11

Probabilistic Abstract Interpretation: Sound Inference and Application to Privacy[a]

José Manuel Calderón Trilla
Galois, Inc.
Michael Hicks
University of Maryland, College Park
Stephen Magill
Galois, Inc.
Piotr Mardziel
Carnegie Mellon University
Ian Sweet
University of Maryland, College Park

Abstract: Bayesian probability models uncertain knowledge and learning from observations. As a defining feature of optimal adversarial behaviour, Bayesian reasoning forms the basis of safety properties in contexts such as privacy and fairness. Probabilistic programming is a convenient implementation of Bayesian reasoning but the adversarial setting imposes obstacles to its use: approximate inference can underestimate adversary knowledge and exact inference is impractical in cases covering large state spaces.

By abstracting distributions, the semantics of a probabilistic language, and inference, jointly termed *probabilistic abstract interpretation*, we demonstrate adversary models both approximate and sound.

We apply the techniques to build a privacy protecting monitor and describe how to trade off the precision and computational cost in its implementation all the while remaining sound with respect to privacy risk bounds.

11.1 Introduction

Bayesian probability is the de facto standard for modeling uncertainty and learning from observations. Adversaries with uncertain information will employ Bayesian reasoning if they wish to optimize the effectiveness of their attacks. Defenders

[a] This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the US Government.

[b] From *Foundations of Probabilistic Programming*, edited by Gilles Barthe and Joost-Pieter Katoen and Alexandra Silva published 2020 by Cambridge University Press.

https://doi.org/10.1017/9781108770750.012 Published online by Cambridge University Press

must build systems assuming such canny attackers or else risk underestimating the knowledge an adversary can gain and the damage it may inflict.

Probabilistic programming, a mechanization of Bayesian reasoning, offers the requisite elements for doing so. It is a tool for describing adversary knowledge; for specifying the systems adversaries interact with, including the experiments or channels through which they make observations; and for defining and verifying guarantees such as those bounding the damage adversaries can inflict.

Unfortunately, using a practical probabilistic programming system may lead one to draw not entirely trustworthy conclusions about an adversary's bounds. Practical systems are necessarily approximate, for reasons of performance and expressiveness, and are often designed with the average or best case in mind, rather than the worst case. However, when sensitive information is at stake, i.e., in a conservative risk-averse analysis, an *over*-approximation of adversarial capabilities is acceptable but an under-approximation is not.

Probabilistic abstract interpretation is a technique addressing exactly this point: it is approximate and thus more practical than exact inference but it can be made approximate in a manner that adversarial risks can be checked soundly, that is, never underestimated. In our case *soundness* means simply that any likelihood is never underestimated. We leverage this guarantee to bound Bayes' vulnerability, a measure of adversary knowledge and privacy risk.

We begin in Section 11.2 with example privacy and algorithmic fairness properties that motivate the approach to follow. In Section 11.3 we describe a language and its probabilistic semantics suitable for defining and verifying those properties. In Section 11.4 we outline abstractions for probability and for probabilistic interpretation of programs which we then instantiate in Section 11.5 and develop in Section 11.6. We then apply abstract interpretation to implement a privacy monitor for limiting adversary knowledge in Section 11.7. In Section 11.8 we discuss closely related works and compare our approach with alternatives. We conclude with Section 11.9.

This chapter collects and expands on a progression of work on the development and use of probabilistic abstract interpretation to compute upper bounds on the likelihoods of outcomes of systems modeled using probabilistic languages (Mardziel et al., 2011, 2013; Sweet et al., 2018).

11.2 Quantitative Properties

Across domains from information security to algorithmic fairness, probability bounds impose limits on the likelihood of undesired outcomes. Consider, for example, disparate impact ratio and the 80-20 rule (Feldman et al., 2015):

Definition 11.1 (Disparate Impact Ratio)**.** Given a population random variable X

https://doi.org/10.1017/9781108770750.012 Published online by Cambridge University Press

with domain X, a jointly distributed sub-population indicator $\mathcal{Z}$, and a decision procedure $f : X \to \{+, -\}$, the *disparate impact ratio* is the likelihood of a positive outcome for a minority population as compared to the likelihood of a positive outcome for the majority population:

$$\mathrm{DIR}(\mathcal{X}, f) \stackrel{\text{def}}{=} \frac{\Pr[f(\mathcal{X}) = + \mid \mathcal{Z} = \text{minority}]}{\Pr[f(\mathcal{X}) = + \mid \mathcal{Z} = \text{majority}]}$$

The *80-20 rule* states that disparate impact ratio should not fall below 0.8 and is the basis of arguments of discrimination in the U.S. where legal restrictions limit the impact of gender, race, and other protected classes on decisions in hiring, housing, lending, and other areas. Example instantiations of this or similar rules are plentiful in the algorithmic fairness literature (Feldman et al., 2015).

Definition 11.2 (Posterior Bayes Vulnerability)**.** Given a prior belief, or background knowledge, r.v. $\mathcal{X}$ about a secret, and a program $f : X \to Y$, the *Posterior Bayes vulnerability* is the probability of the most probable input upon observing a particular program output y.

$$\mathrm{V}(\mathcal{X}, f, y) \stackrel{\text{def}}{=} \max_x \Pr[\mathcal{X} = x \mid f(\mathcal{X}) = y]$$

Bounds on quantities such as Bayes vulnerability are, likewise, plentiful in the quantitative information flow literature. They aim to model the potential risk in an adversary learning the secret input and exploiting it in some manner (Alvim et al., 2012). In the case of Bayes vulnerability, risk measures the chance an adversary will guess the secret input in one try after making a particular observation on a given program (Smith, 2009).

A Privacy Monitor A key thrust throughout this chapter will be the development of a *privacy monitor* for a system that permits querying private information but wishes to enforce bounds on Bayes vulnerability. The setting is motivated by proposals to move personal private data from centralized services to individuals, allowing them tighter controls over their own personal data (Seong et al., 2010; Baden et al., 2009).

An online query interface with a privacy monitor allows interested parties to retrieve only the necessary data with the understanding that different parties will have interest in different aspect of the data. For example, consider an individual's full birth date, which has been shown to be privacy sensitive: along with zip-code[1] and gender, it suffices to uniquely identify 87% of Americans in the 1990 U.S. census (Sweeney, 2000) and 63% in the 2000 census (Golle, 2006). A horoscope application or "happy birthday" application might request only an individual's birth month and day while a different music recommendation application might instead request a

[1] Zip-code is the postal code in the United States.

https://doi.org/10.1017/9781108770750.012 Published online by Cambridge University Press

$$
\begin{array}{lll}
\textit{Variables} & \textit{var} & ::= X \mid Y \mid Z \mid \ldots \\
\textit{Expressions} & \textit{exp} & ::= \textit{var} \mid \textit{const} \mid \textit{op}(\textit{exp}_1, \textit{exp}_2) \\
\textit{Statements} & \textit{stmt} & ::= \textit{var} := \textit{exp} \mid \\
 & & \quad \mathsf{skip} \mid \textit{stmt}_1 \; ; \; \textit{stmt}_2 \mid \\
 & & \quad \mathsf{while}\ \textit{exp}\ \mathsf{do}\ \textit{stmt} \mid \\
 & & \quad \mathsf{if}\ \textit{exp}\ \mathsf{then}\ \textit{stmt}_1\ \mathsf{else}\ \textit{stmt}_2 \mid \\
 & & \quad \mathsf{prob}\ p\ \mathsf{then}\ \textit{stmt}_1\ \mathsf{or}\ \textit{stmt}_2 \mid \\
 & & \quad \textit{var} := \mathsf{uniform}\ \textit{const}_1\ \textit{const}_2
\end{array}
$$

Figure 11.1 `ImpWhile` with probability: syntax.

user's age (i.e., birth year). A traditional access control system might restrict one of these or the other, in order to hide the full date. But doing so excludes some reasonable applications. A privacy monitor using the Bayes vulnerability bound may allow services to query components of the birth date as they like as long as the full birth date is protected up to a given bound.

The privacy monitor is developed in detail in Section 11.7. It appeared originally in Mardziel et al. (2011), and was further developed in Mardziel et al. (2013) and Sweet et al. (2018). The monitor was also extended to consider individual privacy bounds on computations involving multiple parties, each with private inputs (Mardziel et al., 2012).

11.3 Distribution Semantics

This section presents a minimal, imperative probabilistic programming language and its formal, mathematical semantics. We will use the language to model systems of interest, and reason about their properties, including those noted in the prior section.

Figure 11.1 gives the syntax of the language. It is a simple imperative language, which we call `ImpWhile`, with (global) variables, (integer) constants, arithmetic and relational expressions, and statements, which include assignments, no-ops (skip), sequencing, iteration, and conditionals. `ImpWhile` programs manipulate *program states*, which are maps from variables to their current integer values; these values may be updated during execution, where the final state upon termination is termed the *output state*. The language also includes two probabilistic constructs: probabilistic choice and probabilistic uniform assignment. The former, written prob p then $\textit{stmt}_1$ or $\textit{stmt}_2$, has the following semantics: evaluate statement $\textit{stmt}_1$ with probability p (a ratio between 0 and 1) or otherwise evaluate statement $\textit{stmt}_2$.

https://doi.org/10.1017/9781108770750.012 Published online by Cambridge University Press

X, Y, Z	Variable names.
x, y, z	Variable values.
$\Sigma \stackrel{\text{def}}{=} \text{Variables} \to \text{Integers}$	Set of all program states.
$\sigma, \tau \in \Sigma$	Program states.
$\Sigma \stackrel{\text{def}}{=} \Sigma \to [0,1]$	Set of all program state distributions.
$\mathbf{X}, \mathbf{Y}, \mathbf{Z} \in \Sigma$	Program state distributions.
ϵ, ϵ	Initial program state, and the point distribution assigning probability 1 to only the initial state.
$\mathbf{X}(\sigma)$	Probability of state σ in distribution $\mathbf{X}$.
$\mathbf{X}(exp) \stackrel{\text{def}}{=}$ $\sum_{\sigma:exp \text{ is true for } \sigma} \mathbf{X}(\sigma)$	Marginal probability of exp being true in distribution $\mathbf{X}$.
$\mathbf{X}(exp_1 \mid exp_2) \stackrel{\text{def}}{=}$ $\mathbf{X}(exp_1 \wedge exp_2)/\mathbf{X}(exp_2)$	Marginal probability of exp_1 being true conditioned on exp_2 being true in distribution $\mathbf{X}$.
$[\![stmt]\!] : \Sigma \to \Sigma$	Concrete state semantics.
$[\![stmt]\!] : \Sigma \to \Sigma$	Concrete distribution semantics.
$a, b, c \in \mathbb{A}$	Regions; abstract program states of domain $\mathbb{A}$.
$\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{A}$	Abstract distributions of domain $\mathbb{A}$.
$\langle\!\langle stmt \rangle\!\rangle : \mathbb{A} \to \mathbb{A}$	Abstract semantics for state domain $\mathbb{A}$.
$\langle\!\langle stmt \rangle\!\rangle : \mathbb{A} \to \mathbb{A}$	Abstract semantics for distribution domain $\mathbb{A}$.

Table 11.1 *Notations and conventions.*

Likewise, the uniform assignment $X := \mathsf{uniform}\ l\ u$ assigns to X an integer value uniformly at random from the range of integers between l and u inclusive. The probabilistic elements of this language and their semantics derive from foundational work on probabilistic programming (Kozen, 1981) and have appeared in a similar form in the quantitative information flow literature (Clarkson et al., 2009).

Before presenting the semantics of the language, we turn our attention to some notation, gathered in Table 11.3. This notation will help us talk about the properties of systems we are interested in. Non-bold capital letters, such as X, Y, and Z are variable names (as already mentioned). The lowercase counterparts, x, y, and z, are unspecified values attainable by variables. Lowercase Greek σ, τ denote program states, drawn from the set Σ. We write ϵ to denote the *initial state*, which is the state

https://doi.org/10.1017/9781108770750.012 Published online by Cambridge University Press

that maps all variables to 0. Bold capital letters, $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$, denote distributions[2] over program states from the set $\Sigma \stackrel{\text{def}}{=} \Sigma \to [0,1]$.

Some of our conventions for the rest of this chapter depart from the standard probability conventions used in Definitions 11.1 and 11.2 due to the use of an imperative modeling language and the need to manipulate and distinguish distributions. First, we will no longer use random variables over values like $\mathcal{X}$. Instead, we use distributions like $\mathbf{X}$ and write $\mathbf{X}(\sigma)$ to designate the probability of the state σ according to the distribution $\mathbf{X}$. Second, we will no longer use $f(\mathcal{X})$ to designate the r.v. distributing output values of the function f given input values distributed according to r.v. $\mathcal{X}$. Instead we will write $[\![stmt]\!]\mathbf{X}$ to describe the distribution of output states after the evaluation of statement *stmt* starting from a distribution of input states $\mathbf{X}$.

We define two shorthands to make clear the connection between more familiar probabilistic notation and the distribution notations in this chapter. Given a boolean expression *exp*, we will write $\mathbf{X}(exp)$ to denote the marginal probability of the event that the variable *exp* is true. That is, $\mathbf{X}(exp) \stackrel{\text{def}}{=} \sum_{\sigma:exp \text{ is true for } \sigma} \mathbf{X}(\sigma)$. Given boolean expressions exp_1, exp_2, we will write $\mathbf{X}(exp_1 \mid exp_2)$ to denote the marginal probability of exp_1 being true given exp_2 being true. Formally, $\mathbf{X}(exp_1 \mid exp_2) \stackrel{\text{def}}{=} \mathbf{X}(exp_1 \wedge exp_2)/\mathbf{X}(exp_2)$. Probabilities such as $\Pr[f(\mathcal{X}) = + \mid \mathcal{Z} = \text{minority}]$ of Definition 11.1 will now be written as $([\![stmt]\!]\mathbf{X})(Y = + \mid Z = \text{minority})$. In this case we assumed *stmt* is the imperative implementation of f, the variable Y holds its sole output, and Z holds the minority status of the input individual.[3]

Now we present the mathematical semantics of the language in Figure 11.1. We call it the *concrete probabilistic semantics* $[\![stmt]\!]:\Sigma \to \Sigma$ (as distinct from abstract probabilistic semantics which will follow) and it describes the effect of statements on distributions (of states). It is presented in Figure 11.2. The meaning of a statement *stmt* evaluated on a distribution $\mathbf{X}$, written $[\![stmt]\!]\mathbf{X}$, captures informally the process of evaluating *stmt* on states sampled according to $\mathbf{X}$, and collecting the results in a distribution.[4] The probabilistic semantics described at the top of Figure 11.2 is defined in terms distribution operations and combinators in the bottom part. The two shorthand notations for marginal probability and marginal conditioned probability can likewise be defined in terms of these distribution operators. We note that distributions for the language are discrete. For space reasons, we omit many foundational probability theory details.

We can now rephrase our two example properties. Given a population of individuals $\mathbf{X}$ and a program (statement) *stmt* over some set of variables including protected class Z representing the individuals' attributes and producing its outcome in variable

[2] For simplicity, we often use the term *distribution* even when we are technically dealing with sub-distributions.

[3] When writing f in our language, the minority status { minority, majority } and outcome quality { +, - } would be encoded as integers.

[4] A formal treatment of distribution to distribution semantics first defines the intermediate state to distribution semantics to describe probabilistic statements and can be found in Clarkson et al. (2009).

https://doi.org/10.1017/9781108770750.012 Published online by Cambridge University Press

$$
\begin{aligned}
[\![\mathsf{skip}]\!]\mathbf{X} &\stackrel{\text{def}}{=} \mathbf{X} \\
[\![X := exp]\!]\mathbf{X} &\stackrel{\text{def}}{=} \mathbf{X}\,[X \to exp] \\
[\![stmt_1 \;;\; stmt_2]\!]\mathbf{X} &\stackrel{\text{def}}{=} [\![stmt_2]\!]([\![stmt_1]\!]\mathbf{X}) \\
[\![\mathsf{if}\ exp\ \mathsf{then}\ stmt_1\ \mathsf{else}\ stmt_2]\!]\mathbf{X} &\stackrel{\text{def}}{=} [\![stmt_1]\!]([\![exp]\!]\mathbf{X}) + [\![stmt_2]\!]([\![\neg exp]\!]\mathbf{X}) \\
[\![stmt]\!]\mathbf{X} &\stackrel{\text{def}}{=} [\![stmt]\!]([\![stmt_1]\!]([\![exp_1]\!]\mathbf{X})) + [\![\neg exp_1]\!]\mathbf{X} \\
\text{where } stmt &= \mathsf{while}\ exp_1\ \mathsf{do}\ stmt_1 \\
[\![\mathsf{prob}\ q\ \mathsf{then}\ stmt_1\ \mathsf{or}\ stmt_2]\!]\mathbf{X} &\stackrel{\text{def}}{=} [\![stmt_1]\!](q \cdot \mathbf{X}) + [\![stmt_2]\!]((1-q) \cdot \mathbf{X}) \\
[\![X := \mathsf{uniform}\ l\ u]\!]\mathbf{X} &\stackrel{\text{def}}{=} \textstyle\sum_{x=l}^{u} \frac{1}{u-l+1} \cdot \mathbf{X}\,[X \to x]
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{X}\,[X \to exp] &\stackrel{\text{def}}{=} \lambda\sigma.\ \textstyle\sum_{\tau\,|\,\tau[X \to [\![exp]\!]\tau] = \sigma} \mathbf{X}(\tau) \\
\mathbf{X}_1 + \mathbf{X}_2 &\stackrel{\text{def}}{=} \lambda\sigma.\ \mathbf{X}_1(\sigma) + \mathbf{X}_2(\sigma) \\
[\![exp]\!]\mathbf{X} &\stackrel{\text{def}}{=} \lambda\sigma.\ \mathsf{if}\ [\![exp]\!]\sigma\ \mathsf{then}\ \mathbf{X}(\sigma)\ \mathsf{else}\ 0 \\
p \cdot \mathbf{X} &\stackrel{\text{def}}{=} \lambda\sigma.\ p \cdot \mathbf{X}(\sigma) \\
\|\mathbf{X}\| &\stackrel{\text{def}}{=} \textstyle\sum_{\sigma} \mathbf{X}(\sigma) \\
\mathrm{normal}(\mathbf{X}) &\stackrel{\text{def}}{=} \frac{1}{\|\mathbf{X}\|} \cdot \mathbf{X} \\
\mathbf{X} \mid exp &\stackrel{\text{def}}{=} \mathrm{normal}([\![exp]\!]\mathbf{X})
\end{aligned}
$$

Figure 11.2 `ImpWhile` with probability: probabilistic semantics (top) and distribution operators and combinators (bottom).

Y, we rewrite Definition 11.1 as below:

$$
\frac{([\![stmt]\!]\mathbf{X})\,(Y = + \mid Z = \text{minority})}{([\![stmt]\!]\mathbf{X})\,(Y = + \mid Z = \text{majority})} \tag{11.1}
$$

Likewise, given a program $stmt$ processing variable X distributed according to prior states $\mathbf{X}$ to output Y, the posterior Bayes vulnerability given output y, as in Definition 11.2 is expressed as below:

$$
\max_x \{([\![f]\!]\mathbf{X})\,(X = x \mid Y = y)\} \tag{11.2}
$$

Example 11.3. The $\mathbf{Demographics}_{stmt}$ program below computes a distribution for the demographics—just the birth year and day—of a population of individuals.

$$
\begin{aligned}
&\mathbf{Demographics}_{stmt} \stackrel{\text{def}}{=} \\
&\quad BDAY := \mathsf{uniform}\ 0\ 364; \\
&\quad BYEAR := \mathsf{uniform}\ 1956\ 1992
\end{aligned}
$$

If $\mathbf{X}_0 = [\![\mathbf{Demographics}_{stmt}]\!]\epsilon$ then $\mathbf{X}_0(\sigma) = \frac{1}{365*37}$ for states σ that have

https://doi.org/10.1017/9781108770750.012 Published online by Cambridge University Press

$\sigma(BDAY) \in \{0, \cdots, 364\}$ and $\sigma(BYEAR) \in \{1956, \cdots, 1992\}$. $\mathbf{X}_0$ assigns probability 0 to all other states.

As the distribution is uniform it is not likely to represent a realistic population. More realistic distributions, informed from actual demographic reports such as the U.S. census, can be generated via combinations of the probabilistic choice prob and uniform assignment uniform statements. Such a distribution could then be used to represent an adversary's prior knowledge. Further, the abstraction described later in this chapter relaxes the need to exactly capture background knowledge. A discussion of background knowledge can be found in Section 11.8.

Example 11.4. The program $\mathbf{Birthday}_{stmt}$ below determines whether an individual's birth day (of the year) is within the week period starting from what is assumed to be today.

$$
\begin{array}{l}
\mathbf{Birthday}_{stmt} \stackrel{\text{def}}{=} \\
\quad TODAY := 260; \\
\quad \text{if } BDAY \geq TODAY \wedge BDAY < (TODAY + 7) \text{ then} \\
\qquad OUTPUT := 1 \\
\quad \text{else} \\
\qquad OUTPUT := 0
\end{array}
$$

If $\mathbf{X}_1 = \left([\![\mathbf{Birthday}_{stmt}]\!]\mathbf{X}_0\right) \mid (OUTPUT = 0)$ then $\mathbf{X}_1(\sigma) = \frac{1}{358*37}$ for states σ with $\sigma(BDAY) \in \{0, \cdots, 259, 267, \cdots, 364\}$ and $\sigma(BYEAR) \in \{1956, \cdots, 1992\}$. $\mathbf{X}_1$ assigns 0 probability to all other states.

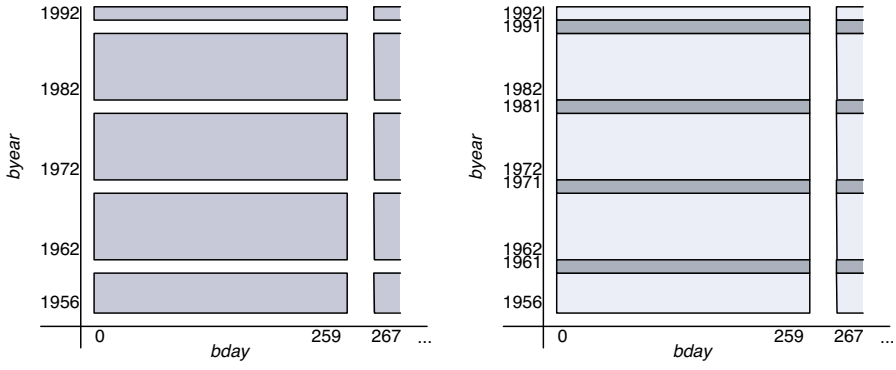
Example 11.5. The program $\mathbf{Decennial}_{stmt}$ determines whether an individual is in a decennial year (their age is a multiple of 10), or otherwise gets lucky with a probabilistic draw.

$$
\begin{array}{l}
\mathbf{Decennial}_{stmt} \stackrel{\text{def}}{=} \\
\quad AGE := 2011 - BYEAR; \\
\quad \text{if } AGE = 20 \vee AGE = 30 \vee \ldots \vee AGE = 60 \\
\qquad \text{then} \\
\qquad\quad OUTPUT := 1 \\
\qquad \text{else} \\
\qquad\quad OUTPUT := 0; \\
\quad \text{prob } 0.1 \text{ then } OUTPUT := 1 \text{ or skip}
\end{array}
$$

Let $\mathbf{X}_2 = ([\![\mathbf{Decennial}_{stmt}]\!]\mathbf{X}_1)$, that is, before conditioning on any particular output. Then $\mathbf{X}_2$ has probability:

- $\mathbf{X}_2(\sigma) = \frac{1}{358*37}$ for states σ with $\sigma(OUTPUT) = 1$, $\sigma(BDAY) \in \{0, \cdots, 259, 267, \cdots, 364\}$, and $\sigma(BYEAR) \in \{1991, 1981, 1971, 1961\}$.

https://doi.org/10.1017/9781108770750.012 Published online by Cambridge University Press



(a) $X_2 \mid (OUTCOME = 0)$ (b) $X_2 \mid (OUTCOME = 1)$

Figure 11.3 Posterior distributions given starting demographics according to **Demographics**$_{stmt}$, **Birthday**$_{stmt}$ outputs in the negative, and **Decennial**$_{stmt}$ outputs in the negative (a) or positive (b).

- $\mathbf{X}_2(\sigma) = \frac{1}{358*37} * \frac{1}{10}$ for states σ with $\sigma(OUTPUT) = 1$, $\sigma(BDAY) \in \{0, \cdots, 259, 267, \cdots, 364\}$, and $\sigma(BDAY) \in \{1956, \cdots, 1992\} \setminus \{1991, 1981, 1971, 1961\}$.
- $\mathbf{X}_2(\sigma) = \frac{1}{358*37} * \frac{9}{10}$ for states σ with $\sigma(OUTPUT) = 0$, $\sigma(BDAY) \in \{0, \cdots, 259, 267, \cdots 364\}$, and $\sigma(BYEAR) \in \{1956, \cdots, 1992\} \setminus \{1991, 1981, 1971, 1961\}$.
- $\mathbf{X}_2(\sigma) = 0$ for all other states σ.

The mass of the positive outcomes, $\|[\![OUTPUT = 1]\!]\mathbf{X}_2\|$, is $\frac{73}{370}$ while the mass of the negative outcomes, $\|[\![OUTPUT = 0]\!]\mathbf{X}_2\|$, is $\frac{297}{370}$. Combining the probabilities above with the masses, we let $X_2^+ = X_2 \mid (OUTCOME = 1)$ and $X_2^- \stackrel{\text{def}}{=} X_2 \mid (OUTPUT = 0)$ have probabilities as below:

- $\mathbf{X}_2^+(\sigma) = \frac{1}{358*37} / \frac{73}{370} = \frac{10}{358*73}$ for states σ with $\sigma(BDAY) \in \{0, \cdots, 259, 267, \cdots, 364\}$ and $\sigma(BYEAR) \in \{1991, 1981, 1971, 1961\}$.
- $\mathbf{X}_2^+(\sigma) = \frac{1}{358*37} * \frac{1}{10} / \frac{73}{370} = \frac{1}{358*73}$ for states σ with $\sigma(BDAY) \in \{0, \cdots, 259, 267, \cdots, 364\}$ and $\sigma(BYEAR) \in \{1991, 1981, 1971, 1961\} \setminus \{1991, 1981, 1971, 1961\}$.
- $\mathbf{X}_2^+(\sigma) = 0$ for all other states σ.
- $\mathbf{X}_2^-(\sigma) = \frac{1}{358*37} * \frac{9}{10} / \frac{297}{370} = \frac{1}{358*33}$ for states σ with $\sigma(BDAY) \in \{0, \cdots, 259, 267, \cdots, 364\}$ and $\sigma(BYEAR) \in \{1991, 1981, 1971, 1961\} \setminus \{1991, 1981, 1971, 1961\}$
- $\mathbf{X}_2^-(\sigma) = 0$ for all other states σ.

The two posterior distributions are visualized in Figure 11.3 with negative outcome

https://doi.org/10.1017/9781108770750.012 Published online by Cambridge University Press

on the left and positive outcome on the right. Darker regions correspond to higher probability.

As described and exemplified, the probabilistic semantics are exact. Computational issues, however, make it difficult to directly apply these semantics in practice. When the space of states becomes large, several aspects of the semantics and definitions become intractable. The assignment statement (see Figure 11.2) and the conditioning operator require sums to enumerate over potentially large number of states or even all possible states. Likewise, properties like Bayes vulnerability refer to all possible marginal states ($X = x$). Finally, while loops may require potentially an infinite number of iterations to evaluate. In the next section we introduce abstractions that can overcome the state-space problems and conclude with a discussion on the problem of adapting abstract interpretation techniques for analyzing looping constructs to the probabilistic case.

11.4 Abstraction

Abstract interpretation is a technique for making tractable the verification of otherwise intractable program properties (Cousot and Cousot, 1977). As the term implies, abstraction is its main principle: instead of reasoning about potentially large sets of program states and behaviours, we abstract them and reason in terms of their abstract properties. We begin by describing the two principal aspects of abstract interpretation in general: an *abstract domain* and *abstract semantics* over that domain.

Definition 11.6 (Abstract Domain)**.** Given a set of concrete objects C an *abstract domain* $\mathbb{A}$ is a set of corresponding abstract elements as defined by two functions:

- an *abstraction* function $\alpha : \mathbf{2}^C \to \mathbb{A}$, mapping sets of concrete elements to abstract elements, and
- a *concretization* function $\gamma : \mathbb{A} \to \mathbf{2}^C$, mapping abstract elements to sets of concrete elements.

In this chapter, C will be instantiated to either program states Σ or distributions Σ over program states. In either case, we assume that concrete program semantics for these elements are given. Because you can view a program's state as a point in a multidimensional space, we often refer specific sets or distributions of program states as *regions*.

We will ignore the abstraction function. For convenience we will consider abstract domains that can be defined as predicates over concrete states. I.e., $\gamma : a \mapsto \{c \in C : \varphi_a(c)\}$ where φ_a is a predicate parameterized by the abstract element a.

The second aspect of abstract interpretation is the *interpretation* part: an abstract

https://doi.org/10.1017/9781108770750.012 Published online by Cambridge University Press

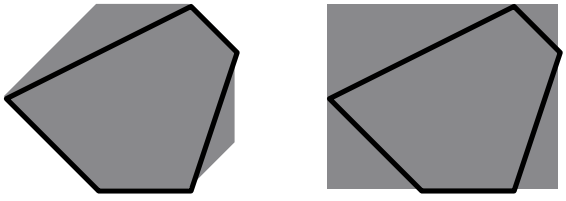


Figure 11.4 The (over)approximation of a polyhedron (black) using an octagon (shaded, left) and an interval (shaded, right).

semantics, written $\langle\langle stmt \rangle\rangle : \mathbb{A} \to \mathbb{A}$. We require that the abstract semantics be *sound* in that it over-approximates the concrete semantics.

Definition 11.7 (Sound Abstraction)**.** Given an abstract domain $\mathbb{A}$ and its abstract semantics, the abstraction is *sound* if whenever $c \in \gamma(a)$ then $[\![stmt]\!]c \in \gamma(\langle\langle stmt \rangle\rangle a)$.

Abstractions generally sacrifice some precision: the abstraction of a set of elements C can be imprecise in that $\gamma(\alpha(C))$ contains strictly more than just C and likewise that $\gamma(\langle\langle stmt \rangle\rangle a)$ contains strictly more elements than $\{[\![stmt]\!]c : c \in \gamma(a)\}$. For this reason, an analysis satisfying Definition 11.7 is called a *may* analysis in that it contains the set of all states that may arise during program execution.

Numeric abstractions A large class of abstractions are designed specifically to model numeric values; in this chapter we restrict ourselves to integer-valued variables. The *interval domain* $\mathbb{I}$ represents "boxes" or non-relational bounded ranges of values for each variable X_i in a state (Cousot and Cousot, 1976):

$$\gamma : \{(l_i, u_i)\}_i \mapsto \{\sigma \in \Sigma : l_i \leq \sigma(X_i) \leq u_i \text{ for every } i\}$$

Abstract elements here are sets of bound pairs, l_i and u_i, forming the lower and upper bound, respectively, for every variable X_i. Intervals are efficient to compute, but imprecise, in that they cannot characterize invariants among variables. More precise, but less efficient numeric domains can be used.

More generally, an abstract domain can be defined in terms of a set of predicates over states, interpreted conjunctively:

$$\gamma : \{\phi_j\}_j \mapsto \{\sigma \in \Sigma : \phi_j(\sigma) \text{ for every } j\}$$

Restrictions on the types of predicates allowed define a family of abstractions. Examples include intervals $\mathbb{I}$ already mentioned, *polyhedra* $\mathbb{P}$ where ϕ_j are restricted to linear inequalities, and *octagons* (Miné, 2001) where the linear inequality coefficients are further restricted to the set $\{-1, 0, 1\}$. Polyhedra and octagons are relational in that they allow precise representations of states that constrain variables

https://doi.org/10.1017/9781108770750.012 Published online by Cambridge University Press

in terms of other variables (note this is not the case for intervals). In terms of tractability, intervals are faster to compute with than octagons which are faster than polyhedra. Precision follows the reverse ordering: polyhedra are more precise than octagons which are more precise than intervals. In other words, intervals can over-approximate the set of points represented by octagons which themselves can over-approximate the set of points represented by polyhedra. This relationship is visualized in Figure 11.4.

Other domains are specifically tailored to efficient analysis of particular types of systems. These include grids (Bagnara et al., 2006) for precisely handling modulo operations and domains designed for analysis of numeric values with overflow/underflow (Simon and King, 2007).

Abstract domains implement a set of standard operations including:

- *Meet*, $a \sqcap b$ is the smallest region containing the set of states in the intersection of $\gamma(a), \gamma(b)$. For convex linear domains this operation is least expensive and is exact.
- *Join*, $a \sqcup b$ is the smallest region containing both $\gamma(a)$ and $\gamma(b)$. For linear convex domains, this is supported by the convex hull operation.
- *Transform*, $a[x \rightarrow exp]$, computes an over-approximation of the state-wise assignment $x \mapsto exp$. In the case of invertible assignments, this operation is supported by linear domains via affine transformation. Non-invertible assignments require special consideration (Mardziel et al., 2011).

Abstraction combinators Abstractions can also be extended disjunctively as in the *powerset* construction (Giacobazzi and Ranzato, 1998). For a base domain $\mathbb{A}$, the *powerset* $\mathbf{2}^{\mathbb{A}}$ domain has concretization:

$$\gamma : \{a_j\}_j \mapsto \{\sigma \in \Sigma : \sigma \in \gamma(a_j) \text{ for some } j\} = \cup_j \gamma(a_j)$$

That is, an abstract element in $\mathbf{2}^{\mathbb{A}}$ is itself a set of base elements from $\mathbb{A}$ and represents the set of states represented by at least one of its constituents base elements.

Abstraction in the manner outlined can also be applied to probability distributions which serve as the concrete elements. Earlier techniques (Monniaux, 2001) attached probability constraints to standard state domains. Given a state domain $\mathbb{A}$ we form the probabilistic (upper bound) domain $\overline{\mathbb{D}}(\mathbb{A})$ that adds a probability bound on all states represented by the base domain elements:

$$\gamma : (a, p) \mapsto \{\mathbf{X} \in \Sigma : \mathbf{X}(\sigma) \leq p \text{ for all } \sigma \in \gamma(a)\}$$

We can combine the probabilistic upper bound construction the powerset construction to define a domain for representing more complex distributions. A more

https://doi.org/10.1017/9781108770750.012 Published online by Cambridge University Press

expressive variant of powerset for probabilistic abstractions imposes a sum bound (as opposed to disjunction of bounds):

$$\gamma : \left\{(a_j, p_j)\right\}_j \mapsto \left\{ \mathbf{X} \in \Sigma : \mathbf{X}(\sigma) \leq \sum_{j:\sigma \in \gamma(a_j)} p_j \text{ for every } \sigma \right\}$$

We emphasize in these abstractions the focus on the upper bounds of probability; such abstractions do not explicit track lower bounds (beyond the assumed trivial 0). That is, for any probabilistic abstraction a and any state σ, there exists $\mathbf{X} \in \gamma(a)$ such that $\mathbf{X}(\sigma) = 0$. Because of this, these upper bound abstractions lack sound definitions of conditioning. Recall Bayes' rule or the definition of conditioning in Figure 11.2 that involves a normalization by total mass of a (sub)distribution. Upper bounds alone cannot exclude the possibility of 0 as the total mass in the denominator. Posterior Bayes vulnerability of Definition 11.2 features conditioning by program output and disparate impact ratio of Definition 11.1 features probability in the denominator. Thus neither of these conditions can be soundly checked using purely upper-bound abstractions of probability described thus far.

11.5 Sound Domains with Conditioning

As suggested, sound inference needs to account for both lower and upper bounds on probability. The *dual-bounded probabilistic* construction does exactly this (Mardziel et al., 2011)[5]. In this Section we define this domain. In the next section we outline representative aspects of the implementation of its abstract semantics and outline soundness proofs.

The construction imposes probability bounds along with several other constraints used to preserve precision in the implementations of abstract operators to follow.

Definition 11.8. Given a state domain $\mathbb{A}$, the *dual-bounded probabilistic* domain $\mathbb{D}(\mathbb{A})$ is occupied by *probabilistic regions* defined by 4-tuples (a, s, p, m). A probabilistic region represent distributions satisfying 4 constraints: $a \in \mathbb{A}$ bounds their support, $s = (\mathrm{s}^{\min}, \mathrm{s}^{\max})$ bounds their number of support points, $p = (\mathrm{p}^{\min}, \mathrm{p}^{\max})$ bounds their probability mass per support point, and $m = (\mathrm{m}^{\min}, \mathrm{m}^{\max})$ bounds their total probability mass (recall we are working with sub-distributions). Formally these conditions define the concretization function:

$$\gamma(a, s, p, m) \mapsto \left\{ \begin{array}{l} \mathbf{X} \in \Sigma : \mathit{support}(\mathbf{X}) \subseteq \gamma(a), \\ \qquad \mathrm{s}^{\min} \leq \|\mathit{support}(\mathbf{X})\| \leq \mathrm{s}^{\max}, \\ \qquad \mathrm{m}^{\min} \leq \|\mathbf{X}\| \leq \mathrm{m}^{\max}, \\ \qquad \mathrm{p}^{\min} \leq \mathbf{X}(\sigma) \leq \mathrm{p}^{\max} \text{ for every } \sigma \in \mathit{support}(\mathbf{X}) \end{array} \right\}$$

[5] This is a generalization of the *probabilistic polyhedra* domain from earlier work (Mardziel et al., 2011).

https://doi.org/10.1017/9781108770750.012 Published online by Cambridge University Press

We will use non-bold lowercase a to denote abstract states and bold lowercase $\mathbf{a}$ to denote dual-bounded probabilistic regions. We will refer to the p parameters as the point bounds, the s parameters as size bounds, the m parameters as mass bounds, and the a parameter as the support bound.

This probabilistic construction applies to base domains that implement the standard set of abstract operations from the prior section in addition to the counting operation:

- Region size, written $\#(a)$, is the number of states (integer vectors in the case of integer-valued states being modeled) in the region, i.e., $\|\gamma(a)\|$. For some domains this is an expensive model counting operation and requires specialized tools such as Latte (De Loera et al., 2008). On the other hand, for other domains like intervals, this operation is trivial.

For convenience in the rest of this chapter, we use two additional operations that can be defined using the standard operations and size.

- Boolean expression conjunction on a region a, written $\langle\langle exp \rangle\rangle\, a$, returns a region containing at least the points in a that satisfy exp. This is the abstract equivalent of $[\![exp]\!]\sigma$ of Figure 11.2.
- Boolean expression count on a region a, written as $a\#exp$, is an upper bound on the number of points in a that satisfy exp.

Example 11.9. In the powerset of probabilistic polyhedra $\mathbb{D}(\mathbb{P})$, we can represent the negative outcome distributions of Figure 11.3(a), before normalization, with two probabilistic polyhedra $\mathbf{a}_1$ and $\mathbf{a}_2$ containing polyhedra a_1 and a_2 bounding regions $0 \leq BDAY \leq 259, 1956 \leq BYEAR \leq 1992$ and $267 \leq BDAY \leq 354, 1956 \leq BYEAR \leq 1992$, respectively. The other parameters for $\mathbf{a}_1$ would be as follows:

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

Notice this over-approximation loses the fact that the states with $BYEAR \in \{1991, 1981, 1971, 1961\}$ have 0 probability in the concrete semantics. This is also evident in that $\mathrm{s}_1^{\min} = \mathrm{s}_1^{\max} = 8580 < \#(a_1) = 9620$, illustrating that the "bounding box" of the polyhedra covers more area than is strictly necessary for precision.

For the positive outcome of Figure 11.3(b), we can use the same two polyhedra a_1 and a_2 with the other parameters for a_1 as follows:

$$\begin{aligned} &\mathrm{p}_1^{\min} = 1/135050 = \tfrac{1}{37*365} * \tfrac{1}{10} \quad && \mathrm{p}_1^{\max} = 10/135050 = \tfrac{1}{37*365} \\ &\mathrm{s}_1^{\min} = 9620 = 260 * 37 && \mathrm{s}_1^{\max} = 9620 = 260 * 37 \\ &\mathrm{m}_1^{\min} = 26/185 && \mathrm{m}_1^{\max} = 26/185 \end{aligned} \tag{11.3}$$

In this case $\mathrm{s}_1^{\min} = \mathrm{s}_1^{\max} = \#(a_1)$, meaning that all covered points are possible,

https://doi.org/10.1017/9781108770750.012 Published online by Cambridge University Press

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 $\textbf{Decennial}_{stmt}$ returning 1 composed of having a decennial (first term) plus not having a decennial (second term).

Notice that the exemplified representations are only some among many reasonable options. First, the use of polyhedra as a base domain was not necessary and intervals alone would have been sufficient. Second, more precise representations could have been constructed by using more than just two probabilistic polyhedra (or intervals). On the other hand, even less precise representations would use only a single probabilistic polyhedron (interval). Further nuances come into play for schemes with powerset abstractions that employ a dynamic number of probabilistic regions that can increase or decrease in the process of evaluating programs. How to best employ the representation power of powerset domains is not trivial remains an open problem.

11.6 Abstract Semantics

The abstract semantics for dual-bounded probabilistic regions is defined identically to concrete semantics in Figure 11.2(top) except with supplanting each of the concrete operations/combinators of Figure 11.2 (bottom) with abstract versions that operate on abstract distributions instead of concrete distributions. Soundness of the abstraction is then shown inductively on language statements from the soundness of the abstracted operations and combinators. We present some of these operations in this section; the full set of operations as well as the corresponding proofs can be found in Mardziel et al. (2013).

Abstract Conjunction The concrete conjunction operation restricts a distribution to states satisfying a boolean expression, nullifying probability mass of states that do not: $[\![exp]\!]\mathbf{X} \stackrel{\text{def}}{=} \lambda\sigma.\ \textsf{if}\ [\![exp]\!]\sigma\ \textsf{then}\ \mathbf{X}(\sigma)\ \textsf{else}\ 0$. Using the expression conjunction and count for abstract states, we develop the abstract conjunction for probabilistic regions as follows.

Definition 11.10. Given a probabilistic region $\mathbf{a}_1 = (a_1, s_1, p_1, m_1)$ and boolean expression exp, let $n = a\#exp$ and $\overline{n} = a\#(\neg exp)$. That is, n is an over-approximation of the number of states in a that satisfy the condition exp and $\overline{n}$ is an over-approximation of the number of points in a that do not satisfy exp. Then, $\langle\langle exp \rangle\rangle\ \mathbf{a}_1$

https://doi.org/10.1017/9781108770750.012 Published online by Cambridge University Press

is the probabilistic region $\mathbf{a}_2 = (a_2, s_2, p_2, m_2)$ defined by the parameters enumerated below.

$$\begin{array}{l|l}
\mathrm{p}_2^{\min} = \mathrm{p}_1^{\min} & \mathrm{s}_2^{\min} = \max\left\{\mathrm{s}_1^{\min} - \overline{n}, 0\right\} \\
\mathrm{p}_2^{\max} = \mathrm{p}_1^{\max} & \mathrm{s}_2^{\max} = \min\left\{\mathrm{s}_1^{\max}, n\right\}
\end{array}$$

$$\mathrm{m}_2^{\min} = \max\left\{\mathrm{p}_2^{\min} \cdot \mathrm{s}_2^{\min},\ \mathrm{m}_1^{\min} - \mathrm{p}_1^{\max} \cdot \min\left\{\mathrm{s}_1^{\max}, \overline{n}\right\}\right\}$$

$$\mathrm{m}_2^{\max} = \min\left\{\mathrm{p}_2^{\max} \cdot \mathrm{s}_2^{\max},\ \mathrm{m}_1^{\max} - \mathrm{p}_1^{\min} \cdot \max\left\{\mathrm{s}_1^{\min} - n, 0\right\}\right\}$$

$$a_2 = \langle\!\langle exp \rangle\!\rangle\, a_1$$

The soundness requirement for this and subsequent operations stipulates an inclusion relation between the concrete variant and the abstract variant. In the case of conjunction the statement is is thus: if $\mathbf{X} \in \gamma(\mathbf{a})$ then $[\![exp]\!]\mathbf{X} \in \gamma(\langle\!\langle exp \rangle\!\rangle\, \mathbf{a})$.

Abstract Plus The concrete plus operation combines mass of two given distributions: $\mathbf{X}_1 + \mathbf{X}_2 \stackrel{\text{def}}{=} \lambda\sigma.\ \mathbf{X}_1(\sigma) + \mathbf{X}_2(\sigma)$. The abstract counterpart over-approximates the result. Specifically, if $\mathbf{X}_1 \in \gamma(\mathbf{a}_1)$ and $\mathbf{X}_2 \in \gamma(\mathbf{a}_2)$ then $\mathbf{X}_1 + \mathbf{X}_2 \in \gamma(\mathbf{a}_1 + \mathbf{a}_2)$. For the remainder of the chapter, we will leave the association between a probabilistic region, $\mathbf{a}$, and its constituents, (a, s, p, m), implicit. When more than one probabilistic region is being discussed, the subscripts of the tuple elements will match the subscript of the region.

The abstract sum of two probabilistic regions is defined differently depending on whether their support regions overlap. In the case they do not overlap, the sum $\mathbf{a}_3$ has $a_3 = a_1 \sqcup a_2$ and parameters as below:

$$\begin{array}{lcl|lcl}
\mathrm{p}_3^{\min} & = & \min\left\{\mathrm{p}_1^{\min}, \mathrm{p}_2^{\min}\right\} & \mathrm{p}_3^{\max} & = & \max\left\{\mathrm{p}_1^{\max}, \mathrm{p}_2^{\max}\right\} \\
\mathrm{s}_3^{\min} & = & \mathrm{s}_1^{\min} + \mathrm{s}_2^{\min} & \mathrm{s}_3^{\max} & = & \mathrm{s}_1^{\max} + \mathrm{s}_2^{\max} \\
\mathrm{m}_3^{\min} & = & \mathrm{m}_1^{\min} + \mathrm{m}_2^{\min} & \mathrm{m}_3^{\max} & = & \mathrm{m}_1^{\max} + \mathrm{m}_2^{\max}
\end{array}$$

Otherwise, a_1 and a_2 overlap. We first determine the minimum and maximum number of points in the intersection that may be support points for both $\mathbf{a}$ and for $\mathbf{b}$. We refer to these counts as the *pessimistic overlap* and *optimistic overlap*, respectively.

Definition 11.11. Given two distributions $\mathbf{X}_1, \mathbf{X}_2$, we refer to the set of states that are in the support of both $\mathbf{X}_1$ and $\mathbf{X}_2$ as the *overlap* of $\mathbf{X}_1, \mathbf{X}_2$. The *pessimistic overlap* of $\mathbf{a}$ and $\mathbf{b}$, denoted $\mathbf{a} \mathbin{\text{☹}} \mathbf{b}$, is the cardinality of the smallest possible overlap between any two distributions $\mathbf{X}_1 \in \gamma(\mathbf{a})$ and $\mathbf{X}_2 \in \gamma(\mathbf{b})$ and the *optimistic overlap* $\mathbf{a} \mathbin{\text{☺}} \mathbf{b}$ is the cardinality of the largest possible overlap. They are computed as follows:

$$\mathbf{a} \mathbin{\text{☹}} \mathbf{b} \stackrel{\text{def}}{=} \max\left\{\mathrm{s}_1^{\min} + \mathrm{s}_2^{\min} - \left(\#(a) + \#(b) - \#(a \sqcap b)\right),\ 0\right\}$$

$$\mathbf{a} \mathbin{\text{☺}} \mathbf{b} \stackrel{\text{def}}{=} \min\left\{\mathrm{s}_1^{\max}, \mathrm{s}_2^{\max}, \#(a \sqcap b)\right\}$$

https://doi.org/10.1017/9781108770750.012 Published online by Cambridge University Press

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

Definition 11.12. The abstract sum of **a** and **b**, written **a** + **b**, is the probabilistic region **c** with parameters as follows:

$$
\begin{aligned}
c &= a \sqcup b \\
\mathrm{p}_3^{\min} &= \begin{cases} \mathrm{p}_1^{\min} + \mathrm{p}_2^{\min} & \text{if } \mathbf{a} \odot \mathbf{b} = \#(c) \\ \min\left\{\mathrm{p}_1^{\min}, \mathrm{p}_2^{\min}\right\} & \text{otherwise} \end{cases} \\
\mathrm{p}_3^{\max} &= \begin{cases} \mathrm{p}_1^{\max} + \mathrm{p}_2^{\max} & \text{if } \mathbf{a} \odot \mathbf{b} > 0 \\ \max\left\{\mathrm{p}_1^{\max}, \mathrm{p}_2^{\max}\right\} & \text{otherwise} \end{cases} \\
\mathrm{s}_3^{\min} &= \max\left\{\mathrm{s}_1^{\min} + \mathrm{s}_2^{\min} - \mathbf{a} \odot \mathbf{b},\ 0\right\} \\
\mathrm{s}_3^{\max} &= \min\left\{\mathrm{s}_1^{\max} + \mathrm{s}_2^{\max} - \mathbf{a} \odot \mathbf{b},\ \#(c)\right\} \\
\mathrm{m}_3^{\min} &= \mathrm{m}_1^{\min} + \mathrm{m}_2^{\min} \quad | \quad \mathrm{m}_3^{\max} = \mathrm{m}_1^{\max} + \mathrm{m}_2^{\max}
\end{aligned}
$$

The setting of parameters in the sum is chosen to be as precise as possible while maintaining soundness: if $\mathbf{X}_1 \in \gamma(\mathbf{a})$ and $\mathbf{X}_2 \in \gamma(\mathbf{b})$ then $\mathbf{X}_1 + \mathbf{X}_2 \in \gamma(\mathbf{a} + \mathbf{b})$. The two cases for $\mathrm{p}_3^{\min}$ derive from: (first case) the overlap between the operands is complete (the support of both is identical) and (second case) there is a possibility of a non-overlapping state that is in support of one of the operands but not the other. Likewise, the cases for $\mathrm{p}_3^{\max}$ derive from: (first case) there is a possibility of support point overlap and (second case) there is no overlap possible between the operands. The size parameters follow from the inclusion-exclusion principle and the mass parameter is a mere sum that does not depend on where the operands distribute their probability mass.

Together the abstract operation soundness claims (see Mardziel et al. (2013) for the rest and their proofs) imply soundness of the abstract semantics:

Theorem 11.13 (Abstraction Semantics Soundness)**.** *If* $\mathbf{X} \in \gamma(\mathbf{a})$ *then* $[\![stmt]\!]\mathbf{X} \in \gamma(\langle\!\langle stmt \rangle\!\rangle\, \mathbf{a})$.

Abstract Normalization Critically, the dual-bounded probabilistic domain allows us to soundly define the conditioning operation which in turn is defined primarily via normalization operation. The normalization of a (sub) distribution produces a distribution whose total mass is equal to 1: $\mathrm{normal}(\mathbf{X}) \stackrel{\text{def}}{=} \frac{1}{\|\mathbf{X}\|} \cdot \mathbf{X}$. If a probabilistic region **a** 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 for transforming an arbitrary probabilistic region into one containing only normalized distributions.

https://doi.org/10.1017/9781108770750.012 Published online by Cambridge University Press

Definition 11.14. Assuming $\mathrm{m}_1^{\min} > 0$, normal($\mathbf{a}_1$) is the normalized region $\mathbf{a}_2$ with:

$$\begin{array}{lcl|lcl} \mathrm{p}_2^{\min} & = & \mathrm{p}_1^{\min}/\mathrm{m}_1^{\max} & \mathrm{s}_2^{\min} & = & \mathrm{s}_1^{\min} \\ \mathrm{p}_2^{\max} & = & \mathrm{p}_1^{\max}/\mathrm{m}_1^{\min} & \mathrm{s}_2^{\max} & = & \mathrm{s}_1^{\max} \\ \mathrm{m}_2^{\min} & = & \mathrm{m}_2^{\max} = 1 & a_2 & = & a_1 \end{array}$$

The normalization operator illustrates the interaction between under and over approximation of probability in the abstraction: to ensure that the over-approximation of a state's probability ($\mathrm{p}^{\max}$) is sound, we must divide by the *under-approximation* of the total probability mass ($\mathrm{m}^{\min}$). This results in abstract normalization that is sound: If $\mathbf{X} \in \gamma(\mathbf{a})$ and $\mathbf{X}$ has non-zero mass, then normal($\mathbf{X}$) $\in \gamma$ (normal($\mathbf{a}$)).

Together with soundness of abstract conjunction presented earlier in this section we arrive at the main goal of this work.

Theorem 11.15 (Soundness of Conditioning)**.** *If* $\mathbf{X} \in \gamma(\mathbf{a})$ *and exp has non-zero marginal probability in* $\mathbf{X}$ *then* $\mathbf{X} \mid exp \in \gamma(\mathbf{a} \mid exp)$.

We can now show how to use our abstraction to soundly over-approximate quantities such as disparate impact ratio (Definition 11.1) and posterior Bayes vulnerability (Definition 11.2). We define upper and lower bounds on the probability of states as well as the marginal probability bounds for boolean expressions according to a dual-bounded probabilistic region.

Definition 11.16. Given probabilistic region $\mathbf{a}$ and a boolean expression *exp*, the *upper and lower bounds on the marginal probability of exp* are defined as $\mathbf{a}_{\min}(exp) \stackrel{\text{def}}{=} \mathrm{m}_1^{\min}$ and $\mathbf{a}_{\max}(exp) \stackrel{\text{def}}{=} \mathrm{m}_1^{\max}$ where the mass bound parameters are those of the probabilistic region $\mathbf{a}_1 = \langle\!\langle exp \rangle\!\rangle \mathbf{a}$. The *upper and lower state bounds* are the bounds on the probability of any single (possible) state and are defined $\mathbf{a}_{\min} \stackrel{\text{def}}{=} \mathrm{p}_1^{\min}$ and $\mathbf{a}_{\max} \stackrel{\text{def}}{=} \mathrm{p}_1^{\max}$.

Corollary 11.17. *The marginal and state probability bounds are sound. That is, for every* $\mathbf{X} \in \gamma(\mathbf{a})$*:*

$$\mathbf{a}_{min}(exp) \leq \mathbf{X}(exp) \leq \mathbf{a}_{max}(exp)$$

For every $\sigma \in support(\mathbf{X})$*:*

$$\mathbf{a}_{min} \leq \mathbf{X}(\sigma) \leq \mathbf{a}_{max}$$

Notice that the state bounds quantities $\mathbf{a}_{\min}$ and $\mathbf{a}_{\max}$ bound the probability of all support states (state with non-zero probability). Quantities such as vulnerability can thus be checked using state bounds without enumerating every possible state.

Returning to disparate impact ratio, let $\mathbf{X}$ be a distribution of individuals with variable Z referring to minority status and presume we have $\mathbf{X} \in \gamma(\mathbf{a})$. Let $\mathbf{a} \stackrel{\text{def}}{=} \langle\!\langle stmt \rangle\!\rangle (\mathbf{a} \mid Z = \text{minority})$ and $\mathbf{b} \stackrel{\text{def}}{=} \langle\!\langle stmt \rangle\!\rangle (\mathbf{a} \mid Z = \text{majority})$.

https://doi.org/10.1017/9781108770750.012 Published online by Cambridge University Press

$$\frac{([\![stmt]\!]\mathbf{X})\,(Y = + \mid Z = \text{minority})}{([\![stmt]\!]\mathbf{X})\,(Y = + \mid Z = \text{majority})} \leq \frac{(\mathbf{a})_{\max}\,(Y = +)}{(\mathbf{b})_{\min}\,(Y = +)}$$

Note that though the abstraction, its semantics, and operations allow us to soundly check a disparate ratio bound, this check is outside the syntax and semantics of the language modeled. Though some languages support both probabilistic interpretation and subsequent manipulation of resulting distributions in the same host language, this is not a goal of our toy language and manipulations of distributions like those in the disparate impact ratio bound above have to be done in a separate language hosting the probabilistic interpreter.

In the next section we show how to use the state probability bound, an indicator of the probability of any support point in a distribution, to construct a vulnerability-based privacy monitor.

Powerset Bounds Single regions are firmly on the tractability side of the tractability/accuracy trade-off. Probabilistic regions can be additively combined using a probabilistic powerset construction of Section 11.4. There an abstract probability distribution is composed of a set of simpler abstract probability distributions (in our case dual-bounded probabilistic regions). The set represents all distributions equal to the distribution sum of the distributions represented by each of the abstract elements:

$$\gamma : \{\mathbf{a}_j\}_j \mapsto \left\{ \mathbf{X} \in \Sigma : \mathbf{X} = \sum_j \mathbf{X}_j \text{ where } \mathbf{X}_j \in \gamma\,(\mathbf{a}_j) \right\}$$

For sound base probabilistic abstractions as per Theorem 11.13 and with sound event probability bounds as per Corollary 11.17, the powerset construction provides similarly sound results. Details including the set operations taking part of the abstract interpretation, the probability bound definitions, and proofs can be found in Mardziel et al. (2013).

Widening A distinguishing aspect of abstract interpretation as compared to other static analysis techniques is its handling of looping programs. Recall the semantics of while:

$$[\![stmt]\!]\mathbf{X} \stackrel{\text{def}}{=} \qquad [\![stmt]\!]([\![stmt_1]\!]([\![exp_1]\!]\mathbf{X})) + [\![\neg exp_1]\!]\mathbf{X}$$
$$\text{where } stmt = \textsf{while } exp_1 \textsf{ do } stmt_1$$

Notice the first term includes the evaluation of the same while statement as we are defining the semantics of. We rewrite this as a monotonic sequence $\mathbf{Y}_i$ via a

https://doi.org/10.1017/9781108770750.012 Published online by Cambridge University Press

recursive definition $\mathbf{X}_i$ (along with the abstract version of the same):

$$\begin{aligned} \mathbf{X}_0 &\stackrel{\text{def}}{=} \mathbf{X} & \mathbf{a}_0 &\stackrel{\text{def}}{=} \mathbf{a} \\ \mathbf{X}_{i+1} &\stackrel{\text{def}}{=} [\![stmt_1]\!]([\![exp_1]\!]\mathbf{X}_i) & \mathbf{a}_{i+1} &\stackrel{\text{def}}{=} \langle\!\langle stmt_1 \rangle\!\rangle\, (\langle\!\langle exp_1 \rangle\!\rangle\, \mathbf{a}_i) \\ \mathbf{Y}_n &\stackrel{\text{def}}{=} \sum_{i=0}^{n} [\![\neg exp_1]\!]\mathbf{X}_i & \mathbf{b}_n &\stackrel{\text{def}}{=} \sum_{i=0}^{n} \langle\!\langle \neg exp_1 \rangle\!\rangle\, \mathbf{a}_i \end{aligned} \tag{11.4}$$

The result of $[\![stmt]\!]\mathbf{X}$ is the fixed-point of the sequence $\mathbf{Y}_i$, a point i at which $\mathbf{Y}_{i+1} = \mathbf{Y}_i$. The problem is that reaching the fixed-point may require large number of iterations. Given the motivation of large state spaces, it is plausible that the number of iterations in a loop is likewise large. Worse yet, in the abstract version of the semantics where abstractions may include concretely unrealizable distributions due to precision loss, the fixed-point may not be achieved in any finite number of steps even if the while loop terminates concretely.

Abstract interpretation employs the *widening* operator to make sure fixed-point computations take only a finite number of iterations. Let $\sqsubseteq$ be an ordering on abstractions respecting the subset relation in their concretizations ($\mathbf{a} \sqsubseteq \mathbf{b}$ implies $\gamma(\mathbf{a}) \subseteq \gamma(\mathbf{b})$).

Definition 11.18. A *widening* operator ∇ is a binary operator that defines for every ascending chain of abstractions $\mathbf{c}_i \sqsubseteq \mathbf{c}_{i+1}$, a chain $\mathbf{c}'_0 \stackrel{\text{def}}{=} \mathbf{c}_0$, $\mathbf{c}'_{i+1} \stackrel{\text{def}}{=} \mathbf{c}'_i \nabla \mathbf{c}_{i+1}$ that over-approximates the original chain ($\mathbf{c}'_i \sqsubseteq \mathbf{c}_i$) and has a finite fixed-point ($\mathbf{c}'_{i+1} = \mathbf{c}'_i$ for some finite i).

The abstract semantics of a while loop written as $\mathbf{b}_n$ of Equation 11.4 constitutes an ascending chain and it can thus be over-approximated with a chain having a finite fixed-point by employing widening. Practically, for a widening operator to be defined for an abstraction, it must come with an ordering and must be able to represent potentially infinite concrete states. For example, the interval domain constraints allows for variable bounds to be one-sided or even unbounded. That is, constraints for variables to ranges like $[c, +\infty]$, $[-\infty, c]$, and $[-\infty, +\infty]$, for constant c, are possible. The state abstractions discussed in this chapter all come with widening operators including the powerset constructions.

Widening for probabilistic abstractions, however, is another matter. For the integer-valued programs discussed, the techniques we described rely heavily on counting states. Merely admitting infinite state counts into the counting arguments of this section result in total loss of precision, distribution representations whose probability bounds are uselessly between 0 and 1. As a result, defining abstractions with non-trivial widening operators for probabilistic semantics with sound conditioning remains an open problem.

https://doi.org/10.1017/9781108770750.012 Published online by Cambridge University Press

11.7 Privacy Monitor

In this section demonstrate an application of state bounds to an implementation of an online query privacy monitor. Such a monitor allows clients to query private information while owner can measure how much has been revealed by the queries and can decide to block queries that would otherwise reveal too much. Mardziel et al. (2011) motivate the use of such a system for retaining individual control over personal information while selling partial access to entities such as advertiser who might themselves derive full financial benefit from only limited access. A birthday cake merchant, for example, might be content with knowing that a user's birthday is within the next week in order to offer them a coupon (as in the program $\mathbf{Birthday}_{stmt}$). An online query scheme allows the merchant to get the information they need and nothing else. On the other hand, repeated queries reveal additional information hence the proposed system tracks the knowledge of each querying party while interacting with the monitor.

The primary tool for monitor is the upper state probability bound with which posterior vulnerability can be soundly estimated. Given a program (the query) *stmt* processing variable X distributed according to prior states $\mathbf{X}$ to output Y and $\mathbf{X} \in \gamma(\mathbf{a})$, we bound the posterior Bayes vulnerability given output y:

$$\max_x \{([\![f]\!]\mathbf{X})(X = x \mid Y = y)\} \leq (\langle\!\langle stmt \rangle\!\rangle \mathbf{a} \mid Y = y)_{\max} \stackrel{\text{def}}{=} \mathrm{V}(\mathbf{a}, stmt, y)$$

Specifically, the right hand inequality gives us a conservative overestimate of the risk in revealing the output y of the program *stmt* to an adversary whose prior knowledge is $\mathbf{X} \in \gamma(\mathbf{a})$ where risk is the likelihood that the adversary can guess value of the secret X correctly in one try. We will refer to the vulnerability bound based on the abstraction $\mathbf{a}$ as $\mathrm{V}(\mathbf{a}, stmt, y)$.

A common objection to privacy properties such as this is that they depend on having the right model of what potential adversaries know. Abstraction alleviates this problem. We need not know $\mathbf{X}$ exactly but only that whatever the adversary knowledge actually is, we capture it in the abstraction $\mathbf{a}$. We return to this point in Section 11.9.

A monitor serves as the gateway to the protected information and will, given a security parameter or vulnerability threshold t, make sure that the risk (in terms of vulnerability) never rises beyond t. This application presumes a querier only observes its interactions with the monitor (it does not infer anything about the secret from any other source).

Given a vulnerability threshold t, a secret state σ, prior adversary knowledge $\mathbf{a}$ with $\mathbf{X}(\sigma) > 0$ for some $\mathbf{X} \in \gamma(\mathbf{a})$, a query *stmt* whose output on the secret state is the value y of variable Y the dynamic monitor has three components: one to determine whether a query should be answered, one to determine what to return to

https://doi.org/10.1017/9781108770750.012 Published online by Cambridge University Press

the querier, and one to revise adversary knowledge:

$$\text{Allow}_t(stmt) : \mathbf{a} \mapsto \begin{cases} \text{true} & \text{if } \mathrm{V}(\mathbf{a}, stmt, y) \leq t \ \forall y \\ \text{false} & \text{otherwise} \end{cases}$$

$$\text{Observe}_t(stmt, y) : \mathbf{a} \mapsto \begin{cases} y & \text{if Allow}_t(stmt) \\ \text{deny} & \text{otherwise} \end{cases}$$

$$\text{Posterior}_t(stmt, y) : \mathbf{a} \mapsto \begin{cases} (\langle\!\langle stmt \rangle\!\rangle \, \mathbf{a}) \mid (Y = y) & \text{if Allow}_t(stmt) \\ \mathbf{a} & \text{otherwise} \end{cases}$$

Note that in the Allow component, by way of a test for all possible outputs y, we are evaluating the "worst-case" posterior vulnerability (Köpf and Basin, 2007). Compared to making this check for conditional vulnerability given only the actual output y from *stmt* evaluated on secret σ, the worst-case has the benefit of being simulatable (Kenthapadi et al., 2005) in that it can be determined without knowledge of the secret value σ. The outcome of the vulnerability check in the Allow component, therefore, leaks no information about it beyond what is assumed to be known by the adversary.

Chaining applications of the monitor on a sequence of programs $stmt_i$ whose outputs are y_i, we define $\mathbf{a}_{i+1} = \text{Posterior}_t(stmt_i, y_i)(\mathbf{a}_i)$ as the sequence of knowledge revisions of a Bayesian adversary proposing queries $stmt_i$ and observing $\text{Observe}_t(stmt_i, y_i)$.

Theorem 11.19. *Assume a Bayesian adversary has prior knowledge* $\mathbf{X} \in \gamma(\mathbf{a}_0)$ *consistent with secret* σ ($\mathbf{X}(\sigma) > 0$) *and prior vulnerability bounded by threshold* t ($\max_\sigma \mathbf{X}(\sigma) \leq t$). *Let* y_i *be a the set of query outputs sampled from* $[\![stmt_i]\!]\sigma$. *If the adversary observes nothing but the sequence* $\text{Observe}_t(stmt_i, y_i)(\mathbf{a}_i)$, *then at no stage will they have a likelihood of more than* t *of guessing the correct secret* σ *in a single try.*

We say that y_i are sampled in the theorem as *stmt* may contain probabilistic statements so more than one output value is possible. The theorem is principally based on the soundness of our abstraction for modeling probabilistic program semantics and conditioning. At each stage in the sequence, the abstraction $\mathbf{a}_i$ includes what the Bayesian adversary knows about the secret given their initial prior knowledge and the outputs of the queries before that point, noting query rejections do not reveal anything about the secret.

Example 11.20. Let us consider the sequence of queries, starting from prior according to $\textbf{Demograhpics}_{stmt}$, evaluating $\textbf{Birthday}_{stmt}$ which returns 0, and then evaluating $\textbf{Birthday261}_{stmt}$ which has $TODAY = 261$. In concrete interpretation we have distributions:

https://doi.org/10.1017/9781108770750.012 Published online by Cambridge University Press

- $\mathbf{X}_0 = [\![\mathbf{Demograhpics}_{stmt}]\!]\epsilon$. At this point $\mathbf{X}_0(\sigma) = \frac{1}{365*37}$, meaning we could reasonably employ the privacy monitor for any threshold t above $\frac{1}{365*37}$.
- $\mathbf{X}_1 = \left([\![\mathbf{Birthday}_{stmt}]\!]\mathbf{X}_0\right) \mid (OUTPUT = 0)$. At this point we have $\mathbf{X}_1(\sigma) = \frac{1}{358}$ for states σ that have $\sigma(BDAY) \in \{0, \cdots, 259, 267, \cdots, 364\}$ (and for some values of *BYEAR*). Thus if the threshold t was below $\frac{1}{358*37}$, the monitor would have already rejected this first query.
- $\mathbf{X}_2' = \left([\![\mathbf{Birthday261}_{stmt}]\!]\mathbf{X}_1\right)$.

The monitor now needs to determine whether to evaluate the report the output of the second birthday query. Were the output of this query to be 0, we would have posterior $\mathbf{X}_2(\sigma) = \frac{1}{357*37}$ but if the query returned 1, we would instead have $\mathbf{X}_2(\sigma) = \frac{1}{1*37}$, pinpointing the day of the year exactly. Thus for any threshold below $\frac{1}{37}$, the monitor must reject this query, regardless of whether it would return 0 or 1 on the true value of *BDAY*.

The example above is described in terms of concrete probabilistic semantics. Given soundness of the corresponding abstract semantics, the safe enforcement of the posterior vulnerability can also be done in the abstract (Theorem 11.19).

11.8 Related Work

The work presented in this chapter has connections to techniques for program analysis, notably abstraction interpretation, and methods for measuring and enforcing privacy. We briefly summarize the most related of these works. When considering privacy, we specifically consider works that do, and do not, require modeling prior knowledge when assessing information leakage.

Abstract Interpretation Static program analyses such as abstract interpretation(Cousot and Cousot, 1977) and symbolic execution (King, 1976; Cadar et al., 2008) model program behaviour over large sets of inputs or starting conditions with the goal of discovering or verifying the absence of undesirable conditions that would be difficult or close to impossible to verify with mere test cases or dynamic analyses. Static techniques employ forms of abstraction to explore the space of executions. The form of abstraction varies and has implications on the differences between the analyses both in terms of their tractability, precision, and soundness in modeling programs. The aspects of abstract interpretation that distinguish it from other static analysis techniques include its limits on the complexity of representation and the use of the widening operator to handle looping programs.

Abstract domains impose limits on the complexity of an analysis by abstract interpretation (recall domains of Section 11.4). Powerset domains typically restrict the number of disjuncts in a representation and the disjuncts themselves are limited by

https://doi.org/10.1017/9781108770750.012 Published online by Cambridge University Press

their underlying domain whose basic logical queries such as satisfiability are trivial or at least easy. This distinguishes abstract interpretation from analyses in which logical representation can grow ever more complex and employ more expressive theories, sometimes to the point where success of analysis depends primarily on an undecidable logical satisfiability test for an enormous formula employing potentially incomplete theories.

The abstract interpretation of probabilistic programs has been tackled by Monniaux (2001) who defined probabilistic program semantics based on over-approximating probabilities of program states. In other work, Di Pierro et al. (2005) described abstract interpretation for probabilistic lambda calculus, and Smith (2008) who used probabilistic abstract interpretation for verification of quantitative program properties. Such works are limited in their (lack of) sound handling of conditioning which is a necessary component of a wide variety of quantitative privacy notions. A theoretical investigation of probabilistic abstract interpretation as built atop traditional abstractions can be found in Cousot and Monerau (2012).

Dynamic Probabilistic Programming Dynamic analysis works characterizes properties of concrete program evaluations. Such an analysis has the benefit of not requiring an abstraction of semantics and hence can be easily adopted to full-featured languages. The analysis described in this chapter, on the other hand, works well for programs containing only linear expressions over integer-valued variables. Adapting such analyses to richer programs is possible but will invariably suffer in precision when modeling language features not specifically designed for in the abstraction.

In the context of this chapter's privacy application, the pertinent aspect of a probabilistic programming system is its ability to accurately or soundly approximate probability or a privacy criterion. Though generally lacking ability to derive exact probability or bounds, dynamic techniques and sampling have been used in privacy contexts. Köpf and Rybalchenko (2010), for example, use sampling to derive information flow bounds.

More recently, Sweet et al. (2018) have shown that sampling can be used to improve the precision of the abstract distribution representation discussed in this chapter. However, in this and other sampling techniques, the soundness guarantees become somewhat subtle. In the case of (Sweet et al., 2018), for example, the authors provide for a confidence bound (a probability over the sampling process) that the derived probability values (like posterior Vulnerability) are within a certain range.

Privacy with Background Knowledge Assumptions *Measurement* of adversary knowledge of private data as it is informed by a program's output has been a well-studied problem since Robling Denning (1982). Clark et al. (2005) define a static analysis that bounds the secret information a straight-line program can leak

https://doi.org/10.1017/9781108770750.012 Published online by Cambridge University Press

in terms of equivalence relations between the inputs and outputs. Backes et al. (2009) automate the synthesis of such equivalence relations and quantify leakage by computing the exact size of equivalence classes. Köpf and Rybalchenko (2010) 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 (2009) 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.

Along with tools, there is a growing number of quantitative information flow measures in the literature, varying according to the operational interpretation of risk. Instances include *Bayes vulnerability* (Smith, 2009) and *Bayes risk* (Chatzikokolakis et al., 2008), *Shannon entropy* (Shannon, 1948), and *guessing entropy* (Massey, 1994). The *g-vulnerability* framework (Alvim et al., 2012) is meant to encompass a more general set of operational interpretations.

Two important aspects of all of these works are (a) whether they deal with absolute or relative information and (b) whether they incorporate background knowledge.

For the first, an example absolute measure is posterior *vulnerability* discussed in this chapter. Relative measures compare the absolute measurements before and after an adversary makes an observation from some scrutinized system. Relative measurements of information further have variants which do not assume particular background knowledge on the adversary's part but instead quantify the worst-case difference in prior and posterior over all possible distributions. Channel capacity in FlowCheck (McCamant and Ernst, 2008) and various definitions of maximum leakage measures in the quantitative information flow literature are examples.

The reliance on having a sense of background knowledge of adversaries is the problematic assumption motivating other popular approaches such as *differential privacy* (Dwork, 2008). Unlike the approaches mentioned above, differential privacy lacks a clear connection to harms induced by privacy loss and attempts at connecting its privacy parameter to harms invariably make assumptions as problematic as those regarding background knowledge (Kifer and Machanavajjhala, 2011).

Though we make the assumption of having adversary background knowledge in our work, our use of probabilistic programming and abstract interpretation alleviates it. First, for cases where a secret is generated by a program which is known by the adversary, the distribution representing their background knowledge can be derived by probabilistic evaluation of said generating program. An over-approximation of the knowledge can likewise be generating using the techniques described in this chapter. Second, probabilistic programs can be viewed as tools for modeling background knowledge and can bring to bear their benefits specifically in terms of concisely describing distributions arising from generative processes. Finally, probabilistic abstract interpretation makes the process easier by not requiring the

exact background knowledge of all adversaries to be known as long as it can be approximated in some abstraction. To this end, we can extend our toy probabilistic language with the "possibilistic" choice statement:

$$stmt ::= \mathsf{poss}\ stmt_1\ \mathsf{and}\ stmt_2 \mid \ldots$$

The meaning of possibilistic choice is that either branch can occur. This cannot be modeled in the concrete semantics but can be approximated in the abstract semantics by making sure that when $\mathbf{b} = [\![\mathsf{poss}\ stmt_1\ \mathsf{and}\ stmt_2]\!]\mathbf{a}$ then $\gamma([\![stmt_1]\!]\mathbf{a}) \subseteq \gamma(\mathbf{b})$ and $\gamma([\![stmt_2]\!]\mathbf{a}) \subseteq \gamma(\mathbf{b})$. That is, the abstraction of possibility must include the abstractions of both branches. As an adversary modeling tool, this lets us remain uncertain which of the branches generates the true adversary knowledge, as long as one of them does.

The techniques described in this chapter can incorporate such a modeling tool by virtue of the imprecise but sound representations employed. The numeric abstractions, however, were not designed with this use-case in mind. Taking advantage of this feature of abstract interpretation for the purpose of modeling uncertainty in knowledge is an open problem.

11.9 Conclusions

In this chapter we have described a probabilistic programming approach with sound probability and inference bounds suitable for specifying fairness and privacy properties. Based on abstract interpretation, the technique allows one to trade off precision for speed of analysis all the while preserving a general soundness criterion. Probabilistic abstract interpretation therefore offers a unique set of benefits from among the probabilistic programming toolkit.

The field of abstract interpretation is an active area of research and offers many open problems. New domains, combinators, and algorithms for efficiently and accurately representing program states and reasoning about larger and more feature rich programs are proposed regularly. Considerations for probability, however, are not as thoroughly investigated. Fundamental aspects of abstraction such as widening remain unavailable for languages with sound conditioning and thus pose a hurdle to the wider adoption of the otherwise extremely successful program analysis technique to probabilistic applications.

References

Alvim, Mário S., Chatzikokolakis, Konstantinos, Palamidessi, Catuscia, and Smith, Geoffrey. 2012. Measuring Information Leakage Using Generalized Gain

https://doi.org/10.1017/9781108770750.012 Published online by Cambridge University Press

Functions. In: *Proceedings of the IEEE Computer Security Foundations Symposium (CSF)*.

Backes, Michael, Köpf, Boris, and Rybalchenko, Andrey. 2009. Automatic Discovery and Quantification of Information Leaks. In: *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*.

Baden, Randy, Bender, Adam, Spring, Neil, Bhattacharjee, Bobby, and Starin, Daniel. 2009. 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)*.

Bagnara, Roberto, Dobson, Katy, Hill, Patricia M, Mundell, Matthew, and Zaffanella, Enea. 2006. Grids: A domain for analyzing the distribution of numerical values. Pages 219–235 of: *International Symposium on Logic-based Program Synthesis and Transformation*. Springer.

Cadar, Cristian, Ganesh, Vijay, Pawlowski, Peter M, Dill, David L, and Engler, Dawson R. 2008. EXE: automatically generating inputs of death. *ACM Transactions on Information and System Security (TISSEC)*, **12**(2), 10.

Chatzikokolakis, Konstantinos, Palamidessi, Catuscia, and Panangaden, Prakash. 2008. On the Bayes risk in information-hiding protocols. *Journal of Computer Security*, **16**(5).

Clark, David, Hunt, Sebastian, and Malacaria, Pasquale. 2005. Quantitative Information Flow, Relations and Polymorphic Types. *Journal of Logic and Computation*, **15**, 181–199.

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

Cousot, Patrick, and Cousot, Radhia. 1976. Static Determination of Dynamic Properties of Programs. In: *Proceedings of the Second International Symposium on Programming*.

Cousot, Patrick, and Cousot, Radhia. 1977. 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)*.

Cousot, Patrick, and Monerau, Michael. 2012. Probabilistic Abstract Interpretation. In: *Proceedings of the European Symposium on Programming (ESOP)*.

De Loera, Jesus A., Haws, David, Hemmecke, Raymond, Huggins, Peter, Tauzer, Jeremy, and Yoshida, Ruriko. 2008. *LattE*. http://www.math.ucdavis.edu/latte.

Di Pierro, Alessandra, Hankin, Chris, and Wiklicky, Herbert. 2005. Probabilistic Lambda-calculus and Quantitative Program Analysis. *Journal of Logic and Computation*, **15**(2), 159–179.

Dwork, Cynthia. 2008. Differential privacy: A survey of results. Pages 1–19 of: *International Conference on Theory and Applications of Models of Computation*. Springer.

Feldman, Michael, Friedler, Sorelle A., Moeller, John, Scheidegger, Carlos, and Venkatasubramanian, Suresh. 2015. Certifying and Removing Disparate Impact. Pages 259–268 of: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Giacobazzi, Roberto, and Ranzato, Francesco. 1998. Optimal domains for disjunctive abstract interpretation. *Science of Computer Programming*, **32**(1-3), 177–210.

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

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

Kifer, Daniel, and Machanavajjhala, Ashwin. 2011. No free lunch in data privacy. Pages 193–204 of: *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM.

King, James C. 1976. Symbolic execution and program testing. *Communications of the ACM*, **19**(7), 385–394.

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

Köpf, Boris, and Rybalchenko, Andrey. 2010. Approximation and Randomization for Quantitative Information-Flow Analysis. In: *Proceedings of the IEEE Computer Security Foundations Symposium (CSF)*.

Kozen, Dexter. 1981. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, **22**(3), 328–350.

Mardziel, Piotr, Magill, Stephen, Hicks, Michael, and Srivatsa, Mudhakar. 2011. Dynamic Enforcement of Knowledge-based Security Policies. In: *Proceedings of the IEEE Computer Security Foundations Symposium (CSF)*.

Mardziel, Piotr, Hicks, Michael, Katz, Jonathan, and Srivatsa, Mudhakar. 2012. Knowledge-Oriented Secure Multiparty Computation. In: *Proceedings of the ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS)*.

Mardziel, Piotr, Magill, Stephen, Hicks, Michael, and Srivatsa, Mudhakar. 2013. Dynamic Enforcement of Knowledge-based Security Policies using Probabilistic Abstract Interpretation. *Journal of Computer Security*, Jan.

Massey, James L. 1994. Guessing and Entropy. In: *Proceedings of the IEEE International Symposium on Information Theory*.

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

Miné, Antoine. 2001. The Octagon Abstract Domain. In: *Proceedings of the Working Conference on Reverse Engineering (WCRE)*.

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

Mu, Chunyan, and Clark, David. 2009. An Interval-based Abstraction for Quantifying Information Flow. *Elec. Notes in Theoretical Computer Science*, **253**(3), 119–141.

Robling Denning, Dorothy Elizabeth. 1982. *Cryptography and data security*. Addison-Wesley Longman Publishing Co., Inc.

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

Shannon, Claude. 1948. A Mathematical Theory of Communication. *Bell System Technical Journal*, **27**.

Simon, Axel, and King, Andy. 2007. Taming the wrapping of integer arithmetic. Pages 121–136 of: *International Static Analysis Symposium*. Springer.

Smith, Geoffrey. 2009. On the Foundations of Quantitative Information Flow. In: *Proceedings of the Conference on Foundations of Software Science and Computation Structures (FoSSaCS)*.

Smith, Michael J. A. 2008. Probabilistic Abstract Interpretation of Imperative Programs using Truncated Normal Distributions. *Electronic Notes in Theoretical Computer Science*, **220**(3), 43–59.

Sweeney, Latanya. 2000. *Simple Demographics Often Identify People Uniquely*. Tech. rept. LIDAP-WP4. Carnegie Mellon University, School of Computer Science, Data Privacy Laboratory.

Sweet, Ian, Trilla, José Manuel Calderón, Scherrer, Chad, Hicks, Michael, and Magill, Stephen. 2018. What's the Over/Under? Probabilistic Bounds on Information Leakage. Pages 3–27 of: Bauer, Lujo, and Küsters, Ralf (eds), *Principles of Security and Trust*. Cham: Springer International Publishing.