defender's strategy and the high values. For a given context X_η and system υ, this learning process is encapsulated by the procedure infer defined below. Given the public history visible to an adversary, the procedure produces a distribution over the secret history conditioned on adversary's view (i.e., consistent with the adversary's observations). We will use this distribution in maximizing the expectation of gain.

$$\text{infer} : \mathit{pubhistory} \to \mathbb{D}\left(\mathcal{S}_\eta \times \mathcal{H}^*\right)$$

```
1 | let infer : (ϵ, ϵ) ↦
2 |   let η ← X_η in
3 |   let h ← η(ϵ, ϵ, ϵ) in
4 |     (η, h)
5 | let infer : (A·a, O·o) ↦
6 |   let (η, H) ← infer(A, O) in
7 |   condition υ(H, A·a, O) = o in
8 |   let h ← η(H, A·a, O·o) in
9 |     (η, H·h)
```

The second case (lines 5–9) determines the adversary's knowledge about $H{\cdot}h$ given a non-empty public history $A{\cdot}a$, $O{\cdot}o$. The adversary first determines an initial knowledge about η and H, which is defined recursively (line 6). This knowledge is then refined (via probabilistic conditioning) by taking into consideration that the system function must have returned o when evaluated on $H, A{\cdot}a, O$ (line 7). Finally, he predicts the next high value h as produced from the high strategy η (line 9).

Gain optimization We can now define P (for "prediction") as the function that, for a given context X_η and system υ, maps the adversary's view to the pair of their optimal expected gain and associated defender's expected loss.

$$P(A, O) \stackrel{\text{def}}{=} \max_{a \in \alpha_S(A,O)} \begin{cases} \mathbb{E}_{\substack{(\cdot,H) \leftarrow \text{infer}\,(A,O) \\ g \leftarrow \gamma(H, A\cdot a, O) \\ s \leftarrow \lambda(H, A\cdot a, O)}} \left[(g, s)\right] & (1) \\ \mathbb{E}_{\substack{(\eta,H) \leftarrow \text{infer}\,(A,O) \\ o \leftarrow \upsilon(H, A\cdot a, O)}} \left[P(A{\cdot}a, O{\cdot}o)\right] & (2) \end{cases}$$

where (1) requires $a \in \mathcal{E}$ or $t = T$, and (2) requires $a \in \mathcal{L}$ and $t < T$, given that $t \stackrel{\text{def}}{=}$ length of A, O. For purposes of maximization, we order pairs lexicographically, favoring g over s, i.e., $(g_1, s_1) < (g_2, s_2)$ whenever $g_1 < g_2$ or $g_1 = g_2$ and $s_1 < s_2$. This is a technicality required for the definition of defender loss when there are multiple optimal adversaries. This definition lets us construct an optimal "demonic" adversary who, when gain is equal, picks actions that maximize loss.

The definition of P recursively maximizes the expected gain over all actions available to the adversary, given the public history at each time step. Whenever an exploit is produced, or when the maximum time T is achieved, the adversary expected gain

is calculated using the result of the gain function evaluated over the public history and the adversary's belief about what the secret history might be. Alternatively, when $t < T$ and the action is a low input, their expected gain is calculated using the gain expected for the adversary's view at the following time step, which includes a prediction of what the next observable might be.

An optimal strategy α_{opt} is defined by replacing max with argmax in the definition of P above.

Proposition 15. *The recurrence P accurately describes the optimal expected adversary gain and the associated defender loss. Also, the strategy α_{opt} is a strategy that realizes both.*

$$\mathbb{G}^T_\gamma\left(X_\eta, \mathcal{S}_\alpha, \upsilon\right) = P_1(\epsilon,\epsilon) = \mathit{gain}_{\gamma,\lambda}(T, \eta \leftarrow X_\eta, \alpha_{opt})$$
$$\mathbb{S}^T_{\gamma,\lambda}\left(X_\eta, \mathcal{S}_\alpha, \upsilon\right) = P_2(\epsilon,\epsilon) = \mathit{loss}_{\gamma,\lambda}(T, \eta \leftarrow X_\eta, \alpha_{opt})$$

4.2.4 Computing worst- and best-case defender loss

The worst- and best-case defender loss can be determined by computing $\mathrm{loss}_{\gamma,\lambda}(T, \eta, \alpha_{opt})$ for every $\eta \in \mathcal{S}_\eta$. Once again, enumerating the set $\mathcal{S}_\eta$ of defender strategies may be too costly, so we assume additional structure on that set and describe how to find the defender strategies that realize the worst- and best loss.

More precisely, we consider that the defender's strategy can be split into two components: (i) a *controllable* one, which is deterministic and can be directly regulated by the defender, and (ii) an *imposed* one, which can be probabilistic and is beyond the defender's control. As an example, consider a defender who can decide when to reset a secret key (via a controlled component), but does not have control over the value it is reset to (it is picked by the imposed component). This expressiveness is particularly relevant for calculating our best- and worst-case metrics, since it allows us to rule out a defender who, knowing beforehand that the adversary is rational, would behave so to minimize the efficacy of the adversary's strategy (e.g., by always picking the secret key that would be guessed last by the rational adversary).

Formally, the imposed and controllable components of a defender strategy are separated as follows. Define $\bar{\mathcal{H}}$ to be a set of *high actions* (distinct from secret high values $\mathcal{H}$), modeling inputs that the controllable component can produce and feed to the imposed component. A *controllable* half is a strategy $\bar{\eta} : \mathit{history} \to \mathbb{D}\left(\bar{\mathcal{H}}\right)$ that produces a high action to be fed to a *imposed* half, which is a function $\eta_f : \mathit{history} \times \bar{\mathcal{H}} \to \mathbb{D}\left(\mathcal{H}\right)$ that takes the high action into account to actually produce a high value. A defender strategy, thus, is a composition $\bar{\eta} \circ \eta_f$:

$(\bar{\eta} \circ \eta_f)\!:\!(H, L, O) \mapsto$
  let $\bar{h} \leftarrow \bar{\eta}(H, L, O)$ **in**
    $\eta_f(H, L, O, \bar{h})$

Our enumeration of high actions is made akin to that of adversary actions in the optimization for gain. We will similarly define a *possibilistic high action strategy* $\bar{\eta}_{\mathcal{S}} : \mathcal{H}^* \times \mathcal{L}^* \times \mathcal{O}^* \to \mathcal{P}\left(\bar{\mathcal{H}}\right)$, which provides the set of high actions available at any point in a scenario. The search for the worst- and best-case defender loss can be thus restricted to strategies generated as follows:

$$\begin{aligned}\text{gen}(\bar{\eta}_{\mathcal{S}}, \eta_f) \stackrel{\text{def}}{=} \{\eta \in \mathit{history} \to \mathbb{D}\left(\mathcal{H}\right) : \\ \eta = \bar{\eta} \circ \eta_f, \\ \bar{\eta} \text{ is deterministic}, \\ \bar{\eta} \text{ is consistent with } \bar{\eta}_{\mathcal{S}}\}\end{aligned}$$

We thus assume that $\text{gen}(\bar{\eta}_{\mathcal{S}}, \eta_f) \subseteq \mathcal{S}_\eta$, and that any other strategy in $\mathcal{S}_\eta$ is a composition of some $\bar{\eta}$ (consistent with $\bar{\eta}_{\mathcal{S}}$) and η_f. From here we proceed similarly as in the optimization of adversary behavior. We define $S^\wedge$ and $S^\vee$ as mappings from the defender's view (the full history of the execution) to their expected worst- and best-case loss, respectively.

$$S^\wedge(H, A, O) \stackrel{\text{def}}{=}$$

$$\max_{\bar{h} \in \bar{\eta}_{\mathcal{S}}(H, A{\cdot}a, O)} \begin{cases} \mathbb{E}_{\substack{h \leftarrow \eta_f(H, A{\cdot}a, O, \bar{h}) \\ s \leftarrow \lambda(H{\cdot}h, A{\cdot}a, O)}}\left[s\right] & (1) \\ \mathbb{E}_{\substack{h \leftarrow \eta_f(H, A{\cdot}a, O, \bar{h}) \\ o \leftarrow \upsilon(H{\cdot}h, A{\cdot}a, O)}}\left[S^\wedge(H{\cdot}h, A{\cdot}a, O{\cdot}o)\right] & (2) \end{cases}$$

where (1) requires $a \in \mathcal{E}$ or $t = T$, and (2) requires $a \in \mathcal{L}$ and $t < T$, given that $a \stackrel{\text{def}}{=} \alpha_{opt}(A, O)$ and $t \stackrel{\text{def}}{=}$ length of H.

The definition of $S^\vee$ for computing the best-case is the same but replacing max with min and $S^\wedge$ with $S^\vee$ in the recursive case. Let η_{wc} and η_{bc} be strategies defined by replacing max and min above with argmax and argmin respectively, and composing with the imposed η_f as described above.

Proposition 16. *The recurrences $S^\wedge$ and $S^\vee$ accurately describe the worst- and best-case defender loss against the optimal adversary. Also, the strategies exhibiting the worst- and best-case loss are η_{wc} and η_{bc}, respectively.*

$$\begin{aligned}\mathbb{S}^{T\wedge}_{\gamma,\lambda}\left(X_\eta, \mathcal{S}_\eta, \mathcal{S}_\alpha, \upsilon\right) &= S^\wedge(\epsilon, \epsilon, \epsilon) \\ &= \mathit{loss}_{\gamma,\lambda}(\eta_{wc}, \alpha_{opt}) \\ \mathbb{S}^{T\vee}_{\gamma,\lambda}\left(X_\eta, \mathcal{S}_\eta, \mathcal{S}_\alpha, \upsilon\right) &= S^\vee(\epsilon, \epsilon, \epsilon) \\ &= \mathit{loss}_{\gamma,\lambda}(\eta_{bc}, \alpha_{opt})\end{aligned}$$

4.3 Experiments

In the following section we perform a sequence of experiments of various scenarios to demonstrate and explore several aspects of risk prediction using our model. First

we present several experiments that describe zero-sum settings, ones in which the loss to defender and gain to adversary are the same quantity. Later we demonstrate scenarios in which this is not the case.

We implemented our model using a simple monadic embedding of probabilistic computing [93] in OCaml, as per Kiselyov and Shan [54]. The recursive procedure P from Section 4.2 is computed using these elements to obtain both the optimal adversary gain and associated defender loss, as well as the strategy that achieves these. To compute the worst- and best-case defender loss we similarly implement the recursive procedures $S^{\wedge}$ and $S^{\vee}$ of Section 4.2.4. The implementation (and experiments from the next section) are available online [4]. Though a big portion of the implementation involves probabilistic programming and inference, there are significant roadblocks to applying the approximate approach for the tasks described in Chapter 6. The biggest issue is that the model in this Chapter involves iterative steps which optimize adversary choices based on the result of probabilistic inference. If an approximation does not provide a definitive optimal choice at step t, then multiple potentially optimal adversary actions would need to be considered and the work for optimizing step $t + 1$ would increase two-fold (or n-fold where n is the number of potentially optimal choices). Though the issue is not insurmountable, it could introduce enough complexity to discount any benefits of the approximation.

Given the limitations of our implementation, the experiments we present are simple and include only limited ranges of secrets, inputs, and observables. We cannot analyze, for example, the full space of passwords in an authentication scheme, addresses in an address space randomization approach, medical diagnoses in a medical information sharing system, or locations in a location-based service. Instead we focus on small examples with which we demonstrate our model and make observations which should carry-over to realistic scenarios. The setting of our experiments is a whimsical one so that the limited range of elements in these experiments does not need to be motivated as realistic.

Our experiments develop several examples on the theme of *stakeouts and raids*. Each example varies different parameters, including whether and how a secret changes, whether the adversary is low- and/or wait-adaptive, and whether there is are costs associated with observations. We describe the general scenario first and in the following experiments we will restrict various parts of the model in order to focus each experiment on a particular aspect.

Consider a pirate captain (the defender) who can choose to hide a stash of gold in one of a set of possible stash locations, modeled as an integer between 0 and 7. At every point in time the rival Captain (the adversary) may pick one of three actions: (i) stake out, at a cost, a location to learn whether the defender's stash is stored there; (ii) raid a location, obtaining a gain of 1 if the stash is found there, and 0 otherwise; or (iii) stall and not stake out or raid at all (at no gain or cost). Concomitantly, every f-th time step (for f to be specified by experiment), the defender can move the stash to a new random location. In the non-zero-sum section later on we will define the

[4] https://github.com/plum-umd/qif

loss to the defender based on whether or not the stash stays hidden, regardless of the cost incurred by the adversary in finding it.

This example can be formally captured by our model as follows. (Whenever we use the variable t in the pseudo-code, it will refer to the length of the high portion of the history.)

- Sets of possible high inputs, low inputs, observations, and exploits are encoded as:

$$\begin{aligned}\mathcal{H} &= \{0, \cdots, 7\} \\ \mathcal{L} &= \{\text{Stakeout}_i\}_{i=0}^{7} \cup \{\text{No-stakeout}\} \\ \mathcal{O} &= \{\text{Found}, \text{Not-found}\} \\ \mathcal{E} &= \{\text{Raid}_i\}_{i=0}^{7} \cup \{\text{No-raid}\}\end{aligned}$$

- The high-input strategy depends on the parameter f that determines the frequency of changing the stash location and is modeled by the function move_f:

```
let move_f:(H·h, A, O)↦
  if t mod f = 0
    then uniform H
    else h
```

  Hence $\mathcal{S}_\eta = \{\text{move}_f\}$. (We are not yet reasoning about worst- and best-case loss, so we do not need to decompose the defender strategy into controllable and imposed components as per Section 4.2.4.)

- The adversary is assumed to know the defender's strategy. The adversary's prior knowledge X_η is equal to X_{move_f}.

  Later in this section we will present an experiment in which the adversary does not exactly know the defender's strategy.

- The possibilistic basis α_S for adversary strategies will vary in the first few experiments. The most restricted will be an adversary that is non-adaptive and cycles through low-inputs via some ordering of the 8 possible stash locations. Given an ordering $\text{ord} : \mathcal{L}^*$, the low input at time t will be to stakeout location $\text{ord}_{(t \mod 8)}$. This adversary will only exploit his knowledge at the final time T. We will see in the second experiment that the order does not matter much and we will use the ordering $\text{ord} = \text{id} \stackrel{\text{def}}{=} 0{\cdot}1 \cdots 7$ for many of the experiments where non-low-adaptive adversaries are modeled.

```
let act_nonadapt_ord:(A, O)↦
  if t = T then E
  else {Stakeout_ord(t mod 8)}
```

  A low adaptive adversary will pick from among all low inputs up to time T at which point they will pick an exploit:

```
let act_lowadapt:(A, O)↦
  if t = T then ℰ
  else ℒ
```

A wait-adaptive, but not low-adaptive adversary will pick low inputs according to a fixed ordering but also have the option of exploiting at any time:

```
let act_waitadapt_ord:(A, O)↦
  {Stakeout_ord(t mod 8)} ∪ ℰ
```

In the most general experiments, the adversary will be fully adaptive as defined by act_all :

```
let act_all :(A, O)↦ℒ ∪ ℰ
```

- The system function, through which the adversary makes observations, is $\upsilon =$ stakeout_maybe, and it determines whether the adversary staked out the correct location or not:

```
let stakeout_maybe:(H·h, A·No-stakeout, O)↦
  Not-found
let stakeout_maybe:(H·h, A·Stakeout_ℓ, O)↦
  if h = ℓ then Found else Not-found
```

- The adversary gain, $\gamma =$ raid_maybe$_c$, computes the cumulative cost incurred for observations made by the adversary. If the adversary decides to raid, the function adds to the cost a value of either 1 or 0 depending on whether the raid was successful or not successful:

```
let raid_maybe_c:(H·h, A·No-raid, O)↦
  −c · count_stakeouts(A)
          (* count how many adversary actions were stakeouts *)
let raid_maybe_c:(H·h, A·Raid_ℓ, O)↦
  let raid_gain ← (if h = ℓ then 1 else 0) in
  let stakeouts ← count_stakeouts(A) in
    raid_gain − c · stakeouts
```

The process is parameterized by the value c of the cost the adversary incurs for each observation.

- In the first few experiments, we will only be concerned with the adversary gain. Later we will separately define the defender loss, $\lambda =$ raid_success, that indicates only the success of the raid if it occurred, ignoring the costs incurred by the adversary.

```
let raid_success :(H·h, A·no-raid, O)↦0
let raid_success :(H·h, A·raid_ℓ, O)↦
  if h = ℓ then 1 else 0
```

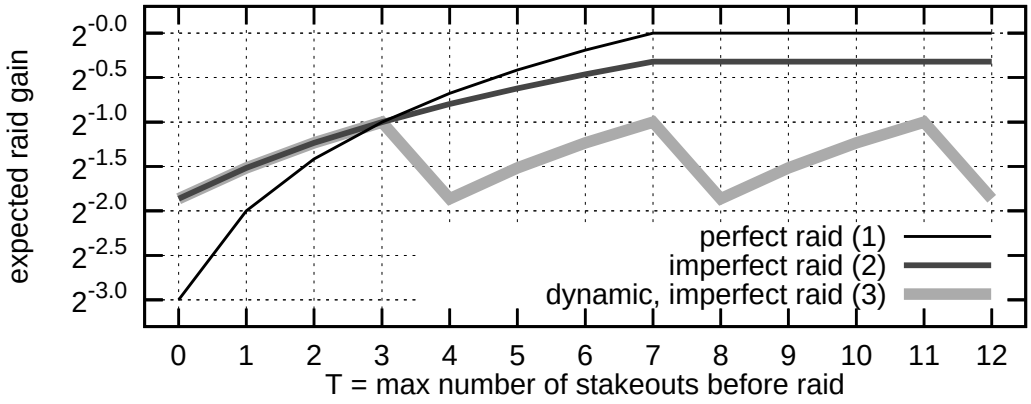


Figure 4.3: Gain in experiment of Section 4.3.1

Expected raid gain over time given stakeouts stakeout_maybe with (1) static stash move_∞ and perfect raids raid_maybe_0, (2) static stash and imperfect raids raid_imperfect, and (3) moving stash move_4 and imperfect raids, all with non-adaptive adversaries act_nonadapt_{id}.

4.3.1 How does gain differ for dynamic secrets, rather than static secrets?

Our first experiment considers the impact on information leakage due to a dynamic secret (using high-input strategy move_f). The adversary here will be non-adaptive and for comparison we also consider a high-input strategy that does not change the secret by setting f to infinity.

In this experiment the stakeout cost to the adversary, c, will be 0. We also consider a variation of a raid that has a chance to fail even if the raid location is correct, and has a chance to accidentally discover a "new" stash even if the raid takes place at the wrong location. This can be modeled by the following gain function:

```
let raid_imperfect : (H·h, A·Raid_ℓ, O) ↦
  let success ← (if h = ℓ then flip 0.8 else flip 0.2) in
  if success then 1.0 else 0.0
```

Figure 4.3 plots how the gain differs when we have a static secret with perfect raids, a static secret with imperfect raids, and a dynamic secret with imperfect raids. The static portion (1) with perfect raid is an example of an analysis achievable by a parallel composition of channels and the vulnerability metric [42]. Adding the imperfect gain function (2) alters the shape of vulnerability over time though in a manner that is *not a mere scaling* of the perfect raid case. The small chance of a successful raid at the wrong location results in higher gains (compared to perfect raid) when knowledge is low. With more knowledge, the perfect raid results in more gain than the imperfect. Adding a dynamically changing stash (3) results in a periodic, non-monotonic, gain; though gain increases in the period of unchanging secret, it falls right after the secret changes. This, in effect, is a recovery of uncertainty, which is

thus not a non-renewable resource [42]. In the following sections we will refer to the period of time in which the secret does not change as an *epoch*.

4.3.2 How does low adaptivity impact gain?

To demonstrate the power of low adaptivity we will use a system function that outputs whether the stash is east or west of the stakeout location. Assuming the stash locations are ordered longitudinally, this function is just a comparison between the stash and stakeout location. The non-adaptive adversary will pick the stakeouts in a fixed order according to act_nonadapt_{id} (we will consider all permutations of the set of locations as possible orders), whereas the low-adaptive adversary will use act according to act_lowadapt.

let stakeout_eastwest$:(H{\cdot}h, A{\cdot}\text{Stakeout}_\ell, O)\mapsto$
  if $h \leq \ell$ **then** Found **else** Not-found

Figure 4.4 demonstrates the expected gain of both types of attackers. For fixed-order (non-adaptive) adversaries, we used all 8! permutations of the 8 stash locations as possible orders and plotted them all as the wide light gray lines in the figure. Though there are many possible orders, the only thing that makes any difference in the gain over time is the position of 7 (the highest stash location) in the ordering as the system function for this input reveals no information whatsoever. All other stakeout locations reveal an equal amount of information in terms of the expected gain. To demonstrate this behavior, we have specifically plotted in the figure the gain for an ordering in which location 7 is staked-out at time 3 (labeled "a fixed strategy"). Gain increases linearly with every non-useless observation. On the other hand, the low-adaptive adversary performs binary search, increasing his gain exponentially.

4.3.3 How does wait adaptivity impact gain?

An adversary that can wait is allowed to attack at any time. Adaptive wait has a significant impact on the gain an adversary might expect. In the simple stakeout/raid example of Figure 4.5, it transforms an ever-bounded vulnerability to one that steadily increases in time.

Roughly, the optimal behavior for a wait-adaptive adversary is to wait until a successful stakeout before attacking. The more observations there are, the higher the chance this will occur. This results in the monotonic trend in gain over time.

There are subtle decisions the adversary makes in order to determine whether to wait and allow the secret to change. For example, in Figure 4.5, if the adversary has to attack at time 5 or earlier and has not yet observed a successful stakeout by time 3, they will wait until time 5 to attack, letting all their accumulated knowledge be invalidated by the change that occurs at time 4. This seems counter-intuitive as their odds of a successful raid at time 3 is 1/5 (they eliminated 3 stash locations from consideration), whereas they will accumulate only 1 relevant stakeout observation by time 5. Having 3 observations seems preferable to 1. The optimal adversary will wait because the *expected* gain at time 5 is actually better than 1/5; there is 1/8 chance that the stakeout at time 5 will pinpoint the stash, resulting in gain of 1, and 7/8

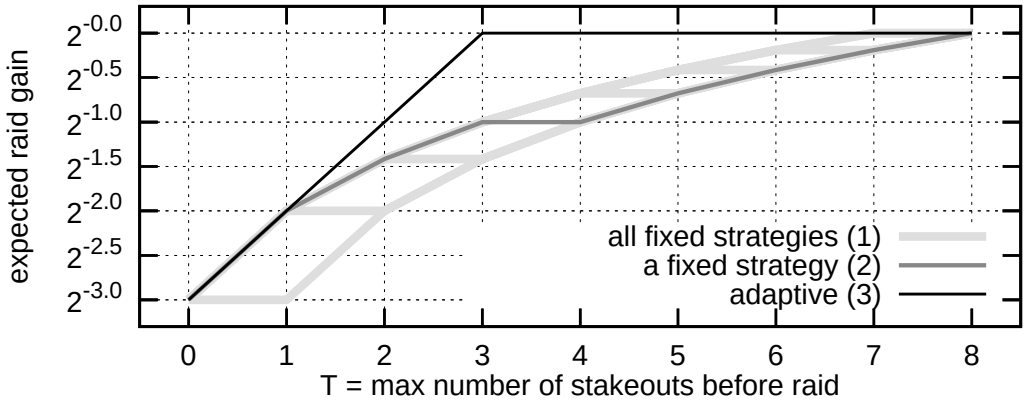


Figure 4.4: Gain in experiment of Section 4.3.2

Expected raid raid_maybe_0 gain over time with static stash move_∞ given stakeouts stakeout_eastwest with (1) all possible non-adaptive ($\text{act\_nonadapt}_{ord}$) orderings ord, (2) one possible non-adaptive ordering (ord = 0·1·2·7·3·4·5·6), and (3) adaptive (act_lowadapt) stakeout locations.

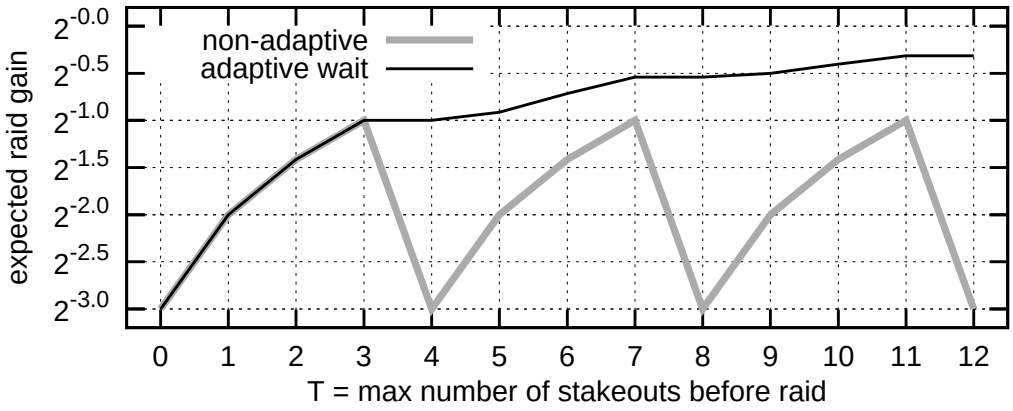


Figure 4.5: Gain in experiment of Section 4.3.3

Expected raid_maybe_0 gain with moving stash move_4 given stakeouts stakeout_maybe with (1) non-adaptive adversary (act_nonadapt_{id}) and (2) wait-adaptive adversary ($\text{act\_waitadapt}_{id}$).

chance it will not, resulting in expected gain of 1/7. The expectation of gain is thus $1/8 \cdot 1 + 7/8 \cdot 1/7 = 1/4 > 1/5$. The adversary thus has better expected gain if they have to attack by time 5 as compared to having to attack by time 3 or 4, despite having to raid with only one observation's worth of knowledge. One can modify the parameters of this experiment so that this does not occur, forcing the adversary to attack before the secret changes. This results in a longer period of constant vulnerability after each stash movement.

This ability to wait is the antithesis of moving-target vulnerability. Though a secret that is changing with time serves to keep the vulnerability at any fixed point low, the vulnerability *for some point* can only increase with time. The defender of our running example would be foolhardy to believe that he is safe from a raid just because it is unlikely to happen on Wednesday, or any other fixed day; if stakeouts continue and the adversary is smart enough to not schedule raid before having performed the stakeouts, the defender will be had. On the other hand, if there is a high enough cost associated with making an observation, the vulnerability against raids can be very effectively bounded (as we show in the next experiment).

In fact, the monotonically increasing vulnerability is a property of any scenario where the adversary can decide when to attack.

Theorem 17. *Gain is monotonically increasing with time:*

$$\mathbb{G}_{\gamma}^{t+1}\left(X_{\eta}, \mathcal{S}_{\alpha}, \upsilon\right) \geq \mathbb{G}_{\gamma}^{t}\left(X_{\eta}, \mathcal{S}_{\alpha}, \upsilon\right)$$

Proof. (Sketch) The theorem holds as an adversary attacking a system with max iteration $T = t + 1$ will have a chance to attack at time t, using the same exact gain function that the adversary would attack were $T = t$, and using the same inputs. Note that the gain functions we defined to model non-wait-adaptive adversaries were actually parameterized by max iteration T, and hence were actually different gain functions (see $\text{act\_nonadapt}_{ord}$ for example). Naturally in the $T = t + 1$ case, the attacker can also wait if the expectation of gain due to waiting one more time step is higher. □

4.3.4 Can gain be bounded by costly observations?

In this experiment we set the cost associated with making stakeouts to a non-zero value. Thus the adversary here has an actual decision to make in regard to making observations.

The results of this scenario are summarized in Figure 4.6 for stakeout costs ranging from 0.00 to 0.12 and in Figure 4.7 for higher costs in the range 0.1400 to 0.1430. In the top half of the first figure, the adversary does not have a choice of when to attack and the optimal behavior is to only perform stakeouts in the epoch that the raid will happen, resulting in the periodic behavior seen in the figure. Higher costs scale down the expected gain.

For wait-adaptive adversaries the result is more interesting. For sufficiently small stakeout costs, the adversary will keep performing stakeouts at all times except for the time period right before the change in the stash location and attack only when the stash is pinpointed. This results in the temporary plateau every 4 steps seen in the bottom half of Figure 4.6. This behavior seems counter-intuitive as one would think the stakeout costs will eventually make observations prohibitively expensive. Every epoch, however, is identical in this scenario, the stash location is uniform in $0, ..., 7$ at the start, and the adversary has 4 observations before it gets reset. If the optimal behavior of the adversary in the first epoch is to stakeout (at most 3 times), then it

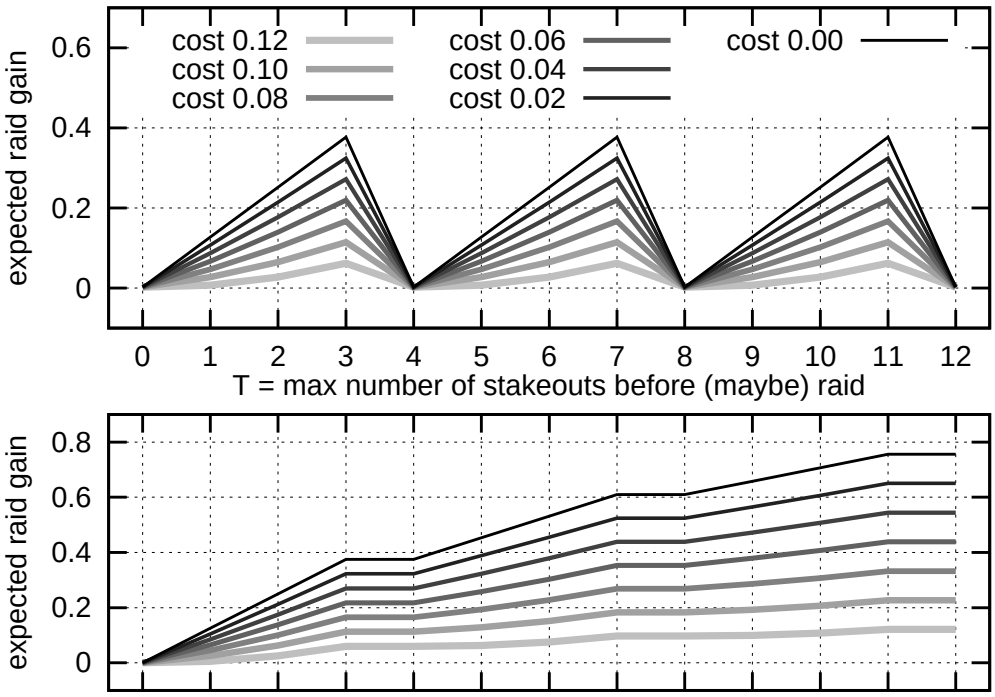


Figure 4.6: Gain in experiment of Section 4.3.4
Expected raid_maybe_c gain with costly stakeouts stakeout_maybe and moving stash move_4 with (top) non-wait-adaptive adversaries (act_nonadapt_{id}) and (bottom) wait-adaptive adversaries ($\text{act\_waitadapt}_{id}$).

is also optimal for them to do it during the second (if they have not yet pinpointed the stash). It is still optimal on the $(n+1)^{th}$ epoch after failures in the first n. As it is, the expectation of gain due to 3 stakeouts is higher than no guesses in any epoch, despite the costs.

The optimal behavior is slightly different for high enough stakeout costs but the overall pattern remains the same. Figure 4.7 shows the result of an adversary staking out in an epoch only if he is allowed enough time to attack at the end of the epoch. That is, for T equal to 1 or 2, the adversary would not stakeout at all, but if T is 3, he will stakeout at times 1,2, and 3. This is because the expected gain given 1 or 2 stakeouts is lower than 0, but for 3 stakeouts, it is greater than 0. As was the case with the lower cost, since the expected gain of observing in an epoch (3 times) is greater than not, the optimal adversary will continue to observe indefinitely if he is given the time.

Note however, that observing indefinitely in these examples does not mean that the expected gain approaches 1. Analytical analysis [5] of this scenario tells us that

[5]The quantity is the solution to the recurrence g_n which computes the expected gain after n full epochs, where $g_0 \stackrel{\text{def}}{=} 0$ and $g_n \stackrel{\text{def}}{=} \frac{1}{8}(1-1\cdot c)+\frac{1}{8}(1-2\cdot c)+\frac{1}{8}(1-3\cdot c)+\frac{5}{8}(g_{n-1}-3\cdot c)$.

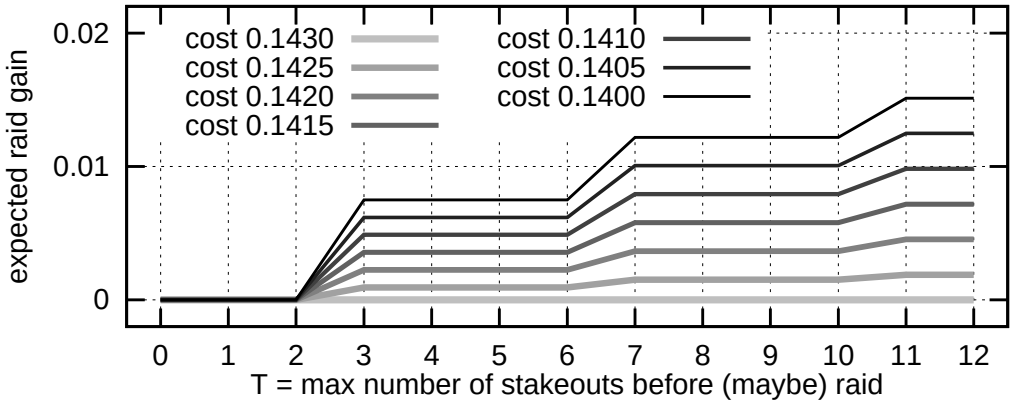


Figure 4.7: Gain detail in experiment of Section 4.3.4
Zoomed-in view of the same setting as Figure 4.6 (top).

after n full epochs, the expected gain is $(1-7c)\left(1-\left(\frac{5}{8}\right)^n\right)$ and in the limit, this quantity approaches $1-7c$. The adversary's gain is thus bounded by $1-7c$ for varying stakeout costs c (the point at which it is optimal to not observe at all is $c = 1/7 \approx 0.1429$).

The fact that the adversary's gain can be bounded arbitrarily close to 0, despite their strategy of performing stakeouts indefinitely, highlights the need for separating the notion of defender loss from that of adversary gain. We revisit this example using a non-zero-sum loss/gain functions in Section 4.3.6.

4.3.5 Does more frequent change necessarily imply less gain?

In this example we demonstrate a counter-intuitive fact that more change is not always better. So far we have used a high-input strategy that is known to the attacker but here he will only know the distribution over a set of possible strategies. Furthermore, guessing the full secret will *require* knowledge of the high-input strategy and therefore will require change to occur sufficiently many times. To construct such an example we will make part of the secret be a permutation which itself governs how (a second part of) the secret changes over time. We will restrict observations to only reveal (that second part of) the secret. Thus in this example the adversary will have to make observations about that second part over several applications of the permutation in order the learn the unobservable part. Delaying application of the permutation would thus delay the adversary learning it.

The setup for this example differs somewhat from the running stakeouts example. Let nbuilds be the number of buildings (in the example figure, nbuilds will be 5), and nfloors = factorial (nbuilds−1) be the number of floors in each of these buildings. The (nbuilds) buildings are in a city where stashes of gold tend to be found. Each building has nfloors floors and each floor of each building is claimed by a pirate gang. A single gang has the same-numbered floor in all the buildings. That is,

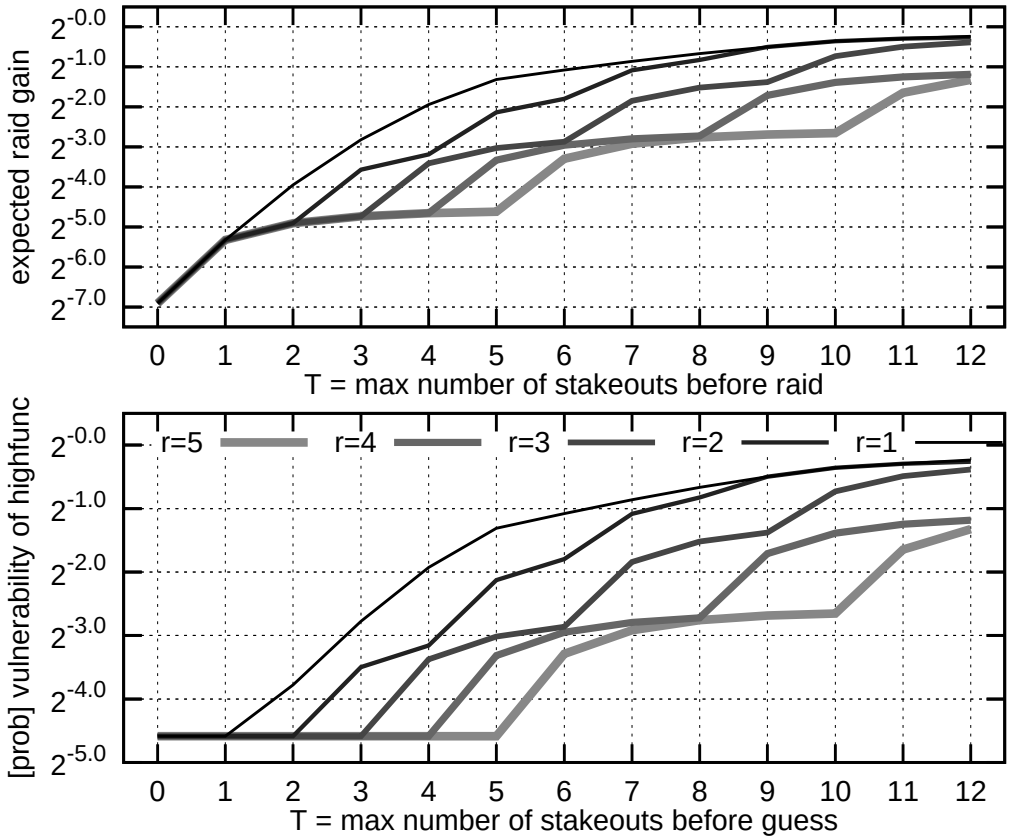


Figure 4.8: Gain and strategy vulnerability in experiment of Section 4.3.5 Changing stash according to $X_{\text{prior\_gangs}\ ()}$ with stakeouts stakeout_building quantifying (top) expected raid raid_apartment gain and (bottom) expected vulnerability of defender strategy η, all with wait-adaptive adversaries (act_all) and but with no low choices.

gang 0 will have floor 0 claimed in every building, gang 1 will have floor 1 in every building, and so on. Concretely, the set of high values in the experiment will be $\mathcal{H} = \{0, \cdots, 4\} \times \{0, \cdots, 23\}$, where the first dimension is the building number and the second is the floor/gang number.

The adversary knows there is a stash hidden on some floor in some building and that every gang moves their stash once in a while from one building to another in a predictable pattern (the floor does not change). They know all nbuilds gangs each have a unique permutation π of $0, ..., \text{nbuilds}-1$ as their stash movement pattern. The adversary knows which floors belong to which gangs (and their permutation). Given r as the stash movement rate parameter, the movement of the stash is governed by one of the following functions, parameterized by gang/floor number gng and that gang's movement permutation π:

let gang_move$_{gng,\pi}$ $:(\epsilon, A, O) \mapsto$

```
  (uniform {0,···,nbuilds−1}, gng)
let gang_move_{gng,π}:(H·(bldng, flr), A, O)↦
  if t mod r = 0
    then (π (bldng), flr)
    else (bldng, flr )
```

The prior belief that the adversary attributes to these strategies is $X_\eta = X_{\text{prior\_gangs}}\ ()$ defined as follows:

```
prior_gangs: () ↦
  let gng ← uniform {0, ···, nfloors−1} in
  let π ← (gen_all_permutations nbuilds)_{gng} in
    gang_move_{gng,π}
```

The function first picks a random gang and generates the permutation that represents that gang's stash movement pattern. It then creates a high-input strategy as defined by $\text{gang\_move}_{gng,\pi}$ that will perform that movement, keeping the gang/floor the same, while moving from building to building (the function picks a random building at time 0).

The adversary sets up stakeouts to observe all the buildings but is only successful at detecting activity half the time, and he cannot tell on which floor the stash activity takes place, just which building. Thus we have $\mathcal{L} = \{\text{Stakeout}\}$ and $\mathcal{O} = \{\text{No-activity}\} \cup \{\text{Activity}_i\}_{i \in \{0,\cdots,\text{nbuilds}\}}$.

```
let stakeout_building: (H·(bldg, flr), L, O) ↦
  if flip 0.5
    then Activity_{bldg}
    else No−Activity
```

The adversary wants to raid the stash but cannot raid the whole building, he needs to know the floor too. Exploits are thus $\mathcal{E} = \mathcal{H}$ and the gain function is as follows:

```
raid_apartment:(H·(bldg, flr), A·(bldg', flr'), O)↦
  if bldg = bldg' and flr = flr' then 1.0 else 0.0
```

Now, the chances of a successful raid after a varying number of stakeouts depends on the stash change rate r. Unintuitively, frequent stash changes lead the adversary to the stash more quickly. Figure 4.8(top) shows the gain in the raid after various number of stakeouts, for four different stash change rates (nbuilds = 5). The chances of a failed observation in this example are not important and are used to demonstrate the trend in gain over a longer period of time. Without this randomness, the gains quickly reach 1.0 after r·(nbuilds−1) observations (exactly enough to learn the initially unknown permutation).

The example has a property that the change function (the permutation π of the gang) needs to be learned in order to determine the floor of the stash accurately. Observing infinitely many stakeouts of the same building would not improve the adversary's chance beyond 1 in nfloors; learning the high-input strategy here absolutely requires learning how it changes the secret. One can see the expected progress in learning the high-input strategy in Figure 4.8(bottom), note the clear association between knowing the strategy and knowing the secret. We theorize that this correlation is a necessary part of examples that have the undesirable property that more change leads to more vulnerability.

4.3.6 How does a non-zero-sum utility/gain impact information flow as compared to zero-sum?

Here we analyze the experiment of Section 4.3.4 but now making a distinction between the gain of the adversary and the loss to the defender. We use the separate loss function raid_success for this purpose.

Figure 4.9 demonstrates how adversary gain and defender utility behave in this stakeouts example for varying lengths of time (T). Each line considers a different cost value c: dotted lines show adversary gain, whereas solid ones show defender loss. Lines incurring the same cost have the same color, for easier comparison. Notice that gain very much depends on the cost of observations. With cost 0.00 the gain is highest, and it steadily decreases as cost increases. At 0.15 it is no longer advantageous for the adversary to observe at all, so the gain flattens at $1/8$ (their initial gain without any observations). On the other hand, defender loss is not as dependent on the observation cost: it is essentially the same for costs 0.00, 0.05, and 0.10, and it follows in magnitude the gain an adversary would receive were cost equal to 0.00. When the cost is high enough (0.15) the loss drops abruptly to 1/8, since in this case the adversary would opt to never observe.

We have already noted in Section 4.3.4 that adversary gain and the resulting defender loss can be bounded at 0 as long as the cost for observations is high enough (at least $1/7$). For intermediate values of cost, the gain and loss are not identical. There, the gain can be bounded in the limit by $1 - c \cdot 1/7$ but *even given this bound, the loss for the defender approaches* 1.

4.3.7 How can a defender be prevented from a catastrophic worst-case behavior?

This experiment shows how to use worst- and best-case defender loss as a means of quantifying secret vulnerability in the face of, respectively, a catastrophic defender and a particularly lucky one. It shows, on one hand, that a scenario can be sufficiently restricted that even the worst-case defender maintains security for some period of time, and on the other hand, that even the most lucky defender will succumb eventually under those same restrictions.

Best- and worst-case defender loss are extra tools in analyzing the information flow of a system. Though the adversary is expecting to act optimally against a population of defender strategies X_η, here we will look at the range of the resulting loss over three, ever larger, sets of defender strategies $X_\eta \subset \mathcal{S}_{\eta 1} \subset \mathcal{S}_{\eta 2} \subset \mathcal{S}_{\eta 3}$. In effect this provides some guarantees that no matter what strategy is chosen by the defender, their loss will be bounded above and below by the worst case and best case loss. The worst-case is perhaps more relevant as it provides a means of quantifying security even in the face of catastrophic defender behavior (like users picking "password" as their password).

To illustrate our point, consider a slight modification to the previous example. Assume that now the adversary expects to be attacking a defender who may or may not choose to move the secret stash every 4th time step. To model this variation

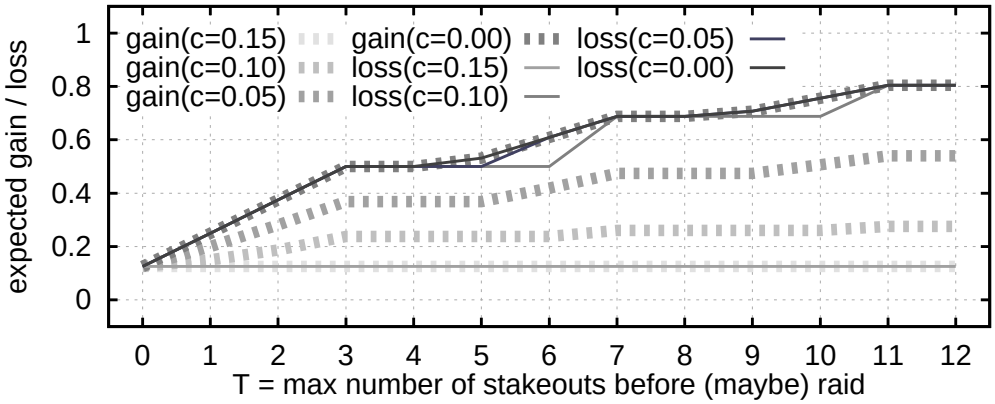


Figure 4.9: Gain and loss in experiment of Section 4.3.6
Expected raid_maybe_c gain (dotted lines) and raid_success loss (solid lines) with costly stakeouts stakeout_maybe and moving stash move_4.

we decompose the defender strategies into controllable and imposed components, as described in Section 4.2.4.

Recall that the controllable component produces a high action that the imposed component uses as input when deciding on what high value to generate. Here we define the high actions as follows:

$$\bar{\mathcal{H}} = \{\text{Move}, \text{No-move}\} \cup \{\text{Move-to}_i\}_{i=0}^{7}$$

In short, the defender can choose to move the stash (but cannot determine where), leave it in place, or move it to a particular location. Assume the adversary thinks the defender's controllable strategy $\bar{\eta}$ considers moving the stash every 4 time steps, but will only do so half the time.

let decide_move_maybe:$(H, A, O) \mapsto$
  let $c \leftarrow$ **flip** 0.5 **in**
    if $t \mod 4 = 0$ **and** c **then** Move
    else No-move

The imposed component of the defender strategy, η_f, controls the actual movement, and is defined as follows:

let move_act:$(H, A, O, \text{Move}) \mapsto$**uniform** $\mathcal{H}$
let move_act:$(H \cdot h, A, O, \text{Move-to}_\ell) \mapsto \ell$
let move_act:$(H \cdot h, A, O, \text{No-move}) \mapsto$h

let move_maybe $\stackrel{\text{def}}{=}$
  (move_act) $\circ$ (decide_move_maybe)

Though the adversary believes he is interacting with this kind of a defender ($X_\eta = X_{\text{move\_maybe}}$), we will also analyze defender loss with respect to three wider sets of

defender strategies, $\mathcal{S}_{\eta 1} \subset \mathcal{S}_{\eta 2} \subset \mathcal{S}_{\eta 3}$. We define the basis functions for these strategies as follows.

For the first, the defender has the control to either change the stash every 4 time steps or not but cannot decide where the stash gets changed to.

```
let η̄_{S_1}:(H, A, O)↦
  if t mod 4 = 0
    then {Move, No-move} else {No-move}
```

The second, wider set of defender strategies, also includes ones in which the defender can move every 4 time steps to anywhere they want but cannot decide where the initial stash is placed:

```
let η̄_{S_2}:(H, A, O)↦
  if t = 0 then {Move}
  else if t mod 4 = 0
    then {Move-to_i}_{i=0}^{7} else {No-move}
```

The third, most permissive one, is the set of strategies like the above but also giving the defender control over the initial stash location:

```
let η̄_{S_3}:(H, A, O)↦
  if t mod 4 = 0
    then {Move-to_i}_{i=0}^{7} else {No-move}
```

The three sets of strategies are generated from compositions of move_act and one of $\bar{\eta}_1$, $\bar{\eta}_2$, $\bar{\eta}_3$, with the extra addition of the sole strategy the adversary believes is employed, move_maybe.

$$\mathcal{S}_{\eta 1} \stackrel{\text{def}}{=} \text{gen}(\bar{\eta}_1, \text{move\_act}) \cup \{\text{move\_maybe}\}$$
$$\mathcal{S}_{\eta 2} \stackrel{\text{def}}{=} \text{gen}(\bar{\eta}_2, \text{move\_act}) \cup \{\text{move\_maybe}\}$$
$$\mathcal{S}_{\eta 3} \stackrel{\text{def}}{=} \text{gen}(\bar{\eta}_3, \text{move\_act}) \cup \{\text{move\_maybe}\}$$

Notice that $X_\eta \subset \mathcal{S}_{\eta 1} \subset \mathcal{S}_{\eta 2} \subset \mathcal{S}_{\eta 3}$.

Given the increasing range of defender strategies, the worst- and best-case will likewise increase in the range between them. The results of this experiment can be seen in Figure 4.10. The line shows the expected loss for the defender drawn from the prior X_η, whereas the shaded regions show the range between worst- and best-case defender loss for varying spaces of defender strategies; lighter regions denote more permissive spaces.

The most permissive set of defender strategies allows the defender to set the initial stash location to the one raided by the adversary at time 0 (before making any observations). This results in the worst-case loss to immediately shoot up to its maximum. On the other hand, the defender can also pick a stash location that the optimal adversary will never check and never raid, keeping the best-case loss at 0.

The less permissive set of strategies prevents the defender from picking the initial stash location. This makes worst- and best-case loss coincide with expected loss until they get a chance to make a best or worst choice at time 4. Then the worst-case shoots up to 1 for the same reason as in the previous case, and the best-case stays

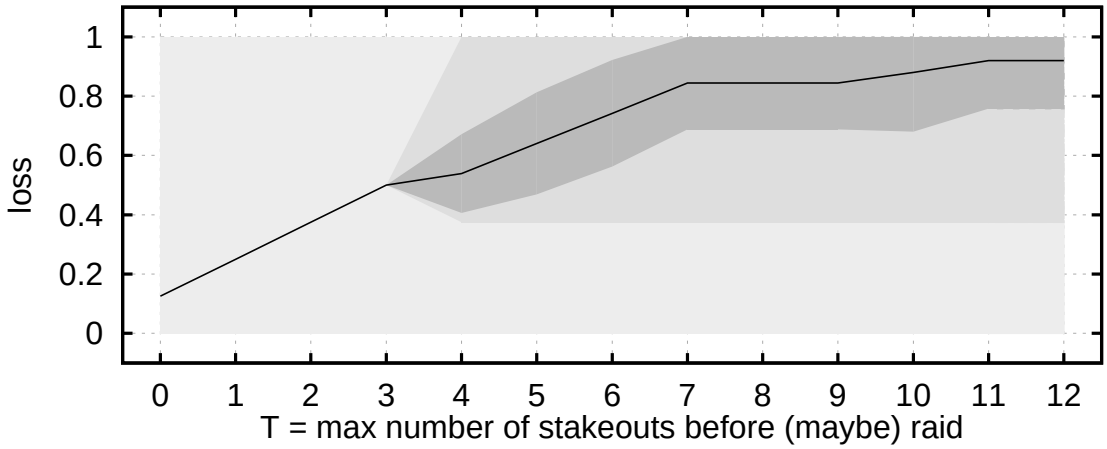


Figure 4.10: Loss in experiment of Section 4.3.7
Expected, worst-, and best-case raid_success loss against adversary with costly (c = 0.05) stakeouts stakeout_maybe and raid raid_maybe$_c$, with adversary expecting moving stash according to $X_{\text{move\_maybe}}$. Expected loss is solid line whereas the range between worst- and best-case loss are shaded; they are shaded lighter for more permissive defender behaviors.

near the level it has reached up to that point; the adversary will never get to observe the correct stash location from here on and can only succeed by a lucky raid, as opposed to having some chance to observe the stash while interacting with expected defender (thus the slight drop in defender loss compared to expected loss).

Finally the most restricted set of strategies only allows the defender to choose to move or not move, but not where to move to. In that situation the worst-case is to not ever move, and the best-case is to move every 4 time steps. This results in the worst-case loss increasing faster than the expected and the best-case loss increasing slower than the expected. The expected itself has a 50/50 chance of moving and is approximately between the worst- and best-case loss.

4.3.8 How can sound over-estimation of adversary gain impact defender loss?

Our final experiment demonstrates the importance of carefully making sure adversary goals are accurately defined if loss is to be correctly quantified. In particular, it is not safe to over-approximate the adversary's gain because doing so may lead to an unsound calculation (i.e., an under-approximation) of loss.

Consider a variation of our introductory example in which the adversary does not need to raid the stash's exact location. Instead of being all-or-nothing, the adversary receives gain proportional to how near the raid is to the stash (assuming stash

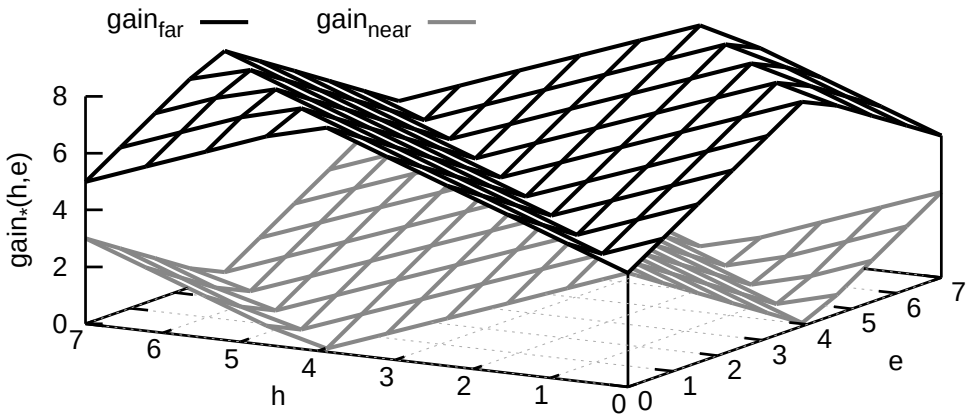


Figure 4.11: Gain functions $\text{gain}_{\text{near}}$ and gain_{far}

Two gain functions: $\text{gain}_{\text{near}}$ is inversely proportional to distance between guess and secret, and an over-approximation, gain_{far}, is proportional to distance but offset so that $\text{gain}_{\text{far}} \geq \text{gain}_{\text{near}}$ on all inputs.

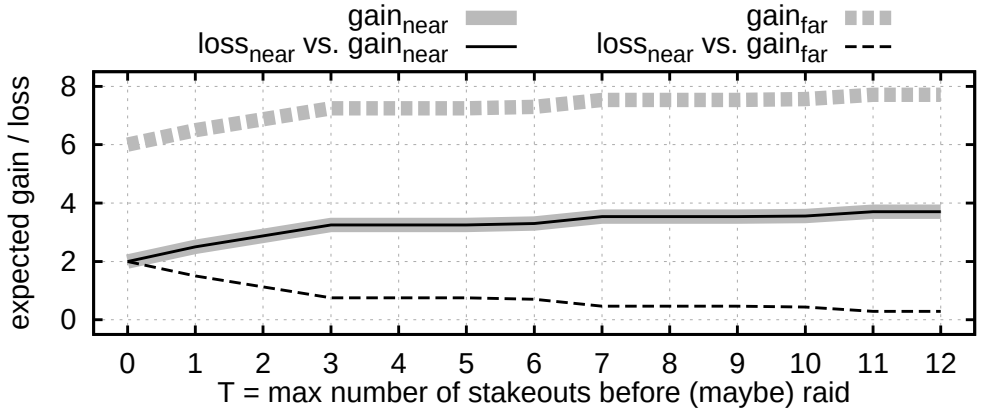


Figure 4.12: Gain and loss in experiment of Section 4.3.8

Expected $\text{gain}_{\text{near}}$ gain with resulting (matching) $\text{loss}_{\text{near}}$ loss (solid lines), and expected gain_{far} gain with resulting (mismatched) $\text{loss}_{\text{near}}$ loss (dashed lines).

locations wrap around at 0 and 7 so that distance between 0 and 7 is 1).[6] This is modeled using the following modification to raid_maybe function:

```
let dist : (x, y) ↦ min{|x − y|, 8 − |x − y|}

let gain_near : (H·h, A·Raid_e, O) ↦
  let raid_gain ← 4 − dist(h, e) in
  let stakeouts ← count_stakeouts(A) in
    raid_gain − 0.05 · stakeouts
```

[6] Say that the pile of gold is so big that it falls into a pyramid-like shape, with a peak in the middle and coins spreading around the area.

Thus, the adversary receives gain of 4 when raiding the exact stash location and as little as 0 when raiding a location as far away from the real stash as possible. The defender's loss function $\text{loss}_{\text{near}}$ is identical to $\text{gain}_{\text{near}}$ above.

It may seem at first glance that we can better safeguard the secret by overestimating the adversary's gain. For example, instead of choosing $\text{gain}_{\text{near}}$, we might assume the adversary receives gain according to a function gain_{far} that is greater than $\text{gain}_{\text{near}}$ on all inputs (that is, overestimates adversary gain):

```
let gain_far:(H·h, A·Raid_e, O)↦
  let raid_gain ← 4 + dist(h, e) in
  let stakeouts ← count_stakeouts(A) in
    raid_gain − 0.05 · stakeouts
```

This function returns adversary gain proportional to the distance between the stash and the raid location (instead of inversely). The two gain functions can be seen in Figure 4.11, for all combinations of stash and raid locations assuming 0 incurred observation costs. The function gain_{far} is plotted in black and is seen above the plot for $\text{gain}_{\text{near}}$ in gray.

Though the alternate gain function is indeed more conservative from the adversary's point of view, it results in very unsound conclusions about the defender's loss. Figure 4.12 demonstrates the adversary gain and defender loss under two different scenarios; one in which the gain and loss functions are both $\text{gain}_{\text{near}}$, and one in which the adversary acts to optimize gain_{far} instead, whereas the defender still loses according to $\text{loss}_{\text{near}}$. It can be seen there, that indeed the expected gain is higher for gain_{far} (thick dashed gray line) than for $\text{gain}_{\text{near}}$ (thick solid gray line) and in that sense gain_{far} does result in a more conservative gain calculation. The associated defender loss is not soundly approximated, however. Instead it can be seen that using the gain_{far} gain function for the adversary reduces the associated loss (dashed black line) as compared to loss with a $\text{gain}_{\text{near}}$-optimized adversary (solid black line). Essentially, it does not matter that gain_{far} is a more conservative gain function than $\text{gain}_{\text{near}}$ in the absolute sense. The relative gains for the adversary encoded in gain_{far} make him optimize his actions so that he can raid as far from the stash as possible, the exact opposite of the behavior induced by $\text{gain}_{\text{near}}$.

4.4 Related Work

In this Chapter we have presented a model for quantifying the information flow of a system whose secrets evolve over time. Our model involves an adaptive adversary, and characterizes the costs and benefits of attacks on the system. We showed that an adaptive view of an adversary is crucial in calculating a system's true vulnerability which could be greatly underestimated otherwise. Likewise we have showed that when the goals of the adversary and that of the defender do not align, both need to be considered to properly estimate risk to the defender which could otherwise be underestimated. We also showed that though adversary uncertainty can effectively be recovered if the secret changes, if the adversary can adaptively wait to attack,

vulnerability can only increase in time. Also, contrary to intuition, we showed that more frequent changes to secrets can actually make them more vulnerable.

Other works in the literature have considered systems with some notion of time-passing. Massey [74] considers systems that can be re-executed several times, whereby new secret and observable values are produced constantly. He conjectured that the flow of information in these systems is more precisely quantified by *directed information*, a form of Shannon entropy that takes causality into consideration, which was later proved correct by Tatikonda and Mitter [103]. Alvim et al. use these works to build a model for *interactive systems* [8], in which secrets and observables may interleave and influence each other during an execution. The main differences between their model and ours are: (i) they see the secret's information content growing with time, rather than evolving; (ii) they consider Shannon-entropy as a metric of information, rather than vulnerability metrics; and (iii) they only consider passive adversaries.

Köpf and Basin [55] propose a model for adaptive attacks on deterministic systems, and show how to calculate bounds on the information leakage of such systems. Our model generalizes theirs in that we consider probabilistic systems. Moreover, we distinguish between the adversarial production of low inputs and of exploits, and allow adversaries to wait until the best time to attack, based on observations of the system, which itself could be influenced by the choice of low inputs.

In the context of Location Based Services, the privacy of a moving individual is closely related to the amount of time he spends in a certain area, and Marconi et al. [65] demonstrate how an adversary with structured knowledge about a user's behavior across time can pose a direct threat to his privacy.

The work of Shokri et al. [97] strives to quantify the privacy of users of location-based services using Markov models and various machine learning techniques for constructing and applying them. Location privacy is a useful application of our framework, as a principal's location may be private, and evolves over time in potentially predictable ways. Shokri et al.'s work employs two phases, one for learning a model of how a principal's location could change over time, and one for de-anonymizing subsequently observed, but obfuscated, location information using this model. Our work focuses on information theoretic characterizations of security in such applications, and permits learning the change function and the secrets in a continual, interleaved (and optimized) fashion. That said, Shokri et al's simpler model and approximate techniques allows them to consider more realistic examples than those described in our work. Subsequent work [98] considers even simpler models (e.g., that a user's locations are independent).

Classic models of QIF in general assume that the secret is fixed across multiple observations, whereas we consider dynamic secrets that evolve over time, and that may vary as the system interacts with its environment. Some approaches capture interactivity by encoding it as a single "batch job" execution of the system. An early work by Gray [52] models the behavior of a system as a probabilistic state machine defining the next state of a system in terms of the prior state and its inputs. These machines induce a probability measure incorporating high inputs and low outputs over which channel capacity is measured. That work does not take into account the

optimization done by adversaries and measures only total information flow (not moving target). Desharnais et al. [35] model systems as a channel matrix of conditional probabilities of whole output traces given whole input traces. Besides creating technical difficulties for computing maximum leakage [8], this approach does not permit low-adaptive or wait-adaptive adversaries, because it lacks the feedback loop present in our model.

O'Neill et al. [86], based on Wittbold and Johnson [104], improve on batch-job models by introducing strategies. The strategy functions of O'Neill et al. are deterministic, whereas ours are probabilistic. And their model does not support wait-adaptive adversaries. So our model of interactivity subsumes theirs.

Clark and Hunt [22], following O'Neill et al., investigate a hierarchy of strategies. *Stream strategies*, at the bottom of the hierarchy, are equivalent to having agents provide all their inputs before system execution as a stream of values. So with stream strategies, adversaries must be non-adaptive. Clark and Hunt show that, for deterministic systems, noninterference against a low-adaptive adversary is the same as noninterference against a non-adaptive adversary. This result does not carry over to quantification of information flow; low-adaptive adversaries derive much more gain than non-adaptive ones as seen in Section 4.3.2. At the top of Clark and Hunt's hierarchy, strategies may be nondeterministic, whereas our model's are probabilistic. Probabilistic choice refines nondeterministic choice [77], so in that sense our model is a refinement of Clark and Hunt's. But probabilities are essential for information-theoretic quantification of information flow. Clark and Hunt do not address quantification, instead focusing on the more limited problem of noninterference. Nor does their model support wait-adaptive adversaries. There are other, nonessential differences between our model and Clark and Hunt's models: Clark and Hunt allow a lattice of security levels, whereas we allow just high and low; our model only produces low outputs, whereas theirs permits outputs at any security level.

Chapter 5

Symmetric Enforcement

Consider a scenario where N parties $P_1, \ldots, P_N$ wish to compute some (known) function $f(s_1, \ldots, s_N)$ of their respective inputs, while ensuring *privacy* of their inputs to the extent possible. If these parties all trust some entity P_T, then each party P_i can simply send its input s_i to this trusted entity, who can in turn evaluate $f(s_1, \ldots, s_n)$ and return the result to each party. In the more general case, where f is a vector-valued function returning outputs $out_1, \ldots, out_N$, the trusted entity gives out_i to party P_i.

Cryptographic protocols for *secure multiparty computation* (SMC) [44, 105] allow the parties to accomplish the same task without the involvement of any trusted entity. (The reader can refer to a recent overview of SMC [59], or a textbook-level treatment [43].) That is, by running a distributed protocol amongst themselves the parties can learn the desired result $f(s_1, \ldots, s_N)$ (or, in the general case, each party P_i learns the result out_i) while ensuring that no information about other party's input is revealed beyond what is implied by the result(s).

Most work on SMC provides an answer to the question of *how* to compute f, but does not address the complementary question of *when* it is "safe" to compute f in the first place, i.e., when the output of f may reveal more information than parties are comfortable with. The two exceptions that we know of [15, 40] decide f's safety *independently* of the parties' inputs and in *isolation* of any (known or assumed) prior knowledge that parties have about each others' inputs.

However, the information implied by a query's result depends both on the parties' inputs and their prior knowledge. As an example of the former, suppose two parties want to compute the "less than or equal" function, $f(s_1, s_2) \stackrel{\text{def}}{=} s_1 \leq s_2$ with variables ranging in $\{1, \ldots, 10\}$. This function could reveal a lot about s_1 to P_2. If $s_2 = 1$ and $f(s_1, s_2)$ returns *true*, then P_2 learns that s_1 can only be the value 1. However, if $s_2 = 5$ then regardless of the output of the function, P_2 only learns that s_1 is one of 5 possibilities, a lower level of knowledge than in the first case.

Additionally, we may deem a pair of queries acceptable in isolation, but allowing their composition would be too revealing. For example, suppose the parties also want to compute "greater than or equal", $f_2(s_1, s_2) \stackrel{\text{def}}{=} s_1 \geq s_2$. When $s_2 = 5$, either query in isolation narrows the values of s_1 to a set of at least 4 possibilities from P_2's perspective. But if f_1 and f_2 both return *true*, P_2 can infer $s_1 = s_2 = 5$.

In Chapter 3 we developed an approach to judging query safety called based on knowledge-threshold security policies. In this Chapter we show how threshold enforcement can be generalized to SMC to address the limitations of current techniques listed above.

Knowledge threshold enforcement relies on reasoning about other parties' knowledge of one's own private data in order to determine whether a given function f is "safe" to compute in a given instance. Our previous work was in an *asymmetric* set-

ting where only one of the parties (say, P_1) was concerned about privacy. The other parties' inputs could be revealed publicly, or at the very least be revealed to P_1; as such, the previous work did not involve SMC at all. At a high level, and specializing to the two-party case, party P_1 knows its own private data s_1 along with P_2's input s_2, and also maintains a *belief* about P_2's knowledge of s_1 (represented as a probability distribution belief). Before agreeing to compute a function $f(s_1, s_2)$, P_1 determines whether computing the residual function $f(\cdot, s_2)$ would reveal "too much information" as determined according to a threshold $0 < t_1 \leq 1$ set by P_1. In particular, P_1 will not compute the function if P_2's belief about the likelihood of a possible secret value (including the actual secret s_1) increases above h_1.[1] If P_1 does reveal $f(s_1, s_2)$ then it determines what P_2 will learn from the output and revises its estimate belief_2 of P_2's knowledge accordingly. It will use this new estimate when considering subsequent functions.

In our prior work, P_1's determination as to whether it should agree to compute f relied in an essential way on the fact that P_1 knows the input s_2 of the other party. In the SMC setting the privacy of *all* parties' inputs should be preserved, so our prior techniques cannot be applied directly. In this chapter we combine knowledge threshold security and SMC, in order to address the question of when it is safe to compute some function f of multiple parties' inputs.

We present two techniques (Section 5.1). The first, which we call the *belief set* method, works as follows. Each P_i maintains a *set* of distributions Beliefs_j for each other principal P_j, one for each possible valuation of s_j. P_j's actual belief belief_j is assumed to be a member of the set Beliefs_j. The same basic procedure as in the prior work, lifted from distributions to sets of distributions, can then be applied by each P_i, and if all participants conservatively agree to participate, they perform the function evaluation via SMC.

The second technique we call *SMC belief tracking*. Rather than have each principal P_i perform the knowledge enforcement procedure individually before the SMC takes place, the enforcement is performed *within the SMC itself*. If the SMC-based-enforcement determines that any of the thresholds h_i will be exceeded by sending a response to P_j then P_j receives a rejection, rather than the actual answer. However, because P_k's knowledge will be different, it could receive a proper answer. By performing enforcement within the SMC, we can look at the *actual* secret values of each of the participants and by accepting/rejecting selectively, we can ensure that no information is revealed by rejection. As we show in Section 5.2 using a series of experiments with our proof-of-concept implementation, SMC belief tracking is strictly more precise (in that fewer queries will be rejected) than belief sets. On the other hand, SMC is known to be very slow, and so implementing enforcement as an SMC could be quite costly.

The work in this chapter is based on a paper published in the 2012 Workshop on Programming Languages and Analysis for Security [69]. The work was a collaboration with Michael Hicks, Jonathan Katz, and Mudhakar Srivatsa.

[1] The release criterion considers all possible values for s_1 — and not just the actual value of s_1 — so that a refusal to participate does not leak any information about s_1.

5.1 Enforcing knowledge thresholds for symmetric queries

In this section we show how to generalize knowledge threshold enforcement from the single-secret scenario to the multi-secret setting of SMC. In this setting, there are N principals, $P_1, ..., P_N$ each with a secret $s_1, ..., s_N$. Each P_i maintains a belief belief_i about the possible values of the other participating principals' secrets. In addition, each P_i has a knowledge threshold h_i that bounds the certainty that the other principals can have about its secret's value.

Next we present an example to illustrate how belief estimation is adapted to the SMC case, and then we use this example to illustrate two possible methods we have devised for enforcing the knowledge threshold, the *belief set* method (Section 5.1.2) and the *SMC belief tracking* method (Section 5.1.4). We prove both methods are sound and discuss their tradeoffs in Section 5.2.

5.1.1 Example

Suppose we have three principals, P_1, P_2, and P_3, each with a net worth $s_1 = 20$, $s_2 = 15$, and $s_3 = 17$, in millions of dollars, respectively. Suppose they wish to compute richest which determines whether P_1 is the richest:

let $\text{richest} : (s_1, s_2, s_3) \mapsto$
$s_1 \geq s_2$ **and** $s_1 \geq s_3$

Using the idealized view, each of P_1, P_2, and P_3 can be seen as sending their secrets to P_T, which evaluates the query to produce the output **true**.

Now suppose that P_1 believes that both P_2 and P_3 have at least \$10 million, but less than \$100 million, with each case equally likely. We will write prior_i for the initial belief of principal i, a distribution over the triple (s_1, s_2, s_3). Thus principal P_1's belief about the three secret variables is the distribution below:

$$\Pr\left(X_{\text{prior}_1} = (20, v_2, v_3)\right) = \frac{1}{8281} \qquad \text{for } v_2, v_3 \in \{10, \cdots, 100\}$$

The beliefs of P_2 and P_3 are defined similarly.

Belief revision proceeds as before: once P_T performs the computation and sends the result, each P_i revises its belief. Given that the output of the example query richest is **true**, principal P_1 would perform revision as below:

let $\text{post}_1 \leftarrow \text{posterior}\ \ \text{prior}_1\ \text{richest}$ **true**

This revised belief additionally disregards possibilities that ascribe s_2 or s_3 to values greater than P_1's own wealth, which is \$20M:

$$\Pr\left(X_{\text{post}_1} = (20, v_2, v_3)\right) = \frac{1}{121} \qquad \text{for } v_2, v_3 \in \{10, \cdots, 20\}$$

The revised beliefs of P_2 and P_3 will be less specific, since each will simply know that P_1's wealth is at least their own and no less than the rest of the parties. Notice that in this example the other parties P_2 and P_3 do not know what P_1's prior or posterior beliefs are. Given that it is they who need to decide whether P_1 knows too much, they will need to maintain some estimate of P_1's knowledge. In the following sections, we will see beliefs written belief_i to designate the actual belief of party i, but also belief_i^j to designate P_i's estimate of the belief of P_j.

5.1.2 Knowledge-based security with belief sets

Now we wish to generalize knowledge threshold enforcement, as described in Section 3.4, to SMC. In the simpler setting P_1 maintained an estimate belief_2^1 of P_2's belief belief_2. In the SMC setting we might imagine that each P_j maintains a belief estimate belief_i^j and then checks that $V^{\max}(\text{belief}_i^j, q) \leq h$ for some threshold h and for all $i \neq j$. If each of these checks succeeds, then P_j is willing to participate.

The snag is that P_j cannot accurately initialize belief_i^j for all $i \neq j$ because it cannot directly represent what P_i knows about s_i—that is, its exact value. So the question is: how can P_j estimate the potential gain in P_i's knowledge about s_j after running query without knowing s_i?

One approach to solving this problem, which we call the *belief set* method, is the following. P_j follows roughly the same procedure as above, but instead of maintaining a single distribution belief_i^j for each remote party P_i, it maintains a *set* of distributions where each distribution in the set applies to a particular valuation of s_i. Let belief be a belief that represents the prior over the secrets of all the parties before taking into account that each party knows their secret. Let $\text{vals}_i = \text{support}(X_{\text{s}_i})$ be set of values of P_i's secret that are possible according to belief. Then, each party P_j initializes this Beliefs_i^j as follows:

```
let Beliefs_i^j ←
  map vals_i (λ v . condition belief_i = v in belief)
```

In the above belief_i designates the i^{th} projection from the tuple belief. Thus Belief_i^j is a set of possible distributions, one per possible valuation of s_i. Using the joint belief in this construction allows us to model correlations in secrets among the various parties. For example, if it were known (by all principals) that only one of the principals in the running example can have secret value equal to 15, then P_2 would know initially, based on this own secret $s_2 = 15$, that P_1's value s_1 cannot be 15. However, P_1 cannot arrive at this conclusion without knowing s_2, which is, of course, outside of its knowledge initially. Since we are starting from a globally held belief, there is no need to distinguish Beliefs_i^j from Beliefs_i^k—they are the same Beliefs_i.

We extend the notion of worst-case posterior vulnerability to operate on sets of beliefs and lift our pseudo-code semantics to operate on sets point-wise, that is, if $\text{S} = \{\text{s}_1, \cdots, \text{s}_n\}$ then $\text{f S} \stackrel{\text{def}}{=} \{\text{f s}_1, \cdots, \text{f s}_n\}$.

Definition 18 (Worst-worst-case Posterior Vulnerability)**.** The *worst-worst-case posterior vulnerability* in a belief set Beliefs for query q is the worst of the worst-case

posterior vulnerabilities over all the beliefs in Beliefs:

$$V^{max}\left(\text{Beliefs}, \text{q}\right) \stackrel{\text{def}}{=} \max_{\text{belief} \in \text{Beliefs}} V^{\max}\left(\text{belief}, \text{q}\right)$$

Now each P_j can use the worst-worst-case posterior vulnerability to gauge risk in participating in a query execution. First the principals agree on the query q. Second, every party P_j checks that no other party can learn too much about their secret (as specified by threshold h_j):

```
forall i ∈ {1, ··· , N} with i ≠ j
  V^max((Beliefs_i)_j, q) ≤ h_j
```

In the third step, if the query is acceptable for all P_j, they participate in a SMC computation to compute $\text{q}\ \text{s}$ where s is the tuple of all of the parties' secrets. As the result o is sent back to each principal P_i, they revise Beliefs_j, the belief sets representing the knowledge of each party P_j:

```
let Posts_j ← posterior Beliefs_j q o
```

Recall that we "lifted" our pseudo-code semantics to operate on sets. Thus the above applies posterior point-wise to each belief in Beliefs_j. We will also assume that any point-wise computation of posterior which includes a conditioning on an impossible predicate does not return and hence does not contribute to the set Posts_j. This can occur since some of the beliefs in the prior Beliefs_j might be inconsistent with q returning o. In essence, this constitutes possibilistic conditioning.

5.1.3 Soundness of belief sets

Now we can show that the belief set procedure is *sound*, in that for all P_i, participating or not participating in a query will never increase another P_j's certainty about P_i's secret above its threshold h_i.

Remark 19. Suppose principals $P_1, ..., P_N$ wish to execute a query q. The secret tuple $s = (s_1, ..., s_N)$ contains all their secrets. Assume that for each P_i:

1. P_i has a belief prior_i.
2. P_i's belief prior_i is consistent with s, that is $\text{s} \in \text{support}\left(X_{\text{prior}_i}\right)$.
3. P_i knows the estimate Beliefs_j of everyone else's knowledge.
4. P_i's belief prior_i is within the estimate of his knowledge, that is, $\text{prior}_i \in \text{Beliefs}_i$.

Suppose $\text{o} \leftarrow \text{q}\ \text{s}$, is the actual output of the query q. Then, the belief of each agent, after learning the output, is $\text{posterior}\ \text{prior}_i\ \text{q}\ \text{o}$, and is a member of the estimated set $\text{posterior}\ \text{Beliefs}_i\ \text{q}\ \text{o}$.

Proof. Lemma2 tells us that $\text{post}_i \leftarrow \text{posterior}\ \text{prior}_i\ \text{q}\ \text{o}$ are the new beliefs of the principals, having learned the output o. By assumption we had $\text{prior}_i \in \text{Beliefs}_i$, and since prior_i was consistent with s, it must be that post_i is well-defined, therefore $\text{post}_i \in \text{posterior}\ \text{Beliefs}_i\ \text{q}\ \text{o}$. □

This remark is merely a lifting of Lemma 2 to sets of beliefs. The more interesting point arises when the principals are also interested in enforcing a knowledge threshold.

Lemma 20. *Suppose the same premise as Remark 19. Also suppose that policy thresholds h_i are public and that each P_i learns either*

- *the output o of the query, or*
- *which principals P_j rejected the query.*

Then, the revised belief of each agent, in the first case is posterior $prior_i$ q o*, and is within the estimate* posterior $Beliefs_i$ q o*, or in the second case, remains unchanged at* $prior_i$*.*

Proof. The lemma effectively states that the policy decisions have no effect on the beliefs; if a query is rejected, learning which principals rejected it reveals nothing. Similarly, if the query is not rejected, the additional information each principal gets (that no one rejected the query), also does not change the belief.

The lemma holds due to the simple fact that the policy decisions based on the worst-worst-case posterior vulnerability do not depend on private information. Every principal could determine, on their own, whether another principal would reject a query (recall the thresholds are assumed to be public). Thus the policy decisions, as a whole, are a simulatable procedure. The rest follows from Remark 19. □

Some subtleties are worth mentioning. First, a premise of the lemma is that Beliefs_i are known by all principals. This fact needs to remain as the query is answered so the same premise will hold for the next query. Fortunately this is the case, as the revised belief sets in the case of policy success, posterior Beliefs_i q o are also known by all participants, as the query q, the output o and the priors Beliefs_i are all known by every party.

A second subtlety is that the queries themselves must be chosen independent of anyone's secret. In some situations, where the principals are actively attempting to maximize their knowledge, and are allowed to propose queries to accomplish this, the query choice can be revealing. This problem is beyond the scope of this work, and we will merely assume the query choice is independent of secrets.

5.1.4 SMC belief tracking: ideal world

Now we present an alternative to the belief set method, in which the decision to participate or not, involving checking thresholds after belief revision, takes place *within the SMC itself.* As such, we call this method *SMC belief tracking.* We present the method using the idealized world with a trusted third party P_T whose function is to stand in place of secure computation.

The first step is that each P_i presents its secret s_i and threshold h_i to P_T, along with the collective belief. Principal P_T then creates several pieces of data he will be maintained as queries are processed according to init_smc below.

Listing 5.1: Secret share initialization

```
let init_smc: (s, h, belief) ↦
  let beliefs ← map {1,···,N} (λi. condition belief_i = s_i in belief) in
  let secret_share ← (s,h, beliefs) in
    secret_share
```

The secret_share stores the secret values, thresholds for policy enforcement, and the initial beliefs of all the parties with beliefs_i referring to the belief of party P_i. Notice that here the belief attributed to party P_i is exact and not a set like it was in the belief-sets approach.

Given the secret share and a query q, the task of the trusted third party is summarized in the query_smc function below.

Listing 5.2: Policy enforcement and query evaluation within SMC

```
let query_smc ((s,h,priors), q) ↦
  let o ← q s in
  let results = map {1,···,N} λ j .
     if ( forall i ∈ {1,···,N} where i ≠ j,
          V^max((priors_j)_i, q) ≤ h_i)
        then (Accept o, posterior priors_j q o)
        else (Reject, priors_j) in
  (* return (results_j)_1 to party P_j *)
  (* new secret share has (results_j)_2 as posterior beliefs *)
```

First, an output o is produced by running the query on the true secret tuple. Then the output for each party P_j is computed along with that party's posterior belief. The process checks that revealing the output to P_j would not violate any other party P_i's threshold h_i, using the same worst-case posterior vulnerability computation we used in the asymmetric case. If all the thresholds pass, the actual output o is produced. Otherwise a rejection is produced. In the first case the belief of party P_j is updated and in the latter case, it is kept the same as it was initially. The trusted third party then returns to each party their respective results.

Importantly, the fact that P_j receives a proper answer or Reject is not (directly) observed by any other P_i; such an observation could reveal information to P_j about s_i. For example, suppose $q_2 \stackrel{\text{def}}{=} s_1 \leq s_2$ and both secrets are (believed to be) between 0 and 9. If $s_2 = 0$ then q_2 s will return **true** only when s_1 is also 0. Supposing $h_1 = 3/5$, then P_2 should receive Reject since there exists a valuation of s_1 (that is, 0) such that P_2 could guess s_1 with probability greater than 3/5. Similar reasoning would argue for reject if $s_2 = 9$, but acceptance in all other cases. As such, if P_1 observes that P_2 receives Reject, it knows that s_2 must be either 0 or 9, independent of h_2; as such, if $h_2 < 1/2$ we have violated the threshold by revealing the result of the query.

This asymmetry means that query_smc may return a result for one participant but not the other, e.g., P_1 might receive Reject because h_2 is too low while P_2 receives the actual answer because h_1 is sufficiently high. Nonetheless each P_i's threshold will be respected.

5.1.5 SMC belief tracking: real world

Lacking a trusted third party in the real world, the participants can use secure multi-party computation and some standard cryptographic techniques to implement P_T's functionality amongst themselves. There are two aspects of P_T that they need to handle: the computation P_T performs, and the hidden state P_T possesses in between queries.

The first aspect is exactly what SMC is designed to do. For the second aspect, we need a way for the participants to maintain P_T's state amongst themselves while preserving its secrecy. (Since we are using the semi-honest adversary model, we do not concern ourselves with integrity; in the malicious setting, standard techniques could be used to enforce integrity.) This state, initially constructed by init_smc (Listing 5.1) consists of (1) the parties' secrets s, (2) policy thresholds h, and (3) the current beliefs beliefs_i. We will refer to this state as Σ_T. We assume Σ_T can be encoded by a binary string of length (exactly) ℓ, for some known ℓ.

The initialization procedure formulated in the idealized world does not output anything to the participants. In the real world, however, the secure computation of init_smc returns *secret shares* of Σ_T to the parties. That is, the secure computation implements the following (randomized) function after computing Σ_T: choose random $c_1, \ldots, c_{N-1} \in \{0,1\}^\ell$ and set $c_N = \Sigma_T \oplus \left(\bigoplus_{i=1}^{N-1} c_i\right)$. Then each party P_i is given c_i.

The query-evaluation procedure query_smc receives $(c_1, ..., c_N)$ along with the query Q. The procedure begins by reconstructing $\Sigma_T = \bigoplus_{i=1}^{N} c_i$, and then proceeds as usual. Upon completion, query_smc computes (new) shares $c'_1, \ldots, c'_N$ of Σ'_T (as before), and gives c'_i to P_i along with the actual output. (At this point, each P_i can erase the old share c_i.)

Note that each time the sharing is done, nothing additional about Σ_T is revealed from any individual fragment ("share") c_i. (Indeed, each c_i is simply a uniform binary string of length ℓ.) In particular, just as in the ideal world, P_i does not learn whether its policy rejected another participant P_j.

Remark 21. Honest (but curious) participants can derive exactly the same knowledge about each other's secrets from the real-world SMC implementation of P_T that they do from interacting with P_T in the idealized world.

There are no theoretical road-blocks to implementing knowledge tracking as an SMC, assuming the shared state's size can be bounded by a public constant like it is assumed in this section. Some high-level languages for SMC such as Wysteria [94] already provide secret sharing at their core. There are, however, practical issues even if a high-level language is used. First is the magnitude of the shared secret. Probability distributions over large state spaces would naturally require significant storage space. Using the approximation to probabilistic computation described in Chapter 6 could help with this problem. Another issue stems from the possibility that the duration of a secure computation, if it were not independent of the inputs, could leak information. Loops, therefore, are usually treated as unrolled sequences of repeated steps, if they are handled at all. Recursive data (like program statements or expressions) is usually not handled at all. The probabilistic interpreter described in Chapter 6 is recursively

defined and the approximate implementation relies on non-trivial tools written using typical programming paradigms at odds with the restrictions of SMC. However, SMC research is active and improvements to performance and convenient specification are ongoing so it is not inconceivable that the practical problems preventing the implementation of knowledge tracking in SMC will alleviate over time.

5.1.6 Soundness of SMC belief tracking

Suppose that no dishonest parties are detected during the runs of SMC belief tracking. Then, by Remark 21, we can justify soundness in the real world by considering the approach in the idealized world.

Lemma 22. *Suppose principals $P_1, ..., P_N$ wish to execute a query q. The secret tuple $s = (s_1, \cdots, s_N)$ contains all their secrets. Each has a public threshold h_i for their policy check. Assume the following for each P_i:*

1. *P_i has a belief $prior_i$ about the secret tuple.*
2. *P_i's belief is consistent with s. That is, $s \in support\left(X_{prior_i}\right)$.*

Suppose $o \leftarrow q\ s$ is an output of the query on the true secret tuple. Then, for each agent P_i:

- *If P_i receives output o from P_T, its revised belief is $posterior\ \ prior_i\ q\ o$.*
- *If P_i is rejected, its belief does not change.*

Specifically, in either case, the procedure query_smc maintains the correct beliefs.

Proof. The proof of this lemma reasons similarly Lemma 20: rejection reveals nothing new, and acceptance tracks beliefs precisely. We can see from Listing 5.2, that the procedure used to determine whether P_j will receive an answer or rejection depends on four things:

- The query q, which is assumed to be public, and chosen independently of secrets.
- P_j's belief, prior_j, about the secrets. This is naturally known by P_j.
- Thresholds h_i for $i \neq j$. These are also assumed to be publicly known.

Since P_j knows all these things, he could determine himself whether P_T will reject him or not. Hence a rejection reveals nothing. In the case P_j receives an answer, we first note the acceptance itself reveals nothing due to the previous argument, and then further, that its belief changes to $\text{posterior}\ \ \text{prior}_j\ \text{q}\ \text{o}$ as claimed due Lemma 2. □

The lemma itself is only useful, however, when its premises hold. Specifically, we require P_T to possess the actual beliefs of the participants to start with. This, in turn, means that the initial init_smc procedure produced them. How the participants arrived at belief, the common belief about the secret variables, used by init_smc to compute prior_j, is beyond the scope of this work. Once the premises hold, however, Lemma 22 states that they will continue to hold; the tracked beliefs will remain correct and thus the protections of the threshold policies will be maintained.

5.2 Experiments

The belief set method and the SMC belief tracking methods present an interesting tradeoff. On the one hand, SMC belief tracking is clearly more precise than belief sets for the simple reason that P_i's estimate of the gain in the other principals' beliefs can consider their secret values exactly without fear that rejection will reveal any information.

On the other hand, SMC belief tracking has two drawbacks. First, the estimate belief_i of what the other parties believe about P_i's secret must be kept hidden from P_i to avoid information leaks. This is unsatisfying from a usability point of view: P_i can be sure that its threshold is not exceeded but cannot see exactly what others know at any point in time. Second, while the performance of SMC has improved quite a bit over time [51], computing a query q via SMC is still orders of magnitude slower than computing it directly. The belief tracking computation of q would already be magnitudes slower than computing q on the actual values, so performing this computation as an SMC will be significantly slower still. Worse, belief tracking is a recursive procedure, since it is an interpreter, and recursive procedures are hard to implement with SMC. So it remains to be seen whether SMC belief tracking can be implemented in a practical sense.

As a step towards a more thorough evaluation of the precision/performance tradeoff, the remainder of this section compares the precision of the belief set and SMC belief tracking methods on three simple queries. We simulate the SMC belief tracking computation by running our normal implementation in the ideal world setup. We find that SMC belief tracking can be significantly more precise than belief sets, but that belief sets can nevertheless be useful.

It is important to note that we use an approximate probabilistic interpreter we describe later in Chapter 6. The inference using that approach is approximate but sound in that the probability in the definition of vulnerability (see Definition 5) is never underestimated (though it could be overestimated).

5.2.1 "Am I the richest?" example (richest).

Consider the running example query richest:

```
let richest : (s1, s2, s3) ↦
  s1 ≥ s2 and s1 ≥ s3
```

If all the principals were to evaluate threshold policies to determine the safety of richest, they would reason about possible revised beliefs of the participants, where the possibilities vary in their valuation of those participants' secret values, as is described in Section 5.1.2. If the principals perform this policy check via SMC, they would do so for only one of those possible valuations.

We can better understand the relationship between the two approaches by looking at the range of possible revised beliefs achievable. For some secret values, a principal might learn little; for others, they might learn a lot. We measure this range in terms of the probability of the most probable secret in a principal's belief, for a given valuation of their own secret.

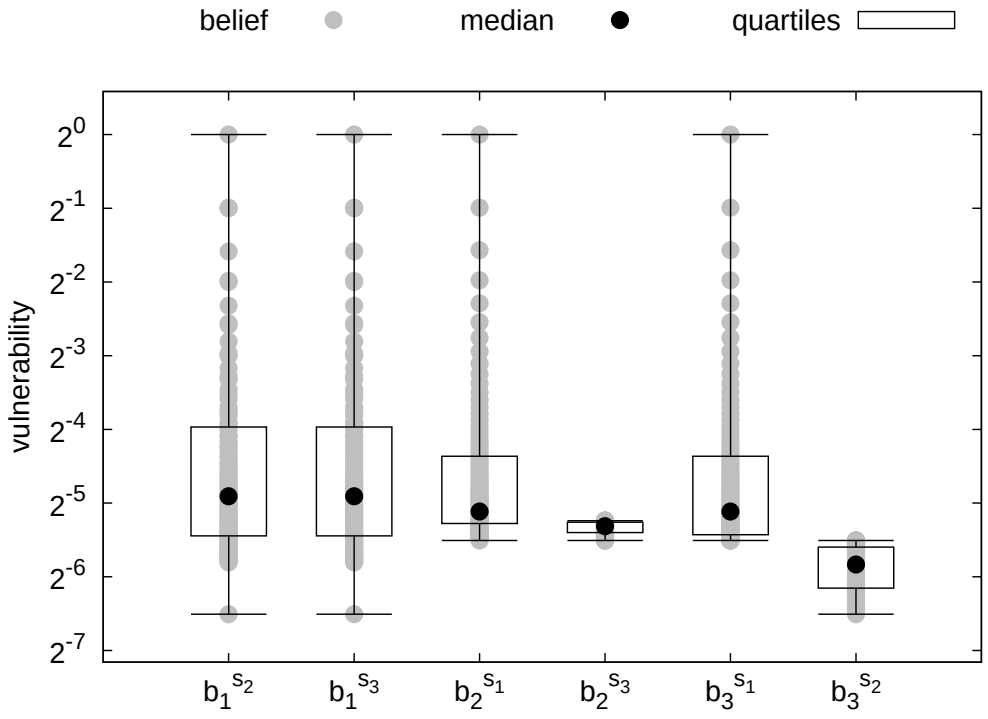


Figure 5.1: Running example richest, plot of vulnerabilities

Figure 5.1 demonstrates the situation for the running example, query richest, starting from the initial belief belief uniformly distributing values in $10 \leq s_1, s_2, s_3 \leq 100$. There are 6 relationships considered, for each principal P_i, during their policy decision to allow P_j, with $j \neq i$, to see the query output, they would compute P_j's potential belief about s_i (labeled $b_j^{s_i}$ in the figures). These beliefs depend on s_j; the figure shows the potential belief for every possible s_j, the median belief achievable over them as well as the 1st and 3rd quartiles, showing the range of P_j's likely knowledge. [2]

Figure 5.2 focuses on the first column of Figure 5.1, the belief belief_1 projected to the second party's secret. It shows the extent of P_1's knowledge about s_2, depending on the value of their secret s_1. At the very top, the most P_1 could learn is when $s_1 = 10$ and the query returns **true**, meaning P_1 was the richest, with the smallest amount of wealth. This lets P_1 conclude that $s_2 = 10$ and $s_3 = 10$. P_1's potential knowledge of s_2 decreases as P_1's wealth grows, up to $s_1 = 65$. At 65, if P_1 is the richest, it is able to narrow s_2 down to 56 values (10 through 65). Starting with $s_1 = 66$, however, P_1 can learn more if the query returns **true**, stating that either P_2 or P_3 is richer than P_1. Further increase in s_1, increases its potential knowledge of s_2, culminating at $s_1 = 99$ which lets P_1 conclude that $s_2 = 100$ with a probability close

[2] This imprecision due to use of approximate inference is the reason why belief sets representing P_2 and P_3's beliefs about each other, or about P_1, in Figure 5.1 appear different, though in actuality they are the same.

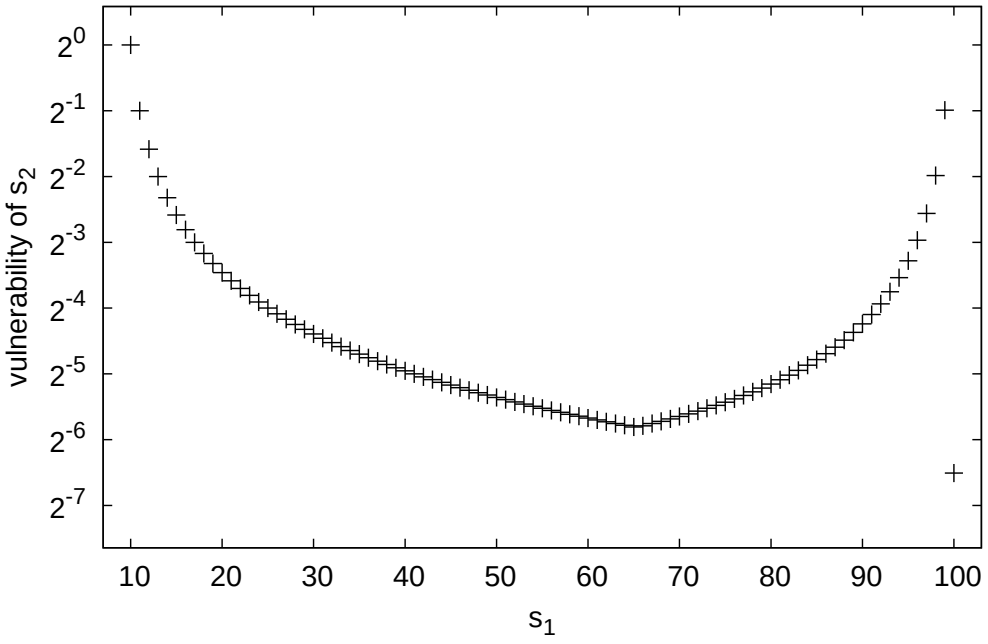


Figure 5.2: Running example richest; $\delta_1^{x_2}$; plot of P_1's max belief about x_2 vs. values of x_1

to 0.5. At $s_1 = 100$, the query can only return **true**, hence P_1 learns nothing, keeping its knowledge of s_2 unchanged at 1/91.

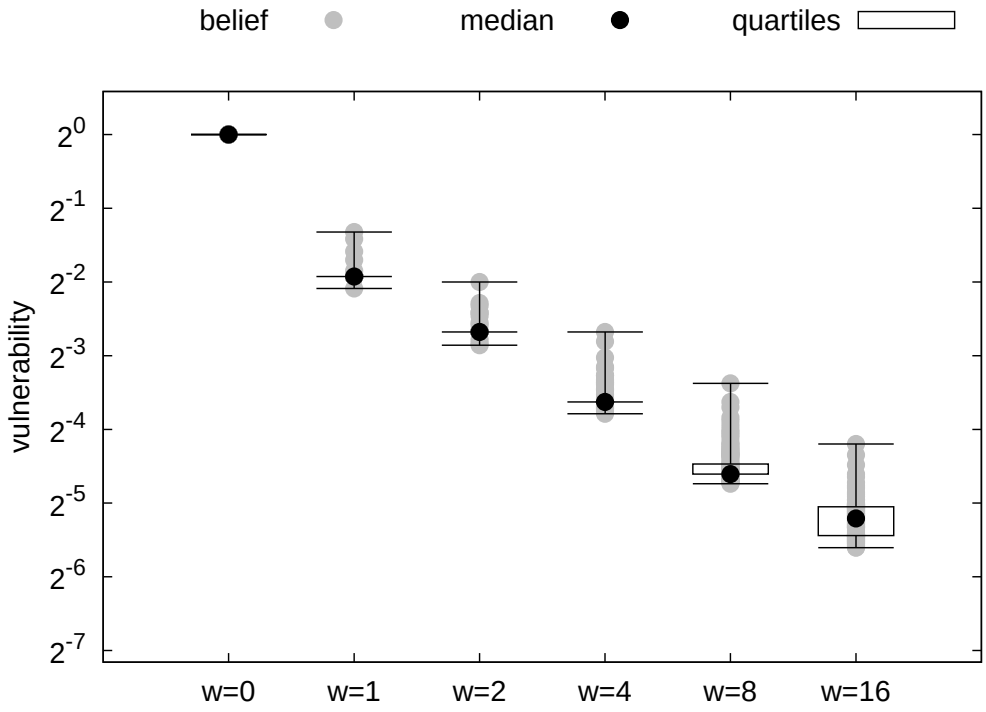
We see that for this query, the belief sets approach would conservatively conclude that all participants could learn s_1 exactly, and that P_1 could learn s_2 and s_3 exactly. On the other hand, it is impossible for P_2 and P_3 to learn each other's values to any confidence.

The benefit of the SMC approach to policy enforcement is that it is free from the overly conservative view of the belief sets approach. In 75% (observing the upper extent of the quartile boxes) or more of the situations, the actual beliefs of the participants do not exceed probability of $2^{-4} \approx 0.06$, which is comparable to the $\frac{1}{91} \approx 0.01$ probability each agent started with. In terms of utility, if the participants set their policy thresholds to as little as 0.06, their policies would allow richest in most cases. The belief sets approach would reject richest for all $h_i < 1$.

Not all queries are pathological for the belief sets approach. We next look at a parameterized query that offers a security vs. utility tradeoff.

5.2.2 "Similar" example

The query similar_w, depicted at the bottom of Figure 5.3, determines whether each principal's secret is within w of the average. The choice of window size w determines




```
let similar_w: (s1, s2, s3) ↦
  let avg ← (s1 + s2 + s3)/3 in
    |s1 − avg| ≤ w and |s2 − avg| ≤ w and |s3 − avg| ≤ w
```

Figure 5.3: similar_w example; plot of vulnerability for a variety of windows w sizes

how much the principals can learn. The plot at the top of the figure shows the possible beliefs after evaluating similar_w with a variety of window values w, with the initial assumption that all values x_i are in uniformly distributed in $1 \leq s_i \leq 100$ (so each of the 100 possibilities has probability 0.01). The scenario is thus completely symmetric in respect to the agents, hence only one of the beliefs is shown in the figure.

When $w = 0$, the query can be completely revealing, as when it returns true, all secrets are equal. Relaxing the window reduces how much each agent can learn. At $w = 2$, each agent, in the worst case, learns every other principal's secret with confidence of 0.25. This worst case already allows non-trivial threshold policies. Going further, with the window set to 16, the query becomes barely revealing, resulting in confidence never reaching over 0.05, comparable to the initial 0.01. Further increase of w can make the query even less revealing to the point of not releasing any information at all, though of course also not providing any utility.

5.2.3 "Millionaires" example.

The common motivating example for SMC is a variant of richest involving a group of millionaires wishing to determine which of them is the richest, without revealing their

exact worth. The query $richest_p$, given at the bottom of Figure 5.4, accomplishes this goal, determining which of 3 participants is (strictly) richer than the other two. An addition to this query has been made to provide a means of injecting noise into its answers to limit potential knowledge gain. The output 0 designates that none of the three were strictly richest. The query is concluded with a step that noises the result. Thus given the parameter p, this query will just randomly return, with probability p, one of the 4 possible outputs.

A significant benefit of using probabilistic programming languages is that the effect of non-determinism described using these probabilistic statements is taken into account; we can determine exactly what a rational agent would conclude from learning the output of such a noised query.

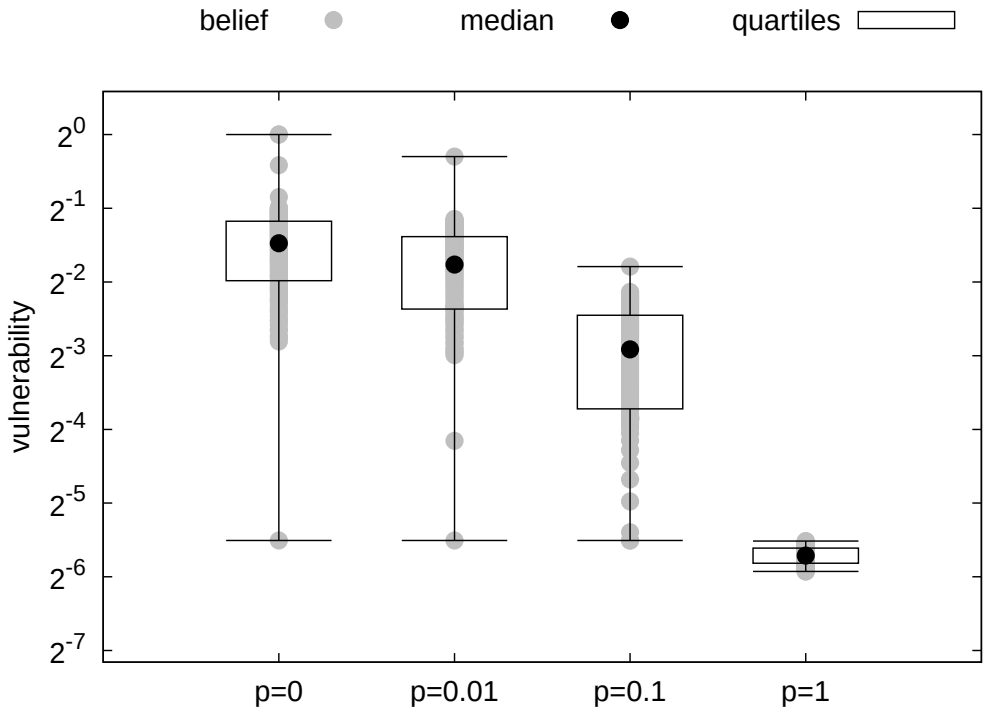
Figure 5.4 summarizes the beliefs of every agent assuming initially each is equally likely to be worth between \$10 and \$100. This scenario is also symmetric hence only the belief of one agent about a single other agent is shown.

With $p = 0$, that is, no chance of random output, the query is potentially fully revealing, but in most cases still keeping the participants below 0.5 certainty. With a 0.01 chance of random output, the worst case no longer results in absolute certainty, though close to it. Randomizing the output with $p = 0.1$, keeps the agents' certainty almost below 0.5 in the worst case. Getting closer to $p = 1$ the beliefs approach the initial ones. At $p = 1$ the query reveals nothing, though our approximate implementation does produce some variation in the upper bound.

5.3 Related Work

Almost all prior work on SMC treats the function f being computed by the parties as given, and is unconcerned with the question of whether the parties should agree to compute f in the first place. The only exceptions we are aware of are two papers [15, 40] that consider SMC in conjunction with *differential privacy* [38, 41]. Dwork et al. [40] show that if f is a differentially private function, then the process of running an SMC protocol that computes f is also differentially private (at least in a computational sense). Beimel et al. [15] observe that if the end goal is a distributed protocol that is differentially private for both parties, then SMC may be overkill to possible, more efficient alternatives. Likewise McGregor et al. [76] show that for some distributed protocols require a minimal amount of error in order to be differentially private. They show, however, that computational relaxations of differential privacy can reduce the required error. Some recent work on secure computation relies on employing oblivious RAM [45] for better handling of large amounts of inputs to secure computations. Oblivious RAM is used to hide memory access patterns which can leak information about private information. Liu et al.'s work [60] attempts to optimize the use of multiple oblivions RAM banks without compromising privacy of the inputs.

The security goal in the symmetric setting of this chapter is the same vulnerability limit that we used in Chapter 3. In that sense the rough relationship between our goal and differential privacy that we described in Section 3.6 holds in the symmetric




```
let richest_p: (s1, s2, s3) ↦
  let o ← if s1 > s2 and s1 > s3 then 1
          else if s2 > s1 and s2 > s3 then 2
          else if s3 > s1 and s3 > s2 then 3
          else 0 in
  if flip 0.5 then o else uniform {0,1,2,3}
```

Figure 5.4: richest_p example; plot of vulnerabilities for a variety of noising probabilities p

setting as well. The claim that differential privacy comes at a greater cost of utility holds less strongly here in the case of enforcement using the very conservative belief sets method we described in this chapter. On the other hand, it is still the case that our methodology allows queries to be answered truthfully without additional noise, which is something differentially private mechanisms do not allow.

5.4 Summary

In this chapter we have presented two methods that apply *knowledge threshold enforcement* to the problem of determining whether participating in a secure multiparty computation could unsafely reveal too much about a participant's secret input. Ours are the first techniques that consider the actual secrets and prior knowledge of participants (potentially gained from previous SMC's) when making this determination, making our approach more permissive (in accepting more functions), and potentially

safer, than techniques that disregard this information. Experiments with the two methods show that the *SMC belief tracking* method is the more permissive of the two, but it remains to be seen whether this method can be implemented efficiently.

Chapter 6

Probabilistic Programming

Consider how we might implement belief tracking and revision to enforce the threshold security property based on Definition 6 (we consider only the trivial identity target in this chapter, that is, we are protecting the whole secret as opposed to some predicate about the secret). A natural choice would be to evaluate queries using a probabilistic programming language with support for conditioning, of which there are many [18, 21, 47, 54, 78, 87, 89, 92]. 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.

In this chapter we revisit the computational aspect of modeling, tracking, and limiting adversary knowledge. We present the work in a separate chapter as the most of the technology developed here has wider use for probabilistic programming in general, not restricted to knowledge tracking and policy enforcement. We briefly summarize knowledge tracking and how this chapter related to probabilistic programming in Section 6.1 for readers wishing to skip the prior three chapters. Next, in Section 6.2, we take another perspective on the demographic birthday query example we saw in Section 3.1. In this chapter we will not use pseudo-code to describe probabilistic programs and instead use a limited syntax whose exact semantics we describe in Section 6.3. We will then introduce our probability distribution abstraction which we make precise in Section 6.4. We then show in Section 6.4.4 how to use the approximate probabilistic inference provided by the abstraction to soundly enforce knowledge threshold we defined previously in Section 6. We make further refinement on our abstraction in Section 6.5 and perform numerous experiments using the implementation of the approach in Section 6.6.

This chapter is based on the same papers that form the basis of Chapter 3. The papers were published in the 2011 Computer Security Foundations Symposium [70], a 2013 issue of the Journal of Computer Security [72], and a University of Maryland technical report released in 2011 [71]. The work was done in collaboration with Michael Hicks, Stephen Magill, and Mudhakar Srivatsa. In this chapter we focus on the computational aspects of probabilistic programming and policy evaluation which were omitted in Chapter 3.

6.1 Knowledge Tracking and Probabilistic Programming

In Section 2.2 we saw how we can use probabilistic programs to describe prior background knowledge of adversaries and how model the posterior knowledge after the adversary learns the outputs of programs or queries. Additionally we saw in Section 2.1 how to measure the vulnerability or risk associated with adversary knowledge. Chap-

ter 3 applies these elements to the problem of determining when to answer a query and when to reject it. The proposed approach rejects queries if there is a possibility that the level of adversary knowledge after learning the output of the query would constitute a vulnerability beyond a given threshold. This is done carefully to make sure that the decision procedure involved is simulatable by the adversary, so that it does not leak information by merely rejecting to answer a query.

Though large parts of Chapter 3 deal with issues specific to the problem of deciding to answer a query, the core of the approach lies in the ability to perform probabilistic inference given program outputs. This chapter focuses almost entirely on the computational issues with performing this inference. The core is an approximate probabilistic semantics of programs with the addition of conditioning that has a strong soundness guarantee. The approximation is based on interpretation and conditioning of a set of distributions which generally has a more concise representation that a single concrete one. Soundness (Theorem 27) tells us that the approximate semantics over-approximate the concrete semantics. The design of the approximation of distributions lets us compute an upper bound on the probability of any element. Soundness, then, guarantees that the true probability cannot exceed this upper bound.

6.2 Example: Birthday revisited

An astute reader of our bday_query example of Section 3.1 would have noticed many regularities in the structure of probability distributions that represented adversary beliefs. In this example we will again look at the query in conjunction with a slightly more complex one in a manner that highlights the uniformities in beliefs. Our approach to approximate probabilistic inference will naturally follow.

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* $h_d = 0.2$ 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 $h_{dy} = 0.05$, 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 ($m = 1$):

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

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 23.

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

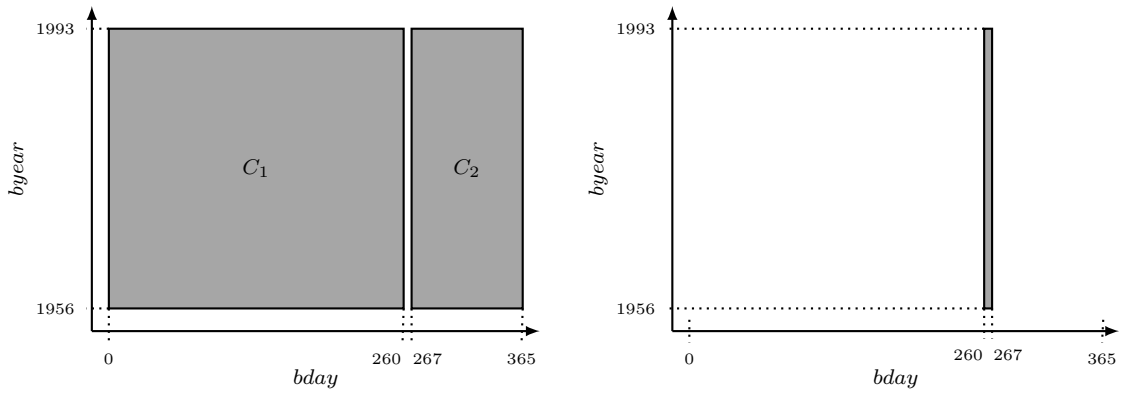
This program refers to Bob's secret variable *bday*, and also uses non-secret variables *today*, which represents the current day and is here set to be 260, and *output*, which is set to True if the user's birthday is within the next seven days (we assume *output* is initially False).

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 (see Figure 6.1(a)), where again all points in each are equally likely (with probability masses $m_1 = \frac{260}{358} \approx 0.726, m_2 = \frac{98}{358} \approx 0.274$):

$$\begin{array}{l} C_1 = 0 \leq bday < 260,\ 1956 \leq byear < 1993 \\ C_2 = 267 \leq bday < 365,\ 1956 \leq byear < 1993 \end{array}$$

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



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

Figure 6.1: Example 1: revised beliefs

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 $h_d = 0.2$ (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 3.4 we proposed 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* [53]: 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. The pif statement in the code below defines a non-deterministic conditional given a probability of executing the branch.

Example 24.

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

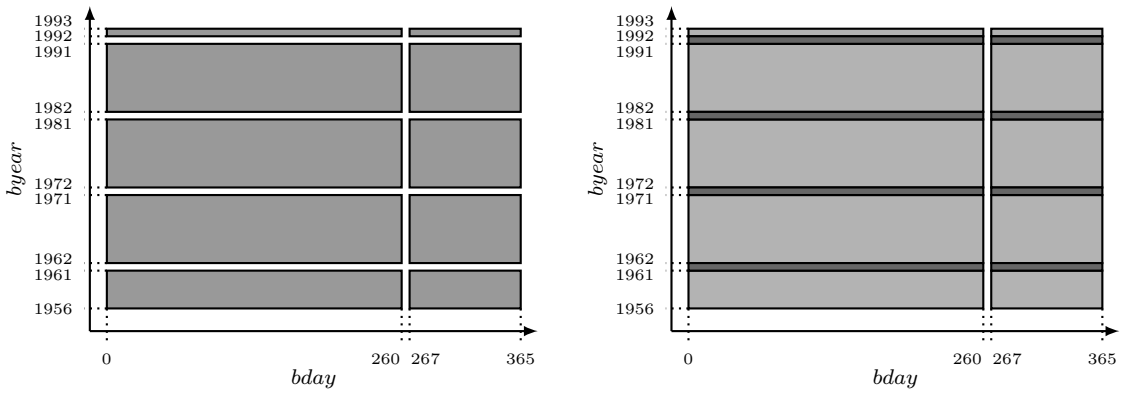
age is (or will be) an exact decade, or if the user wins the luck of the draw (one chance in ten), as implemented by the probabilistic if statement.

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 knowledge, combined with the knowledge gained from the first query, as shown in Figure 6.2(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 6.2(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 $h_{dy} = 0.05$, 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 6.4 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, probabilistic 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 $\mathrm{s}^{\max} = \mathrm{s}^{\min} = \#(C)$. The representation of the convex polyhedra themselves is unimportant at this point. In Section 6.4.1 we will look at them more closely and see how they are used for representing sets of program states.

With the seven-tuple representation, we could choose to collapse the sets of polyhedra given in Figure 6.2. For example, we could represent Figure 6.2(a) with two probabilistic polyhedra P_1 and P_2 bounded by the C_1 and C_2 defined in Example 23 (see Figure 6.1(a)), 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



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

Figure 6.2: Example 2: most precise revised beliefs

(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 6.4. 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 23 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 24, 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 6.2(b) in a similar manner, using the same two polyhedra C_1 and C_2, the other parameters for C_1 are as follows:

$$\begin{aligned}
&\mathrm{p}_1^{\min} = 1/135050 = \tfrac{1}{37*365} * \tfrac{1}{10} \quad \mathrm{p}_1^{\max} = 10/135050 = \tfrac{1}{37*365} \\
&\mathrm{s}_1^{\min} = 9620 = 260 * 37 \qquad\qquad \mathrm{s}_1^{\max} = 9620 = 260 * 37 \\
&\mathrm{m}_1^{\min} = 26/185 \qquad\qquad\qquad\ \mathrm{m}_1^{\max} = 26/185
\end{aligned} \tag{24.1}$$

In this case $s_1^{\min} = s_1^{\max} = \#(C_1)$, meaning that all covered points are possible, but $p_1^{\min} \neq p_1^{\max}$ as some points are more probable than others (i.e., those in the darker band). An astute reader might notice that here $m_1^{\min} \neq p_1^{\min} * s_1^{\min}$ and $m_1^{\max} \neq p_1^{\max} * s_1^{\max}$. The benefit of these seemingly redundant total mass quantities in the representation is that they can sometimes be computed precisely. In this case $m_1^{\min} = m_1^{\max} = \frac{4}{37} * \frac{260}{365} + \frac{1}{10} * \frac{33}{37} * \frac{260}{365}$. 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 chapter, 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 $10/135050 = p_1^{\max}$ (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 bounds in our abstraction. In our example, this results in $p_1^{\max}/(m_1^{\min} + m_2^{\min}) = \frac{10}{135050} / \left(\frac{26}{185} + \frac{49}{925}\right) \approx 0.0004 \leq h_d = 0.05$.

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 on probabilistic programming, though the use of lower/upper probability measures for modeling uncertain probability is an idea at least decades old [33]. Each of the components of a probabilistic polyhedron play a role in producing the lower bound on total mass. While $s_1^{\min}, s_1^{\max}, p_1^{\min}$, and $m_1^{\max}$ 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* [27] (a.k.a. boxes) or *octagons* [79] 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 6.6. Probabilistic intervals are not ideal for every query; in Section 6.6.4 we describe queries for which the probabilistic octagons or polyhedra domains are necessary.

6.3 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 [24]. We retain Clarkson et al.'s notation in lieu of the probabilistic program code of the prior chapters. The language presented here will be formally defined, though it is simpler. The various notations we present below let us carefully define semantics of probabilistic programs without confusion stemming from the presence of multiple distributions referring to the same variables. The main difference from previous chapters is the use of probability distributions (noted as δ) represented by and manipulated as functions from program states (noted by σ, τ) to real values. When we wrote random variables of the form X_{exp} in the prior chapters we implicitly assumed that the potentially probabilistic expression exp is evaluated to produce the distribution governing the values of the random variable. The language in this chapter is imperative. If the statement P has the same effect on some state variable as exp, then we would write $[\![P]\!]\delta$ to describe the equivalent of X_{exp}, explicitly designating the evaluation and the environment (distribution of program states).

6.3.1 Core language

The programming language we use for queries is given in Figure 6.3. 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:

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

The language is essentially standard, though we limit the form of expressions to support our abstract interpretation-based semantics (Section 6.4). 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.

6.3.2 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

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

Figure 6.3: Core language syntax

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 $fv(\sigma)$ is defined by $domain(\sigma)$, sometimes we will refer to this set as just the *domain* of σ. We will also use the this notation for distributions; $fv(\delta) \stackrel{\text{def}}{=} domain(domain(\delta))$. In the context of distributions, *domain* will also refer to the set $fv(\delta)$ as opposed to $domain(\delta)$.

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

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

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

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

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

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

distribution can be constructed by scaling a distribution according to its mass:

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

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: $\text{support}\,(\delta) \stackrel{\text{def}}{=} \{\sigma \mid \delta(\sigma) > 0\}$.

The agent evaluates a query in light of the querier's initial belief using a probabilistic semantics. Figure 6.4 defines a semantic function $[\![\cdot]\!]$ whereby $[\![S]\!]\delta = \delta'$

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

where

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

Figure 6.4: Probabilistic semantics for the core language and index of state/distribution operations

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. As noted by Clarkson [24], the existence of the fixed-point derives from the continuity of the semantics and the ordering on distributions which was proved by Kozen [58]. For an implementation, however, the evaluation of a loop can be performed naïvely, 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 $\sigma \in \text{support}\,(\delta)$.

In Section 6.4 we make use of an additional convenience statement, $\mathsf{uniform}\ x\ n_1\ n_2$ (equivalent to a series of probabilistic conditionals) intended to assign a uniform value in the range $\{n_1, ..., n_2\}$ to the variable x.

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

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.

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

6.3.3 Belief and security

Clarkson et al. [24] 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 normal$(\delta)(\sigma)$. Finally, we extract the attacker-visible output of the sampled state by projecting away the high variables: $\hat{\sigma}_L = \hat{\sigma} \upharpoonright L$. 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 $\hat{\delta}_H = (\delta' \wedge \hat{\sigma}_L) \upharpoonright H$. 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 [24].

6.4 Belief revision via abstract interpretation

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 [30] 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 [13, 91]). 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.

6.4.1 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 $fv(\beta)$ to be the set of variables occurring in β; we also extend this to sets, writing $fv(\{\beta_1, \ldots, \beta_n\})$ for $fv(\beta_1) \cup \ldots \cup fv(\beta_n)$.

Definition 25. 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.

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

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 $\mathit{fv}(\sigma)$, we can view σ as a point in an N-dimensional space, where $N = |\mathit{fv}(\sigma)|$. 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 $x \in \mathit{fv}(C)$, 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, $\mathrm{f}_x(C) = \pi_{\mathit{fv}(C)-\{x\}}(C)$, where $\pi_V(C)$ is a polyhedron C' such that $\gamma_{\mathbb{C}}(C') = \{\sigma \mid \tau \in \gamma_{\mathbb{C}}(C) \wedge \sigma = \tau \restriction V\}$. So $C' = \mathrm{f}_x(C)$ implies $x \notin \mathit{fv}(C')$. 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 $C_A \swarrow\!\!\!\searrow C_B$ 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 $C_A \swarrow\!\!\!\searrow C_B = \{C_A, C_B\}$.

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

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

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

We will write $fv(P) \stackrel{\text{def}}{=} fv(C)$ 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 6.2, Equation 24.1 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 6.3.2). 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 [24]. 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 Section B.8 we use an alternative, though equivalent, semantics of loops defined as an infinite sum of distributions (which in the abstraction, are represented as probabilistic polyhedra), with decreasing probability mass. In the alternative semantics, reaching a fixed point is equivalent to reaching a term of the infinite sum that has zero mass. Thus 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. More details about this can be found in the Appendix, Section B.8. The zero 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 $\mathit{iszero}(P)$ 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 $\mathit{iszero}(P)$ (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 $\mathbb{P}$.

6.4.3 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 6.4. 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 Appendix B.

The abstract program semantics is then exactly the semantics from Figure 6.4, but making use of the abstract operations defined here, rather than the operations on distributions defined in Section 6.3.2. 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 27. *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 6.3.3.) 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 [26, 30]. 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 27 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. The proofs for the various cases and the theorem can be found in Appendix B.

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:

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

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 $V = fv(C_1) - \{y\}$, then for every $\sigma_V \in \gamma_{\mathbb{C}}(C_1 \restriction V)$ we have $\mathrm{h}_y^{\min} \leq |\{\sigma \in \gamma_{\mathbb{C}}(C_1) : \sigma \restriction V = \sigma_V\}| \leq \mathrm{h}_y^{\max}$. 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 $S = \{\sigma \in \gamma_{\mathbb{C}}(C_1) \mid \sigma \restriction V = \sigma_V\}$. 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 $\mathrm{p}_1^{\min}$.

- $\mathrm{p}_2^{\max}$: 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 6.5 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

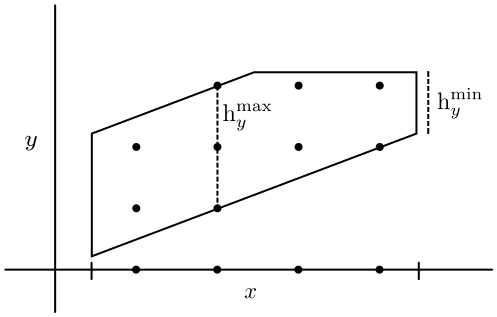


Figure 6.5: Example of a forget operation in the abstract domain $\mathbb{P}$
In this case, $h_y^{\min} = 1$ and $h_y^{\max} = 3$. Note that $h_y^{\max}$ is precise while $h_y^{\min}$ is an under-approximation. If $s_1^{\min} = s_1^{\max} = 9$ then we have $s_2^{\min} = 3$, $s_2^{\max} = 4$, $p_2^{\min} = p_1^{\min} \cdot 1$, $p_2^{\max} = p_2^{\max} \cdot 4$.

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 $h_y^{\min} = 1$. 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 $h_y^{\min}$ in Figure 6.5. 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 $h_y^{\max}$.

Lemma 28. *If* $\delta \in \gamma_{\mathbb{P}}(P)$ *then* $f_y(\delta) \in \gamma_{\mathbb{P}}(f_y(P))$.

We can define an abstract version of projection using forget:

Definition 29. Let $f_{\{x_1,x_2,\ldots,x_n\}}(P) = f_{\{x_2,\ldots,x_n\}}(f_{x_1}(P))$. Then $P \restriction V' = f_{(fv(P)-V')}(P)$.

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

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:

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

The abstract implementation of this operation strongly depends on the invertibility of the assignment. Intuitively, the set $\{\tau \mid \tau\left[x \to [\![E]\!]\tau\right] = \sigma\}$ 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 30. *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.

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 $\delta_1 + \delta_2 \in \gamma_{\mathbb{P}}(P_1 + P_2)$.

[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.

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 31. 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 $P_1 \mathbin{\dot{\frown}} P_2$, 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* $P_1 \mathbin{\ddot{\frown}} P_2$ is the cardinality of the largest possible overlap. Formally, we define these as follows.

$$\begin{array}{l} P_1 \mathbin{\dot{\frown}} P_2 \stackrel{\text{def}}{=} \max\left[\mathrm{s}_1^{\min} + \mathrm{s}_2^{\min} - \left(\#(C_1) + \#(C_2) - \#(C_1 \sqcap_{\mathbb{C}} C_2)\right),\ 0\right] \\ P_1 \mathbin{\ddot{\frown}} P_2 \stackrel{\text{def}}{=} \min\left[\mathrm{s}_1^{\max}, \mathrm{s}_2^{\max}, \#(C_1 \sqcap_{\mathbb{C}} C_2)\right] \end{array}$$

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 32. 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.

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

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

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.

$$C_3 \ = \ C_1 \times C_2$$

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

Lemma 34. *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.

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

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

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

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 35. 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.

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

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 $\overline{n} = C_1 \# \neg B$, 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 $\max\big(\mathrm{s}_1^{\min} - n, 0\big)$.

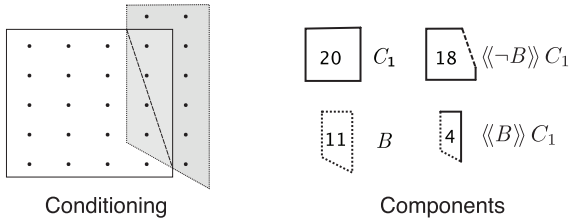


Figure 6.6: Example of distribution conditioning in the abstract domain $\mathbb{P}$.

Figure 6.6 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 6.6. 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 36. *If $\delta \in \gamma_{\mathbb{P}}(P)$ then $\delta \wedge B \in \gamma_{\mathbb{P}}(P \wedge B)$.*

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

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

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

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

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

Lemma 38. *If $\delta_1 \in \gamma_{\mathbb{P}}(P_1)$ then $p \cdot \delta_1 \in \gamma_{\mathbb{P}}(p \cdot P_1)$.*

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

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

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

6.4.4 Policy Evaluation

Here we show how to implement the vulnerability threshold test of Definition 6 using probabilistic polyhedra. To make the definition simpler, let us first introduce a bit of notation.

Notation 41. 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 42. 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 $\mathrm{p}_4^{\max} \leq h$, then we write *tsecure*$_h(S, P_1)$.

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 h.

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 43. *Let δ be an attacker's initial belief. If $\delta \in \gamma_{\mathbb{P}}(P_1)$ and tsecure$_h(S, P_1)$, then S is threshold secure for threshold h when evaluated with initial belief δ.*

The proof of this theorem follows from the soundness of the abstraction (Theorem 27), noting the direct parallels of threshold security definitions for distributions (Definitions 6) and probabilistic polyhedra (Definition 42).

6.4.5 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 6.4.1 can be implemented. Most of the operations listed there are standard to 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* [27], $\mathbb{C}_{\mathbb{I}}$, 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 [79], $\mathbb{C}_{\mathbb{O}}$, 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 6.7. This fact is relevant when computing the various equivalent operations to those listed for polyhedra in Section 6.4.1: 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.

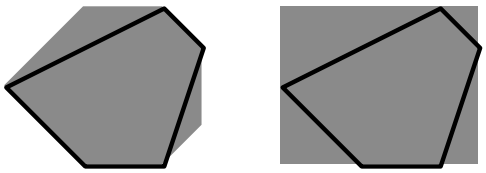


Figure 6.7: The (over)approximation of a polyhedron using an octagon (left) and an interval (right).

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 26, 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 44. 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 $\gamma_{\mathbb{C}}(C) \subseteq \gamma_{\mathbb{C}_{\mathbb{O}}}(C_O)$.

6.5 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 45. 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.

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

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 [81], elements of Δ need not be disjoint. While enforcing disjointness would simplify determining the most probable points for policy evaluation (see Section 6.5.2), 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

[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.

are zero.

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

6.5.1 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.

Theorem 46. *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 27), via soundness of the individual abstract operations in relation to their concrete versions. The full proof is shown in appendix C.

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 47. 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 48. *$\gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta) \subseteq \gamma_{\mathcal{P}_n(\mathbb{P})}(\lfloor \Delta \rfloor_m)$ 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 6.6.3.

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

Project $\Delta \upharpoonright V \stackrel{\text{def}}{=} \{P \upharpoonright V \mid P \in \Delta\}$

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

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

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 24. 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 $\{\sigma \mid \sigma \models B\} = \{\sigma \mid \sigma \models B_1 \vee \cdots \vee B_m\}$, each disjunct B_i contains no further disjunctions, and $\{\sigma \mid \sigma \models B_i \wedge B_j\} = \emptyset$ for all $i \neq j$ (B_i are disjoint).

Conditioning is thus defined as follows:

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

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 $|\Delta| \cdot |\mathrm{ddnf}\,(B)|$. 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. $\Delta \times P' \stackrel{\text{def}}{=} \{P \times P' \mid P \in \Delta\}$ (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.

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

Definition 49. The normalization a probabilistic polyhedra P_1 relative to a probabilistic polyhedron set Δ, written $\mathrm{normal}_\Delta(P_1)$, is the probabilistic polyhedron P_2 defined as follows whenever $\mathrm{M}^{\min}(\Delta) > 0$.

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

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

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

Powersets of Intervals and Octagons

Following the probabilistic extensions to the interval and octagon domains described in Section 6.4.5, 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.

6.5.2 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 50. *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 51. 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 52. 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 $P_i \in \Delta$. 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 53. 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., $\sigma_j \in \gamma_{\mathbb{C}}(L_j)$.

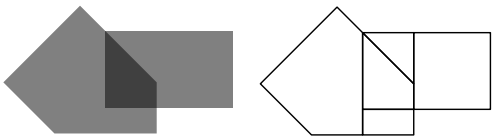


Figure 6.8: Example of a poly partition of two overlapping convex polyhedra (shaded), resulting in 5 disjoint convex polyhedra (outlined).

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

Lemma 54. $\max_{\sigma \in R} \Delta^{max}(\sigma) = \max_{\sigma} \Delta^{max}(\sigma)$ *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 naïve way possible: we maintain a bag of polyhedra, splitting any overlapping pairs, until no overlapping regions remain, as in the pseudo-code below.

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

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 55. 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 56. 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 $\Delta_3 = \Delta_2 \upharpoonright L = \{P'_i\}$. If for every $\sigma_L \in \gamma_{\mathcal{P}(\mathbb{C})}(\{C'_i\})$ with $\neg \mathit{iszero}(\Delta_2 \wedge \sigma_L)$ we have $\Delta_4 = (\Delta_2 | \sigma_L) \upharpoonright H$ and $max_{pp}(\Delta_4) \leq h$, then we write $\mathit{tsecure}_h(S, \Delta_1)$.

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 57. *Let δ be an attacker's initial belief. If $\delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta)$ and $tsecure_h(S, \Delta)$, then S is threshold secure for threshold h when evaluated with initial belief δ.*

Note that the process described in computing threshold security involves merging probabilistic polyhedra via the simplification operation (Definition 47); 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 6.6.3. 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.

6.6 Experiments

We have implemented an interpreter for the language in Figure 6.3 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 [4] (ppl-0.11.2). Counting calculations are done using the LattE [32] 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.

6.6.1 Benchmark Programs

We applied our implementation to several queries. The timings measure, for each query, the construction of a prebelief, 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, prebelief, and further details in Appendix A.

Birthday We benchmark the birthday queries described in Section 6.2:

- **bday 1** The first birthday query (Example 23).
- **bday 1+2+special** The sequence of the first two birthday queries (Example 23 and then the same code, but with *today* increased by 1) followed by the special birthday query (Example 24). Below we refer to this benchmark as the *birthday query sequence*.

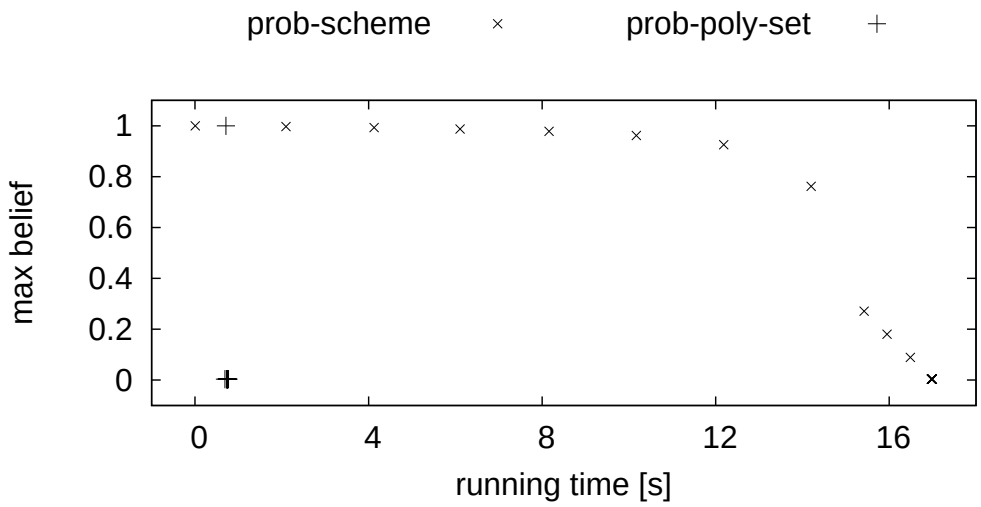
We consider *small* and *large* variants of these two queries: the former assumes the birth year ranges from 1956 to 1992, while the latter uses the range 1910 to 2010.

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.

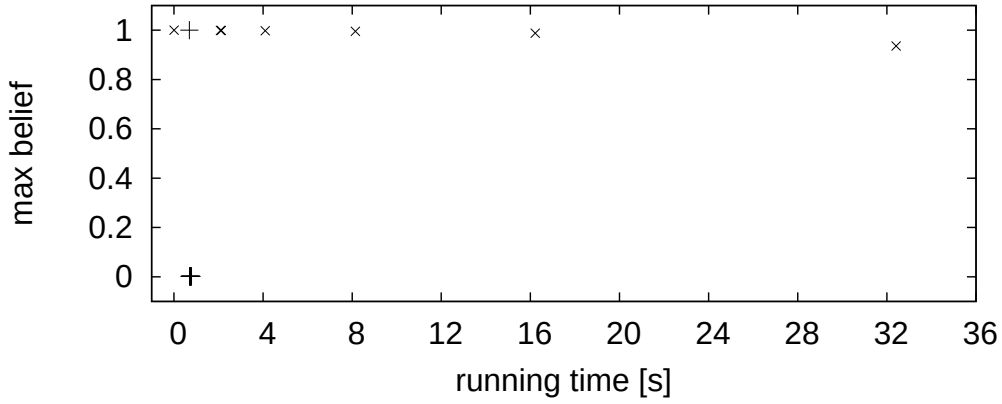
Photo This query is a direct encoding of a real targeted advertisement that Facebook includes on their information page [2]. The query itself checks whether the user is female, engaged, and is in a certain age range, indicative of interest in (wedding) photography service. There are three secret variables in this example: gender, relationship status, and birth year.

Travel This query is another adaptation of a Facebook advertisement case study [1], 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 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 6.6.4 and their full specification is shown in Appendix A.5.



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



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

Figure 6.9: Query evaluation comparison

6.6.2 Comparison to Enumeration

Figure 6.9(a) illustrates the result of running the **bday 1 (small)** query using our implementation and one using Probabilistic Scheme [92], which is capable of sound probability estimation after partial enumeration. Each × 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 6.9(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 6.10, discussed shortly.

6.6.3 Performance analysis

Figures 6.10-6.14 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 A.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 6.11. 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 6.10-6.14, 95% or more of the running time is spent in LattE, performing counting. Figure 6.15 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

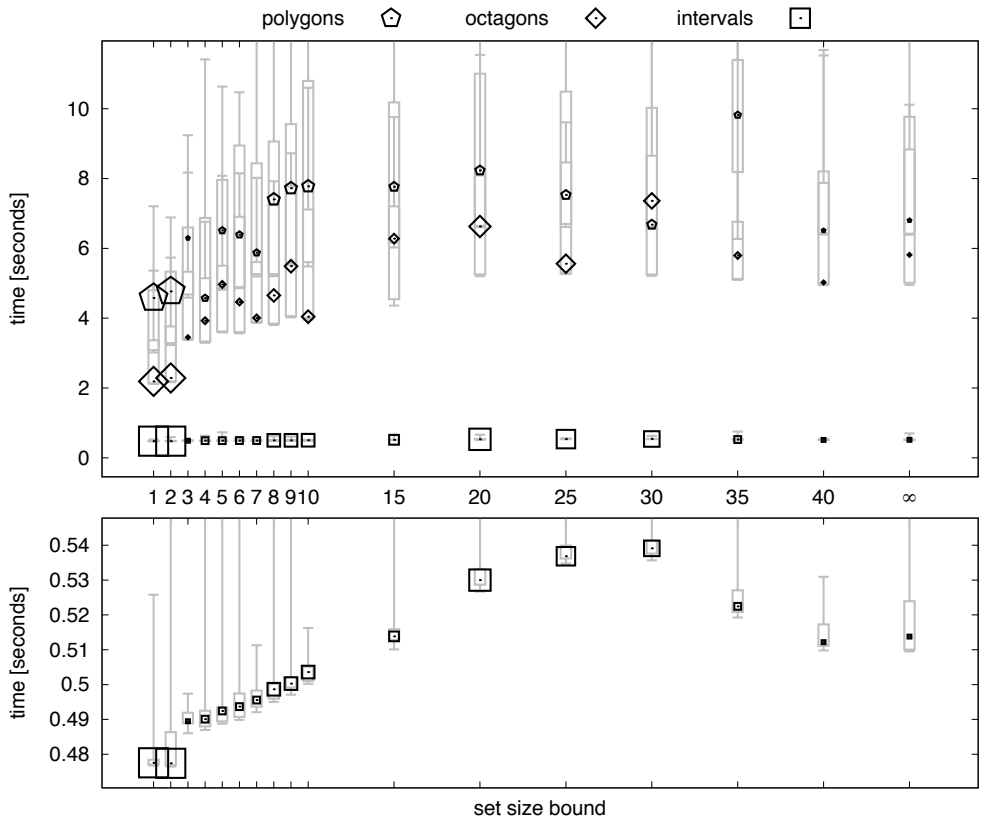


Figure 6.10: birthday query sequence benchmarks

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 6.12 and 6.14 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 A.1 in the appendix) generally show the trend that the maximum belief improves (decreases) as the region bound increases, though there are exceptions. A good example appears in Figure 6.10 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

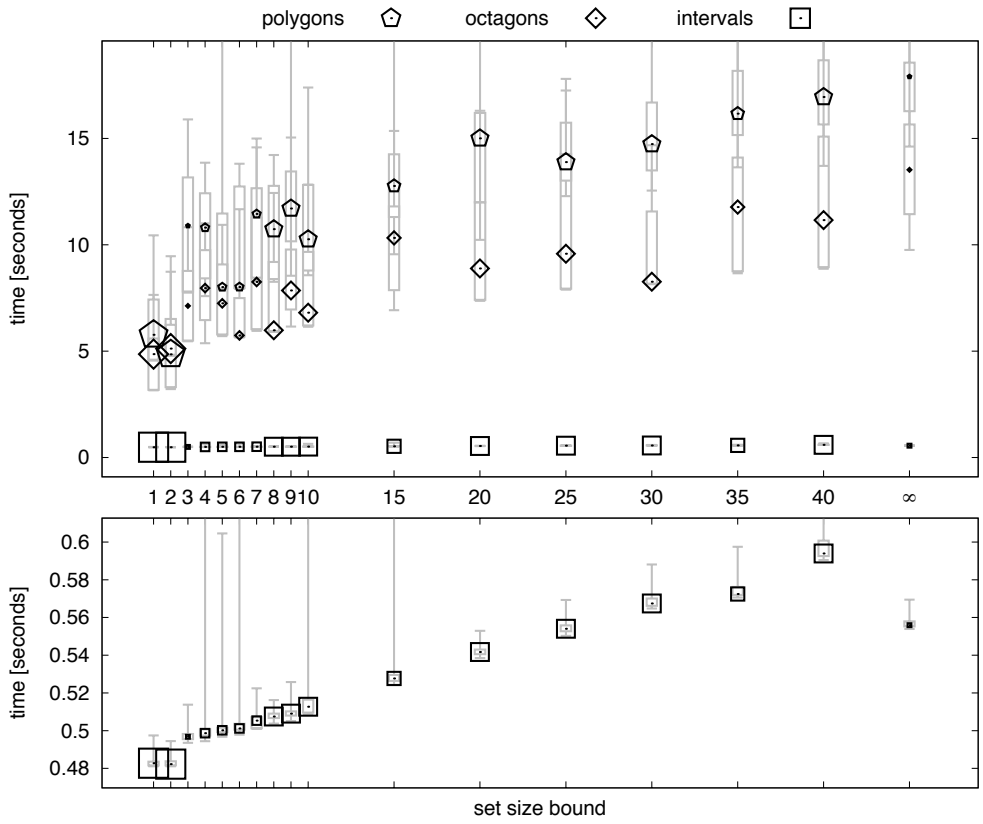


Figure 6.11: birthday (large) query sequence benchmarks

(or intervals, octagons), however, does not produce the exact belief anymore. The same occurs in Figure 6.11 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 6.10. 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 6.6.4 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. 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

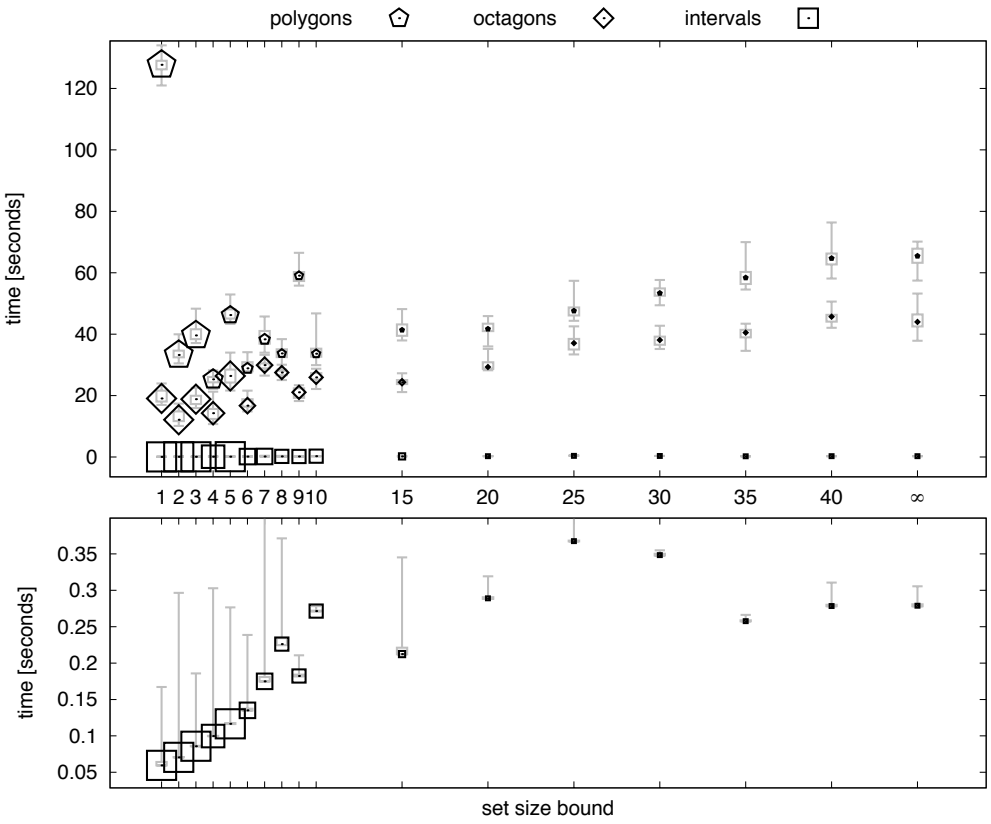


Figure 6.12: pizza query benchmarks

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 6.16. 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 6.5.1 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 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

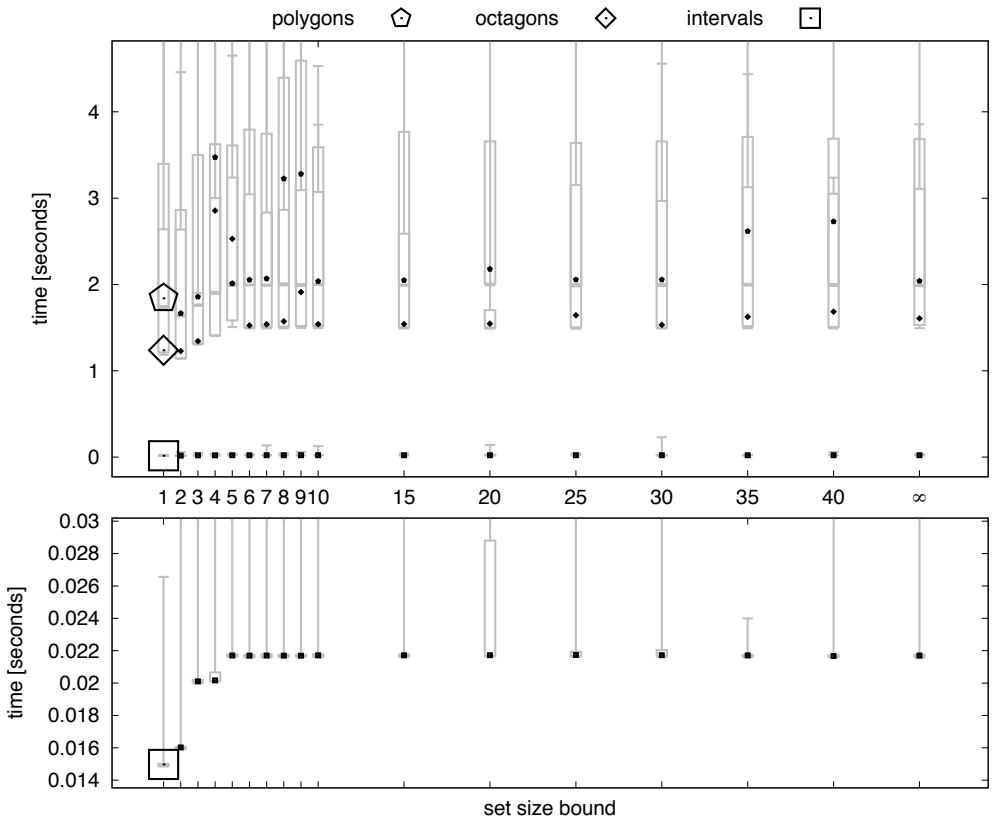


Figure 6.13: photo query benchmarks

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 6.8.

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 6.16 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 distributions, for some merging order.

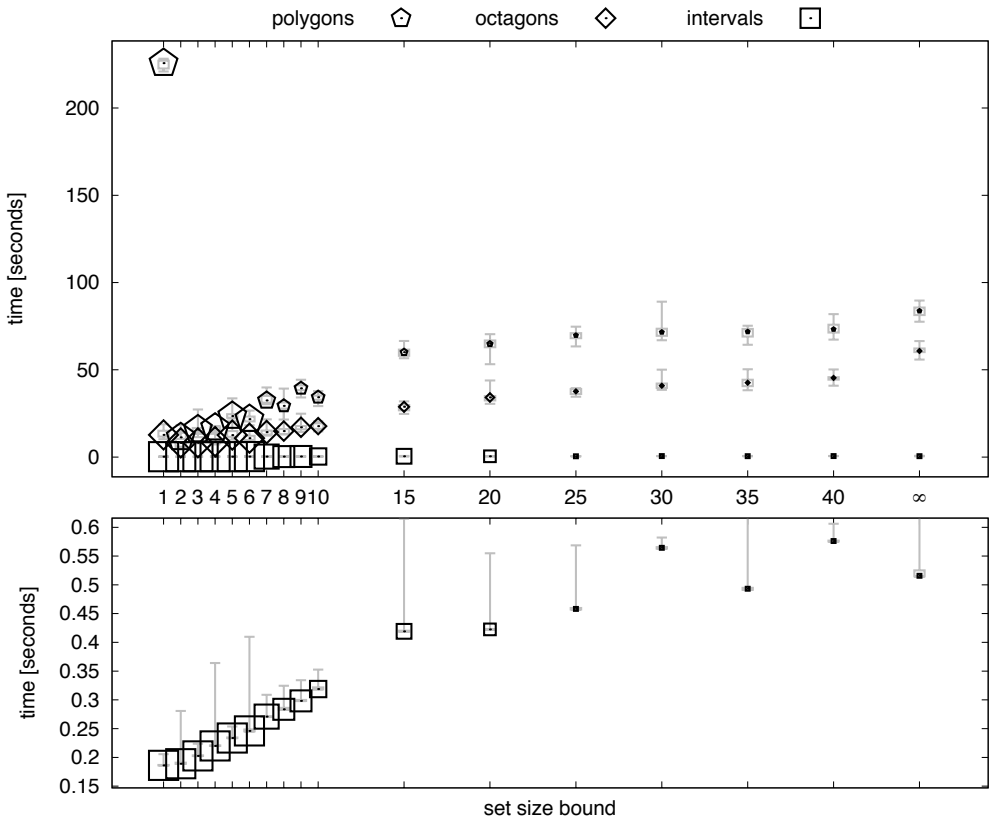


Figure 6.14: travel query benchmarks

- 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.

6.6.4 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 A.5).

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 $|x - x_1| + |y - y_1| \leq d$.

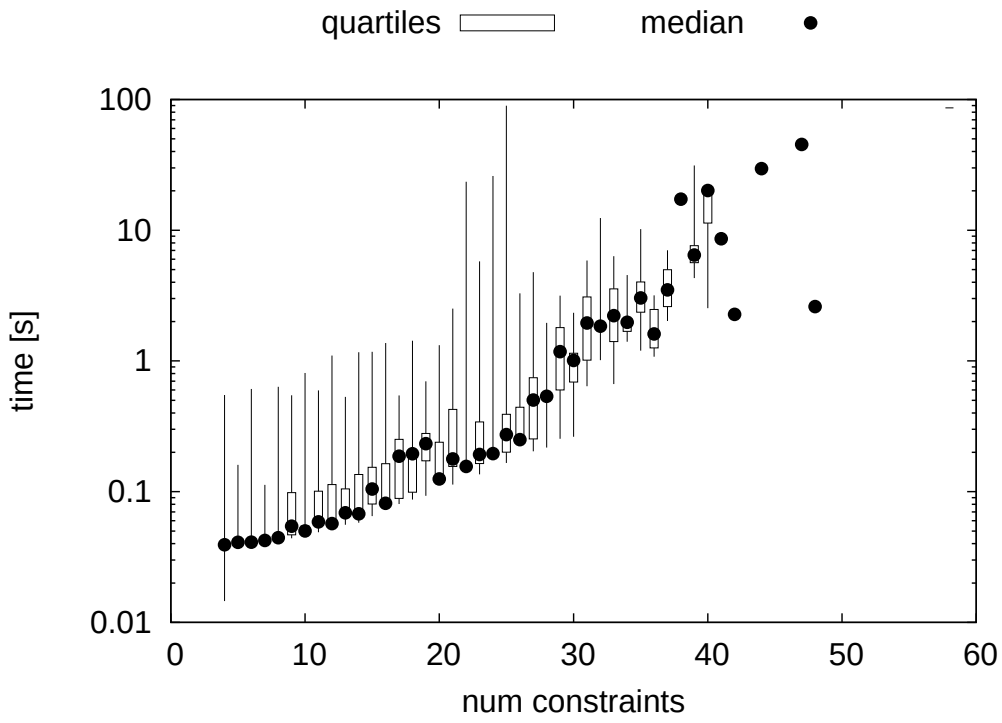


Figure 6.15: LattE benchmarks

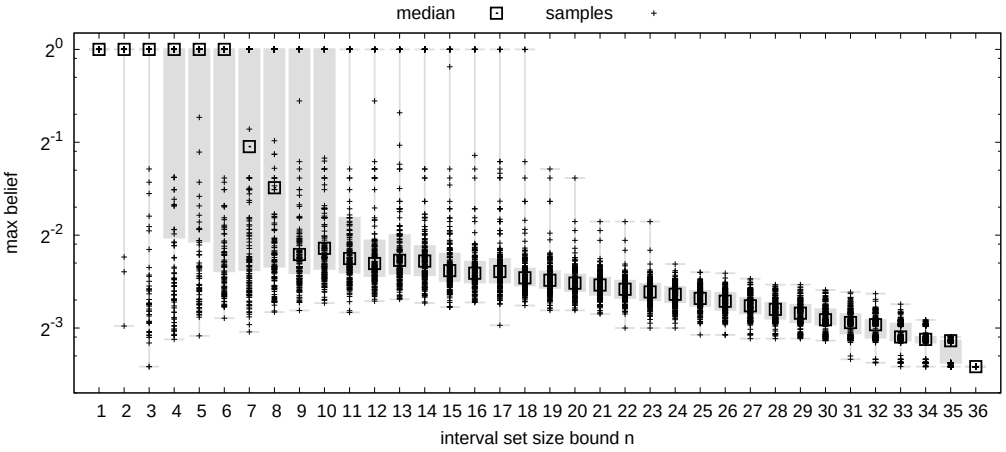


Figure 6.16: birthday query sequence precision variation

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:

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

$$-x_1 - y_1 \leq d - x - y$$

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.

6.7 Related work

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 [47, 87] or full enumeration [92] 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 [21], graphical models [78], and factor graphs [18, 89] 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 [81] gives an abstract interpretation for probabilistic programs based on over-approximating probabilities of points. Di Pierro describes abstract interpretation for probabilistic lambda calculus [36]. Smith [101] describes probabilistic abstract interpretation for verification of quantitative program properties. Cousot [31] 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.

6.8 Improving Performance

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: [21], graphical models [78], or factor graphs [18, 89]. Ideally abstract domains like used in our system and ones better suited for nominal values, could be integrated (i.e. via reduced product [29]) 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 6.16. Heuristically picking the right merge order is a known challenge in abstract interpretation-based analyses.

Chapter 7

Conclusion

In this thesis we showed how to use explicit models of adversary knowledge to address the issue of query safety in information sharing thereby addressing the main goal of this thesis: **which queries are safe?**

In Chapter 3 we showed how a user wishing to protect private information can maintain these models of adversary knowledge and determine when queries are safe to answer. This incorporates a measure of risk based on adversary knowledge, thereby providing an answer to Question 1: How does one measure information? Using the approach the user can answer queries in an on-line manner, as they come, without having to know anything about past or future queries. A user does this by actively maintaining the knowledge of the adversary and how it changes due to learning answers to queries. This addresses Question 2: How does the adversary obtain information? Finally the approach gives the user a simulatable means of refusing to answer a query in order to maintain a risk bound. This addresses Question 3: How can a user make sure that the adversary does not attain too much information?

In Chapter 5 we showed how the same approach and similar though extended answers to the three questions can apply to collaborative queries with multiple parties interested in protecting their private information, yet still performing queries over it. We showed two approaches that limit adversary knowledge in this setting, both designed to overcome the inability of the parties to directly model each other's knowledge: one based on tracking sets of adversary beliefs, and one that uses secure computations to enforce knowledge limits. We experimentally showed that though the first of these is very conservative, it can still allow safe information sharing in some cases.

In Chapter 4 we showed how the knowledge models can be applied in settings where the secrets change over time. The work provided a more nuanced answer to Questions 1 and 2 for this setting. In particular we saw how to make predictions about risks at future points in time, incorporating the intervening evolution of secrets and interactions with the adversary. We considered adaptive adversaries that can decide when to exploit their knowledge as well as decide the parameters of the queries they observe, both based on their prior observations. We showed that adaptability results in significantly higher risk that monotonically increases with time. We also showed that, counter to intuition, more frequent changes to the secret can lead to higher risk.

Finally, in Chapter 6 we presented an approach for modeling knowledge and limiting risk that is approximate but sound, letting users trade off precision (and in some sense utility) for performance. We extended abstract interpretation techniques to consider probabilities and carefully defined a sound abstract semantics and a conditioning operation. We experimentally evaluated an implementation of the approach relative to naïve probabilistic interpretation by enumeration showing how our technique is resistant to the state-space explosion problem. We also demonstrated the

precision/performance tradeoff with an evaluation of abstractions of various complexities.

7.1 Future Directions

There are two main avenues along which the work presented in this thesis can be extended and refined: modeling and implementation.

Models A big part of this thesis, particularly in Chapter 4, dealt with presenting a suitable model for some real-world information-sharing setting. We have showed in that chapter how important it is to take into account the capability of the adversary in such models but stopped short of fully accounting for the behavior of the user or defender, who was assumed to be passive. One of the outcomes we hope from our work in the future is useful lessons that can be implemented by defenders. But the restricted power of defenders in our models limits the range of scenarios we can presently explore. Integrating defender *choice* into our model brings technical difficulties but also potential for useful results.

Implementation Experimentation is a vital tool in exploring the consequences of the models we presented. As such, the computational aspects of the evaluation of our models cannot be overlooked. Our work on approximate but sound probabilistic programming has been our first attempt to addressing those computational aspects but yet is still severely limited both in terms of performance and expressibility of the programs it applies to. To incorporation of a widening operator [1] is an opportunity we have yet to take advantage of. Our restriction to linear expressions can potentially be alleviated by using linearization techniques or incorporating abstractions that sidestep the issue like bit-vectors.

7.2 Final Remarks

In general our work lacks considerations for human factors. The bigger hurdle to our proposal for personal data control may not be technical but social. Lacking proper incentives and understanding, typical computer users rarely take the effort to take security into account. Even if they do, the most strict of security guarantees can be broken by users not following directions. Related branches of research consider how best to address social issues inherent in information security (or how to address them at all). Bridging those works to our approaches on modeling adversary knowledge may reveal new opportunities.

We have motivated our work with a speculative goal for automated techniques to which we (the humans) can offload our security decisions. This work, however, can be a tool for exploration in which the humans take a big role. The scenarios we demonstrated in this thesis are only few instances of quite general models. It is our

[1] Widening is a technique in abstract interpretation that greatly aids in the analysis of loops.

hope that this thesis will make it easier for other researchers or security experts to develop information security scenarios, interpret their results, and inform the human behavior that is the source of one of the biggest problems in information security today.

Appendix A

Example queries

We provide here the queries and prebeliefs we used for the experiments in Section 6.6. The queries are described as functions from some set of inputs to some set of outputs. The exact syntax is as follows.

$$\begin{array}{l}\textsf{querydef}\ \mathit{queryname}\ \mathit{in}_1 \cdots \mathit{in}_n \rightarrow \mathit{out}_1 \cdots \mathit{out}_m\ : \\ \quad \mathit{querybody}\end{array}$$

Query definitions that follow sometimes include preprocessing statements of the form:

$$\#\textsf{define}\ x\ =\ \mathit{exp}$$

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.

$$\begin{array}{l}\textsf{query}\ \mathit{queryname}\ : \\ \quad \mathit{in}_1\ :=\ \mathit{val}_1; \\ \quad \cdots \\ \quad \mathit{in}_n\ :=\ \mathit{val}_n\end{array}$$

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.

$$\begin{array}{l}\textsf{secret}: \\ \mathit{s\_bday}\ :=\ 270\ ; \\ \mathit{s\_byear}\ :=\ 1980 \\ \\ \textsf{belief}:\end{array}$$

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

The two queries used were as follows.

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

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

The statistics described in comparison to the enumeration approach (Section 6.6.2) include the time spent processing this initial setup as well as time processing one birthday query. Figure 6.11 benchmarks two bday queries followed by a spec year query.

- A single bday query alone.

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

- Two bday queries followed by a spec query.

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

A.2 Birthday (large)

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

secret :

$s\_bday := 270$;
$s\_byear := 1980$

belief :

uniform s_bday 0 364 ;
uniform s_byear 1910 2010

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.2.1 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 $pizza$: $\rightarrow$ out

$$
\begin{array}{l}
\#\mathsf{define}\ age\ =\ 2010 - s\_birth\_year \\
\#\mathsf{define}\ lr\_lat\ =\ 38967884 \\
\#\mathsf{define}\ ul\_lat\ =\ 39003178 \\
\#\mathsf{define}\ lr\_long\ =\ -76958199 \\
\#\mathsf{define}\ ul\_long\ =\ -76925926 \\
\mathsf{if}\ s\_in\_school\_type \geq 4\ \mathsf{then} \\
\quad in\_school\ :=\ 1 \\
\mathsf{else} \\
\quad in\_school\ :=\ 0\ ; \\
\mathsf{if}\ age \geq 18\ \wedge\ age \leq 28\ \mathsf{then} \\
\quad age\_criteria\ :=\ 1 \\
\mathsf{else} \\
\quad age\_criteria\ :=\ 0\ ; \\
\mathsf{if}\ s\_address\_lat \leq ul\_lat \\
\quad \wedge\ s\_address\_lat \geq lr\_lat \\
\quad \wedge\ s\_address\_long \geq lr\_long \\
\quad \wedge\ s\_address\_long \leq ul\_long \\
\mathsf{then} \\
\quad in\_box\ :=\ 1 \\
\mathsf{else} \\
\quad in\_box\ :=\ 0\ ; \\
\mathsf{if}\ (in\_school = 1\ \vee\ age\_criteria = 1) \\
\quad \wedge\ in\_box = 1\ \mathsf{then} \\
\quad out\ :=\ 1 \\
\mathsf{else} \\
\quad out\ :=\ 0
\end{array}
$$

A.3 Photo

The photo query is a direct encoding of a case study that Facebook includes on their advertising information page [2]. 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.

$$\mathsf{secret}:$$

$s\_birth\_year := 1983$;
$s\_gender := 0$;
$s\_relationship\_status := 0$

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 $cm\_advert : \rightarrow out$

#define $age = 2010 - s\_birth\_year$
if $age \geq 24 \wedge age \leq 30$ then
  $age\_sat := 1$
else
  $age\_sat := 0$;
if $s\_gender = 1$
 $\wedge\, s\_relationship\_status = 3$
 $\wedge\, age\_sat = 1$ then
  $out := 1$
else
  $out := 0$

A.4 Travel

This example is another Facebook advertising case study [1]. 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
```

A.5 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
  $who\_closer := 1$

A.6 Benchmark Results

Table A.1 tabulates performance results for all of the benchmark programs, for each possible base domain (intervals, octagons, and polyhedra labeled □, ◇, and ⬠ 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 A.1: Query evaluation benchmarks

Appendix B

Soundness proofs for $\mathbb{P}$

B.1 Projection

The proof of projection relies heavily on splitting up the support of a distribution into equivalence classes based on the states they project to. We will have $\sigma, \sigma' \in \text{support}(\delta)$ belonging to the same equivalence class iff $\sigma \upharpoonright V = \sigma' \upharpoonright V$. The details are formalized in the following definition.

Definition 58. *Equivalence classes under projection.*

- $[\sigma_V]_\delta^V$ is an equivalence class of elements of $\text{support}(\delta)$ that project to σ_V (when projected to variables V). Formally, $[\sigma_V]_\delta^V \stackrel{\text{def}}{=} \{\sigma \in \text{support}(\delta) \mid \sigma \upharpoonright V = \sigma_V\}$.
- $\overline{[\sigma_V]}_\delta^V$ is a subset of $\text{support}(\delta)$ that project to anything but σ_V or formally $\overline{[\sigma_V]}_\delta^V \stackrel{\text{def}}{=} \{\sigma \in \text{support}(\delta) \mid \sigma \upharpoonright V \neq \sigma_V\}$.
- $[\sigma_V]_C^V$ is a subset of $\gamma_{\mathbb{C}}(C)$ that project to σ_V (when projected to variables V). Formally, $[\sigma_V]_C^V \stackrel{\text{def}}{=} \{\sigma \in \gamma_{\mathbb{C}}(C) \mid \sigma \upharpoonright V = \sigma_V\}$.
- $\overline{[\sigma_V]}_C^V$ is a subset of $\gamma_{\mathbb{C}}(C)$ that project to anything but σ_V or formally $\overline{[\sigma_V]}_C^V \stackrel{\text{def}}{=} \{\sigma \in \gamma_{\mathbb{C}}(C) \mid \sigma \upharpoonright V \neq \sigma_V\}$.

Remark 59. Let $V \subseteq V' \subseteq \mathit{fv}(\delta)$, $\sigma \in \text{support}(\delta)$, and $\sigma_V, \sigma'_V \in \text{support}(\delta \upharpoonright V)$.

(i) $(\sigma \upharpoonright V') \upharpoonright V = \sigma \upharpoonright V$

The sets of $\left\{[\sigma_V]_\delta^V\right\}_{\sigma_V \in \text{support}(\delta \upharpoonright V)}$ form a partition of $\text{support}(\delta)$, equivalently the following two claims.

(ii) $\mathit{support}(\delta) = \bigcup_{\sigma_V \in \mathit{support}(\delta \upharpoonright V)} [\sigma_V]_\delta^V$

(iii) $[\sigma_V]_\delta^V \cap [\sigma'_V]_\delta^V = \emptyset$ *whenever* $\sigma_V \neq \sigma'_V$

Likewise, for any $\sigma_V \in \text{support}(\delta \upharpoonright V)$, the sets of $\left\{[\sigma_{V'}]_\delta^{V'}\right\}_{\sigma_{V'} \in [\sigma_V]_{\delta \upharpoonright V'}^V}$ form a partition of $[\sigma_V]_\delta^V$, implying also the following claim.

(iv) $[\sigma_V]_\delta^V = \bigcup_{\sigma_{V'} \in [\sigma_V]_{\delta \upharpoonright V'}^V} [\sigma_{V'}]_\delta^{V'}$

The equivalence classes in terms of the concrete support sets as related to the abstract support sets are expressed in the following manner.

(v) $[\sigma_V]_\delta^V \subseteq [\sigma_V]_C^V$ *and* $\overline{[\sigma_V]}_\delta^V \subseteq \overline{[\sigma_V]}_C^V$ *whenever* $\mathit{support}(\delta) \subseteq \gamma_{\mathbb{C}}(C)$

Finally, the concrete projection operation can be rewritten in terms of the equivalence classes, a fact we will repeatedly use in the proofs to follow without explicitly stating it.

(vi) $\delta \restriction V = \lambda\sigma_V.\ \sum_{\sigma\in[\sigma_V]^V_\delta} \delta(\sigma)$

Proof. All of these are merely expansions of the various definitions involved. Note that the two parts of Remark 59(v) are not contradictory as $[\sigma_V]^V_\delta$ and $\overline{[\sigma_V]}^V_\delta$ are not set complements of each other when viewed as subsets of $\gamma_{\mathbb{C}}(C)$, though they are complements when viewed as subsets of support (δ). □

Lemma 60 (Conservation of Mass)**.** *If* $V \subseteq fv(\delta)$ *then* $\|\delta\| = \|\delta \restriction V\|$.

Proof. Let us consider the terms of the projected mass sum.

$$\begin{aligned}\|\delta \restriction V\| &= \sum_{\sigma_V\in\text{support}(\delta\restriction V)} \left(\sum_{\sigma\in[\sigma_V]^V_\delta} \delta(\sigma)\right)\\ &= \sum_{\sigma\in\text{support}(\delta)} \delta(\sigma)\\ &= \|\delta\|\end{aligned}$$

The terms in the double sum are the same as those in the single sum as all terms of the first are accounted for in the second due to Remark 59(ii) and none are double counted due to Remark 59(iii). □

Definition 61. *Concrete forget* can be defined in terms of a projection to all but one variable. That is, $\mathrm{f}_x(\delta) \stackrel{\text{def}}{=} \delta \restriction fv(\delta) - \{x\}$. Also, $\mathrm{f}_{x_1,\cdots,x_n}(\delta) \stackrel{\text{def}}{=} \mathrm{f}_{x_2,\cdots,x_n}(\mathrm{f}_{x_1}(\delta))$.

The correspondence between repeated concrete forget and a projection involving removal of more than one variable will be demonstrated shortly.

Lemma 62 (Order of Projection)**.** *If* $V \subseteq V' \subseteq fv(\delta)$ *then* $(\delta \restriction V') \restriction V = \delta \restriction V$.

Proof. Let $\sigma_V \in \delta \restriction V$.

$$\begin{aligned}((\delta \restriction V') \restriction V)(\sigma_V) &= \sum_{\sigma_{V'}\in[\sigma_V]^V_{\delta\restriction V'}} \left(\sum_{\sigma\in[\sigma_{V'}]^{V'}_\delta} \delta(\sigma)\right) &(62.1)\\ &= \sum_{\sigma\in\bigcup_{\sigma_{V'}\in[\sigma_V]^V_{\delta\restriction V'}}} \delta(\sigma) &(62.2)\\ &= \sum_{\sigma\in[\sigma_V]^V_\delta} \delta(\sigma) &(62.3)\\ &= (\sigma \restriction V)(\sigma_V) &(62.4)\end{aligned}$$

The collapse of the double sums on (62.1) to (62.2) is due to the correspondence between the terms of the double sum and the single sum due to Remark 59(ii) and Remark 59(iii). The equality of the union of equivalence classes, (62.2) to (62.3) is due to Remark 59(iv). □

Corollary 63. $\delta \restriction V = \mathrm{f}_{x_1,\cdots,x_n}(\delta)$ where $fv(\delta) - V = \{x_1, \cdots, x_n\}$.

Proof. Let us show this by induction on the size of $\overline{V} \stackrel{\text{def}}{=} fv(\delta) - V$. When $\left|\overline{V}\right| = 0$ or $\left|\overline{V}\right| = 1$, the claim holds vacuously or by definition of concrete forget, respectively.

Let us assume the claim for $\left|\overline{V}\right| = m - 1 < n$ and consider the case when $\left|\overline{V}\right| = m \leq n$.

$$\begin{aligned}
\delta \restriction V &= (\delta \restriction V \cup \{x_1\}) \restriction V && [\text{ by Lemma 62 }] \\
&= \mathrm{f}_{x_1}(\delta) \restriction V \\
&= \mathrm{f}_{x_2,\cdots,x_m}(\mathrm{f}_{x_1}(\delta)) && [\text{ by induction }] \\
&= \mathrm{f}_{x_1,\cdots,x_m}(\delta)
\end{aligned}$$

Thus, by induction, the claim holds for $m = n$. □

Remark 64 (Counting Variations)**.** Two simple counting arguments are required for the further proofs.

(i) If m objects are distributed fully into two bins, with one of the bins having space for no more than a objects, then the other must have at least $m - a$ objects in it.

(ii) If m objects are to be packed into bins of sizes $a_1, \cdots, a_n$, with $\sum_i a_i \geq m$, the least number of bins that can be used to fit all m objects is greater or equal to $\lceil m/a^* \rceil$ where $a^* \geq \max_i a_i$.

Proof. Part (i) is immediate. For part (ii), consider some optimal set of bins used to pack the m objects. This set of bins would also let one pack m items assuming each bin had space for exactly a^* objects as this is an upper bound on the size of each bin. Thus the space of solutions to the original packing problem is a subset of the space of solutions to the altered packing problem where all bins are increased to fit a^* items. Thus the solution for the original cannot use fewer bins than the optimal solution for the altered problem. For this alternate problem, the minimum number of bins used to pack all m items is exactly $\lceil m/a^* \rceil$ by a generalization of the pigeonhole principle. □

Lemma 28 (Soundness of Forget)**.** *If* $\delta \in \gamma_{\mathbb{P}}(P)$ *then* $\mathrm{f}_y(\delta) \in \gamma_{\mathbb{P}}(\mathrm{f}_y(P))$.

Proof. Let $\delta \in \gamma_{\mathbb{P}}(P)$, $V = fv(\delta) - \{y\}$, and $\delta_2 = \delta \restriction V$. By assumption δ has the following properties.

$$\text{support}\,(\delta) \subseteq \gamma_{\mathbb{C}}(C) \tag{64.1}$$
$$\mathrm{s}^{\min} \leq |\text{support}\,(\delta)| \leq \mathrm{s}^{\max} \tag{64.2}$$
$$\mathrm{m}^{\min} \leq \|\delta\| \leq \mathrm{m}^{\max} \tag{64.3}$$
$$\forall \sigma \in \text{support}\,(\delta)\ .\ \mathrm{p}^{\min} \leq \delta(\sigma) \leq \mathrm{p}^{\max} \tag{64.4}$$

Let $P_2 = \mathrm{f}_y(P)$. P_2 thus has the following properties.

$$C_2 = \mathrm{f}_y(C) \tag{64.5}$$
$$\mathrm{p}_2^{\min} = \mathrm{p}^{\min} \cdot \max\left[\mathrm{h}_y^{\min} - \left(\#(C) - \mathrm{s}^{\min}\right), 1\right] \tag{64.6}$$
$$\mathrm{p}_2^{\max} = \mathrm{p}^{\max} \cdot \min\left[\mathrm{h}_y^{\max}, \mathrm{s}^{\max}\right] \tag{64.7}$$
$$\mathrm{s}_2^{\min} = \lceil \mathrm{s}^{\min}/\mathrm{h}_y^{\max} \rceil \tag{64.8}$$
$$\mathrm{s}_2^{\max} = \min\left[\#(C_2), \mathrm{s}^{\max}\right] \tag{64.9}$$
$$\mathrm{m}_2^{\min} = \mathrm{m}^{\min} \tag{64.10}$$
$$\mathrm{m}_2^{\max} = \mathrm{m}^{\max} \tag{64.11}$$

The quantities $\mathrm{h}_y^{\min}$ and $\mathrm{h}_y^{\max}$ are defined to exhibit the following properties.

$$\mathrm{h}_y^{\min} \leq \min_{\sigma_V \in \gamma_{\mathbb{C}}(C_2)} \left|[\sigma_V]_C^V\right| \tag{64.12}$$
$$\mathrm{h}_y^{\max} \geq \max_{\sigma_V \in \gamma_{\mathbb{C}}(C_2)} \left|[\sigma_V]_C^V\right| \tag{64.13}$$

To show that $\delta_2 \in \gamma_{\mathbb{P}}(\mathrm{f}_y(P))$ we need to show the following.

$$\text{support}\,(\delta_2) \subseteq \gamma_{\mathbb{C}}(C_2) \tag{64.14}$$
$$\mathrm{s}_2^{\min} \leq |\text{support}\,(\delta_2)| \leq \mathrm{s}_2^{\max} \tag{64.15}$$
$$\mathrm{m}_2^{\min} \leq \|\delta_2\| \leq \mathrm{m}_2^{\max} \tag{64.16}$$
$$\forall \sigma_V \in \text{support}\,(\delta_2)\ .\ \mathrm{p}_2^{\min} \leq \delta_2(\sigma_V) \leq \mathrm{p}_2^{\max} \tag{64.17}$$

Let us show each of these in turn.

Claim 64.14 – Support. Let $\sigma_V \in \text{support}\,(\delta_2)$. Thus $\delta_2(\sigma_V) = \sum_{\sigma \in [\sigma_V]_\delta^V} \delta(\sigma) > 0$ so there exists $\sigma \in [\sigma_V]_\delta^V$ with $\delta(\sigma) > 0$. So $\sigma \in \text{support}\,(\delta)$. Therefore, by (64.1), $\sigma \in \gamma_{\mathbb{C}}(C)$, therefore $\sigma_V \in \gamma_{\mathbb{C}}(C_2)$ by definition of polyhedron forget. Therefore $\text{support}\,(\delta_2) \subseteq \gamma_{\mathbb{C}}(C_2)$. □

Claim 64.15 Support points. First let us show the following claim.

$$\max_{\sigma_V \in \text{support}(\delta_2)} \left|[\sigma_V]_\delta^V\right| \leq \mathrm{h}_y^{\max} \tag{64.18}$$

By construction of $\mathrm{h}_y^{\max}$, we have $\mathrm{h}_y^{\max} \geq \max_{\sigma_V \in \gamma_{\mathbb{C}}(C_2)} \left|[\sigma_V]_C^V\right|$. Now, $\text{support}\,(\delta_2) \subseteq \gamma_{\mathbb{C}}(C_2)$ by (64.14). Also for any $\sigma_V \in \text{support}\,(\delta_2)$, we have $[\sigma_V]_\delta^V \subseteq [\sigma_V]_C^V$ by Remark 59(v). Therefore $\max_{\sigma_V \in \gamma_{\mathbb{C}}(C_2)} \left|[\sigma_V]_C^V\right| \geq \max_{\sigma_V \in \text{support}(\delta_2)} \left|[\sigma_V]_C^V\right| \geq \max_{\sigma_V \in \text{support}(\delta)} \left|[\sigma_V]_\delta^V\right|$. Thus concluding $\mathrm{h}_y^{\max} \geq \max_{\sigma_V \in \text{support}(\delta_2)} \left|[\sigma_V]_\delta^V\right|$.

Consider the elements of support (δ) as they map via state projection to elements of support (δ_2). Let us view the elements of the later as bins, with the elements of the former as objects to pack into the bins. By (64.18), we know no bin has more than $\mathrm{h}_y^{\max}$ objects, thus we can apply Remark 64 to conclude there are at least $\lceil |\text{support}\,(\delta)|\,/\mathrm{h}_y^{\max}\rceil$ non-empty bins, or in other words, $|\text{support}\,(\delta_2)| \geq \lceil |\text{support}\,(\delta)|\,/\mathrm{h}_y^{\max}\rceil$. This is itself at least as large as $\lceil \mathrm{s}^{\min}/\mathrm{h}_y^{\max}\rceil = \mathrm{s}_2^{\min}$ by (64.2). Therefore $|\text{support}\,(\delta_2)| \geq \mathrm{s}_2^{\min}$.

For the other side of the inequality, note that the number of bins used, or $|\text{support}\,(\delta_2)|$ cannot exceed $|\text{support}\,(\delta)| \leq \mathrm{s}^{\max}$ itself. It also cannot exceed $|\gamma_{\mathbb{C}}(C_2)| = \#(C_2)$ given (64.14). Therefore $\text{support}\,(\delta_2) \leq \min\left[\#(C_2), \mathrm{s}^{\max}\right]$, concluding requirement (64.15). □

Claim 64.16 Mass. This requirement holds trivially due to Lemma 60 and assumption (64.3). □

Claim 64.17 Probability. Let us first show the following claim.

$$\min_{\sigma_V \in \text{support}(\delta_2)} \left|[\sigma_V]_\delta^V\right| \geq \mathrm{h}_y^{\min} + \mathrm{s}^{\min} - \#(C) \tag{64.19}$$

Let $\sigma_V \in \text{support}\,(\delta_2)$. Let us consider the size of $\overline{[\sigma_V]_\delta^V}$.

$$\begin{aligned}
\left|\overline{[\sigma_V]_\delta^V}\right| &\leq \left|\overline{[\sigma_V]_C^V}\right| && [\text{ by Remark 59(v) }]\\
&= \#(C) - \left|[\sigma_V]_C^V\right| \\
&\leq \#(C) - \min_{\tau_V \in \gamma_{\mathbb{C}}(C_2)} \left|[\tau_V]_C^V\right| \\
&\leq \#(C) - \mathrm{h}_y^{\min} && [\text{ by (64.12) }]
\end{aligned}$$

Let us view now the elements of support (δ) as mapping (via projection) into two bins, $[\sigma_V]_\delta^V$ and $\overline{[\sigma_V]_\delta^V}$. By the argument above, we know the second bin cannot hold more than $\#(C) - \mathrm{h}_y^{\min}$ elements, thus, by Remark 64 (i), it must be the case that the first bin contains at least $|\text{support}\,(\delta)| - \left(\#(C) - \mathrm{h}_y^{\min}\right)$ elements. This itself is no smaller than $\mathrm{s}^{\min} - \#(C) + \mathrm{h}_y^{\min}$ by (64.2). Therefore $\left|[\sigma_V]_\delta^V\right| \geq \mathrm{s}^{\min} - \#(C) + \mathrm{h}_y^{\min}$ and thus claim (64.19) holds.

Consider now $\sigma_V \in \text{support}\,(\delta_2)$. By (64.4) and the concrete projection definition, it must be the case that $\delta_2(\sigma_V) = \sum_{\sigma \in [\sigma_V]_\delta^V} \delta(\sigma) \geq \mathrm{p}^{\min}$. Also,

$$\begin{aligned}
\delta_2(\sigma_V) &= \sum_{\sigma \in [\sigma_V]_\delta^V} \delta(\sigma) \\
&\geq \sum_{\sigma \in [\sigma_V]_\delta^V} \mathrm{p}^{\min} && [\text{ by (64.4) }]
\end{aligned}$$

$$
\begin{aligned}
&= \left|[\sigma_V]_\delta^V\right| \cdot \mathrm{p}^{\min} \\
&\geq \left(\mathrm{h}_y^{\min} + \mathrm{s}^{\min} - \#(C)\right) \cdot \mathrm{p}^{\min} && [\text{ by (64.19) }]
\end{aligned}
$$

Therefore, $\delta_2(\sigma_V) \geq \mathrm{p}^{\min} \cdot \min\left[1, \mathrm{h}_y^{\min} + \mathrm{s}^{\min} - \#(C)\right] = \mathrm{p}_2^{\min}$, concluding one inequality of the last condition.

For the other inequality, let us once more consider a general $\sigma_V \in \text{support}\,(\delta_2)$.

$$
\begin{aligned}
\delta_2(\sigma_V) &= \sum_{\sigma \in [\sigma_V]_\delta^V} \delta(\sigma) \\
&\leq \sum_{\sigma \in [\sigma_V]_\delta^V} \mathrm{p}^{\max} && [\text{ by (64.4) }] \\
&= \left|[\sigma_V]_\delta^V\right| \cdot \mathrm{p}^{\max} \\
&\leq \mathrm{h}_y^{\max} \cdot \mathrm{p}^{\max} && [\text{ by (64.18) }]
\end{aligned}
$$

Since $[\sigma_V]_\delta^V \subseteq \text{support}\,(\delta)$, we have $\left|[\sigma_V]_\delta^V\right| \leq |\text{support}\,(\delta)| \leq \mathrm{s}^{\max}$ (by (64.2)). Thus we can also bound $\delta_2(\sigma_V)$ by $\mathrm{s}^{\max}\cdot\mathrm{p}^{\max}$. Therefore, $\delta_2(\sigma_V) \leq \mathrm{p}^{\max}\cdot\min\left[\mathrm{h}_y^{\max}, \mathrm{s}^{\max}\right]$, completing the last claim. □

□

Lemma 65 (Soundness of Projection)**.** *If $\delta \in \gamma_{\mathbb{P}}(P)$ then $\delta \upharpoonright V \in \gamma_{\mathbb{P}}(P \upharpoonright V)$.*

Proof. Let us show this by induction on the size of $\overline{V} \stackrel{\text{def}}{=} fv(\delta) - V$. When $\left|\overline{V}\right| = 0$ there is no projection to be done, when $\left|\overline{V}\right| = 1$, the claim holds by Lemma 28. Let us assume the claim holds for $\left|\overline{V}\right| = n-1$ and look at the case where $\left|\overline{V}\right| = n$.

Let us write $\overline{V} = \{x_1, \cdots, x_n\}$. Thus $\delta \upharpoonright V = \mathrm{f}_{x_1,\cdots,x_n}(\delta)$ by Corollary 63. By definition of forget, we also have $\mathrm{f}_{x_1,\cdots,x_n}(\delta) = \mathrm{f}_{x_2,\cdots,x_n}(\mathrm{f}_{x_1}(\delta))$ and $\mathrm{f}_{x_1,\cdots,x_n}(P) = \mathrm{f}_{x_2,\cdots,x_n}(\mathrm{f}_{x_1}(P))$. By Lemma 28, we know that $\mathrm{f}_{x_1}(\delta) \in \gamma_{\mathbb{P}}(\mathrm{f}_{x_1}(P))$, therefore, by induction, $\delta \upharpoonright V = \mathrm{f}_{x_2,\cdots,x_n}(\mathrm{f}_{x_1}(\delta)) \in \mathrm{f}_{x_2,\cdots,x_n}(\mathrm{f}_{x_1}(P)) = P \upharpoonright V$. □

B.2 Assignment

We begin with some useful notation.

Notation 66. Let σ be a state, E be an expression, x be a variable, $S \subseteq \mathbf{State}$, $V \subseteq \mathbf{Var}$.

- $\sigma\,[x \to E] \stackrel{\text{def}}{=} \sigma\,[x \to [\![E]\!]\sigma]$
- $S\,[x \to E] \stackrel{\text{def}}{=} \{\sigma\,[x \to E] \mid \sigma \in S\}$.
- $S \upharpoonright V \stackrel{\text{def}}{=} \{\sigma \upharpoonright V \mid \sigma \in S\}$

Definition 67. A state σ is *feasible* for $x \to E$ iff $\sigma \in \mathbf{State}[x \to E]$. We will say that σ is merely feasible if the assignment is clear from the context.

Definition 68. $t_{x \to E}$ is the function from **State** to feasible states (for $x \to E$) defined by $t_{x \to E} : \sigma \mapsto \sigma[x \to E]$

Definition 69. The inverted equivalence class for σ under assignment $x \to E$ is the set of states that map to σ. We define two varieties, one over all possible states and one for just the states in the support of a distribution.

- $\langle \sigma \rangle^{x \to E} \stackrel{\text{def}}{=} \{\tau \mid \tau[x \to E] = \sigma\}$
- $\langle \sigma \rangle_{\delta}^{x \to E} \stackrel{\text{def}}{=} \{\tau \in \text{support}(\delta) \mid \tau[x \to E] = \sigma\}$

Note that σ is feasible iff $\langle \sigma \rangle^{x \to E} \neq \emptyset$.

Definition 70. An assignment $x \to E$ is invertible iff $t_{x \to E}$ is invertible. We will denote $t_{x \to E}^{-1}$ as the inverse of $t_{x \to E}$, if it is invertible. The invertability of $t_{x \to E}$ is characterized by the existence of the inverse, having the property, that for every $\sigma \in \mathbf{State}$, we have $t_{x \to E}^{-1}(t_{x \to E}(\sigma)) = \sigma$. Equivalently, for every feasible state σ, $t_{x \to E}\left(t_{x \to E}^{-1}(\sigma)\right) = \sigma$.

We can also characterize invertability via inverted equivalence classes. $x \to E$ is invertible iff for every feasible σ, $\left|\langle \sigma \rangle^{x \to E}\right| = 1$.

We will say E is invertible if the variable is clear from the context.

Note that since $t_{x \to E}$ only changes the x component of a state, the inverse, $t_{x \to E}^{-1}$, also only changes the x component, if the inverse exists. This doesn't mean, however, that the inverse can be represented by an assignment of some E' to x. Furthermore, since our language for expressions lacks division and non-integer constants, no assignment's inverse can be represented by an assignment.

Definition 71. The expression E is *integer linear* iff $E = n_1 \times x_1 + \cdots + n_m \times x_m$, where n_i are integer constants, and x_i are variables. We assume that all the variables in a given context are present. We will generally use x_i and n_i to refer to the contents of a integer linear expression.

From now on, we will assume all expressions E are integer linear. Programs containing non-linear expressions are just not handled by our system at this stage and linear expressions not fully specified are equivalent to integer linear expressions with $n_i = 0$ for variables unused in the original expression.

Lemma 72. *$x_1 \to E$ is non-invertible iff $n_1 = 0$. In other words, $x_1 \to E$ is non-invertible iff E doesn't depend on x_1.*

Proof. ($\Rightarrow$) Assume otherwise. Thus E is non-invertible but $n_1 \neq 0$. So we have a feasible state σ with $\left|\langle \sigma \rangle^{x_1 \to E}\right| \neq 1$. Since feasible states have non empty inverted equivalence sets, it must be that $\left|\langle \sigma \rangle^{x_1 \to E}\right| \geq 2$. So let $\tau, \tau' \in \langle \sigma \rangle^{x_1 \to E}$ with $\tau \neq \tau'$. So $\tau[x_1 \to E] = \tau'[x_1 \to E] = \sigma$. Since assignment to x_1 doesn't change the state other than in its value of x_1, τ and τ' can only differ in their value for x_1.

But since τ and τ' are identical after the assignment, we have,

$$\begin{aligned}\tau\left[x_1 \to E\right](x_1) &= n_1\tau(x_1) + n_2\tau(x_2) + \cdots + n_m\tau(x_m)\\ &= n_1\tau(x_1) + n_2\tau'(x_2) + \cdots + n_m\tau'(x_m)\\ &= n_1\tau'(x_1) + n_2\tau'(x_2) + \cdots + n_m\tau'(x_m)\\ &= \tau'\left[x_1 \to E\right](x_1)\end{aligned}$$

Canceling out the common $\tau'(x_i)$ terms, we have $n_1\tau(x_1) = n_1\tau'(x_1)$ and since $n_1 \neq 0$, we conclude $\tau(x_1) = \tau'(x_1)$, contradicting $\tau \neq \tau'$.

($\Leftarrow$) Let σ be a feasible state and let $\tau \in \langle\sigma\rangle^{x_1 \to E}$. Let $\tau' = \tau\left[x_1 \to \tau(x_1) + 1\right]$. Since E doesn't depend on x_1, we have $[\![E]\!]\tau = [\![E]\!]\tau'$ and therefore $\tau'\left[x_1 \to E\right] = \tau\left[x_1 \to E\right] = \sigma$ and so we have $\tau, \tau' \in \langle\sigma\rangle^{x_1 \to E}$ with $\tau \neq \tau'$, therefore E is non-invertible. □

Lemma 73. *Assume $x \to E$ is non-invertible. σ is feasible iff $\sigma\left[x \to E\right] = \sigma$.*

Proof. ($\Rightarrow$) Let σ be feasible. Thus $\sigma = \tau\left[x \to E\right]$ for some $\tau \in \mathbf{State}$. Since E doesn't depend on x by Lemma 72, we have $\left(\tau\left[x \to E\right]\right)\left[x \to E\right] = \tau\left[x \to E\right] = \sigma$. So $\sigma\left[x \to E\right] = \sigma$.

($\Leftarrow$) Assume $\sigma\left[x \to E\right] = \sigma$. Thus $\sigma \in \mathbf{State}\left[x \to E\right]$ by definition. □

Lemma 74. *Assume $x \to E$ is non-invertible. Let δ be a distribution with $x \in fv(\delta)$ and let $V = fv(\delta) - \{x\}$. If σ is feasible, then $\langle\sigma\rangle_\delta^{x \to E} = [\sigma \upharpoonright V]_\delta^V$.*

Proof. Let $\tau \in \langle\sigma\rangle_\delta^{x \to E}$. So $\tau\left[x \to E\right] = \sigma$ and $\tau \in \mathrm{support}\left(\delta\right)$. But the assignment only changes x, thus $\tau \upharpoonright V = \sigma \upharpoonright V$, therefore $\tau \in [\sigma \upharpoonright V]_\delta^V$. Thus $\langle\sigma\rangle_\delta^{x \to E} \subseteq [\sigma \upharpoonright V]_\delta^V$.

Let $\tau \in [\sigma \upharpoonright V]_\delta^V$. So $\tau \in \mathrm{support}\left(\delta\right)$ and $\tau \upharpoonright V = \sigma \upharpoonright V$. Since E doesn't depend on x, we have $\tau\left[x \to E\right] = \sigma\left[x \to E\right] = \sigma$; the second equality follows from Lemma 73 as σ is feasible by assumption. So $\tau \in \langle\sigma\rangle_\delta^{x \to E}$. Therefore $[\sigma \upharpoonright V]_\delta^V \subseteq \langle\sigma\rangle_\delta^{x \to E}$. □

Remark 75. Assume $x \to E$ is invertible. For every feasible σ, we have $\langle\sigma\rangle^{x \to E} = \left\{t_{x \to E}^{-1}(\sigma)\right\}$.

Proof. Invertability tells us that $\langle\sigma\rangle^{x \to E}$ has only one element. The function $t_{x \to E}^{-1}$, given the feasible σ, produces an element of $\langle\sigma\rangle^{x \to E}$, as $\left(t_{x \to E}^{-1}(\sigma)\right)\left[x \to E\right] = \sigma$. □

Definition 76. We define an alternate means of assignment, $\delta\langle x \to E\rangle$. Let $V = fv(\delta) - \{x\}$.

- If $x \to E$ is invertible, then

$$\begin{aligned}\delta\langle x \to E\rangle = \lambda\sigma.\ &\textbf{if } \sigma \text{ is feasible}\\ &\textbf{then } \delta\left(t_{x \to E}^{-1}(\sigma)\right)\\ &\textbf{else } 0\end{aligned}$$

- If $x \to E$ is not invertible, then

$$\delta\langle x \to E\rangle = \lambda\sigma.\ \textbf{if } \sigma \text{ is feasible} \\ \textbf{then } \delta \upharpoonright V\,(\sigma \upharpoonright V) \\ \textbf{else } 0$$

Lemma 77. *For any δ, $\delta\,[x \to E] = \delta\langle x \to E\rangle$.*

Proof. Let $\delta' = \delta\,[x \to E]$ and $\delta'' = \delta\langle x \to E\rangle$.

$$\begin{aligned}\delta'(\sigma) &= \sum_{\tau \mid \tau[x\to E]=\sigma} \delta(\tau) \\ &= \sum_{\tau \in \langle\sigma\rangle_\delta^{x\to E}} \delta(\tau)\end{aligned}$$

Case 1: $x \to E$ is invertible

If σ is feasible, $\langle\sigma\rangle^{x\to E}$ has only one element, $\sigma^{-1} = t_{x\to E}^{-1}(\sigma)$, by Remark 75. So $\delta'(\sigma) = \delta(\sigma^{-1}) = \delta''(\sigma)$. Note that when σ^{-1} is not in support (δ) then $\delta'(\sigma) = 0 = \delta''(\sigma)$.

If σ is not feasible then $\langle\sigma\rangle^{x\to E} = \emptyset$ so $\delta'(\sigma) = 0 = \delta''(\sigma)$.

Case 2: $x \to E$ is non-invertible

If σ is feasible, then by Lemma 74 we have $\langle\sigma\rangle_\delta^{x\to E} = [\sigma \upharpoonright V]_\delta^V$.

$$\begin{aligned}\delta'(\sigma) &= \sum_{\tau \in \langle\sigma\rangle_\delta^{x\to E}} \delta(\tau) \\ &= \sum_{\tau \in [\sigma\upharpoonright V]_\delta^V} \delta(\tau) \\ &= (\delta \upharpoonright V)\,(\sigma \upharpoonright V) \\ &= \delta''(\sigma)\end{aligned}$$

If σ is not feasible then $\langle\sigma\rangle_\delta^{x\to E} = \emptyset$ so $\delta'(\sigma) = 0 = \delta''(\sigma)$.

□

Lemma 78. *Assume $x \to E$ is invertible, then support $(\delta) = \left\{t_{x\to E}^{-1}(\sigma) \mid \sigma \in \text{support}\,(\delta\langle x \to E\rangle)\right\}$.*

Proof. Let $\delta_2 = \delta\langle x \to E\rangle$. Let $\tau \in \text{support}\,(\delta)$. So $\sigma \stackrel{\text{def}}{=} \tau\,[x \to E] \in \text{support}\,(\delta_2)$ and $t_{x\to E}^{-1}(\sigma) = \tau$. So $\tau \in \left\{t_{x\to E}^{-1}(\sigma) \mid \sigma \in \text{support}\,(\delta_2)\right\}$.

Let $\tau \in \left\{t_{x\to E}^{-1}(\sigma) \mid \sigma \in \text{support}\,(\delta_2)\right\}$. So $\tau = t_{x\to E}^{-1}(\sigma)$ for some $\sigma \in \text{support}\,(\delta_2)$. So there exists $\tau' \in \text{support}\,(\delta)$ such that $\tau'\,[x \to E] = \sigma$. But $t_{x\to E}^{-1}(\sigma) = t_{x\to E}^{-1}(t_{x\to E}(\tau')) = \tau'$ so $\tau' = \tau$ as $\tau = t_{x\to E}^{-1}(\sigma)$. So $\tau \in \text{support}\,(\delta)$. □

Lemma 30 (Soundness of Assignment)**.** *If $\delta \in \gamma_{\mathbb{P}}(P)$ then $\delta\,[x \to E] \in \gamma_{\mathbb{P}}(P\,[x \to E])$.*

Proof. Let $V = fv(\delta) - \{x\}$. By assumption, we have the following.

$$\text{support}\,(\delta) \subseteq \gamma_{\mathbb{C}}(C) \tag{78.1}$$
$$\mathrm{s}^{\min} \leq |\text{support}\,(\delta)| \leq \mathrm{s}^{\max} \tag{78.2}$$
$$\mathrm{m}^{\min} \leq \|\delta\| \leq \mathrm{m}^{\max} \tag{78.3}$$
$$\forall \sigma \in \text{support}\,(\delta)\ .\ \mathrm{p}^{\min} \leq \delta(\sigma) \leq \mathrm{p}^{\max} \tag{78.4}$$

Let $P_2 = P\,[x \to E]$ and $\delta_2 = \delta\,[x \to E] = \delta\langle x \to E\rangle$. Lemma 77 lets us use $\delta\,[x \to E]$ or $\delta\langle x \to E\rangle$ interchangeably.

We consider two cases. **Case 1: $x \to E$ is invertible**

In this case, P_2 is defined with $C_2 = C\,[x \to E]$ and all other parameters as in P. Thus we need to show the following.

$$\text{support}\,(\delta_2) \subseteq \gamma_{\mathbb{C}}(C_2) \tag{78.5}$$
$$\mathrm{s}^{\min} = \mathrm{s}_2^{\min} \leq |\text{support}\,(\delta_2)| \leq \mathrm{s}_2^{\max} = \mathrm{s}^{\max} \tag{78.6}$$
$$\mathrm{m}^{\min} = \mathrm{m}_2^{\min} \leq \|\delta_2\| \leq \mathrm{m}_2^{\max} = \mathrm{m}^{\max} \tag{78.7}$$
$$\forall \sigma \in \text{support}\,(\delta_2)\ .\ \mathrm{p}^{\min} = \mathrm{p}_2^{\min} \leq \delta_2(\sigma) \leq \mathrm{p}_2^{\max} = \mathrm{p}^{\max} \tag{78.8}$$

Claim 78.5 Support. By definition, $\gamma_{\mathbb{C}}(C_2) = \{\sigma\,[x \to E] \mid \sigma \in \gamma_{\mathbb{C}}(C)\}$. Let $\tau \in \text{support}\,(\delta_2)$, so we have $\sigma \in \text{support}\,(\delta) \subseteq \gamma_{\mathbb{C}}(C)$ with $\sigma\,[x \to E] = \tau$. So $\tau \in \gamma_{\mathbb{C}}(C_2)$. So $\tau \in \gamma_{\mathbb{C}}(C_2)$ and thus $\text{support}\,(\delta_2) \subseteq \gamma_{\mathbb{C}}(C_2)$. □

Claim 78.6 Support points. By Lemma 78 we have $\text{support}\,(\delta) = \left\{t^{-1}_{x\to E}(\sigma) \mid \sigma \in \text{support}\,(\delta_2)\right\}$. Inverse functions are necessarily injective over their domain, and since $\text{support}\,(\delta_2)$ are all feasible (thus in the domain of the inverse), we have $\left|\left\{t^{-1}_{x\to E}(\sigma) \mid \sigma \in \text{support}\,(\delta_2)\right\}\right| = |\text{support}\,(\delta_2)|$. So $|\text{support}\,(\delta)| = |\text{support}\,(\delta_2)|$. This, together with (78.2), completes the claim. □

Claim 78.7 Mass. Note again that $\text{support}\,(\delta_2) \subseteq \textbf{State}\,[x \to E]$. That is, all possible states are feasible. So we can write:

$$\begin{aligned}
\|\delta_2\| &= \sum_{\sigma \in \text{support}(\delta_2)} \delta_2(\sigma) \\
&= \sum_{\sigma \in \text{support}(\delta_2)} \delta(t^{-1}_{x\to E}(\sigma)) && [\text{ by defn. of } \delta_2 \text{ }] \\
&= \sum_{\tau \in \text{support}(\delta)} \delta(\tau) && [\text{ by Lemma 78 }] \\
&= \|\delta\|
\end{aligned}$$

The above, together with (78.3), completes this claim. □

Claim 78.8 Probability. Since $\text{support}\,(\delta_2)$ are feasible, we have, for every $\sigma \in \text{support}\,(\delta_2)$, $\delta_2(\sigma) = \delta(t^{-1}_{x\to E}(\sigma))$. But also, $t^{-1}_{x\to E}(\sigma) \in \text{support}\,(\delta)$. Taking this, and (78.4), completes this claim, and soundness in the invertible case. □

Case 2: $x \to E$ **is non-invertible** In this case, P_2 is defined via the forget operation. If $P_1 = \mathrm{f}_x(P)$ and $C_1 = (B_1, V_1)$, then $P_2 = P[x \to E]$ has $C_2 = (B_1 \cup \{x = E\}, V_1 \cup \{x\})$, and all other parameters as in P_1.

We need to show the following four claims.

$$\text{support}\,(\delta_2) \subseteq \gamma_{\mathbb{C}}(C_2) \tag{78.9}$$

$$\mathrm{s}_1^{\min} = \mathrm{s}_2^{\min} \leq |\text{support}\,(\delta_2)| \leq \mathrm{s}_2^{\max} = \mathrm{s}_1^{\max} \tag{78.10}$$

$$\mathrm{m}_1^{\min} = \mathrm{m}_2^{\min} \leq \|\delta_2\| \leq \mathrm{m}_2^{\max} = \mathrm{m}_1^{\max} \tag{78.11}$$

$$\forall \sigma \in \text{support}\,(\delta_2)\ .\ \mathrm{p}_1^{\min} = \mathrm{p}_2^{\min} \leq \delta_2(\sigma) \leq \mathrm{p}_2^{\max} = \mathrm{p}_1^{\max} \tag{78.12}$$

Recall the definition of δ_2:

$$\begin{aligned}\delta\langle x \to E\rangle = \lambda\sigma.\ &\textbf{if } \sigma \text{ is feasible}\\ &\textbf{then } \delta \restriction V\,(\sigma \restriction V)\\ &\textbf{else } 0\end{aligned}$$

Claim 78.9 Support. Let $\sigma \in \text{support}\,(\delta_2)$. So $\sigma \restriction V \in \text{support}\,(\delta \restriction V)$. so there exists $\tau \in \text{support}\,(\delta) \subseteq \gamma_{\mathbb{C}}(C)$ with $\tau \restriction V = \sigma \restriction V$. So $\tau \restriction V \in \gamma_{\mathbb{C}}(\mathrm{f}_x(C)) = \gamma_{\mathbb{C}}(C_1)$. So $\tau \in \gamma_{\mathbb{C}}((B_1, V_1 \cup \{x\}))$ as the add dimension operation leaves x unconstrained. The non-constraint of x also tells us that $\sigma \in \gamma_{\mathbb{C}}((B_1, V_1 \cup \{x\}))$ as we have $\sigma \restriction V = \tau \restriction V$.

Since $\sigma \in \text{support}\,(\delta_2)$, σ is feasible so σ satisfies the $x = E$ constraint as $\sigma = \tau[x \to E]$ for some τ. Thus, overall, we have $\sigma \in \gamma_{\mathbb{C}}((B_1 \cup \{x = E\}, V_1 \cup \{x\})) = \gamma_{\mathbb{C}}(C_2)$. □

Claim 78.10 Support points. Let $\delta_1 = \delta \restriction V = \mathrm{f}_x(\delta)$. By soundness of forget (Lemma 28), we have the following.

$$\mathrm{s}_1^{\min} = \mathrm{s}_2^{\min} \leq |\text{support}\,(\delta_1)| \leq \mathrm{s}_2^{\max} = \mathrm{s}_1^{\max}$$

All we need to show, then, is the following.

$$|\text{support}\,(\delta_1)| = |\text{support}\,(\delta_2)| \tag{78.13}$$

Let us show this by establishing a bijection f between the two sets. Let us define $f : \text{support}\,(\delta_1) \to \text{support}\,(\delta_2)$ via $f : \sigma_V \mapsto \sigma_V \cup \{x = [\![E]\!]\sigma_V\}$.

To show f is injective, let σ_V, σ'_V be such that $f(\sigma_V) = f(\sigma'_V)$. Since f does not change any part of the state other than adding x, it must be that $\sigma_V = \sigma'_V$.

To show that f is surjective, consider $\sigma \in \text{support}\,(\delta_2)$. So σ is feasible, so $\sigma[x \to E] = \sigma$ by Lemma 73. Also $\sigma \restriction V \in \text{support}\,(\sigma_V)$, considering the definition of σ_2. Since E doesn't depend on x, we can write $[\![E]\!]\sigma = [\![E]\!]\sigma \restriction V$, therefore $f(\sigma \restriction V) = \sigma \restriction V \cup \{x = [\![E]\!]\sigma \restriction V\} = \sigma[x \to E] = \sigma$.

Since f is injective and surjective, it is a bijection and thus $|\text{support}\,(\delta_1)| = |\text{support}\,(\delta_2)|$. □

Claim 78.11 Mass. Let $\delta_1 = \delta \upharpoonright V = \mathrm{f}_x(\delta)$. Let us show the following claim.

$$LHS = \text{support}\,(\delta_1) = \{\sigma \upharpoonright V \mid \sigma \in \text{support}\,(\delta_2)\} = RHS \tag{78.14}$$

Let $\sigma_V \in \text{support}\,(\delta_1)$. So there exists $\sigma \in \text{support}\,(\delta)$ with $\sigma \upharpoonright V = \sigma_V$. So $\sigma\,[x \to E] \in \text{support}\,(\delta_2)$. But the assignment doesn't change anything but x, so it must be that $(\sigma\,[x \to E]) \upharpoonright V = \sigma \upharpoonright V$, therefore $\sigma_V = \sigma \upharpoonright V \in \{\tau \upharpoonright V \mid \tau \in \text{support}\,(\delta_2)\}$. Thus $LHS \subseteq RHS$.

On the other side, let $\sigma \in \text{support}\,(\delta_2)$, so $\sigma = \tau\,[x \to E]$ for some $\tau \in \text{support}\,(\delta)$, by the original definition of distribution assignment. So $\tau \upharpoonright V \in \text{support}\,(\delta_1)$. But $(\tau\,[x \to E]) \upharpoonright V = \tau \upharpoonright V$ as the assignment doesn't change anything but x. So $\sigma \upharpoonright V = (\tau\,[x \to E]) \upharpoonright V = \tau\,[x \to E] \in \text{support}\,(\delta_1)$, concluding that $RHS \subseteq LHS$, and thus $LHS = RHS$.

Note that this, together with (78.13), show that not only are the sets equal, but also no two elements of $\text{support}\,(\delta_2)$ can map, via projection to V, to the same element of $\text{support}\,(\delta_1)$.

By soundness of forget (Lemma 28), we have the following.

$$\mathrm{m}_1^{\min} = \mathrm{m}_2^{\min} \leq \|\delta_1\| \leq \mathrm{m}_2^{\max} = \mathrm{m}_1^{\max}$$

Again, we proceed to show that $\|\delta_1\| = \|\delta_2\|$.

$$\begin{aligned}
\|\delta_1\| &= \sum_{\sigma_V \in \text{support}(\delta_1)} \delta_1(\sigma_V) \\
&= \sum_{\sigma \in \text{support}(\delta_2)} \delta_1(\sigma \upharpoonright V) && [\text{ by (78.13) and (78.14) }] \\
&= \sum_{\sigma \in \text{support}(\delta_2)} \delta_2(\sigma) && [\text{ by defn. of } \delta_2 \text{ }] \\
&= \|\delta_2\|
\end{aligned}$$

□

Claim 78.12 Probability. Let $\sigma \in \text{support}\,(\delta_2)$. So σ is feasible, so $\delta_2(\sigma) = (\delta \upharpoonright V)\,(\sigma \upharpoonright V) > 0$. Therefore $\sigma \upharpoonright V \in \text{support}\,(\delta \upharpoonright V)$. Thus, by soundness of forget (Lemma 28), we have $\mathrm{p}_2^{\min} = \mathrm{p}_1^{\min} \leq (\delta \upharpoonright V)\,(\sigma \upharpoonright V) \leq \mathrm{p}_1^{\max} = \mathrm{p}_2^{\max}$, concluding the claim and the lemma. □

□

B.3 Plus

Definition 79. Let $\mathit{overlap}(\delta_1, \delta_2) = \text{support}\,(\delta_1) \cap \text{support}\,(\delta_2)$.

Lemma 80. *If* $\delta_1 \in \gamma_{\mathbb{P}}(P_1)$ *and* $\delta_2 \in \gamma_{\mathbb{P}}(P_2)$ *then* $P_1 \mathbin{⍥} P_2 \leq |\mathit{overlap}(\delta_1, \delta_2)| \leq P_1 \mathbin{☺} P_2$.

Proof. We first note that for any sets A, B, it is the case that $|A \cup B| = |A| + |B| - |A \cap B|$ (often called the "inclusion-exclusion principle"). Rearranging the equation we also have $|A \cap B| = |A| + |B| - |A \cup B|$.

We will make use of this formula with $A = \text{support}\,(\delta_1)$, $B = \text{support}\,(\delta_2)$, and $U = \text{support}\,(\delta_1) \cup \text{support}\,(\delta_2)$.

Lower Bound We first show the lower bound. Expanding the definitions of $P_1 \mathbin{\dot{\frown}} P_2$ and $\mathit{overlap}(\delta_1, \delta_2)$, this reduces to showing the following.

$$\max((\mathrm{s}_1^{\min} - n_1) + (\mathrm{s}_2^{\min} - n_2) - n_3,\ 0) \leq |\text{support}\,(\delta_1) \cap \text{support}\,(\delta_2)|$$

Clearly we have $0 \leq |\text{support}\,(\delta_1) \cap \text{support}\,(\delta_2)|$, so it remains to show that the following holds.

$$(\mathrm{s}_1^{\min} - n_1) + (\mathrm{s}_2^{\min} - n_2) - n_3 \leq |\text{support}\,(\delta_1) \cap \text{support}\,(\delta_2)|$$

Expanding the definitions of n_1, n_2 from Definition 31, we obtain

$$(\mathrm{s}_1^{\min} - (\#(C_1) - n_3)) + (\mathrm{s}_2^{\min} - (\#(C_2) - n_3)) - n_3 \leq |\text{support}\,(\delta_1) \cap \text{support}\,(\delta_2)|$$

and rearranging yields the following.

$$\mathrm{s}_1^{\min} + \mathrm{s}_2^{\min} - (\#(C_1) + \#(C_2) - n_3) \leq |\text{support}\,(\delta_1) \cap \text{support}\,(\delta_2)|$$

This follows from the rearranged inclusion-exclusion principle provided we can show $\mathrm{s}_1^{\min} \leq |\text{support}\,(\delta_1)|$, $\mathrm{s}_2^{\min} \leq |\text{support}\,(\delta_2)|$, and $\#(C_1) + \#(C_2) - n_3 \geq |\text{support}\,(\delta_1) \cup \text{support}\,(\delta_2)|$. The first two follow directly from our assumptions that $\delta_1 \in \gamma_{\mathbb{P}}(P_1)$ and $\delta_2 \in \gamma_{\mathbb{P}}(P_2)$. For the third condition, we reason as follows.

We have from our assumptions that $\gamma_{\mathbb{C}}(C_1) \supseteq \text{support}\,(\delta_1)$ and $\gamma_{\mathbb{C}}(C_2) \supseteq \text{support}\,(\delta_2)$. Thus, we have

$$\gamma_{\mathbb{C}}(C_1) \cup \gamma_{\mathbb{C}}(C_2) \supseteq \text{support}\,(\delta_1) \cup \text{support}\,(\delta_2)$$

and finally

$$|\gamma_{\mathbb{C}}(C_1) \cup \gamma_{\mathbb{C}}(C_2)| \geq |\text{support}\,(\delta_1) \cup \text{support}\,(\delta_2)|$$

Utilizing the inclusion-exclusion principle, we have

$$|\gamma_{\mathbb{C}}(C_1)| + |\gamma_{\mathbb{C}}(C_2)| - |\gamma_{\mathbb{C}}(C_1) \cap \gamma_{\mathbb{C}}(C_2)| \geq |\text{support}\,(\delta_1) \cup \text{support}\,(\delta_2)|$$

Since we have $|\gamma_{\mathbb{C}}(C)| = \#(C)$, we can rewrite this to the following.

$$\#(C_1) + \#(C_2) - |\gamma_{\mathbb{C}}(C_1) \cap \gamma_{\mathbb{C}}(C_2)| \geq |\text{support}\,(\delta_1) \cup \text{support}\,(\delta_2)|$$

It remains to show that $|\gamma_{\mathbb{C}}(C_1) \cap \gamma_{\mathbb{C}}(C_2)| = n_3$. We have that $\gamma_{\mathbb{C}}(C_1 \sqcap_{\mathbb{C}} C_2) = \gamma_{\mathbb{C}}(C_1) \cap \gamma_{\mathbb{C}}(C_2)$ (that is, $\sqcap_{\mathbb{C}}$ is precise). This allows us to complete the final step, concluding that n_3, which is defined as $\#(C_1 \sqcap_{\mathbb{C}} C_2)$ is equal to $|\gamma_{\mathbb{C}}(C_1) \cap \gamma_{\mathbb{C}}(C_2)|$.

Upper Bound We next show that the upper bound holds. Our goal is to show the following.

$$P_1 \odot P_2 \geq |overlap(\delta_1, \delta_2)|$$

Expanding our definitions yields the following formula.

$$\min(\mathrm{s}_1^{\max}, \mathrm{s}_2^{\max}, n_3) \geq |\mathrm{support}\,(\delta_1) \cap \mathrm{support}\,(\delta_2)|$$

We first note that the following holds.

$$|\mathrm{support}\,(\delta_1) \cap \mathrm{support}\,(\delta_2)| \leq |\mathrm{support}\,(\delta_1)| \leq \mathrm{s}_1^{\max}$$

Thus $\mathrm{s}_1^{\max}$ is a sound upper bound. Similarly, we have

$$|\mathrm{support}\,(\delta_1) \cap \mathrm{support}\,(\delta_2)| \leq |\mathrm{support}\,(\delta_2)| \leq \mathrm{s}_2^{\max}$$

which shows that $\mathrm{s}_2^{\max}$ is a sound upper bound. Finally, we note that our assumptions give us $\mathrm{support}\,(\delta_1) \subseteq \gamma_{\mathbb{C}}(C_1)$ and $\mathrm{support}\,(\delta_1) \subseteq \gamma_{\mathbb{C}}(C_1)$. Thus we have the following.

$$\mathrm{support}\,(\delta_1) \cap \mathrm{support}\,(\delta_2) \subseteq \gamma_{\mathbb{C}}(C_1) \cap \gamma_{\mathbb{C}}(C_2)$$

We showed previously that $n_3 = |\gamma_{\mathbb{C}}(C_1) \cap \gamma_{\mathbb{C}}(C_2)|$. Thus we have

$$|\mathrm{support}\,(\delta_1) \cap \mathrm{support}\,(\delta_2)| \leq n_3$$

which shows that n_3 is a sound upper bound.

Since all of $\mathrm{s}_1^{\max}, \mathrm{s}_2^{\max}$, and n_3 are sound upper bounds, their minimum is also a sound upper bound. □

Lemma 81.

$$|\mathit{support}\,(\delta_1 + \delta_2)| = |\mathit{support}\,(\delta_1)| + |\mathit{support}\,(\delta_2)| - |\mathit{overlap}(\delta_1, \delta_2)|$$

Proof. First we note that $\mathrm{support}\,(\delta_1 + \delta_2) = \{\sigma \mid \delta_1(\sigma) + \delta_2(\sigma) > 0\}$. Since the range of δ_1 and δ_2 is $[0, 1]$, we have that $\delta_1(\sigma) + \delta_2(\sigma) > 0$ if and only if either $\delta_1(\sigma) > 0$ or $\delta_2(\sigma) > 0$. Thus, we have $\sigma \in \mathrm{support}\,(\delta_1 + \delta_2)$ if and only if $\sigma \in \mathrm{support}\,(\delta_1)$ or $\sigma \in \mathrm{support}\,(\delta_2)$, which implies $\mathrm{support}\,(\delta_1 + \delta_2) = \mathrm{support}\,(\delta_1) \cup \mathrm{support}\,(\delta_2)$.

Next, we note that for any sets A, B we have $|A \cup B| = |A| + |B| - |A \cap B|$. Utilizing this statement with $A = \mathrm{support}\,(\delta_1)$ and $B = \mathrm{support}\,(\delta_2)$ completes the proof. □

Lemma 33 (Soundness of Plus)**.** *If* $\delta_1 \in \gamma_{\mathbb{P}}(P_1)$ *and* $\delta_2 \in \gamma_{\mathbb{P}}(P_2)$ *then* $\delta_1 + \delta_2 \in \gamma_{\mathbb{P}}(P_1 + P_2)$.

Proof. Suppose $\delta_1 \in \gamma_{\mathbb{P}}(P_1)$ and $\delta_2 \in \gamma_{\mathbb{P}}(P_2)$. Then we have the following.

$$\text{support}\,(\delta_1) \subseteq \gamma_{\mathbb{C}}(C_1) \tag{81.1}$$
$$\mathrm{s}_1^{\min} \leq |\text{support}\,(\delta_1)| \leq \mathrm{s}_1^{\max} \tag{81.2}$$
$$\mathrm{m}_1^{\min} \leq \|\delta_1\| \leq \mathrm{m}_1^{\max} \tag{81.3}$$
$$\forall \sigma \in \text{support}\,(\delta_1)\,.\ \mathrm{p}_1^{\min} \leq \delta_1(\sigma) \leq \mathrm{p}_1^{\max} \tag{81.4}$$

and

$$\text{support}\,(\delta_2) \subseteq \gamma_{\mathbb{C}}(C_2) \tag{81.5}$$
$$\mathrm{s}_2^{\min} \leq |\text{support}\,(\delta_2)| \leq \mathrm{s}_2^{\max} \tag{81.6}$$
$$\mathrm{m}_2^{\min} \leq \|\delta_2\| \leq \mathrm{m}_2^{\max} \tag{81.7}$$
$$\forall \sigma \in \text{support}\,(\delta_2)\,.\ \mathrm{p}_2^{\min} \leq \delta_2(\sigma) \leq \mathrm{p}_2^{\max} \tag{81.8}$$

The definition of abstract plus has special cases when either of the arguments are zero, that is, if $\mathit{iszero}(P_1)$ or $\mathit{iszero}(P_2)$. Without the loss of generality, let us assume $\mathit{iszero}(P_2)$ and thus by definition $P_1 + P_2 = P_1$. Since $\gamma_{\mathbb{P}}(P_2) = \{0_{\mathbf{Dist}}\}$, where $0_{\mathbf{Dist}}$ is the distribution assigning probability of 0 to every state. Therefore $\delta_2 = 0_{\mathbf{Dist}}$ and thus $\delta_1 + \delta_2 = \delta_1$. But we already have $\delta_1 \in \gamma_{\mathbb{P}}(P_1)$ by assumption, hence we are done in this case.

In the case when not $\mathit{iszero}(P_1)$ and not $\mathit{iszero}(P_2)$ we must show the following.

$$\text{support}\,(\delta_1 + \delta_2) \subseteq \gamma_{\mathbb{C}}(C_1 \sqcup_{\mathbb{C}} C_2) \tag{81.9}$$
$$\max\left[\mathrm{s}_1^{\min} + \mathrm{s}_2^{\min} - P_1 \mathbin{\odot} P_2, 0\right] \leq |\text{support}\,(\delta_1 + \delta_2)| \tag{81.10}$$
$$|\text{support}\,(\delta_1 + \delta_2)| \leq \min\left[\mathrm{s}_1^{\max} + \mathrm{s}_2^{\max} - P_1 \mathbin{\oslash} P_2, \#(C_3)\right] \tag{81.11}$$
$$\mathrm{m}_1^{\min} + \mathrm{m}_2^{\min} \leq \|\delta_1 + \delta_2\| \leq \mathrm{m}_1^{\max} + \mathrm{m}_2^{\max} \tag{81.12}$$

We also must show the conditions on $\mathrm{p}^{\min}$ and $\mathrm{p}^{\max}$ for the sum.

Condition (81.9) follows from (81.1) and (81.5) and the fact that $\sqcup_{\mathbb{C}}$ over-approximates union. The key step is noting that $\text{support}\,(\delta_1 + \delta_2) = \text{support}\,(\delta_1) \cup \text{support}\,(\delta_2)$. To show this we consider some $\sigma \in \text{support}\,(\delta_1 + \delta_2)$. We have that $(\delta_1 + \delta_2)(\sigma) > 0$ which, expanding the definition of $+$, yields $\delta_1(\sigma) + \delta_2(\sigma) > 0$. Since the range of δ_1 and δ_2 is $[0, 1]$, this implies that either $\delta_1(\sigma) > 0$ or $\delta_2(\sigma) > 0$ and thus $\sigma \in \text{support}\,(\delta_1)$ or $\sigma \in \text{support}\,(\delta_2)$.

Conditions (81.10) and (81.11) follow from (81.2) and (81.6) and Lemmas 80 and 81. We have $\mathrm{s}_1^{\min} \leq |\text{support}\,(\delta_1)|$ from (81.2) and $\mathrm{s}_2^{\min} \leq |\text{support}\,(\delta_2)|$ from (81.6). Monotonicity of addition then gives us

$$\mathrm{s}_1^{\min} + \mathrm{s}_2^{\min} \leq |\text{support}\,(\delta_1)| + |\text{support}\,(\delta_2)|$$

From Lemma 80 we have $|\mathit{overlap}(\delta_1, \delta_2)| \leq P_1 \mathbin{\odot} P_2$ and thus

$$-P_1 \mathbin{\odot} P_2 \leq -\,|\mathit{overlap}(\delta_1, \delta_2)|$$

Combining with the above yields

$$\mathrm{s}_1^{\min} + \mathrm{s}_2^{\min} - P_1 \circledcirc P_2 \leq |\mathrm{support}\,(\delta_1)| + |\mathrm{support}\,(\delta_2)| - |\mathit{overlap}(\delta_1, \delta_2)|$$

We can then rewrite the right-hand side according to Lemma 81 to obtain

$$\mathrm{s}_1^{\min} + \mathrm{s}_2^{\min} - P_1 \circledcirc P_2 \leq |\mathrm{support}\,(\delta_1 + \delta_2)|$$

which is condition (81.10).

Condition (81.11) follows the same reasoning. We have $|\mathrm{support}\,(\delta_1)|+|\mathrm{support}\,(\delta_2)| \leq \mathrm{s}_1^{\max}+\mathrm{s}_2^{\max}$ by (81.2) and (81.6). We then apply Lemma 80 and 81 to obtain condition (81.11).

For Condition (81.12), note that

$$\|\delta_1 + \delta_2\| = \sum_\sigma \big(\delta_1(\sigma) + \delta_2(\sigma)\big) = \sum_\sigma \delta_1(\sigma) + \sum_\sigma \delta_2(\sigma)$$

This is then equivalent to $\|\delta_1\| + \|\delta_2\|$. We have shown that $\|\delta_1 + \delta_2\| = \|\delta_1\| + \|\delta_2\|$. Condition (81.12) then follows from monotonicity of addition applied to (81.3) and (81.7)

We now consider the $\mathrm{p}^{\min}$ and $\mathrm{p}^{\max}$ conditions. Let $P_3 = P_1 + P_2$ and $\delta_3 = \delta_1 + \delta_2$. We must show.

$$\forall \sigma \in \mathrm{support}\,(\delta_3)\ .\ \mathrm{p}_3^{\min} \leq \delta_3(\sigma) \leq \mathrm{p}_3^{\max}$$

The values $\mathrm{p}_3^{\min}$ and $\mathrm{p}_3^{\max}$ are defined by cases and we consider these cases separately. In one case, we have that $\mathrm{p}^{\min}$ of the sum is $\min(\mathrm{p}_1^{\min}, \mathrm{p}_2^{\min})$. This is always a sound choice. To see why, suppose $\sigma \in \mathrm{support}\,(\delta_1 + \delta_2)$. Then $\sigma \in \mathrm{support}\,(\delta_1)$ or $\sigma \in \mathrm{support}\,(\delta_2)$. If $\sigma \in \mathrm{support}\,(\delta_1)$, then $(\delta_1 + \delta_2)(\sigma) = \delta_1(\sigma) + \delta_2(\sigma)$ is at least $\mathrm{p}_1^{\min}$. Similarly, if $\sigma \in \mathrm{support}\,(\delta_2)$ then $(\delta_1 + \delta_2)(\sigma) \geq \delta_2(\sigma)$.

Similarly, the value $\mathrm{p}_1^{\max} + \mathrm{p}_2^{\max}$ is always a sound choice for $\mathrm{p}_3^{\max}$. Consider $\sigma \in \mathrm{support}\,(\delta_3)$. Then $\sigma \in \mathrm{support}\,(\delta_1)$ or $\sigma \in \mathrm{support}\,(\delta_2)$. If $\sigma \in \mathrm{support}\,(\delta_1)$ and $\sigma \notin \mathrm{support}\,(\delta_2)$, then we have

$$\delta_3(\sigma) = \delta_1(\sigma) + \delta_2(\sigma) = \delta_1(\sigma)$$

By (81.4) we then have $\delta_3(\sigma) \leq \mathrm{p}_1^{\max}$ and thus $\delta_3(\sigma) \leq \mathrm{p}_1^{\max} + \mathrm{p}_2^{\max}$ as desired.

Similarly, if $\sigma \notin \mathrm{support}\,(\delta_1)$ and $\sigma \in \mathrm{support}\,(\delta_2)$ then by (81.8) we have

$$\delta_3(\sigma) = \delta_2(\sigma) \leq \mathrm{p}_2^{\max} \leq \mathrm{p}_1^{\max} + \mathrm{p}_2^{\max}$$

Finally, if $\sigma \in \mathrm{support}\,(\delta_1)$ and $\sigma \in \mathrm{support}\,(\delta_2)$ then by (81.4) we have $\delta_1(\sigma) \leq \mathrm{p}_1^{\max}$. By (81.8) we have $\delta_2(\sigma) \leq \mathrm{p}_2^{\max}$. Combining these we have $\delta_1(\sigma) + \delta_2(\sigma) \leq \mathrm{p}_1^{\max} + \mathrm{p}_2^{\max}$ which is equivalent to $\delta_3(\sigma) \leq \mathrm{p}_3^{\max}$ as desired.

Next we consider the $P_1 \circledcirc P_2 = \#(C_3)$ case for $\mathrm{p}_3^{\min}$. We must show that $\mathrm{p}_1^{\min} + \mathrm{p}_2^{\min}$ is a sound lower bound on $\delta_3(\sigma)$ for $\sigma \in \mathrm{support}\,(\delta_3)$. We have by Lemma 80 that $P_1 \circledcirc P_2 \leq |\mathit{overlap}(\delta_1, \delta_2)|$. Since $P_1 \circledcirc P_2 = \#(C_3)$ and $\#(C_3) \geq |\mathit{overlap}(\delta_1, \delta_2)|$, we have that $\#(C_3) = |\mathit{overlap}(\delta_1, \delta_2)|$. Expanding the definition of $\mathit{overlap}(\delta_1, \delta_2)$ yields

$$|\mathrm{support}\,(\delta_1) \cap \mathrm{support}\,(\delta_2)| = \#(C_3) \tag{81.13}$$

We have from (81.9) that $\mathrm{support}\,(\delta_1 + \delta_2) \subseteq \gamma_{\mathbb{C}}(C_3)$ and from the proof of (81.9) we have that $\mathrm{support}\,(\delta_1 + \delta_2) = \mathrm{support}\,(\delta_1) \cup \mathrm{support}\,(\delta_2)$. Combining these yields

$$|\mathrm{support}\,(\delta_1) \cup \mathrm{support}\,(\delta_2)| \leq \#(C_3)$$

Combining this with (81.13) yields

$$|\mathrm{support}\,(\delta_1) \cup \mathrm{support}\,(\delta_2)| \leq |\mathrm{support}\,(\delta_1) \cap \mathrm{support}\,(\delta_2)|$$

For any sets A, B, we have that $|A \cup B| \geq |A \cap B|$ and thus the above inequality implies the following.

$$|\mathrm{support}\,(\delta_1) \cup \mathrm{support}\,(\delta_2)| = |\mathrm{support}\,(\delta_1) \cap \mathrm{support}\,(\delta_2)|$$

The fact that the size of the intersection and the size of the union of $\mathrm{support}\,(\delta_1)$ and $\mathrm{support}\,(\delta_2)$ is identical implies that $\mathrm{support}\,(\delta_1) = \mathrm{support}\,(\delta_2)$. This implies that for all σ, we have $\sigma \in \mathrm{support}\,(\delta_1)$ if and only if $\sigma \in \mathrm{support}\,(\delta_2)$.

Now consider $\sigma \in \mathrm{support}\,(\delta_3)$. We have $\sigma \in \mathrm{support}\,(\delta_1)$ or $\sigma \in \mathrm{support}\,(\delta_2)$, as before, but now we can strengthen this to $\sigma \in \mathrm{support}\,(\delta_1)$ *and* $\sigma \in \mathrm{support}\,(\delta_2)$. By (81.4) we have $\mathrm{p}_1^{\min} \leq \delta_1(\sigma)$ and by (81.8) we have $\mathrm{p}_2^{\min} \leq \delta_2(\sigma)$. Thus we have

$$\mathrm{p}_1^{\min} + \mathrm{p}_2^{\min} \leq \delta_1(\sigma) + \delta_2(\sigma)$$

which was our goal.

Finally we consider the $P_1 \circledcirc P_2 = 0$ case for $\mathrm{p}_3^{\max}$ (the "otherwise" case in Definition 32). Consider a $\sigma \in \mathrm{support}\,(\delta_3)$. We must show that $\delta_3(\sigma) \leq \max(\mathrm{p}_1^{\max}, \mathrm{p}_2^{\max})$. We have that either $\sigma \in \mathrm{support}\,(\delta_1)$ or $\sigma \in \mathrm{support}\,(\delta_2)$. We cannot have both since $P_1 \circledcirc P_2 = 0$ which, by Lemma 80 implies that $|\mathit{overlap}(\delta_1, \delta_2)| = 0$. If $\sigma \in \mathrm{support}\,(\delta_1)$ then by (81.4) we have $\delta_1(\sigma) \leq \mathrm{p}_1^{\max}$. We have $\sigma \notin \mathrm{support}\,(\delta_2)$ and thus $\delta_2(\sigma) = 0$. Thus we reason that

$$\delta_1(\sigma) + \delta_2(\sigma) = \delta_1(\sigma) \leq \mathrm{p}_1^{\max} \leq \max(\mathrm{p}_1^{\max}, \mathrm{p}_2^{\max})$$

Similarly, if $\sigma \in \mathrm{support}\,(\delta_2)$ then we apply (81.8) to obtain

$$\delta_1(\sigma) + \delta_2(\sigma) = \delta_2(\sigma) \leq \mathrm{p}_2^{\max} \leq \max(\mathrm{p}_1^{\max}, \mathrm{p}_2^{\max})$$

□

B.4 Product

Lemma 34 (Soundness of Product)**.** *If* $\delta_1 \in \gamma_{\mathbb{P}}(P_1)$ *and* $\delta_2 \in \gamma_{\mathbb{P}}(P_2)$ *then* $\delta_1 \times \delta_2 \in \gamma_{\mathbb{P}}(P_1 \times P_2)$.

Proof. By assumption, we have the following for $i = 1, 2$.

$$\text{support}\,(\delta_i) \subseteq \gamma_{\mathbb{C}}(C_i) \tag{81.14}$$
$$\mathrm{s}_i^{\min} \leq |\text{support}\,(\delta_i)| \leq \mathrm{s}_i^{\max} \tag{81.15}$$
$$\mathrm{m}_i^{\min} \leq \|\delta_i\| \leq \mathrm{m}_i^{\max} \tag{81.16}$$
$$\forall \sigma \in \text{support}\,(\delta_i)\ .\ \mathrm{p}_i^{\min} \leq \delta(\sigma_i) \leq \mathrm{p}_i^{\max} \tag{81.17}$$

Let $\delta_3 = \delta_1 \times \delta_2$ and $P_3 = P_1 \times P_2$. Recall the definition of P_3.

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

We must show the following four claims.

$$\text{support}\,(\delta_3) \subseteq \gamma_{\mathbb{C}}(C_3) \tag{81.18}$$
$$\mathrm{s}_3^{\min} \leq |\text{support}\,(\delta_3)| \leq \mathrm{s}_3^{\max} \tag{81.19}$$
$$\mathrm{m}_3^{\min} \leq \|\delta_3\| \leq \mathrm{m}_3^{\max} \tag{81.20}$$
$$\forall \sigma \in \text{support}\,(\delta_3)\ .\ \mathrm{p}_3^{\min} \leq \delta_3(\sigma) \leq \mathrm{p}_3^{\max} \tag{81.21}$$

Also, recall the definition of concrete product.

$$\delta_1 \times \delta_2 = \lambda(\sigma_1, \sigma_2).\ \delta_1(\sigma_1) \cdot \delta_2(\sigma_2)$$

Let $V_1 = \mathit{fv}(\delta_1)$ and $V_2 = \mathit{fv}(\delta_2)$.

Claim 81.18 – Support. Let $\sigma = (\sigma_1, \sigma_2) \in \text{support}\,(\delta_3)$. Thus it must be that $\delta_1(\sigma_1) > 0$ and $\delta_2(\sigma_2) > 0$, thus, by (81.14), $\sigma_1 \in \text{support}\,(\delta_1) \subseteq \gamma_{\mathbb{C}}(C_1)$ and $\sigma_2 \in \text{support}\,(\delta_2) \subseteq \gamma_{\mathbb{C}}(C_2)$, therefore $\sigma \in \gamma_{\mathbb{C}}(\delta_3)$. □

Claim 81.19 – Support points. Using (81.15) we get the following.

$$\mathrm{s}_1^{\min} \cdot \mathrm{s}_2^{\min} \leq |\text{support}\,(\delta_1)| \cdot |\text{support}\,(\delta_2)| \leq \mathrm{s}_1^{\max} \cdot \mathrm{s}_2^{\max}$$

Likewise, the size of support (δ_3) can be equated as follows.

$$|\text{support}\,(\delta_3)| = \left| \left\{ (\sigma_1, \sigma_2) \;\middle|\; \begin{array}{l} \sigma_1 \in \text{support}\,(\delta_1)\,, \\ \sigma_2 \in \text{support}\,(\delta_2) \end{array} \right\} \right|$$

$$= |\text{support}\,(\delta_1)| \cdot |\text{support}\,(\delta_2)|$$

This completes the claim as $\mathrm{s}_3^{\min} = \mathrm{s}_1^{\min} \cdot \mathrm{s}_2^{\min}$ and $\mathrm{s}_3^{\max} = \mathrm{s}_1^{\max} \cdot \mathrm{s}_2^{\max}$. □

Claim 81.20 – Mass.

$$\begin{aligned}
\|\delta_3\| &= \sum_{\sigma \in \text{support}(\delta_3)} \delta(\sigma) \\
&= \sum_{(\sigma_1,\sigma_2) \in \text{support}(\delta_3)} \delta_1(\sigma_1) \cdot \delta_2(\sigma_2) \\
&= \sum_{\sigma_1 \in \text{support}(\delta_1)} \left(\sum_{\sigma_2 \in \text{support}(\delta_2)} \delta_1(\sigma_1) \cdot \delta_2(\sigma_2) \right) \\
&= \sum_{\sigma_1 \in \text{support}(\delta_1)} \delta_1(\sigma_1) \sum_{\sigma_2 \in \text{support}(\delta_2)} \delta_2(\sigma_2) \\
&= \sum_{\sigma_1 \in \text{support}(\delta_1)} \delta_1(\sigma_1) \cdot \|\delta_2\| \\
&= \|\delta_1\| \cdot \|\delta_2\|
\end{aligned}$$

□

Likewise, by (81.16), we have the following.

$$\mathrm{m}_1^{\min} \cdot \mathrm{m}_2^{\min} \leq \|\delta_1\| \cdot \|\delta_2\| \leq \mathrm{m}_1^{\max} \cdot \mathrm{m}_2^{\max}$$

This completes the claim as $\mathrm{m}_3^{\min} = \mathrm{m}_1^{\min} \cdot \mathrm{m}_2^{\min}$ and $\mathrm{m}_3^{\max} = \mathrm{m}_1^{\max} \cdot \mathrm{m}_2^{\max}$.

Claim 81.21 – Probability. Let $\sigma = (\sigma_1, \sigma_2) \in \text{support}\,(\delta_3)$. Thus $\sigma_1 \in \text{support}\,(\delta_1)$ and $\sigma_2 \in \text{support}\,(\delta_2)$. Also, $\delta_3\,(\sigma) = \delta_1(\sigma_1) \cdot \delta_2(\sigma_2)$. By (81.17), we have $\mathrm{p}_1^{\min} \leq \delta_1(\sigma_1) \leq \mathrm{p}_1^{\max}$ and $\mathrm{p}_2^{\min} \leq \delta_2(\sigma_2) \leq \mathrm{p}_2^{\max}$. Therefore

$$\mathrm{p}_3^{\min} = \mathrm{p}_1^{\min} \cdot \mathrm{p}_2^{\min} \leq \delta_3(\sigma) \leq \mathrm{p}_1^{\max} \cdot \mathrm{p}_2^{\max} = \mathrm{p}_3^{\max}$$

This completes the claim and the proof. □

□

B.5 Conditioning

Definition 82. Given a set of states S and a boolean expression B, let $S|B$ be the subset of S that satisfy the condition B and $S|\overline{B}$ be the subset of S that do not satisfy the condition. Formally,

$$\begin{aligned}
S|B &\stackrel{\text{def}}{=} \{\sigma \in S \mid [\![B]\!]\sigma = \textbf{ true }\} \\
S|\overline{B} &\stackrel{\text{def}}{=} \{\sigma \in S \mid [\![B]\!]\sigma = \textbf{ false }\}
\end{aligned}$$

Lemma 36 (Soundness of Conditioning)**.** *If* $\delta \in \gamma_{\mathbb{P}}(P)$ *then* $\delta \wedge B \in \gamma_{\mathbb{P}}(P \wedge B)$.

Proof. Let $\delta_2 = \delta \wedge B$. Recall the definition of the conditional distribution:

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

Let $P_2 = P \wedge B$. The construction of P_2 produces the following parameters.

$$\begin{array}{lll|l}
\mathrm{p}_2^{\min} & = \mathrm{p}^{\min} & & \mathrm{s}_2^{\min} = \max\left[\mathrm{s}^{\min} - \overline{n}, 0\right] \\
\mathrm{p}_2^{\max} & = \mathrm{p}^{\max} & & \mathrm{s}_2^{\max} = \min\left[\mathrm{s}^{\max}, n\right] \\
\mathrm{m}_2^{\min} & = \max\left[\mathrm{p}_2^{\min} \cdot \mathrm{s}_2^{\min},\ \mathrm{m}^{\min} - \mathrm{p}^{\max} \cdot \min\left[\mathrm{s}^{\max}, \overline{n}\right]\right] & & \\
\mathrm{m}_2^{\max} & = \min\left[\mathrm{p}_2^{\max} \cdot \mathrm{s}_2^{\max},\ \mathrm{m}^{\max} - \mathrm{p}^{\min} \cdot \max\left[\mathrm{s}^{\min} - n, 0\right]\right] & & \\
C_2 & = \langle\!\langle B \rangle\!\rangle\, C & &
\end{array}$$

The quantities n and $\overline{n}$ are defined in such a way that n over-approximates the number of support points of δ that satisfy B, whereas $\overline{n}$ over-approximates the number of support points of δ that do not satisfy B. Also, $\langle\!\langle B \rangle\!\rangle\, C$ is defined to contain at least the points in C that satisfy B. Making these properties precise gives us the following.

$$|\mathrm{support}\,(\delta)\,|B| \leq n \tag{82.1}$$
$$\left|\mathrm{support}\,(\delta)\,|\overline{B}\right| \leq \overline{n} \tag{82.2}$$
$$\gamma_{\mathbb{C}}(C)|B \subseteq \gamma_{\mathbb{C}}(\langle\!\langle B \rangle\!\rangle\, C) \tag{82.3}$$

By assumption we have the following.

$$\mathrm{support}\,(\delta) \subseteq \gamma_{\mathbb{C}}(C) \tag{82.4}$$
$$\mathrm{s}^{\min} \leq |\mathrm{support}\,(\delta)| \leq \mathrm{s}^{\max} \tag{82.5}$$
$$\mathrm{m}^{\min} \leq \|\delta\| \leq \mathrm{m}^{\max} \tag{82.6}$$
$$\forall \sigma \in \mathrm{support}\,(\delta)\ .\ \mathrm{p}^{\min} \leq \delta(\sigma) \leq \mathrm{p}^{\max} \tag{82.7}$$

We need to show the following four claims.

$$\mathrm{support}\,(\delta_2) \subseteq \gamma_{\mathbb{C}}(C_2) \tag{82.8}$$
$$\mathrm{s}_2^{\min} \leq |\mathrm{support}\,(\delta_2)| \leq \mathrm{s}_2^{\max} \tag{82.9}$$
$$\mathrm{m}_2^{\min} \leq \|\delta_2\| \leq \mathrm{m}_2^{\max} \tag{82.10}$$
$$\forall \sigma \in \mathrm{support}\,(\delta_2)\ .\ \mathrm{p}_2^{\min} \leq \delta_2(\sigma) \leq \mathrm{p}_2^{\max} \tag{82.11}$$

Claim 82.8 – Support. Let $\sigma \in \mathrm{support}\,(\delta_2)$. Thus it must be that $\sigma \in \mathrm{support}\,(\delta)$ and $[\![B]\!]\sigma =$ **true** . By (82.4), we have $\sigma \in \gamma_{\mathbb{C}}(C)$, therefore $\sigma \in \gamma_{\mathbb{C}}(C_2)$ as $\{\sigma \in \gamma_{\mathbb{C}}(C) \mid [\![B]\!]\sigma =$ **true** $\} \subseteq \gamma_{\mathbb{C}}(C_2)$ by construction of C_2. □

Claim 82.9 – Support points. Let us write support (δ) as a union of two disjoint sets.

$$\text{support}\,(\delta) = \text{support}\,(\delta)\,|B \cup \text{support}\,(\delta)\,|\overline{B}$$

Given the disjointness of the two, we also have the following.

$$|\text{support}\,(\delta)| = |\text{support}\,(\delta)\,|B| + \left|\text{support}\,(\delta)\,|\overline{B}\right|$$

Now note that support $(\delta_2) = \text{support}\,(\delta)\,|B$. Thus we can write $|\text{support}\,(\delta_2)| = |\text{support}\,(\delta)| - \left|\text{support}\,(\delta)\,|\overline{B}\right|$. We can therefore estimate the size of the support of δ_2 in the following manner.

$$\begin{aligned} |\text{support}\,(\delta_2)| &= |\text{support}\,(\delta)| - \left|\text{support}\,(\delta)\,|\overline{B}\right| \\ &\leq |\text{support}\,(\delta)| \\ &\leq \mathrm{s}^{\max} && [\text{ by } (82.5)\text{ }] \end{aligned}$$

Therefore, using (82.1) and the above, we have $|\text{support}\,(\delta_2)| \leq \min\left[\mathrm{s}^{\max}, n\right] = \mathrm{s}_2^{\max}$.

Going in the other direction, we can write as follows.

$$\begin{aligned} |\text{support}\,(\delta_2)| &= |\text{support}\,(\delta)| - \left|\text{support}\,(\delta)\,|\overline{B}\right| \\ &\geq \mathrm{s}^{\min} - \left|\text{support}\,(\delta)\,|\overline{B}\right| && [\text{ by } (82.5)\text{ }] \\ &\geq \mathrm{s}^{\min} - \overline{n} && [\text{ by } (82.2)\text{ }] \end{aligned}$$

Since all sets are trivially of size at least 0, we have $|\text{support}\,(\delta_2)| \geq \max\left[\mathrm{s}^{\min} - \overline{n}, 0\right] = \mathrm{s}_2^{\min}$. □

Claim 82.11 – Probability. Note that we will show the probability claim before the mass as we will use the truth of the probability claim in the mass arguments.

Let $\sigma \in \text{support}\,(\delta_2)$. By definition of δ_2, we have $\delta_2(\sigma) = \delta(\sigma)$. Thus $\sigma \in \text{support}\,(\delta)$ so by (82.7) we have:

$$\mathrm{p}_2^{\min} = \mathrm{p}^{\min} \leq \delta(\sigma) = \delta_2(\sigma) \leq \mathrm{p}^{\max} = \mathrm{p}_2^{\max}$$

□

Claim 82.10 – Mass. Let us first show the following bound on the size of support $(\delta)\,|\overline{B}$.

$$\max\left[\mathrm{s}^{\min} - n, 0\right] \leq \left|\text{support}\,(\delta)\,|\overline{B}\right| \leq \min\left[\mathrm{s}^{\max}, \overline{n}\right] \tag{82.12}$$

Since $|\text{support}\,(\delta)| = |\text{support}\,(\delta)\,|B| + \left|\text{support}\,(\delta)\,|\overline{B}\right|$, we can say $\left|\text{support}\,(\delta)\,|\overline{B}\right| = |\text{support}\,(\delta)| - |\text{support}\,(\delta)\,|B|$ and continue to the bound in the following manner.

$$\left|\text{support}\,(\delta)\,|\overline{B}\right| = |\text{support}\,(\delta)| - |\text{support}\,(\delta)\,|B|$$

$$\begin{aligned}
&\geq \mathrm{s}^{\min} - |\mathrm{support}\,(\delta)\,|B| && [\text{ by } (82.5)\ ]\\
&\geq \mathrm{s}^{\min} - n && [\text{ by } (82.1)\ ]
\end{aligned}$$

Therefore $\left|\mathrm{support}\,(\delta)\,|\overline{B}\right| \geq \max\left[\mathrm{s}^{\min} - n, 0\right]$ as claimed. For the other end of the inequality, note that we have $\left|\mathrm{support}\,(\delta)\,|\overline{B}\right| \leq |\mathrm{support}\,(\delta)| \leq \mathrm{s}^{\max}$ by (82.5). Also, by (82.2), $\left|\mathrm{support}\,(\delta)\,|\overline{B}\right| \leq \overline{n}$. Therefore $\left|\mathrm{support}\,(\delta)\,|\overline{B}\right| \leq \max\left[\mathrm{s}^{\max}, \overline{n}\right]$, completing our bound.

Now, let us write $\|\delta\|$ in two parts.

$$\begin{aligned}
\|\delta\| &= \sum_{\sigma \in \mathrm{support}(\delta)} \delta(\sigma)\\
&= \sum_{\sigma \in \mathrm{support}(\delta)|B} \delta(\sigma) + \sum_{\sigma \in \mathrm{support}(\delta)|\overline{B}} \delta(\sigma)\\
&= \|\delta_2\| + \sum_{\sigma \in \mathrm{support}(\delta)|\overline{B}} \delta(\sigma)
\end{aligned}$$

Therefore $\|\delta_2\| = \|\delta\| - \sum_{\sigma \in \mathrm{support}(\delta)|\overline{B}} \delta(\sigma)$.

$$\begin{aligned}
\|\delta_2\| &= \|\delta\| - \sum_{\sigma \in \mathrm{support}(\delta)|\overline{B}} \delta(\sigma)\\
&\leq \mathrm{m}^{\max} - \sum_{\sigma \in \mathrm{support}(\delta)|\overline{B}} \delta(\sigma) && [\text{ by } (82.6)\ ]\\
&\leq \mathrm{m}^{\max} - \sum_{\sigma \in \mathrm{support}(\delta)|\overline{B}} \mathrm{p}^{\min} && [\text{ by } (82.7)\ ]\\
&= \mathrm{m}^{\max} - \left|\mathrm{support}\,(\delta)\,|\overline{B}\right| \cdot \mathrm{p}^{\min}\\
&\leq \mathrm{m}^{\max} - \max\left[\mathrm{s}^{\min} - n, 0\right] \cdot \mathrm{p}^{\min} && [\text{ by } (82.12)\ ]
\end{aligned}$$

Also, we can bound the mass using our other already proven conditions.

$$\begin{aligned}
\|\delta_2\| &= \sum_{\sigma \in \mathrm{support}(\delta_2)} \delta_2(\sigma)\\
&\leq \sum_{\sigma \in \mathrm{support}(\delta_2)} \mathrm{p}_2^{\max} && [\text{ by } (82.11)\ ]\\
&= |\mathrm{support}\,(\delta_2)| \cdot \mathrm{p}_2^{\max}\\
&\leq \mathrm{s}_2^{\max} \cdot \mathrm{p}_2^{\max} && [\text{ by } (82.9)\ ]
\end{aligned}$$

Combining the bounds, we have half of our probability condition.

$$\|\delta_2\| \leq \mathrm{m}_2^{\max}$$

$$= \min\left[\mathrm{p}_2^{\max} \cdot \mathrm{s}_2^{\max}, \mathrm{m}^{\max} - \mathrm{p}^{\min} \cdot \max\left[\mathrm{s}^{\min} - n, 0\right]\right]$$

For the other half, we proceed similarly.

$$\begin{aligned}
\|\delta_2\| &= \|\delta\| - \sum_{\sigma \in \mathrm{support}(\delta)|\overline{B}} \delta(\sigma) \\
&\geq \mathrm{m}^{\min} - \sum_{\sigma \in \mathrm{support}(\delta)|\overline{B}} \delta(\sigma) && [\text{ by } (82.6)] \\
&\geq \mathrm{m}^{\min} - \sum_{\sigma \in \mathrm{support}(\delta)|\overline{B}} \mathrm{p}^{\max} && [\text{ by } (82.7)] \\
&= \mathrm{m}^{\min} - \left|\mathrm{support}\,(\delta)\,|\overline{B}\right| \cdot \mathrm{p}^{\max} \\
&\geq \mathrm{m}^{\min} - \min\left[\mathrm{s}^{\max}, \overline{n}\right] \cdot \mathrm{p}^{\max} && [\text{ by } (82.12)]
\end{aligned}$$

And likewise another bound using our other conditions.

$$\begin{aligned}
\|\delta_2\| &= \sum_{\sigma \in \mathrm{support}(\delta_2)} \delta_2(\sigma) \\
&\geq \sum_{\sigma \in \mathrm{support}(\delta_2)} \mathrm{p}_2^{\min} && [\text{ by } (82.11)] \\
&= \left|\mathrm{support}\,(\delta_2)\right| \cdot \mathrm{p}_2^{\min} \\
&\geq \mathrm{s}_2^{\min} \cdot \mathrm{p}_2^{\min} && [\text{ by } (82.9)]
\end{aligned}$$

Combining the two bounds, we have the final element of our proof.

$$\begin{aligned}
\|\delta_2\| &\geq \mathrm{m}_2^{\min} \\
&= \max\left[\mathrm{p}_2^{\min} \cdot \mathrm{s}_2^{\min}, \mathrm{m}^{\min} - \mathrm{p}^{\max} \cdot \min\left[\mathrm{s}^{\max}, \overline{n}\right]\right]
\end{aligned}$$

□

□

B.6 Scalar product

Lemma 38. *If* $\delta_1 \in \gamma_{\mathbb{P}}(P_1)$ *then* $p \cdot \delta_1 \in \gamma_{\mathbb{P}}(p \cdot P_1)$.

Proof. By assumption we have the following.

$$\begin{aligned}
&\mathrm{support}\,(\delta_1) \subseteq \gamma_{\mathbb{C}}(C_1) \\
&\mathrm{s}_1^{\min} \leq \left|\mathrm{support}\,(\delta_1)\right| \leq \mathrm{s}_1^{\max} \\
&\mathrm{m}_1^{\min} \leq \|\delta_1\| \leq \mathrm{m}_1^{\max}
\end{aligned}$$

$$\forall \sigma \in \text{support}\,(\delta_1)\ .\ \mathrm{p}_1^{\min} \leq \delta_1(\sigma) \leq \mathrm{p}_1^{\max}$$

Let $\delta_2 = p \cdot \delta_1$ and $P_2 = p \cdot P_1$. Let us assume that $p \neq 0$. In this case we need to show the following.

$$\begin{aligned}
&\text{support}\,(\delta_1) = \text{support}\,(\delta_2) \subseteq \gamma_{\mathbb{C}}(C_2) = \gamma_{\mathbb{C}}(C_1)\\
&\mathrm{s}_1^{\min} = \mathrm{s}_2^{\min} \leq |\text{support}\,(\delta_2)| = |\text{support}\,(\delta_2)| \leq \mathrm{s}_2^{\max} = \mathrm{s}_1^{\max}\\
&p \cdot \mathrm{m}_1^{\min} = \mathrm{m}_2^{\min} \leq \|\delta_1\| \leq \mathrm{m}_2^{\max} = p \cdot \mathrm{m}_1^{\max}\\
&\forall \sigma \in \text{support}\,(\delta_2)\ .\\
&\quad p \cdot \mathrm{p}_1^{\min} = \mathrm{p}_2^{\min} \leq \delta_2(\sigma) \leq \mathrm{p}_2^{\max} = p \cdot \mathrm{p}_1^{\max}
\end{aligned}$$

The first two conditions are trivially satisfied given the lack of change in the various parameters. For the mass condition, note that $\|\delta_2\| = \sum_\sigma \delta_2(\sigma) = \sum_\sigma p \cdot \delta_1(\sigma) = p \cdot \|\delta_1\|$. The probability condition is also trivially satisfied as $\delta_2(\sigma) = p \cdot \delta_1(\sigma)$.

In the case that $p = 0$, the abstract scalar product is defined with $\mathrm{s}_2^{\min} = \mathrm{s}_2^{\max} = \mathrm{p}_2^{\min} = \mathrm{p}_2^{\max} = \mathrm{m}_2^{\min} = \mathrm{m}_2^{\max} = 0$ and $C_2 = \emptyset_{\mathcal{C}}$. In this case note that $\text{support}\,(\delta_2) = \emptyset = \gamma_{\mathbb{C}}(\emptyset_{\mathcal{C}})$, and thus the conditions hold trivially. □

B.7 Uniform

Lemma 83 (Soundness of Uniform)**.** *If* $\delta \in \gamma_{\mathbb{P}}(P)$ *and* $S = \mathsf{uniform}\ x\ n_1\ n_2$ *then* $[\![S]\!]\delta \in \gamma_{\mathbb{P}}(\langle\!\langle S \rangle\!\rangle\ P)$.

Proof. Recall the semantics of the statement.

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

The distribution δ_2 is defined as follows.

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

The abstract semantics are similar.

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

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

By construction, we have $\delta_2 \in P_2$ thus the lemma follows from Lemma 28 (Soundness of Forget) and Lemma 34 (Soundness of Product). □

B.8 While loops

Definition 84. First we have some preliminary definitions. Given some set of variables, we have the following, where each distribution or state in each statement is understood to be defined over the same set of variables.

- Two distributions are *ordered*, or $\delta_1 \leq \delta_2$ iff for every state σ, $\delta_1(\sigma) \leq \delta_2(\sigma)$.
- Two probabilistic polyhedra are *ordered*, or $P_1 \sqsubseteq_{\mathbb{P}} P_2$ iff for every $\delta_1 \in \gamma_{\mathbb{P}}(P_1)$, there exists $\delta_2 \in \gamma_{\mathbb{P}}(P_2)$ with $\delta_1 \leq \delta_2$.
- The *zero distribution* δ is the unique distribution with $\delta(\sigma) = 0$ for every σ. We will use $0_{\mathbf{Dist}}$ to refer to this distribution.
- A *zero probabilistic polyhedron* P is one whose concretization contains only the zero distribution, that is $\gamma_{\mathbb{P}}(P) = \{0_{\mathbf{Dist}}\}$. We write $\mathit{iszero}(P)$ when P is a zero probabilistic polyhedron.

Lemma 85. *Let P_i be consistent probabilistic polyhedra, that is, $\gamma_{\mathbb{P}}(P_i) \neq \emptyset$. Then, $P_1 + P_2 \sqsubseteq_{\mathbb{P}} P_1$ iff $\mathit{iszero}(P_2)$.*

Proof. In the forward direction, we have $P_1 + P_2 \sqsubseteq_{\mathbb{P}} P_1$. Now, let us consider a P_2 with not $\mathit{iszero}(P_2)$. Thus there is $\delta_2 \in \gamma_{\mathbb{P}}(P_2)$ with $\|\delta_2\| > 0$. Let $\delta_1 \in \gamma_{\mathbb{P}}(P_1)$ be the distribution in $\gamma_{\mathbb{P}}(P_1)$ maximizing mass, that is $\|\delta_1\| \geq \|\delta_1'\|$ for every $\delta_1' \in \gamma_{\mathbb{P}}(P_1)$. By Lemma 33, $\delta_1 + \delta_2 \in \gamma_{\mathbb{P}}(P_1 + P_2)$ and by the definition of $P_1 + P_2 \sqsubseteq_{\mathbb{P}} P_1$, there must be $\delta_3 \in \gamma_{\mathbb{P}}(P_1)$ with $\delta_1 + \delta_2 \leq \delta_3$. Thus $\|\delta_3\| \geq \|\delta_1 + \delta_2\| = \|\delta_1\| + \|\delta_2\| > \|\delta_1\|$. This contradicts that δ_1 was mass maximizing in $\gamma_{\mathbb{P}}(P_1)$.

In the backward direction, our definition of abstract plus makes $P_1 + P_2$ identical to P_1. Thus $P_1 + P_2 = P_1 \sqsubseteq_{\mathbb{P}} P_1$. □

Definition 86. Given a statement $S = \mathsf{while}\ B\ \mathsf{do}\ S'$, a distribution δ and a probabilistic polyhedron P, let us define a few useful items.

- $\omega(f) \stackrel{\text{def}}{=} \lambda\delta.\ f([\![S']\!](\delta \wedge B)) + \delta \wedge \neg B$
- $\delta_1 \stackrel{\text{def}}{=} \delta$
- $\delta_{i+1} \stackrel{\text{def}}{=} [\![S']\!](\delta_i \wedge B)$
- $\Delta_n \stackrel{\text{def}}{=} \sum_{i=1}^{n} (\delta_i \wedge \neg B)$
- $\bot_{\mathbf{Dist}}$ is the function that takes in any distribution and produces the zero distribution $0_{\mathbf{Dist}}$, that is $\bot_{\mathbf{Dist}}(\delta) = 0_{\mathbf{Dist}}$.

Similarly we have the abstract versions of the definitions.

- $\Omega(F) \stackrel{\text{def}}{=} \lambda P.\ F(\langle\!\langle S' \rangle\!\rangle\ (P \wedge B)) + P \wedge \neg B$
- $P_1 \stackrel{\text{def}}{=} P$

- $P_{i+1} \stackrel{\text{def}}{=} \langle\langle S' \rangle\rangle (P_i \wedge B)$
- $\Phi_n \stackrel{\text{def}}{=} \sum_{i=1}^{n} (P_i \wedge \neg B)$
- $\bot_{\mathbb{P}}$ is a function that takes in any probabilistic polyhedron and produces a zero probabilistic polyhedron, that is $iszero(\bot_{\mathbb{P}}(P))$ for every P.

The semantics of while loops are defined as such:

$$[\![S]\!] = [\![\mathsf{while}\ B\ \mathsf{do}\ S']\!] = \mathrm{lfp}(\omega)$$

$$\langle\langle S \rangle\rangle = \langle\langle \mathsf{while}\ B\ \mathsf{do}\ S' \rangle\rangle = \mathrm{lfp}(\Omega)$$

While such definitions are of theoretical interest, they are not particularly useful for implementations, given our lack of a widening operator. Thus, our security checks will always be conditioned on termination of the abstract interpretation, defined below. We show that termination of the abstract interpretation implies termination of all corresponding concrete executions. This is crucial, as our concrete semantics (due to Clarkson et al. [24]) assumes termination to avoid leaks. To make this termination condition explicit, we provide an alternate concrete semantics for terminating while loops and show that this gives results equivalent to those of the original semantics.

Definition 87. The termination of $[\![S]\!]\delta$ is defined as follows.

- If S is an elementary statement (assignment, skip, uniform), then $[\![S]\!]\delta$ terminates.
- If S is a sequence, if statement, or a probabilistic choice statement, then $[\![S]\!]\delta$ terminates iff the various evaluations steps to evaluate S terminate. This depends on the statement type, for $S = S_1\ ;\ S_2$, for example, it means that $[\![S_1]\!]\delta$ terminates and so does $[\![S_2]\!]([\![S_1]\!]\delta)$.
- If $S = \mathsf{while}\ B\ \mathsf{do}\ S_1$ is a while statement, then $[\![S]\!]\delta$ terminates iff there exists n with $\delta_n = 0_{\mathbf{Dist}}$ and the evaluation steps as per definition of δ_i terminate for all i up to n.

The termination of $\langle\langle S \rangle\rangle P$ is framed similarly, except in the while case, we require the existence of n with $iszero(P_n)$ and the termination of the abstract evaluations as in the definitions of P_i for all i up to n.

The Δ_i and Φ_i capture exactly the concrete and abstract values when termination is assumed.

$$\begin{aligned}\omega^1(\bot_{\mathbf{Dist}})(\delta) &= \delta \wedge \neg B \\ &= \Delta_1\end{aligned}$$

$$\omega^2(\bot_{\mathbf{Dist}})(\delta) = ([\![S']\!](\delta \wedge B)) \wedge \neg B + \delta \wedge \neg B$$

$$= \delta_2 \wedge \neg B + \delta_1 \wedge \neg B$$
$$= \Delta_2$$

$$\begin{aligned}\omega^i(\bot_{\mathbf{Dist}})(\delta) &= \omega^{i-1}(\bot_{\mathbf{Dist}})([\![S']\!]\delta \wedge B) + \delta \wedge \neg B \\ &= \Delta_{i-1} + \delta \wedge \neg B \\ &= \Delta_i\end{aligned}$$

Likewise $\Omega^i(\bot_{\mathbb{P}})(P) = \Phi_i$.

Definition 88. Terminating semantics of while loops are as follows.

$$[\![\mathsf{while}\ B\ \mathsf{do}\ S_1]\!]\delta = \Delta_n$$

Where n is the least index with $\delta_n = 0_{\mathbf{Dist}}$. Likewise for the abstract case.

$$\langle\!\langle \mathsf{while}\ B\ \mathsf{do}\ S_1 \rangle\!\rangle\, P = \Phi_n$$

Where n is the least index with $\mathit{iszero}(P_n)$.

Lemma 89. *If* $[\![\mathsf{while}\ B\ \mathsf{do}\ S_1]\!]\delta$ *is terminating, then* $\Delta_n = (\mathrm{lfp}(\omega))\,(\delta)$*, noting that* $\mathrm{lfp}(\omega)$ *is the original semantics of a while loop.*

Proof. As noted in [80], the evaluation of a while loop on a distribution is equal to an infinite sum:

$$[\![S]\!]\delta = \sum_{i=1}^{\infty} \delta_i \wedge \neg B$$

By the termination assumption we have an n with $\delta_n = 0_{\mathbf{Dist}}$. Now, since $\delta_{i+1} = [\![S']\!]\delta_i \wedge B$ hence the mass of δ_{i+1} cannot exceed the mass of δ_i, it is the case that if $\delta_n = 0_{\mathbf{Dist}}$, then $\delta_i = 0_{\mathbf{Dist}}$ for every $i \geq n$. Thus the infinite sum above can be shortened.

$$\begin{aligned}[\![S]\!]\delta &= \sum_{i=1}^{\infty} \delta_i \wedge \neg B \\ &= \sum_{i=1}^{n} \delta_i \wedge \neg B + 0_{\mathbf{Dist}} \\ &= \Delta_n\end{aligned}$$

□

Remark 90 (Composition of Termination)**.** If $\langle\!\langle S \rangle\!\rangle\, P$ terminates, then so must the evaluation of all of its components as defined by the semantics. This is immediate from the definition of termination.

B.9 Soundness of Abstraction

Theorem 27. *For all P, δ, if $\delta \in \gamma_{\mathbb{P}}(P)$ and $\langle\!\langle S \rangle\!\rangle\, P$ terminates, then $[\![S]\!]\delta$ terminates and $[\![S]\!]\delta \in \gamma_{\mathbb{P}}(\langle\!\langle S \rangle\!\rangle\, P)$.*

Proof. Let us show this by structural induction on S. As base cases we have the following.

- $S = \mathsf{skip}$. In this case we have $[\![S]\!]\delta = \delta$ and $\langle\!\langle S \rangle\!\rangle\, P = P$. Termination is not an issue and the claim holds by assumption.
- $S = x := E$. Here non-termination is also not a possibility given non-recursive definition of assignment. Also, by Lemma 30 (Soundness of Assignment) we have $[\![S]\!]\delta \in \gamma_{\mathbb{P}}(\langle\!\langle S \rangle\!\rangle\, P)$.
- $S = \mathsf{uniform}\ x\ n_1\ n_2$. Again, there is no termination issues and the claim follows from Lemma 83 (Soundness of Uniform).

Let us thus assume the claim for sub-statements of S and show it for S itself. Note that the inductive assumption is general for all δ, P with $\delta \in \gamma_{\mathbb{P}}(P)$. S has several cases.

- $S = S_1\ ;\ S_2$. By the termination remark, we know $\langle\!\langle S_1 \rangle\!\rangle\, P$ terminates and thus by induction $[\![S_1]\!]\delta$ terminates and is in $\gamma_{\mathbb{P}}(\langle\!\langle S_1 \rangle\!\rangle\, P)$. We then apply induction once more with S_2 to find that $[\![S_2]\!]([\![S_1]\!]\delta) = [\![S]\!]\delta$ terminates and is in $\gamma_{\mathbb{P}}(\langle\!\langle S_2 \rangle\!\rangle\, (\langle\!\langle S_1 \rangle\!\rangle\, P)) = \gamma_{\mathbb{P}}(\langle\!\langle S \rangle\!\rangle\, P)$.
- $S = \mathsf{if}\ B\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2$. By the termination remark, we know that $\langle\!\langle S_1 \rangle\!\rangle\, (P \wedge B)$ and $\langle\!\langle S_2 \rangle\!\rangle\, (P \wedge \neg B)$ terminate. By Lemma 36 (Soundness of Conditional) we have $\delta \wedge B \in \gamma_{\mathbb{P}}(P \wedge B)$ and $\delta \wedge \neg B \in \gamma_{\mathbb{P}}(P \wedge \neg B)$. We thus apply induction to both sub-statements to conclude that $[\![S_1]\!](\delta \wedge B)$ and $[\![S_2]\!](\delta \wedge \neg B)$ both terminate and are in $\gamma_{\mathbb{P}}(\langle\!\langle S_1 \rangle\!\rangle\, (P \wedge B))$ and $\gamma_{\mathbb{P}}(\langle\!\langle S_2 \rangle\!\rangle\, (P \wedge \neg B))$ respectively. Finally we apply Lemma 33 (Soundness of Plus) to conclude $[\![S]\!]\delta = [\![S_1]\!](\delta \wedge B) + [\![S_2]\!](\delta \wedge \neg B) \in \gamma_{\mathbb{P}}(\langle\!\langle S_1 \rangle\!\rangle\, (P \wedge B) + \langle\!\langle S_2 \rangle\!\rangle\, (P \wedge \neg B)) = \gamma_{\mathbb{P}}(\langle\!\langle S \rangle\!\rangle\, P)$.
- $S = \mathsf{pif}\ p\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2$. This case is identical to the previous except we use Lemma 38 (Soundness of Scalar Product) in place of Lemma 36 (Soundness of Conditional).
- $S = \mathsf{while}\ B\ \mathsf{do}\ S_1$.

For this last case we must first show a claim. For every δ', P' with $\delta' \in \gamma_{\mathbb{P}}(P')$, and every i we have the following.

$$\delta'_i \in \gamma_{\mathbb{P}}(P'_i) \tag{90.1}$$
$$\Delta'_i \in \gamma_{\mathbb{P}}(\Phi'_i) \tag{90.2}$$

Let us show this claim by induction on i. As the base case we have $\delta'_1 = \delta'$ and $\Delta'_1 = \delta'_1 \wedge \neg B = \delta' \wedge \neg B$. Also $P'_1 = P'$ and $\Phi'_1 = P'_1 \wedge \neg B = P' \wedge \neg B$. By assumption we had $\delta' \in \gamma_{\mathbb{P}}(P')$ so the first part of our claim holds trivially. For the other we apply Lemma 36 (Soundness of Conditional) to conclude $\Delta'_1 \in \gamma_{\mathbb{P}}(\Phi'_1)$.

Let us assume the claim holds for all $i < n$ and show that it holds for n.

We have, by definition, $\delta'_n = [\![S_1]\!]\left(\delta'_{n-1} \wedge B\right)$ and $P'_n = \langle\!\langle S_1 \rangle\!\rangle \left(P'_{n-1} \wedge B\right)$. By the (inner) induction assumption, we have $\delta'_{n-1} \in \gamma_{\mathbb{P}}(P'_{n-1})$ so by Lemma 36 we have $\delta'_{n-1} \wedge B \in \gamma_{\mathbb{P}}(P'_{n-1} \wedge B)$. Since $\langle\!\langle S \rangle\!\rangle\, P$ terminates, then so must $\langle\!\langle S_1 \rangle\!\rangle\, P'_{n-1} \wedge B$ by the termination remark. Thus, by the (outer) induction hypothesis, we know that $[\![S_1]\!]\left(\delta'_{n-1} \wedge B\right) = \delta'_n \in \gamma_{\mathbb{P}}(\langle\!\langle S_1 \rangle\!\rangle \left(P'_{n-1} \wedge B\right)) = \gamma_{\mathbb{P}}(P'_n)$.

For the second part of the claim, we have $\Delta'_n = \Delta'_{n-1} + \delta'_n \wedge \neg B$ and $\Phi'_n = \Phi'_{n-1} + P'_n \wedge \neg B$. By (inner) induction we know $\Delta'_{n-1} \in \gamma_{\mathbb{P}}(\Phi'_{n-1})$. By the first part of the claim above we know $\delta'_n \in \gamma_{\mathbb{P}}(P'_n)$ so by Lemma 36 (Soundness of Conditional) we have $\delta'_n \wedge \neg B \in \gamma_{\mathbb{P}}(P'_n \wedge \neg B)$. Now we apply Lemma 33 (Soundness of Plus) to conclude $\Delta'_n = \Delta'_{n-1} + \delta'_n \wedge \neg B \in \gamma_{\mathbb{P}}(\Phi'_{n-1} + P'_n \wedge \neg B) = \gamma_{\mathbb{P}}(\Phi'_n)$, finishing the claim.

Now, since $\langle\!\langle S \rangle\!\rangle\, P'$ terminates, it must be that $\langle\!\langle S \rangle\!\rangle\, P' = \Phi'_n$ for some n, according to the terminating semantics. Furthermore we have the following, also by definition of termination.

$$\mathit{iszero}(P'_n \wedge \neg B) \tag{90.3}$$

This is the case since $\mathit{iszero}(P'_n)$ and the fact that the conditioning operation preserves $\mathit{iszero}(\cdot)$.

Therefore by (90.1) we can conclude that $\delta_n = 0_{\mathbf{Dist}}$ as $\gamma_{\mathbb{C}}(P_n) = \{0_{\mathbf{Dist}}\}$. Therefore $[\![S]\!]\delta$ terminates and by Lemma 89 we have $[\![S]\!]\delta = \Delta_n$. The issue of whether n is the least index with $\delta_n = 0_{\mathbf{Dist}}$ is irrelevant as if it were not, the larger sum includes only additional $0_{\mathbf{Dist}}$ terms. By (90.2), we have $\Delta_n \in \gamma_{\mathbb{P}}(\Phi_n)$ and we are done as $\Phi_n = \langle\!\langle S \rangle\!\rangle\, P$ according to the terminating semantics. □

B.10 Normalization

Lemma 40. *If $\delta_1 \in \gamma_{\mathbb{P}}(P_1)$ then $\mathit{normal}(\delta_1) \in \gamma_{\mathbb{P}}(\mathit{normal}(P_1))$.*

Proof. By assumption we have the following.

$$\begin{aligned}
&\text{support}\,(\delta_1) \subseteq \gamma_{\mathbb{C}}(C_1)\\
&\mathrm{s}_1^{\min} \leq |\text{support}\,(\delta_1)| \leq \mathrm{s}_1^{\max}\\
&\mathrm{m}_1^{\min} \leq \|\delta_1\| \leq \mathrm{m}_1^{\max}\\
&\forall \sigma \in \text{support}\,(\delta_1)\ .\ \mathrm{p}_1^{\min} \leq \delta_1(\sigma) \leq \mathrm{p}_1^{\max}
\end{aligned}$$

If $\|\delta_1\| = 0$ then $\text{normal}(\delta_1)$ is undefined. Since $\mathrm{m}_1^{\min} \leq \|\delta_1\|$, it must be that $\mathrm{m}_1^{\min} = 0$ as well, and thus $\text{normal}(P_1)$ is likewise undefined.

Let us now assume $\|\delta_1\| > 0$. Let $\delta_2 = \text{normal}(\delta_1)$ and $P_2 = \text{normal}(P_1)$. We have two sub-cases, either $\mathrm{m}_1^{\min} = 0$ or $\mathrm{m}_1^{\min} > 0$. In the first sub case, P_2 is defined as follows.

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

Since $\text{support}\,(\delta_2) = \text{support}\,(\delta_1)$, it must be that $\text{support}\,(\delta_2) \subseteq \gamma_{\mathbb{C}}(C_2)$ as $C_2 = C_1$. Likewise, the number of support point is is unchanged in both the concrete operation and the abstract one, hence the number of support points condition for soundness are satisfied as well. Also, the probability per point in any distribution does not exceed 1 hence the $\mathrm{p}_2^{\max}$ condition is satisfied. As for $\mathrm{p}_2^{\min}$, note that if $\sigma \in \text{support}\,(\delta_2) = \text{support}\,(\delta_1)$, we have $\delta_2(\sigma) = \delta_1(\sigma)/\|\delta_1\| \geq \mathrm{p}_1^{\min}/\|\delta_1\| \geq \mathrm{p}_1^{\min}/\mathrm{m}_1^{\max}$, by assumption. Finally, $\|\delta_2\| = 1$ hence the $\mathrm{m}_2^{\min}$ and $\mathrm{m}_2^{\max}$ conditions are satisfied.

In the other case, we have $\mathrm{p}_1^{\min} > 0$. Here P_2 is defined as follows.

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

The support, support points, total mass, and $\mathrm{p}_2^{\min}$ conditions are satisfied for the same reason as in the previous case. For $\mathrm{p}_2^{\max}$, let $\sigma \in \text{support}\,(\delta_2) = \text{support}\,(\delta_1)$ and we have the following.

$$\begin{aligned} \delta_2(\sigma) &= \delta_1(\sigma)/\|\delta_1\| \\ &\leq \mathrm{p}_1^{\max}/\|\delta_1\| \\ &\leq \mathrm{p}_1^{\max}/\mathrm{m}_1^{\min} \end{aligned}$$

□

B.11 Security

Before we prove the security theorem, let us show that the definition of abstract conditioning on a state is sound.

Lemma 91. *If $\delta \in \gamma_{\mathbb{P}}(P)$ and $\sigma_V \in \boldsymbol{State}_V$ with $V \subseteq fv(\delta)$ then $\delta \wedge \sigma_V \in \gamma_{\mathbb{P}}(P \wedge \sigma_V)$*

Proof. Recall the definition of $P \wedge \sigma_V$.

$$P \wedge \sigma_V = P \wedge B$$

With $B = \bigwedge_{x \in V} (x = \sigma_V(x))$. Let us show that $\delta \wedge \sigma_V = \delta \wedge B$, the rest will follow from Lemma 36.

The definition of $\delta \wedge \sigma_V$ is as follows.

$$\delta \wedge \sigma = \lambda\sigma.\ \textbf{if}\ \sigma \restriction V = \sigma_V\ \textbf{then}\ \delta(\sigma)\ \textbf{else}\ 0$$

Meanwhile, $\delta \wedge B$ is defined as follows.

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

The correspondence is immediate as $[\![B]\!]\sigma = \ \textbf{true}\ $ if and only if $\sigma \upharpoonright V = \sigma_V$ as per construction of B. □

Theorem 43. *Let δ be an attacker's initial belief. If $\delta \in \gamma_{\mathbb{P}}(P)$ and $tsecure_t(S, P)$, then S is threshold secure for threshold t when evaluated with initial belief δ.*

Proof. Let us consider the contrapositive. That is, assuming $\delta \in \gamma_{\mathbb{P}}(P)$, if S is not threshold secure for t and initial belief δ, then it is not the case that $tsecure_t(S, P)$.

Let $\delta_2 = [\![S]\!]\delta$ and $\delta_3 = \delta_2 \upharpoonright L$. Since S is not secure, we have $\sigma_L \in \text{support}\,(\delta_3)$ and $\sigma'_H \in \mathbf{State}_H$ with $(\text{normal}((\delta_2 \wedge \sigma_L) \upharpoonright H))(\sigma'_H) > t$. This implies that $(\delta_2 \wedge \sigma_L) \upharpoonright H \neq 0_{\mathbf{Dist}}$ and therefore $\delta_2 \wedge \sigma_L \neq 0_{\mathbf{Dist}}$ as projection preserves mass.

If $\langle\!\langle S \rangle\!\rangle\, P$ is not terminating, then we are done as termination is a condition for $tsecure_t(S, P)$. So let us assume $\langle\!\langle S \rangle\!\rangle\, P$ is terminating. Let $P_2 = \langle\!\langle S \rangle\!\rangle\, P$. By Theorem 27, we have $\delta_2 \in \gamma_{\mathbb{P}}(P_2)$. By Lemma 91, $\delta_2 \wedge \sigma_L \in \gamma_{\mathbb{P}}(P_2 \wedge \sigma_L)$. Therefore not $iszero(P \wedge \sigma_L)$ as $\delta_2 \wedge \sigma_L \neq 0_{\mathbf{Dist}}$. Continuing, by Lemma 65, $(\delta_2 \wedge \sigma_L) \upharpoonright H \in \gamma_{\mathbb{P}}((P_2 \wedge \sigma_L) \upharpoonright H)$ and finally, by Lemma 40, we have $\text{normal}((\delta_2 \wedge \sigma_L) \upharpoonright H) \in \gamma_{\mathbb{P}}(\text{normal}((P_2 \wedge \sigma_L) \upharpoonright H))$. Let $\delta_4 = \text{normal}((\delta_2 \wedge \sigma_L) \upharpoonright H)$ and $P_4 = \text{normal}((P_2 \wedge \sigma_L) \upharpoonright H)$. Since $\sigma'_H \in \text{support}\,(\delta_4)$, we have $\delta_4(\sigma'_H) \leq \mathrm{p}_4^{\max}$. Since $\delta_4(\sigma'_H) > t$, we have $t < \mathrm{p}_4^{\max}$.

Also, let $P_3 = P_2 \upharpoonright L$. By Lemma 65, we have $\delta_3 \in \gamma_{\mathbb{P}}(P_3)$ so $\sigma_L \in \gamma_{\mathbb{C}}(C_3)$. We already had that not $iszero(P \wedge \sigma_L)$ above. Thus σ_L is indeed the witness to the failure of $tsecure_t(S, P_1)$. □

Appendix C

Soundness proofs for $\mathcal{P}_n(\mathbb{P})$

C.1 Useful Lemmas

We begin with some lemmas that give properties of the concretization function for powersets of probabilistic polyhedra and addition on sets.

Lemma 92. *If* $\Delta = \Delta_1 \cup \Delta_2$ *then* $\gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta) = \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_1) + \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_2)$.

Proof. From the definition of $\gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta)$ we have

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

Applying $\Delta = \Delta_1 \cup \Delta_2$ and associativity of $+$ allows us to conclude

$$\gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta) = \sum_{P_1 \in \Delta_1} \gamma_{\mathbb{P}}(P_1) + \sum_{P_1 \in \Delta_2} \gamma_{\mathbb{P}}(P_2)$$

Again applying the definition of $\gamma_{\mathcal{P}_n(\mathbb{P})}(\ldots)$, we have

$$\gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta) = \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_1) + \gamma_{\mathcal{P}_n(\mathbb{P})}(P_2)$$

□

Lemma 93. *If* $D_1 \subseteq D_1'$ *and* $D_2 \subseteq D_2'$ *then* $D_1 + D_2 \subseteq D_1' + D_2'$.

Proof. According to the definition of addition for sets, we have

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

Consider some $\delta \in D_1 + D_2$. We have $\delta = \delta_1 + \delta_2$ with $\delta_1 \in D_1$ and $\delta_2 \in D_2$. Since $D_1 \subseteq D_1'$, we have $\delta_1 \in D_1'$. Similarly, since $D_2 \subseteq D_2'$, we have $\delta_2 \in D_2'$. Since

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

we have $\delta = \delta_1 + \delta_2 \in D_1' + D_2'$. □

C.2 Bounding Operation

Lemma 48 (Soundness of Bounding Operation)**.** $\gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta) \subseteq \gamma_{\mathcal{P}_n(\mathbb{P})}(\lfloor \Delta \rfloor_n)$.

Proof. According to Definition 47, there are two cases for $\lfloor \Delta \rfloor_n$. If $|\Delta| \leq n$ then we have $\lfloor \Delta \rfloor_n = \Delta$ and thus $\gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta) = \gamma_{\mathcal{P}_n(\mathbb{P})}(\lfloor \Delta \rfloor_n)$.

If $|\Delta| > n$, we reason by induction on $|\Delta|$. Since $n \geq 1$, we have that $|\Delta| \geq 2$ and thus we can partition Δ into $\Delta_1 \cup \{P_1, P_2\}$. Applying Definition 47 we then have $\lfloor \Delta \rfloor_n = \lfloor \Delta_1 \cup \{P_1 + P_2\} \rfloor_n$. The inductively-passed set has size one less than the original, allowing us to apply the inductive hypothesis to conclude the following.

$$\gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_1 \cup \{P_1 + P_2\}) \subseteq \gamma_{\mathcal{P}_n(\mathbb{P})}(\lfloor \Delta_1 \cup \{P_1 + P_2\} \rfloor_n)$$

Our conclusion will follow provided we can show

$$\gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta) \subseteq \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_1 \cup \{P_1 + P_2\})$$

Lemma 92 allows us to rewrite this to

$$\gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta) \subseteq \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_1) + \gamma_{\mathcal{P}_n(\mathbb{P})}(\{P_1 + P_2\}) \tag{93.1}$$

We have $\Delta = \Delta_1 \cup \{P_1, P_2\}$ and thus by Lemma 92 we have

$$\gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta) = \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_1) + \gamma_{\mathcal{P}_n(\mathbb{P})}(\{P_1, P_2\})$$

By Lemma 93, we will have (93.1) provided we can show

$$\gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_1) \subseteq \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_1)$$

which is immediate, and

$$\gamma_{\mathcal{P}_n(\mathbb{P})}(\{P_1, P_2\}) \subseteq \gamma_{\mathcal{P}_n(\mathbb{P})}(\{P_1 + P_2\})$$

The latter is proven by applying the definitions of $\gamma_{\mathcal{P}_n(\mathbb{P})}(\{P_1, P_2\})$ and $\gamma_{\mathcal{P}_n(\mathbb{P})}(\{P_1 + P_2\})$, resulting in a goal of

$$\gamma_{\mathbb{P}}(P_1) + \gamma_{\mathbb{P}}(P_2) \subseteq \gamma_{\mathbb{P}}(P_1 + P_2)$$

which follows directly from Lemma 33. □

C.3 Distributive Operations

The soundness proofs for the majority of the operations on elements of $\mathcal{P}_n(\mathbb{P})$ are sound for exactly the same reason: the operations distribute over $+$, allowing us to reduce soundness for the powerset case to soundness for the case of a single probabilistic polyhedron. We start with the Lemma that is used to structure such a proof.

Lemma 94. *Consider* $f : \mathbb{P} \to \mathbb{P}$, $F : \mathcal{P}_n(\mathbb{P}) \to \mathcal{P}_n(\mathbb{P})$, *and* $f^\flat : \textbf{\textit{Dist}} \to \textbf{\textit{Dist}}$. *Suppose the following all hold for all* δ_i, P_i.

1. $f^\flat(\delta_1 + \ldots + \delta_n) = f^\flat(\delta_1) + \ldots + f^\flat(\delta_n)$
2. $F(\{P_1, \ldots, P_n\}) = \{f(P_1), \ldots, f(P_n)\}$
3. $\delta \in \gamma_{\mathbb{P}}(P) \Rightarrow f^\flat(\delta) \in \gamma_{\mathbb{P}}(f(P))$

Then $\delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta)$ *implies* $f^\flat(\delta) \in \gamma_{\mathcal{P}_n(\mathbb{P})}(F(\Delta))$.

Proof. Suppose $\delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta)$ and $\Delta = \{P_1, \ldots, P_n\}$. We have the following by definition of $\gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta)$.

$$\gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta) = \gamma_{\mathbb{P}}(P_1) + \ldots + \gamma_{\mathbb{P}}(P_n)$$

Applying the definition of addition on sets, we obtain

$$\gamma_{\mathbb{P}}(P_1) + \ldots + \gamma_{\mathbb{P}}(P_n) = \{\delta_1 + \ldots + \delta_n \mid \delta_i \in \gamma_{\mathbb{P}}(P_i)\}$$

Thus, we have that $\delta = \delta_1 + \ldots + \delta_n$ where $\delta_i \in \gamma_{\mathbb{P}}(P_i)$. By premise 3 we then have $f^\flat(\delta_i) \in \gamma_{\mathbb{P}}(f(P_i))$ for all i.

We now consider $\gamma_{\mathcal{P}_n(\mathbb{P})}(F(\Delta))$. By premise 2 we have that this is $\gamma_{\mathcal{P}_n(\mathbb{P})}(\{f(P_1), \ldots, f(P_n)\})$. Applying the definition of $\gamma_{\mathcal{P}_n(\mathbb{P})}$, this is equal to $\gamma_{\mathbb{P}}(f(P_1)) + \ldots + \gamma_{\mathbb{P}}(f(P_n))$.

Expanding the definition of $+$ for sets, we have that

$$\gamma_{\mathcal{P}_n(\mathbb{P})}(F(\Delta)) = \{\delta_1 + \ldots + \delta_n \mid \delta_i \in \gamma_{\mathbb{P}}(f(P_i))\}$$

Since $f^\flat(\delta_i) \in \gamma_{\mathbb{P}}(f(P_i))$ for all i we have $\sum_i (f^\flat(\delta_i)) \in \gamma_{\mathcal{P}_n(\mathbb{P})}(F(\Delta))$ and thus, by premise 1 we have $f^\flat(\sum_i \delta_i) \in \gamma_{\mathcal{P}_n(\mathbb{P})}(F(\Delta))$ and thus $f^\flat(\delta) \in \gamma_{\mathcal{P}_n(\mathbb{P})}(F(\Delta))$ as desired. □

Lemma 95 (Soundness of Forget)**.** *If* $\delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta)$ *then* $\mathrm{f}_y(\delta) \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\mathrm{f}_y(\Delta))$.

Proof. We will apply Lemma 94 with $f^\flat = \lambda\delta.\ \delta \upharpoonright (fv(\delta) - \{y\})$, $f = \lambda P.\ \mathrm{f}_y(P)$, and $F = \lambda\Delta.\ \mathrm{f}_y(\Delta)$. Lemma 28 gives us premise 3. The definition of $\mathrm{f}_y(\Delta)$ satisfies premise 2. Let $V = fv(\delta) - \{y\}$. It remains to show premise 1, which states

$$(\delta_1 + \ldots + \delta_n) \upharpoonright V = \delta_1 \upharpoonright V + \ldots + \delta_n \upharpoonright V$$

We show this for the binary case, from which the n-ary version above follows.

$$(\delta_1 + \delta_2) \upharpoonright V = \delta_1 \upharpoonright V + \delta_2 \upharpoonright V$$

Expanding the definition of projection, we then obtain the following goal.

$$\lambda\sigma_V \in \mathbf{State}_V.\ \sum_{\sigma' | (\sigma' \restriction V = \sigma_V)} (\delta_1 + \delta_2)(\sigma') = \left(\lambda\sigma_V \in \mathbf{State}_V.\ \sum_{\sigma' | (\sigma' \restriction V = \sigma_V)} \delta_1(\sigma')\right) + \left(\lambda\sigma_V \in \mathbf{State}_V.\ \sum_{\sigma' | (\sigma' \restriction V = \sigma_V)} \delta_2(\sigma')\right)$$

We can now apply the definition of $+$ for distributions to the right-hand side to obtain a goal of

$$\lambda\sigma_V \in \mathbf{State}_V.\ \sum_{\sigma' | (\sigma' \restriction V = \sigma_V)} (\delta_1 + \delta_2)(\sigma') = \lambda\sigma_V \in \mathbf{State}_V.\ \left(\sum_{\sigma' | (\sigma' \restriction V = \sigma_V)} \delta_1(\sigma') + \sum_{\sigma' | (\sigma' \restriction V = \sigma_V)} \delta_2(\sigma')\right)$$

These functions are equal if they give equal results for all inputs. Thus, we must show the following for all σ_V.

$$\sum_{\sigma' | (\sigma' \restriction V = \sigma_V)} (\delta_1 + \delta_2)(\sigma') = \left(\sum_{\sigma' | (\sigma' \restriction V = \sigma_V)} \delta_1(\sigma') + \sum_{\sigma' | (\sigma' \restriction V = \sigma_V)} \delta_2(\sigma')\right)$$

Finally, applying the definition of $+$ for distributions to the left-hand side of the equality yields

$$\sum_{\sigma' | (\sigma' \restriction V = \sigma_V)} \left(\delta_1(\sigma') + \delta_2(\sigma')\right) = \left(\sum_{\sigma' | (\sigma' \restriction V = \sigma_V)} \delta_1(\sigma') + \sum_{\sigma' | (\sigma' \restriction V = \sigma_V)} \delta_2(\sigma')\right)$$

This follows by associativity and commutativity of $+$. □

Lemma 96 (Soundness of Projection)**.** *If $\delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta)$ and $V \subseteq fv(\delta)$ then $\delta \restriction V \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta \restriction V)$.*

Proof. Inductive application of Lemma 95 (Soundness of Forget) as was the case in the base domain. □

Lemma 97 (Soundness of Assignment)**.** *If $\delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta)$ then $\delta\,[x \to E] \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta\,[x \to E])$.*

Proof. As in Lemma 95, we apply Lemma 94. We have premises 3 (by Lemma 30) and 2 (by definition) and must show premise 1. This means showing that

$$(\delta_1 + \delta_2)\,[x \to E] = \delta_1\,[x \to E] + \delta_2\,[x \to E]$$

Expanding the definition of assignment, we must show that the following

$$\lambda\sigma. \sum_{\tau \mid \tau[x \to [\![E]\!]\tau] = \sigma} (\delta_1 + \delta_2)(\tau)$$

is equal to

$$\left(\lambda\sigma. \sum_{\tau \mid \tau[x \to [\![E]\!]\tau] = \sigma} \delta_1(\tau)\right) + \left(\lambda\sigma. \sum_{\tau \mid \tau[x \to [\![E]\!]\tau] = \sigma} \delta_1(\tau)\right)$$

Again applying the definition of $+$ for distributions and using extensional equality for functions yields the following goal, which follows by associativity and commutativity of $+$.

$$\forall\sigma. \left(\sum_{\tau \mid \tau[x \to [\![E]\!]\tau] = \sigma} \Big(\delta_1(\tau) + \delta_2(\tau)\Big) = \sum_{\tau \mid \tau[x \to [\![E]\!]\tau] = \sigma} \delta_1(\tau) + \sum_{\tau \mid \tau[x \to [\![E]\!]\tau] = \sigma} \delta_1(\tau)\right)$$

□

Lemma 98 (Soundness of Scalar Product)**.** *If* $\delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta)$ *then* $p \cdot \delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(p \cdot \Delta)$.

Proof. This proof follows the same format as the others in this section. We apply Lemma 30 with the definition of scalar product for powersets and Lemma 38. We must show

$$p \cdot (\delta_1 + \delta_2) = p \cdot \delta_1 + p \cdot \delta_2$$

Expanding according to the definition of scalar product and $+$ for distributions, we obtain the following as a goal.

$$\lambda\sigma.\ p \cdot (\delta_1(\sigma) + \delta_2(\sigma)) = \lambda\sigma.\ p \cdot \delta_1(\sigma) + p \cdot \delta_2(\sigma)$$

The result follows by distributivity of $\cdot$ over $+$. □

Lemma 99 (Soundness of Conditioning)**.** *If* $\delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta)$ *then* $\delta \wedge B \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta \wedge B)$.

Proof. Again we apply Lemma 30, this time using Lemma 36 to satisfy premise 3. We let $f^\flat = \lambda\delta.\ \delta \wedge B$, $f = \lambda P.\ P \wedge B$, and $F = \lambda\Delta.\ \Delta \wedge B$. We must show

$$(\delta_1 + \delta_2) \wedge B = \delta_1 \wedge B + \delta_2 \wedge B$$

Applying the definition of conditioning and addition for distributions, we have to show the following for all σ.

$$\textbf{if } [\![B]\!]\sigma \textbf{ then } (\delta_1 + \delta_2)(\sigma) \textbf{ else } 0 = \big(\textbf{if } [\![B]\!]\sigma \textbf{ then } \delta_1(\sigma) \textbf{ else } 0\big) + \big(\textbf{if } [\![B]\!]\sigma \textbf{ then } \delta_2(\sigma) \textbf{ else } 0\big)$$

We proceed via case analysis. If $[\![B]\!]\sigma = \textbf{false}$ then we have $0 = 0 + 0$, which is a tautology. If $[\![B]\!]\sigma = \textbf{true}$, we have to show

$$(\delta_1 + \delta_2)(\sigma) = \delta_1(\sigma) + \delta_2(\sigma)$$

which follows directly from the definition of $+$ on distributions. □

C.4 Other Powerset Lemmas

We now show the lemmas for operations in the powerset domain that do not immediately follow from distributivity over plus of the operations in the base domain.

Lemma 100 (Soundness of Product)**.** *If* $\delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta)$ *and* $\delta' \in \gamma_{\mathcal{P}_n(\mathbb{P})}(P')$ *and* $fv(\Delta) \cap fv(P') = \emptyset$ *then* $\delta \times \delta' \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta \times P')$.

Proof. Let $\Delta = \{P_1, \ldots, P_n\}$. We first expand definitions in our goal, obtaining

$$\delta \times \delta' \in \gamma_{\mathbb{P}}(P_1 \times P') + \ldots + \gamma_{\mathbb{P}}(P_n \times P')$$

Applying the definition of addition for sets, we obtain a goal of

$$\delta \times \delta' \in \left\{ \sum_i \delta_i \mid \delta_i \in \gamma_{\mathbb{P}}(P_i \times P') \right\}$$

This holds provided we can find $\delta_i \in \gamma_{\mathbb{P}}(P_i \times P')$ such that $\delta \times \delta' = \sum_i \delta_i$. We have from $\delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta)$ that $\delta = \sum_j \delta_j$ for some $\delta_j \in \gamma_{\mathbb{P}}(P_j)$. We then have from Lemma 34 and $\delta' \in \gamma_{\mathcal{P}_n(\mathbb{P})}(P')$ and $fv(\Delta) \cap fv(P') = \emptyset$ that $\delta_j \times \delta' \in \gamma_{\mathbb{P}}(P_j \times P')$ for all j. We now show that the δ_i we were searching for are these $\delta_j \times \delta'$. To do so, we must show that $\delta \times \delta' = \sum_j (\delta_j \times \delta')$. We have $\delta = \sum_j \delta_j$ and thus the result follows by distributivity of $\times$ over $+$, which we show now.

Goal: $\times$ distributes over $+$ We want to show the following when $domain(\delta_1) = domain(\delta_2)$ and $domain(\delta_1) \cap domain(\delta') = \emptyset$.

$$(\delta_1 + \delta_2) \times \delta' = \delta_1 \times \delta' + \delta_2 \times \delta'$$

Expanding the definition of $+$ and of $\times$, we obtain

$$\lambda(\sigma, \sigma').\ \big(\delta_1(\sigma) + \delta_2(\sigma)\big) \cdot \delta'(\sigma') = \lambda(\sigma, \sigma').\ \big(\delta_1(\sigma) \cdot \delta'(\sigma') + \delta_2(\sigma) \cdot \delta'(\sigma')\big)$$

This holds due to distributivity of $\cdot$ over $+$. □

Lemma 101 (Soundness of Addition)**.** *If* $\delta_1 \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_1)$ *and* $\delta_2 \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_2)$ *then* $\delta_1 + \delta_2 \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_1 + \Delta_2)$.

Proof. First let us take care of the special cases that occur when $\mathit{iszero}(\Delta_1)$ or $\mathit{iszero}(\Delta_2)$. Without the loss of generality let us say $\mathit{iszero}(\Delta_2)$. The sum is defined to be identical to Δ_1. Since $\mathit{iszero}(\Delta_2)$, it must be that $\gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_2)$ contains only the zero distribution $0_{\mathbf{Dist}}$, therefore $\delta_2 = 0_{\mathbf{Dist}}$. Therefore $\delta_1 + \delta_2 = \delta_1$ and by assumption, $\delta_1 \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_1) = \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_1 + \Delta_2)$.

In the case where Δ_1 and Δ_2 are both non-zero, we have $\Delta_1 + \Delta_2 = \lfloor \Delta_1 \cup \Delta_2 \rfloor_n$. Suppose $\delta_1 \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_1)$ and $\delta_2 \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_2)$. By Lemma 92 we have $\gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_1) + \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_2) = \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_1 \cup \Delta_2)$. The set $\gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_1) + \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_2)$ is $\{\delta_1' + \delta_2' \mid \delta_1' \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_1) \wedge \delta_2' \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_2)\}$. Our distributions δ_1 and δ_2 satisfy these conditions and thus are in $\gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_1 \cup \Delta_2)$. It remains to show that $\gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_1 \cup \Delta_2) \subseteq \gamma_{\mathcal{P}_n(\mathbb{P})}(\lfloor \Delta_1 \cup \Delta_2 \rfloor_n)$, but this is exactly Lemma 48. □

C.5 Main Soundness Theorem for Powerset Domain

The main soundness theorem is an identical restatement of the main soundness theorem in the base domain and the proof is likewise identical, save for replacement of the relevant base domain definitions and lemmas with the powerset ones. The only corresponding lemma which has not yet been proven follows below.

Lemma 102. *Let Δ_i be consistent probabilistic polyhedron sets, that is, $\gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_i) \neq \emptyset$. Then, $\Delta_1 + \Delta_2 \sqsubseteq_{\mathbb{P}} \Delta_1$ iff $\mathit{iszero}(\Delta_2)$.*

Proof. The proof is identical to the Lemma 85, replacing the base domain lemmas and definitions with the powerset ones. □

Theorem 46 (Soundness of Abstraction)**.** *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 $[\![S]\!]\delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\langle\langle S \rangle\rangle\, \Delta)$.*

Proof. The proof is identical to the main soundness proof for the base domain (Theorem 27), replacing definitions and lemmas about the base domain abstraction with the corresponding definitions and lemmas about the powerset domain. □

Lemma 103 (Soundness of Normalization)**.** *If $\delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta)$ then $\mathit{normal}(\delta) \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\mathit{normal}(\Delta))$.*

Proof. Whenever $\|\delta\| = 0$, the normalization in the concrete sense is undefined, likewise it is undefined in the abstract sense. So let us assume $\|\delta\| > 0$.

Let $\underline{m} = \sum_i \mathrm{m}_i^{\min}$ and $\overline{m} = \sum_i \mathrm{m}_i^{\max}$. By assumption we have $\delta = \sum_i \delta_i$ with

$$\delta_i \in \gamma_{\mathbb{P}}(P_i) \tag{103.1}$$

Thus we have $\|\delta\| = \sum_i \|\delta_i\|$ and we conclude $\underline{m} = \sum_i \mathrm{m}_i^{\min} \leq \|\delta\| \leq \sum_i \mathrm{m}_i^{\max} = \overline{m}$ via (103.1).

$$\underline{m} \leq \|\delta\| \leq \overline{m} \tag{103.2}$$

Let $\delta' = \text{normal}(\delta) = \frac{1}{\|\delta\|}\delta = \sum_i \frac{1}{\|\delta\|}\delta_i$, due to linearity of scalar product. Let us thus show that $\frac{1}{\|\delta\|}\delta_i \in \gamma_{\mathbb{P}}(\text{normal}(P_i)(\underline{m}, \overline{m})) = \gamma_{\mathbb{P}}(\text{normal}(\Delta))$ which would conclude the proof. Let us write $P_{i'} = \text{normal}(P_i)(\underline{m}, \overline{m})$ and $\delta_{i'} = \frac{1}{\|\delta\|}\delta_i$. We must thus show the following.

$$\text{support}\,(\delta_{i'}) \subseteq \gamma_{\mathbb{C}}(C_{i'}) \tag{103.3}$$
$$\text{s}_{i'}^{\min} \leq |\text{support}\,(\delta_{i'})| \leq \text{s}_{i'}^{\max} \tag{103.4}$$
$$\text{m}_{i'}^{\min} \leq \|\delta_{i'}\| \leq \text{m}_{i'}^{\max} \tag{103.5}$$
$$\forall \sigma \in \text{support}\,(\delta_{i'})\ .\ \text{p}_{i'}^{\min} \leq \delta_{i'}(\sigma) \leq \text{p}_{i'}^{\max} \tag{103.6}$$

Claim (103.3) holds trivially as $\text{support}\,(\delta_{i'}) = \text{support}\,(\delta_i)$, $C_{i'} = C_i$, and (103.1). Claim (103.4) holds due to the same reasoning.

For (103.5), in the case where $\underline{m} > 0$, we reason, via (103.2), as follows.

$$\text{m}_i^{\min} \leq \|\delta_i\| \leq \text{m}_i^{\max}$$
$$\frac{\text{m}_i^{\min}}{\overline{m}} \leq \frac{1}{\|\delta\|}\|\delta_i\| \leq \frac{\text{m}_i^{\max}}{\underline{m}}$$
$$\frac{\text{m}_i^{\min}}{\overline{m}} \leq \|\frac{1}{\|\delta\|}\delta_i\| \leq \frac{\text{m}_i^{\max}}{\underline{m}}$$
$$\text{m}_{i'}^{\min} = \frac{\text{m}_i^{\min}}{\overline{m}} \leq \|\delta_{i'}\| \leq \frac{\text{m}_i^{\max}}{\underline{m}} = \text{m}_{i'}^{\max}$$

If $\underline{m} = 0$, the definition of normalization makes $\text{m}_{i'}^{\max} = 1$, which is also sound as all distributions have mass no more than 1.

The (103.6) claim is shown using reasoning identical to the mass claim above. □

Lemma 50 (Soundness of Simple Maximal Bound Estimate)**.** *If* $\delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\{P_i\})$ *and* $P = \sum_i P_i$ *then* $\max_\sigma \delta(\sigma) \leq \text{p}^{\max}$.

Proof. By assumption we have $\delta = \sum_i \delta_i$ with $\delta_i \in \gamma_{\mathbb{P}}(P_i)$ thus by Lemma 33 (Soundness of Plus), we have $\delta \in \gamma_{\mathbb{P}}(\sum_i P_i) = \gamma_{\mathbb{P}}(P)$, thus for every $\sigma \in \text{support}\,(\delta)$, $\delta(\sigma) \leq \text{p}^{\max}$, hence $\max_\sigma \delta(\sigma) \leq \text{p}^{\max}$. □

The above lemma shows soundness of the very simple method of estimating the maximum probability but in the implementation we use the method based on poly partitioning and the following lemma.

Lemma 54. $max_{pp}\,(\Delta) \stackrel{\text{def}}{=} \max_{\sigma \in R} \Delta^{max}\,(\sigma) = \max_\sigma \Delta^{max}\,(\sigma)$ *where* $\mathcal{L}$ *is a poly partition of* Δ *and* R *is a representative set of* $\mathcal{L}$.

Proof. Let $\mathcal{L}$ be the poly partition of $\Delta = \{C_i\}$ as in the statement of the lemma. Let us first show a claim: if $\sigma, \sigma' \in L \in \mathcal{L}$ then

$$A \stackrel{\text{def}}{=} \{C \in \Delta \mid \sigma \in \gamma_{\mathbb{C}}(C)\} = \{C \in \Delta \mid \sigma' \in \gamma_{\mathbb{C}}(C)\} \stackrel{\text{def}}{=} B \tag{103.7}$$

Let $C \in A$. Thus $\sigma \in \gamma_{\mathbb{C}}(C)$ so by Definition 53 (2), we have $\sigma \in \gamma_{\mathbb{C}}(L')$ for some $L' \in \mathcal{L}$. By (1) it must be that $L = L'$ and by (3), we have $\gamma_{\mathbb{C}}(L) = \gamma_{\mathbb{C}}(L') \subseteq \gamma_{\mathbb{C}}(C)$. Therefore $\sigma' \in \gamma_{\mathbb{C}}(C)$ and thus $C \in B$, showing $A \subseteq B$. The other direction is identical, concluding $A = B$ as claimed. □.

Now we can get back to the main lemma. Let σ^* be the state with $\Delta^{\max}(\sigma^*) = \max_\sigma \Delta^{\max}(\sigma)$. Thus $\sigma^* \in \gamma_{\mathbb{C}}(L)$ for some $L \in \mathcal{L}$, by Definition 53 (2). Let σ_L be any representative of L, that is $\sigma_L \in \gamma_{\mathbb{C}}(L)$.

$$\begin{aligned}
\Delta^{\max}(\sigma^*) &= \sum_i P_i^{\max}(\sigma^*) \\
&= \sum_{i \mid \sigma^* \in \gamma_{\mathbb{C}}(C_i)} \mathrm{p}_i^{\max} \\
&= \sum_{i \mid \sigma_L \in \gamma_{\mathbb{C}}(C_i)} \mathrm{p}_i^{\max} && [\text{ by (103.7) }] \\
&= \sum_i P_i^{\max}(\sigma_L) \\
&= \Delta^{\max}(\sigma_L)
\end{aligned}$$

Now we see that $\max_\sigma \Delta^{\max}(\sigma) = \Delta^{\max}(\sigma^*) = \Delta^{\max}(\sigma_L) = max_{pp}(\Delta)$ as claimed. □

Before we prove the security theorem, let us show that the definition of abstract conditioning on a state is sound.

Lemma 104. *If $\delta \in \gamma_{\mathbb{P}}(\Delta)$ and $\sigma_V \in \boldsymbol{State}_V$ with $V \subseteq fv(\delta)$ then $\delta \wedge \sigma_V \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta \wedge \sigma_V)$*

Proof. Recall the definition of $\Delta \wedge \sigma_V$.

$$\Delta \wedge \sigma_V = \Delta \wedge B$$

With $B = \bigwedge_{x \in V}(x = \sigma_V(x))$. Let us show that $\delta \wedge \sigma_V = \delta \wedge B$, the rest will follow from Lemma 99.

The definition of $\delta \wedge \sigma_V$ is as follows.

$$\delta \wedge \sigma = \lambda\sigma.\ \textbf{if}\ \sigma \upharpoonright V = \sigma_V\ \textbf{then}\ \delta(\sigma)\ \textbf{else}\ 0$$

Meanwhile, $\delta \wedge B$ is defined as follows.

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

The correspondence is immediate as $[\![B]\!]\sigma = \ \textbf{true}\ $ if and only if $\sigma \upharpoonright V = \sigma_V$ as per construction of B. □

Theorem 57 (Soundness for Threshold Security)**.** *Let δ be an attacker's initial belief. If $\delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta)$ and $tsecure_t(S, \Delta)$, then S is threshold secure for threshold t when evaluated with initial belief δ.*

Proof. Let us consider the contrapositive. That is, assuming $\delta \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta)$, if S is not threshold secure for t and initial belief δ, then it is not the case that $tsecure_t(S, \Delta)$.

Let $\delta_2 = [\![S]\!]\delta$ and $\delta_3 = \delta_2 \restriction L$. Since S is not secure, we have $\sigma_L \in \text{support}\,(\delta_3)$ and $\sigma'_H \in \mathbf{State}_H$ with $(\text{normal}((\delta_2 \wedge \sigma_L) \restriction H))(\sigma'_H) > t$. This implies that $(\delta_2 \wedge \sigma_L) \restriction H \neq 0_{\mathbf{Dist}}$ and therefore $\delta_2 \wedge \sigma_L \neq 0_{\mathbf{Dist}}$ as projection preserves mass.

If $[\![S]\!]\Delta$ is not terminating, then we are done as termination is a condition for $tsecure_t(S, \Delta)$. So let us assume $\langle\!\langle S \rangle\!\rangle\, \Delta$ is terminating. Let $\Delta_2 = \langle\!\langle S \rangle\!\rangle\, \Delta$. By Theorem 46, we have $\delta_2 \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_2)$. By Lemma 104, $\delta_2 \wedge \sigma_L \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_2 \wedge \sigma_L)$. Therefore not $iszero(\Delta_2 \wedge \sigma_L)$. Continuing, by Lemma 96, $(\delta_2 \wedge \sigma_L) \restriction H \in \gamma_{\mathcal{P}_n(\mathbb{P})}((\Delta_2 \wedge \sigma_L) \restriction H)$ and finally, by Lemma 103, we have $\delta_4 \stackrel{\text{def}}{=} \text{normal}((\delta_2 \wedge \sigma_L) \restriction H) \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\text{normal}((\Delta_2 \wedge \sigma_L) \restriction H))$. Let $\Delta_4 = \text{normal}((\Delta_2 \wedge \sigma_L) \restriction H)$.

By Remark 52, we have $\delta_4(\sigma'_H) \leq \max_\sigma \Delta_4^{\max}(\sigma)$ and by Lemma 54 we have $\max_\sigma \Delta_4^{\max}(\sigma) = max_{pp}(\Delta_4)$. But $\delta_4(\sigma'_H) > t$ so $max_{pp}(\Delta_4) > t$, a potential failure of $tsecure_t(S, \Delta)$.

To finish the proof we need to make sure that σ_L was indeed a valid witness to the failure of $tsecure_t(S, P_1)$. Let $\Delta_3 = \{P''_i\} = \Delta_2 \restriction L$. By Lemma 96, we have $\delta_3 \in \gamma_{\mathcal{P}_n(\mathbb{P})}(\Delta_3)$ so $\delta_3 = \sum_i \delta'_i$ with $\delta'_i \in \gamma_{\mathbb{P}}(P''_i)$. Since $\sigma_L \in \text{support}\,(\delta_3)$ it must be that $\delta_3(\sigma_L) > 0$ and thus $\delta'_i(\sigma_L) > 0$ for at least one i. Thus $\sigma_L \in \text{support}\,(\delta'_i) \subseteq \gamma_{\mathbb{C}}(C''_i)$ for at least one i and therefore $\sigma_L \in \gamma_{\mathcal{P}(\mathbb{C})}(\{C''_i\})$. Also, we have already shown that not $iszero(\Delta_2 \wedge \sigma_L)$, thus σ_L is indeed the witness as needed. □

Bibliography

[1] Facebook ads: A guide to targeting and reporting. https://www.americanexpress.com/us/small-business/openforum/articles/facebook-ads-a-guide-to-targeting-and-reporting-1, 2011.

[2] Facebook ads: Case studies. http://socialfresh.com/facebook-advertising-examples, 2011.

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

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

[5] Statement of rights and responsibilities. `http://www.facebook.com/legal/terms`, November 2013.

[6] 3 million teens leave facebook in 3 years: The 2014 facebook demographic report. http://istrategylabs.com/2014/01/3-million-teens-leave-facebook-in-3-years-the-2014-facebook-demographic-report, January 2014.

[7] Nathanael Leedom Ackerman, Cameron E Freer, and Daniel M Roy. Noncomputable conditional distributions. In *Proceedings of the IEEE Symposium on Logic in Computer Science (LICS)*, 2011.

[8] Mário S. Alvim, Miguel E. Andrés, and Catuscia Palamidessi. Quantitative information flow in interactive systems. *Journal of Computer Security*, 20(1):3–50, 2012.

[9] Mário S. Alvim, Konstantinos Chatzikokolakis, Pierpaolo Degano, and Catuscia Palamidessi. Differential privacy versus quantitative information flow. *Computing Research Repository (CoRR)*, 2010.

[10] Mário S. Alvim, Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Geoffrey Smith. Measuring information leakage using generalized gain functions. In *Proceedings of the IEEE Computer Security Foundations Symposium (CSF)*, 2012.

[11] Michael Backes, Boris Köpf, and Andrey Rybalchenko. Automatic discovery and quantification of information leaks. In *IEEE Security and Privacy*, 2009.

[12] 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.

[13] Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. Widening operators for powerset domains. 8(4):449–466, 2006.

[14] Gilles Barthe and Boris Köpf. Information-theoretic bounds for differentially private mechanisms. In *Proceedings of the IEEE Computer Security Foundations Symposium (CSF)*, 2011.

[15] Amos Beimel, Kobbi Nissim, and Eran Omri. Distributed private data analysis: Simultaneously solving how and what. In *International Cryptology Conference (CRYPTO)*, 2008.

[16] Frédéric Besson, Nataliia Bielova, and Thomas Jensen. Browser randomisation against fingerprinting: A quantitative information flow approach. In *Proceedings of the Nordic Conference in Secure IT Systems (NordSec)*.

[17] 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.

[18] 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.

[19] Christelle Braun, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. Quantitative notions of leakage for one-try attacks. In *Proceedings of the Conference on Mathematical Foundations of Programming Semantics (MFPS)*, volume 249, pages 75–91, 2009.

[20] Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Prakash Panangaden. Anonymity protocols as noisy channels. *Information and Computation*, 206(2–4):378–401, 2008.

[21] 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.

[22] David Clark and Sebastian Hunt. Non-interference for deterministic interactive programs. In *Proceedings of the*, 2008.

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

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

[25] Michael R. Clarkson and Fred B. Schneider. Quantification of integrity. *Mathematical Structures in Computer Science*, 25:207–258, 2015.

[26] Agostino Cortesi. Widening operators for abstract interpretation. In *Proceedings of the IEEE International Conference on Software Engineering and Formal Methods (SEFM)*, 2008.

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

[28] 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.

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

[30] 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.

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

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

[33] Arthur P. Dempster. Upper and lower probabilities induced by a multivalued mapping. *The Annals of Mathematical Statistics*, 38(2):325–339, 1967.

[34] Dorothy E. Denning. *Cryptography and Data Security*. Addison-Wesley, Reading, Massachusetts, 1982.

[35] Josee Desharnais, Radha Jagadeesan, Vineet Gupta, and Prakash Panangaden. The metric analogue of weak bisimulation for probabilistic processes. In *Proceedings of the IEEE Symposium on Logic in Computer Science (LICS)*, 2002.

[36] 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.

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

[38] Cynthia Dwork. Differential privacy. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, 2006.

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

[40] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 486–503, 2006.

[41] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Theory of Cryptography Conference (TCC)*, 2006.

[42] Barbara Espinoza and Geoffrey Smith. Min-entropy as a resource. In *Information and Computation*, 2013.

[43] Oded Goldreich. *Foundations of Cryptography, vol. 2: Basic Applications*. Cambridge University Press, Cambridge, UK, 2004.

[44] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game, or a completeness theorem for protocols with honest majority. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 1987.

[45] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM*, 43:431–473, 1996.

[46] 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.

[47] 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.

[48] Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. Probabilistic programming. In *Proceedings of the International Conference on Software Engineering (ICSE)*, 2014.

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

[50] 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.

[51] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *Proceedings of the USENIX Security Symposium*, 2011.

[52] James W. Gray III. Toward a mathematical foundation for information flow security. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 1991.

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

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

[55] 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.

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

[57] Boris Köpf and Geoffrey Smith. Vulnerability bounds and leakage resilience of blinded cryptography under timing attacks. In *Proceedings of the*, 2010.

[58] Dexter Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 22(3):328–350, 1981.

[59] Yhuda Lindell and Benny Pinkas. Secure multiparty computation for privacy-preserving data mining. *Journal of Privacy and Confidentiality*, 1(1):59–98, 2009.

[60] Chang Liu, Yan Huang, Elaine Shi, Jonathan Katz, and Michael Hicks. Automating efficient ram-model secure computation. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2014.

[61] 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.

[62] Pasquale Malacaria. Assessing security threats of looping constructs. In *Proceedings of the ACM SIGPLAN Conference on Principles of Programming Languages (POPL)*, 2007.

[63] Pasquale Malacaria. Algebraic foundations for information theoretical, probabilistic and guessability measures of information flow. *Computing Research Repository (CoRR)*, abs/1101.3453, 2011.

[64] Pasquale Malacaria and Han Chen. Lagrange multipliers and maximum information leakage in different observational models. In Úlfar Erlingsson and Marco Pistoia, editor, *Proceedings of the ACM SIGPLAN Workshop on Programming*

Languages and Analysis for Security (PLAS), pages 135–146, Tucson, AZ, USA, June 2008. ACM.

[65] Luciana Marconi, Roberto Di Pietro, Bruno Crispo, and Mauro Conti. Time warp: How time affects privacy in lbss. In *Proceedings of the International Conference on Information & Communications Security (ICICS)*, pages 325–339, 2010.

[66] Piotr Mardziel, Mario Alvim, Michael Hicks, and Michael Clarkson. Quantifying information flow for time-varying data. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2014.

[67] Piotr Mardziel, Mário S. Alvim, and Michael Hicks. Adversary gain vs defender loss in quantified information flow. In *Workshop on Foundations of Computer Security (FCS)*, July 2014.

[68] Piotr Mardziel, Mário S. Alvim, Michael Hicks, and Michael R. Clarkson. Quantifying information flow for dynamic secrets. Technical report, University of Maryland, College Park, 2014. (extended technical report).

[69] 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.

[70] 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.

[71] 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.

[72] Piotr Mardziel, Stephen Magill, Michael Hicks, and Mudhakar Srivatsa. Dynamic enforcement of knowledge-based security policies using abstract interpretation. *Journal of Computer Security*, 21(4):463–532, 2013.

[73] Massey. Guessing and entropy. In *Proceedings of the IEEEInternational Symposium on Information Theory (ISIT)*, 1994.

[74] James L. Massey. Causality, feedback and directed information. In *Proceedings of the International Symposium on Information Theory and its Applications (ISITA)*, 1990.

[75] 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.

[76] Andrew McGregor, Ilya Mironov, Toniann Pitassi, Omer Reingold, Kunal Talwar, and Salil Vadhan. The limits of two-party differential privacy. In *Proceedings of the IEEESymposium on Foundations of Computer Science (FOCS)*, 2010.

[77] Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer, New York, 2005.

[78] 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.

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

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

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

[82] Ira S. Moskowitz, Richard E. Newman, Daniel P. Crepeau, and Allen R. Miller. Covert channels and anonymizing networks. In *Workshop on Privacy in the Electronic Society 2003*, pages 79–88, 2003.

[83] Ira S. Moskowitz, Richard E. Newman, and Paul F. Syverson. Quasi-anonymous channels. In *Proceedings of the International Conference on Communication, Network, and Information Security (CNIS)*, pages 126–131. IASTED, 2003.

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

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

[86] Kevin R. O'Neill, Michael R. Clarkson, and Stephen Chong. Information-flow security for interactive programs. In *Proceedings of the IEEE Computer Security Foundations Symposium (CSF)*, 2006.

[87] 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.

[88] Thomas Paul, Antonino Famulari, and Thorsten Strufe. A survey on decentralized online social networks. *Computer Networks*, 75, Part A(0):437 – 452, 2014.

[89] 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.

[90] Pliam. On the incomparability of entropy and marginal guesswork in brute-force attacks. In *Proceedings of the International Conference in Cryptology in India (INDOCRYPT)*, number 1977 in Lecture Notes in Computer Science, pages 67–79. Springer-Verlag, 2000.

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

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

[93] Norman Ramsey and Avi Pfeffer. Stochastic lambda calculus and monads of probability distributions. In *Proceedings of the ACM SIGPLAN Conference on Principles of Programming Languages (POPL)*, 2002.

[94] Aseem Rastogi, Matthew A. Hammer, and Michael Hicks. Wysteria: A programming language for generic, mixed-mode multiparty computations. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, 2014.

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

[96] 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.

[97] 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.

[98] Reza Shokri, George Theodorakopoulos, Carmela Troncoso, Jean-Pierre Hubaux, and Jean-Yves Le Boudec. Protecting location privacy: optimal strategy against localization attacks. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2012.

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

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

[101] 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.

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

[103] Sekhar Tatikonda and Sanjoy K. Mitter. The capacity of channels with feedback. *IEEETransactions on Information Theory*, 55(1):323–349, 2009.

[104] J. Todd Wittbold and Dale Johnson. Information flow in nondeterministic systems. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 144–161, 1990.

[105] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Proceedings of the IEEESymposium on Foundations of Computer Science (FOCS)*, 1986.