

Information Security and Cryptography

Mário S. Alvim

Konstantinos Chatzikokolakis

Annabelle McIver · Carroll Morgan

Catuscia Palamidessi · Geoffrey Smith

The Science of Quantitative Information Flow

Information Security and Cryptography

Series Editors

David Basin
Kenny Paterson

Advisory Board

Michael Backes
Gilles Barthe
Ronald Cramer
Ivan Damgård
Andrew D. Gordon
Joshua D. Guttman
Christopher Kruegel
Ueli Maurer
Tatsuaki Okamoto
Adrian Perrig
Bart Preneel

Mário S. Alvim • Konstantinos Chatzikokolakis
Annabelle McIver • Carroll Morgan
Catuscia Palamidessi • Geoffrey Smith

The Science of Quantitative Information Flow



Mário S. Alvim
Computer Science Department
Universidade Federal de Minas Gerais
Belo Horizonte, Brazil

Annabelle McIver
Department of Computing
Macquarie University
Sydney, NSW, Australia

Catuscia Palamidessi
Inria Saclay and LIX
École Polytechnique
Institut Polytechnique de Paris
Palaiseau, France

Konstantinos Chatzikokolakis
Department of Informatics
and Telecommunications
University of Athens
Athens, Greece

Carroll Morgan
School of Computer Science
& Engineering
University of New South Wales
Trustworthy Systems, Data61
CSIRO
Sydney, NSW, Australia

Geoffrey Smith
School of Computing
& Information Sciences
Florida International University
Miami, FL, USA

ISSN 1619-7100 ISSN 2197-845X (electronic)
Information Security and Cryptography
ISBN 978-3-319-96129-3 ISBN 978-3-319-96131-6 (eBook)
<https://doi.org/10.1007/978-3-319-96131-6>

© Springer Nature Switzerland AG 2020

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

The authors dedicate this book as follows:

Mário S. Alvim to his mother, Maria Angélica, his stepfather, Mario, his brothers, Marco Antônio and Marcus Vinícius, and his husband, Trevor.

Kostas Chatzikokolakis to his father, Thymios.

Annabelle McIver to her daughter, Eleanor, and her parents, Anne and Ted.

Carroll Morgan to the policy of diversity and tolerance deliberately instituted and actively sustained at Data61's Trustworthy Systems Group.

Catuscia Palamidessi to her husband, Dale Miller, and their children, Alexis and Nadia Miller.

Geoffrey Smith to his parents, Marilyn and Seward, his wife, Elena, his sons, Daniel and David, and his cockatiel, Yoshi.



Cockatiel *Yoshi* as a probabilistic channel C that maps a top-secret document X to a (randomly generated) pile of shredded paper Y

Preface

Information Flow is the transfer of information from a source (who knows the information) to a target (who does not yet know it). In history, that topic has sometimes been studied in order to *impede* flow (e.g. Caesar’s Cipher from millennia ago), and sometimes to *facilitate* it (e.g. Shannon’s work in the 1940’s). Usually, however, the aims are a careful mixture of the two: to let information flow to those who need to know it, but to keep it from those who must not have it. That is the focus of our contemporary perspective –facilitate some flows, impede others– and our main (but not exclusive) concern here is computer systems.

But *first*: what is so special about now? Information-flow security is a critical problem today because of recent technological developments and their –largely uncontrolled– spread to many hands: all the way from everyday home users to super-skilled hackers, and all over the earth. Data is being collected more than ever before (smart phones, surveillance cameras, “loyalty” cards); networks then enable its transmission to unknown (or unintended) destinations; and powerful corporate and governmental agents gain financial and/or political benefits by collecting and analyzing that data. And, of course, there are the criminals.

Because so much is flowing, and so many have access to it, and we know so little specifically about who they are, we can no longer protect our information by relying on the people through whose hands it passes. Thus the standard technologies like *access control* and *encryption* are insufficient, because there we require the entities granted access to our data to handle it appropriately, and that implied trust might well be misplaced: a smartphone app could legitimately need access to our location, for example, but then leak that information to some other party, perhaps maliciously — but also perhaps just by accident.

Thus instead we must try to generate, process, and transfer our data with systems that protect *themselves*, that are safe no matter who accesses them or how they might abuse that access. It demands a fundamental, rigorous approach; and that fundamental rigor is exactly the *science* that we are striving for.

Thus, *second*: how can it be done? Early rigorous work in information-flow security (since the 1970’s) suggested ways in which programs could be analyzed to see whether the program variables an adversary “could see” might depend on variables that were not supposed to be seen: our secrets. If there was no dependence, then the program was secure; but if there was *any dependence at all*, then the program was deemed insecure. That “depends or not” criterion was later realized to be too coarse, however: even a password-checking program, no matter how carefully constructed, would be deemed insecure, because **Access Denied** still unavoidably exhibits a dependence — on what the password *is not*.

Quantitative information flow solves the “depends or doesn’t”, the “black or white” problem by relativizing information leaks, recognizing that it’s not really that clear-cut — some leaks are more important than others, and thus some are tolerable (e.g. leaking what a password isn’t, provided it’s only infrequently). A typical quantitative approach is to use Shannon’s information theory to measure the “entropy” of a secret (roughly, how hard it is to guess) before a system is run, and then to determine what the entropy would become after the program is run (by analyzing the source code, which we assume is available to our adversary). The difference between the two entropies, before minus after, is then how many bits have flowed from the system (escaped, if that flow is not desirable) and —again roughly— if it’s a small proportion of the bits that should remain secret, then the actual impact might be considered to be quite limited. Further, because the flow is quantified, the impact can actually be reasoned about rather than merely regretted. That technique realizes a powerful insight, and it works well in many situations: quantifying secrecy in the Shannon style (via entropy) provides the needed nuance to escape the earlier “all or nothing” judgments. For example, if the amount of entropy leaked by a failed login is indeed very small, it is exactly there that quantitative reasoning allows us to calculate with “very small” and “how often” and compare the result to “tolerable”.

But much more recently still, it was suggested that Shannon’s approach could be generalized, taken further, because in some situations also *it* turned out to be too inflexible: were the numbers it produced, how many bits escaped, really the numbers we needed to know? The generalization was to allow a *selection* of entropies —many more than just Shannon’s alone— whose characteristics were derived empirically from a study of the possible adversaries’ motivations and capabilities. Which secrets do they really want, and which ones would they not bother to steal? What exactly can they do with their knowledge about the secret? That last step —the generalized entropies— completes the conceptual trajectory from “Does information flow at all?” (simple dependence) through “How many bits of information flow?” (Shannon leakage) to finally (at least for the moment) “What is the *value* to the adversary of the information that flows?” or, dually, “What damage to us is caused by that flow, and how much would we spend (or should we have spent) to prevent it?” Generalized entropies (of which Shannon entropy is a special case) are captured by what we call “loss functions”; dually, we also consider generalized “vulnerabilities”, captured by “gain functions”. Furthermore, loss- and gain functions enable a connection with the science of program *development*, where specification programs are “refined” into implementation programs that satisfy those specifications both in terms of functionality and security. (Shannon-entropy leakage is not usually a compositional criterion; and yet compositionality is essential for reliable program construction. The use of generalized entropies, however, *is* compositional.)

For all of those reasons, our study of *the science of quantitative information flow* aims to understand fundamentally how sensitive information “flows” as it is processed by an authorized entity (e.g. our computer program), and to ensure that those flows are acceptable to us in terms of the quantified damage they might cause. And here —as we will emphasize— it is important to understand “flows” in a very broad sense: indeed flow occurs whenever sensitive information is *correlated* with observable outputs, allowing an adversary to make *inferences* about the sensitive information. Such correlations can be blatant, as when a sensitive file is copied to some publicly observable place, but they can also be subtle, as when a medical database outputs a patient’s country as “United States” if the patient has diabetes and as “USA” if not: in that case the patient’s diabetes status “flows” to the country output in a way that probably was not intended.

Extant studies of information flow encompass a variety of domains –such as non-interference, anonymity, unlinkability, secure multi-party computation, differential privacy, statistical databases, side channels, voting, and anonymous communication and publishing– and we have tried to do the same. Something that makes those studies challenging, and our study as well, is that perfection is often unachievable, because *some* undesirable flows cannot be helped. Publishing statistics about a database of medical records necessarily involves revealing some information about the individual records: keeping those records completely private is not an option in that case. Indeed there are many practical reasons for accepting flows that –in a perfect world– we would prefer not to have:

- Sometimes a flow is *intentional*: we *want* to learn something from our statistical database.
- Sometimes a flow is due to *side channels* that are hard or impossible to control fully.
- Sometimes a flow is in exchange for a *service*, one which for example might need our location.
- Sometimes a flow is in exchange for *efficiency*, as when a weaker but more efficient anonymous communication system is used instead of a stronger but less efficient protocol.

All of those support our belief that we must not (only) ask *whether* there is an information flow, and not even (only) *how many* bits of Shannon entropy might flow. We try to study instead *how much damage* an information flow would cause; and because of the generality of that approach, the earlier two are special cases.

The six authors of this book come from a number of distinct research domains, including process calculi, privacy, type systems for secure information flow, and programming-language semantics and refinement. As we all came to understand information flow better, we recognized that our efforts shared deep commonalities; and so, merging our earlier specialties, we have been working intensively as a group together since about 2010. This book is our comprehensive treatment of *quantitative information flow (QIF)* as we currently understand it — and we hope that it will lead to further and wider collaboration with those who might read it.

Much of what we present here is based on material already published, but by no means all of it — it is not at all merely “a collection of papers”. Instead we have tried hard to write a unified and self-contained text, hoping as we did that to find better terminology and notation than we might have used before, and then in some cases even rewriting whole presentations from scratch to take advantage of it. As well, in many cases we have also replaced earlier mathematical proofs with new ones that are clearer and more self-contained.

Finally, while this book is mainly focused on the systematic development of the theory of quantitative information flow, we also demonstrate the theory’s practical utility by including (in Part V) case studies showing how quantitative-information-flow analysis can be applied to a number of interesting realistic scenarios.

Intended readership

Our intended reader is anyone interested in the mathematical foundations of computer security. As far as the required technical background is concerned, we have tried to make the main story understandable to anyone with just a basic knowledge of discrete probability, though sometimes deeper concepts are used. But, in those cases, we have tried to minimize the need for prior familiarity by presenting the necessary material within our text.

It is worth clarifying however that this book is not aimed at readers interested in the legal, ethical, or sociological aspects of information flow. While it is clear that some information flows are beneficial and others are harmful, we make no effort to address the question of which are which.

And finally, we recognize that information flow is in fact a *general phenomenon* with relevance beyond security. So while the theory developed here has largely been motivated by the question of how to limit the leakage of sensitive information, that same theory can no doubt be applied fruitfully in diverse contexts such as machine learning, recommendation systems, and robotics. (Interestingly, in those contexts information flow would typically be seen as a *good* thing.) For this reason, readers outside the field of security may also profit from reading this book.

Organization and structure

We now briefly describe the overall structure of the book.

In Part I, we motivate the study of quantitative information flow, and we give an informal overview of some of its important concepts by discussing information leakage in a very simple context.

In Part II, we begin our detailed development by explaining what a *secret* X actually is, or at least what we consider it to be: a *probability distribution* π that specifies the adversary’s knowledge about the likelihood of X ’s possible values. We also consider how π can be used in quantifying either X ’s *vulnerability* or (complementarily) the adversary’s *uncertainty* about X , observing that there are *many* reasonable ways to do that, depending on the operational scenario, and showing that a single framework, based on “gain functions” (or dually “loss functions”), can encompass them all.

In Part III, we move from secrets to *systems*, modeled as information-theoretic channels that process secret information and possibly leak some of it to their public outputs. We develop a rich family of gain-function–leakage measures to quantify the damage a channel’s leakage might cause, carefully considering the *operational significance* of such measures and developing theory that supports *robust* judgments about leakage.

In Part IV, we consider a more detailed model of systems as *programs* written in a simple probabilistic imperative programming language, enabling *compositional reasoning* about information leakage. Here, with assignment statements to program variables we can treat secrets that *change over time*. For that we introduce a mathematical technique that generalizes both channels (which leak secrets) and assignments (which update them). The technique is based on *Hidden Markov Models*.

Finally, in Part V we present a number of *case studies* showing how one can apply quantitative–information-flow analysis to many interesting realistic scenarios — including anonymity protocols, side-channel attacks on cryptography, voting protocols, and even differential privacy in statistical databases. Those chapters are intended to be somewhat self-contained, and readers interested in applications might wish to browse through them early.

Details of presentation

We sometimes format a definition, theorem, or paragraph in a box to give it greater visual prominence, as we have done in this paragraph. Our intent in doing that is to express our judgments, necessarily subjective, about which things are particularly significant or interesting.

The main text has been kept essentially free of literature citations and historical remarks — instead they are collected in a final section “Chapter Notes” for each chapter. The bibliography is, similarly, organized chapter by chapter.

Cited authors can be found alphabetically in the index, where they appear within square brackets, for example “[Claude E. Shannon]”. A glossary appears just before the index, and its entries are in order of first occurrence in the main text. The entry usually reads “*see something*”, without a page number, in the hope that the something on its own will be enough to jog the memory. If it isn’t, the index entry for “something” itself should be consulted to get a page reference.

Possible usage as a textbook

We have used draft chapters from Parts I, II, and III in a master’s-level course on the foundations of cybersecurity that also included extensive coverage of cryptography.

For a full-semester course, we envisage that a course based on Parts I, II, and III and selected chapters from Part V could be taught at both the advanced undergraduate and master’s levels. Part IV is more advanced mathematically, and is probably more suitable for doctoral students.

To facilitate the use of the book as a course textbook, we have included a section of *Exercises* at the end of most chapters. Solutions to these exercises are available to qualified instructors.

Language issues

Turning finally to questions of language: we come from six different countries (Brazil, Greece, the United Kingdom, Australia, Italy, and the United States) — which had the advantage that the sun never set on this book’s preparation: at all times at least one of us could be found hard at work on it. But such diversity also raises issues of spelling and usage. For the sake of consistency we have made an essentially arbitrary choice to follow American conventions throughout.

Also, with respect to the thorny question of personal pronouns, we have chosen to refer to the *defender* (i.e. the person or entity trying to protect sensitive information) as “he” or “him”, to the *adversary* as “she” or “her”, and to the *authors* and *readers* of this book as “we” or “us”. When there are several points of view, for example in multi-party protocols, we will occasionally use the neuter “it”. While assigning genders to the defender and adversary is of course arbitrary (and some readers might indeed prefer the opposite assignment), it has the advantages of avoiding the syntactic awkwardness of “he or she” and, more importantly, of enabling us to write with greater clarity and precision.

Acknowledgments

Our many collaborators have made profound contributions to our understanding of quantitative information flow — and we are particularly grateful to Arthur Américo, Miguel Andrés, Nicolás Bordenabe, Chris Chen, Michael R. Clarkson, Pierpaolo Degano, Kai Engelhardt, Barbara Espinoza, Natasha Fernandes, Jeremy Gibbons, Michael Hicks, Yusuke Kawamoto, Boris Köpf, Piotr Mardziel, Larissa Meinicke, Ziyuan Meng, Tahiry Rabehaja, Andre Scedrov, Fred B. Schneider, Tom Schrijvers, David M. Smith, Marco Stronati, and Roland Wen.

The authors are grateful for support from Digiteo and the Inria équipe associée Princess. Also, Mário S. Alvim was supported by the Computer Science Department at Universidade Federal de Minas Gerais (DCC/UFGM), by the National Council for Scientific and Technological Development (CNPq), by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), and by the Fundação de Amparo à Pesquisa de Minas Gerais (FAPEMIG). Konstantinos Chatzikokolakis was supported by the Centre national de la recherche scientifique (CNRS), by the Institut national de recherche en sciences et technologies du numérique (Inria), and by the Department of Informatics and Telecommunications of the National and Kapodistrian University of Athens. Annabelle McIver was supported by the Department of Computing at Macquarie University and the Optus Macquarie Cyber Security Hub, Carroll Morgan by the Trustworthy Systems Group of CSIRO’s Data61 and the School of Engineering and Computer Science at the University of New South Wales, and both of them by the Australian Research Council and the Information Security Group at ETH Zürich. Catuscia Palamidessi was supported by the Institut national de recherche en sciences et technologies du numérique (Inria), by her ERC grant HYPATIA and by the ANR project REPAS. Geoffrey Smith was supported by the School of Computing and Information Sciences at Florida International University and by the National Science Foundation under grant CNS-1116318.

Belo Horizonte
Athens
Sydney
Sydney
Paris
Miami

April 2020

Mário S. Alvim
Konstantinos Chatzikokolakis
Annabelle McIver
Carroll Morgan
Catuscia Palamidessi
Geoffrey Smith

Contents

Preface	vii
I Motivation	1
1 Introduction	3
1.1 A first discussion of information leakage	5
1.1.1 Secrets	5
1.1.2 Bayes vulnerability	5
1.1.3 Deterministic channels	6
1.1.4 Posterior distributions and hyper-distributions	7
1.1.5 Posterior Bayes vulnerability	8
1.1.6 Quantifying leakage	9
1.2 Looking ahead	10
1.3 Exercises	11
1.4 Chapter notes	12
II Secrets and How to Measure Them	15
2 Modeling secrets	17
2.1 Secrets and probability distributions	17
2.2 Shannon entropy	18
2.3 Bayes vulnerability	20
2.4 A more general view	21
2.5 Exercises	22
2.6 Chapter notes	22
3 On g-vulnerability	25
3.1 Basic definitions	25
3.1.1 Graphing g -vulnerability	27
3.2 A catalog of gain functions	29
3.2.1 The identity gain function	29
3.2.2 Gain functions induced from distance functions	30
3.2.3 Binary gain functions	31
3.2.4 Gain functions for a password database	33

3.2.5	A gain function that penalizes wrong guesses	34
3.2.6	A gain function for a medical diagnosis scenario	35
3.2.7	A loss function that gives guessing entropy	35
3.2.8	A loss function that gives Shannon entropy	37
3.3	Classes of gain functions	39
3.3.1	Finite-valued, non-negative vulnerabilities: the class $\mathbb{G}\mathcal{X}$	39
3.3.2	Finitely many actions: $\mathbb{G}^{\text{fin}}\mathcal{X}$	40
3.3.3	Non-negative gain functions: $\mathbb{G}^+\mathcal{X}$	40
3.3.4	One-bounded gain functions: $\mathbb{G}^\ddagger\mathcal{X}$	41
3.4	Mathematical properties	41
3.4.1	Gain function algebra	42
3.5	On “absolute” versus “relative” security	43
3.6	Exercises	44
3.7	Chapter notes	44
III	Channels and Information Leakage	47
4	Channels	49
4.1	Channel matrices	49
4.2	The effect of a channel on the adversary’s knowledge	51
4.3	From joint distributions to hyper-distributions	54
4.4	Abstract channels	57
4.5	More on abstract channels	61
4.6	A first look at channel compositions	63
4.6.1	Convex combinations of channels	63
4.6.2	Cascading and the Data-Processing Inequality	64
4.7	Exercises	65
4.8	Chapter notes	66
5	Posterior vulnerability and leakage	71
5.1	Posterior g -vulnerability and its basic properties	71
5.2	Multiplicative and additive g -leakage	80
5.3	A closer look at posterior Bayes vulnerability and Bayes leakage	82
5.4	Measuring leakage with Shannon entropy	84
5.5	More properties of posterior g -vulnerability and g -leakage	86
5.5.1	A matrix-based formulation of posterior g -vulnerability	86
5.5.2	A trace-based formulation of posterior g -vulnerability	87
5.5.3	A linear-programming formulation	90
5.6	Example channels and their leakage	91
5.7	Max-case posterior g -vulnerability	93
5.8	Exercises	94
5.9	Chapter notes	97
6	Robustness	101
6.1	The need for robustness	101
6.2	Approaches to robustness	103
6.3	Exercises	103
6.4	Chapter notes	103

7 Capacity	107
7.1 Multiplicative Bayes capacity	107
7.2 Additive Bayes capacity	111
7.3 General capacities	116
7.4 Multiplicative capacities	117
7.4.1 Fixed g , maximize over π	117
7.4.2 Fixed π , maximize over g	118
7.4.3 Maximize over both g and π	119
7.5 Additive capacities	119
7.5.1 Fixed g , maximize over π	119
7.5.2 Fixed π , maximize over g	120
7.5.3 Maximize over both g and π	123
7.6 Obtaining bounds on leakage	124
7.6.1 The additive miracle theorem	124
7.6.2 Improved miracle bounds	124
7.6.3 Examples	125
7.7 Exercises	127
7.8 Chapter notes	127
8 Composition of channels	131
8.1 Compositions of (concrete) channel matrices	131
8.1.1 Parallel composition	132
8.1.2 External fixed-probability choice	133
8.1.3 External conditional choice	134
8.1.4 External (general) probabilistic choice	135
8.1.5 Internal fixed-probability choice	136
8.1.6 Internal conditional choice	137
8.1.7 Internal (general) probabilistic choice	137
8.1.8 Cascading	137
8.2 Compositions of abstract channels	138
8.2.1 The issue of compositionality	138
8.2.2 Parallel composition	139
8.2.3 External fixed-probability choice	139
8.2.4 External conditional choice	140
8.2.5 External (general) probabilistic choice	140
8.2.6 The internal choices, and cascading	140
8.3 Exercises	142
8.4 Chapter notes	143
9 Refinement	147
9.1 Refinement: for the <i>customer</i> ; for the <i>developer</i>	147
9.2 Structural refinement: the developer's point of view	148
9.2.1 Structural refinement for deterministic channels	148
9.2.2 Structural refinement for probabilistic channels	150
9.3 Testing refinement: the customer's point of view	152
9.4 Soundness of structural refinement	153
9.5 Completeness of structural refinement: the Coriaceous theorem	154
9.6 The structure of abstract channels under refinement	157
9.7 Refinement and monotonicity	159
9.7.1 Compositionality for contexts	159
9.7.2 Monotonicity with respect to refinement	160

Contents

9.8	Why does refinement (\sqsubseteq) have to be so complicated?	160
9.8.1	Who gets to define refinement, anyway?	160
9.8.2	A subjective argument: keeping the customer satisfied	162
9.8.3	An objective argument: compositional closure	164
9.9	Capacity is unsuitable as a criterion for refinement	166
9.10	Exercises	167
9.11	Chapter notes	167
10	The Dalenius perspective	171
10.1	Dalenius scenarios	172
10.2	Compositional closure for Dalenius contexts	175
10.2.1	Safety and necessity with respect to Dalenius contexts	175
10.2.2	Justifying refinement: an example	176
10.3	Bounding Dalenius leakage	177
10.4	Chapter notes	179
11	Axiomatics	183
11.1	An axiomatic view of vulnerability	183
11.2	Axiomatization of prior vulnerabilities	185
11.2.1	Soundness and completeness of V_g with respect to continuous, convex functions	186
11.3	Axiomatization of posterior vulnerabilities	188
11.3.1	Possible definitions of posterior vulnerabilities	189
11.4	Applications of axiomatization to understanding leakage measures	197
11.5	Chapter notes	199
12	The geometry of hypers, gains and losses	205
12.1	Barycentric representation of gain/loss functions	208
12.2	Barycentric representation of hypers and their refinement	210
12.3	Primitive hyper-distributions and their refinements	213
12.4	Hyper-distributions are not a lattice under refinement	216
12.5	A geometric proof of antisymmetry of refinement	218
12.6	Exercises	220
12.7	Chapter notes	220
IV	Information Leakage in Sequential Programs	223
13	Quantitative information flow in sequential computer programs	225
13.1	Markovs don't leak; and channels don't update	226
13.2	Specifications and implementations: a review	228
13.2.1	When is one <i>program</i> better than another, and why?	228
13.2.2	When is one <i>channel</i> better than another, and why?	229
13.2.3	Programs and channels together: what is "better" for both?	230
13.3	Aligning functional refinement with information-flow refinement	230
13.3.1	Generalizing Hoare logic for probability	230
13.3.2	Using loss functions	231
13.3.3	Refinement in general	232
13.3.4	Initial-final correlations, and Dalenius	233
13.4	Larger information-flow-aware programs	235
13.4.1	Sequential composition	235
13.4.2	On the terms <i>prior</i> , <i>posterior</i> , <i>initial</i> and <i>final</i>	240

13.4.3	Conditionals	241
13.4.4	The power of the adversary: <i>gedanken</i> experiments	242
13.4.5	Iteration	243
13.5	Syntax for probabilistic choice	243
13.6	Summary	246
13.7	Exercises	246
13.8	Chapter notes	247
14	Hidden-Markov modeling of <i>QIF</i> in sequential programs	255
14.1	Concrete Hidden Markov Models	255
14.1.1	<i>A priori</i> versus <i>a posteriori</i> reasoning — in more detail	257
14.2	Operations on and specializations of concrete HMM's	258
14.2.1	Pure-channel and pure-markov HMM's	258
14.2.2	Sequential composition of concrete HMM's	258
14.2.3	General (concrete) HMM's	260
14.3	Abstract Hidden Markov Models	260
14.3.1	Sequential (Kleisli) composition of abstract HMM's	261
14.4	Syntax and abstract-HMM semantics of <i>QIF</i> -programs	264
14.4.1	Probabilistic assignment	264
14.4.2	Information flow via channels: leaking with PRINT	265
14.4.3	External probabilistic choice	266
14.4.4	(Internal probabilistic choice)	267
14.4.5	Sequential composition	268
14.4.6	Conditional	268
14.4.7	Iteration	268
14.4.8	Local variables	268
14.5	Leaks caused by conditionals and by external choice	270
14.6	Examples of small <i>QIF</i> programs	272
14.6.1	First example: Bertrand's Boxes	272
14.6.2	Second example: Goldfish or piraña?	274
14.6.3	Third example: Repeated independent runs	275
14.7	Underlying and unifying structures: a summary	275
14.8	Exercises	278
14.9	Chapter notes	279
15	Program algebra for <i>QIF</i>	283
15.1	Semantics, logic, and program algebra	283
15.2	Static visibility declarations; multiple variables	284
15.3	Simple examples of program derivations in <i>QIF</i>	286
15.3.1	The Encryption Lemma	286
15.3.2	From qualitative proofs to quantitative proofs	287
15.3.3	The One-Time Pad	287
15.4	Algebraic rules for reordering statements	290
15.5	Larger example 1: Oblivious Transfer	291
15.6	Larger example 2: Two-party conjunction, or <i>The Lovers' protocol</i>	296
15.7	Sub-protocols and declassification	298
15.8	Refinement and quantitative analyses	298
15.9	Exercises	301
15.10	Chapter notes	304

16 Iteration and nontermination	307
16.1 Why iteration is “different”	307
16.2 Classical nontermination	307
16.3 Nontermination for markovs and channels	308
16.3.1 Nontermination for markovs	308
16.3.2 Nontermination for channels	310
16.3.3 Applying abstract channels and markovs to sub-hypers	310
16.3.4 The semantic model for nontermination	311
16.4 The algebra of nontermination in <i>QIF</i>	311
16.5 A refinement order on <i>sub</i> -hyper-distributions	313
16.6 From nontermination to termination	316
16.7 Example of (certain) termination: how to design a password checker	317
16.8 A taxonomy of refinement orders	319
16.9 Exercises	321
16.10 Chapter notes	321
17 A demonic lattice of information	325
17.1 A <i>deterministic</i> lattice of information — the original	325
17.1.1 Historical introduction, intuition and abstraction	325
17.1.2 Structural definition of refinement for deterministic channels	328
17.1.3 Testing, soundness and completeness: deterministic	329
17.2 Our probabilistic partial order	330
17.3 Basic structure of the demonic lattice	331
17.4 Examples of demonically nondeterministic channels	334
17.5 Testing, soundness and completeness: demonic	336
17.6 A reformulation of demonic testing	337
17.7 Reduced demonic channels	339
17.8 Compositional closure	339
17.9 “Weakest pre-tests” and source-level reasoning	342
17.10 Exercises	345
17.11 Chapter notes	346
V Applications	351
18 The Crowds protocol	353
18.1 Introduction to Crowds, and its purpose	353
18.2 Modeling the Crowds protocol	354
18.3 Bayes vulnerability and Bayes leakage	357
18.4 Explanation of the paradox	358
18.4.1 Modified Crowds	358
18.4.2 Vulnerability of the original protocol	359
18.5 Why φ matters, even for uniform priors	360
18.5.1 Probable innocence as no lion leakage	361
18.6 Refinement: increasing φ is always safe	361
18.7 Multiple paths	363
18.7.1 Paths recreated by the initiator	363
18.7.2 Paths repaired by the last working node	364
18.7.3 Multiple detections and deviating from the protocol	365
18.8 Exercises	365
18.9 Chapter notes	366

19 Timing attacks on blinded and bucketed cryptography	369
19.1 Cryptographic background	369
19.2 A first leakage bound	370
19.3 A better leakage bound	372
19.4 Analytic results about $\text{cap}_b(n)$	374
19.5 Analytic proofs	378
19.6 Another proof of Theorem 19.5	384
19.7 Chapter notes	385
20 Defense against side channels	389
20.1 Evaluating a defense against side channels	389
20.2 <i>QIF</i> exploration of the fast-exponentiation algorithm	391
20.2.1 Cost/benefit analysis	394
20.3 Chapter notes	395
21 Multi-party computation: The Three Judges protocol	399
21.1 Introduction to The Three Judges	400
21.2 Developing an implementation of the Three Judges	401
21.2.1 First attempt	401
21.2.2 Second development attempt (sketch)	402
21.2.3 Successful development	403
21.2.4 Two-party exclusive-or	405
21.2.5 Summary	406
21.3 Exercises	409
21.4 Chapter notes	409
22 Voting systems	413
22.1 Elections and privacy risks	413
22.2 An illustrative and simplified <i>QIF</i> model for elections	414
22.2.1 The tallying	414
22.2.2 The casting	415
22.2.3 The Dalenius perspective: casting then tallying	416
22.3 Election by simple majority: first past the post	417
22.3.1 <i>QIF</i> channels for simple-majority elections: two examples	417
22.4 Election by preferences: instant run-off	418
22.4.1 <i>QIF</i> channels for instant-run-off elections: two examples	419
22.5 Gain functions for privacy of elections: a first example	419
22.6 The effect of small electorates in general	421
22.7 Case studies of small-electorate impact	422
22.7.1 First past the post, in small electorates	422
22.7.2 Instant run-off in small electorates	426
22.8 Chapter notes	429
23 Differential privacy	433
23.1 Notation and definition	434
23.2 Mechanisms as information-theoretic channels	435
23.3 The relation between differential privacy and multiplicative g -leakage	436
23.3.1 Bounds on leakage do not imply differential privacy	438
23.4 Exercises	439
23.5 Chapter notes	441
Glossary and Index	445

List of definitions, theorems, examples, *etc.*

Theorem 1.1	8	Example 4.17	62
Corollary 1.2	9	Definition 4.18	64
Definition 2.1	17	Example 5.1	71
Conjecture 2.2	20	Definition 5.2	72
Definition 2.3	20	Example 5.3	73
Definition 3.1	25	Example 5.4	73
Definition 3.2	26	Example 5.5	75
Example 3.3	26	Theorem 5.6	77
Definition 3.4	27	Theorem 5.7	78
Definition 3.5	30	Theorem 5.8	78
Theorem 3.6	30	Theorem 5.9	79
Definition 3.7	30	Theorem 5.10	79
Definition 3.8	31	Definition 5.11	80
Definition 3.9	39	Theorem 5.12	80
Definition 3.10	40	Theorem 5.13	81
Definition 3.11	40	Example 5.14	81
Definition 3.12	41	Theorem 5.15	83
Theorem 3.13	41	Example 5.16	83
Theorem 3.14	42	Theorem 5.17	84
Definition 4.1	50	Theorem 5.18	86
Example 4.2	52	Example 5.19	87
Theorem 4.3	53	Theorem 5.20	87
Example 4.4	56	Definition 5.21	87
Definition 4.5	56	Lemma 5.22	88
Definition 4.6	57	Theorem 5.23	88
Definition 4.7	57	Theorem 5.24	89
Corollary 4.8	58	Example 5.25	89
Definition 4.9	59	Example 5.26	89
Theorem 4.10	59	Algorithm 5.27	90
Corollary 4.11	59	Algorithm 5.28	91
Example 4.12	59	Definition 5.29	93
Definition 4.13	60	Theorem 5.30	94
Definition 4.14	61	Theorem 5.31	94
Example 4.15	61	Definition 7.1	107
Theorem 4.16	62	Theorem 7.2	108

Corollary 7.3	108	Example 9.15	158
Corollary 7.4	108	Definition 9.16	160
Theorem 7.5	109	Definition 9.17	162
Example 7.6	109	Example 9.18	163
Theorem 7.7	110	Definition 9.19	165
Theorem 7.8	110	Example 9.20	166
Definition 7.9	111	Definition 10.1	173
Example 7.10	111	Theorem 10.2	173
Theorem 7.11	112	Example 10.3	173
Theorem 7.12	113	Theorem 10.4	175
Definition 7.13	116	Definition 10.5	177
Theorem 7.14	118	Theorem 10.6	178
Example 7.15	118	Theorem 10.7	178
Definition 7.16	121	Theorem 10.8	179
Definition 7.17	121	Definition 11.1	185
Lemma 7.18	121	Definition 11.2	185
Lemma 7.19	122	Definition 11.3	186
Theorem 7.20	122	Theorem 11.4	187
Theorem 7.21	122	Theorem 11.5	187
Example 7.22	123	Definition 11.6	188
Theorem 7.23	124	Definition 11.7	188
Theorem 7.24	125	Definition 11.8	189
Definition 8.1	132	Definition 11.9	190
Definition 8.2	133	Theorem 11.10	190
Definition 8.3	134	Theorem 11.11	191
Definition 8.4	135	Theorem 11.12	191
Definition 8.5	136	Lemma 11.13	191
Definition 8.6	138	Theorem 11.14	192
Definition 8.7	138	Definition 11.15	193
Definition 8.8	139	Theorem 11.16	193
Definition 8.9	139	Example 11.17	194
Definition 8.10	140	Theorem 11.18	194
Definition 8.11	140	Theorem 11.19	194
Definition 8.12	141	Corollary 11.20	195
Lemma 8.13	141	Definition 11.21	195
Definition 8.14	141	Theorem 11.22	196
Lemma 8.15	141	Example 11.23	196
Definition 9.1	149	Definition 12.1	210
Theorem 9.2	149	Lemma 12.2	213
Theorem 9.3	149	Definition 12.3	216
Theorem 9.4	150	Lemma 12.4	218
Definition 9.5	150	Corollary 12.5	218
Definition 9.6	151	Example 12.6	220
Theorem 9.7	151	Definition 13.1	232
Example 9.8	151	Definition 14.1	258
Corollary 9.9	152	Definition 14.2	260
Definition 9.10	152	Definition 14.3	262
Theorem 9.11	153	Theorem 14.4	263
Theorem 9.12	155	Definition 16.1	309
Theorem 9.13	156	Definition 16.2	313
Theorem 9.14	157	Definition 16.3	314

Definition 16.4	314	Theorem 18.3	363
Lemma 16.5	315	Theorem 18.4	363
Corollary 16.6	315	Theorem 19.1	371
Theorem 16.7	315	Definition 19.2	372
Theorem 16.8	316	Theorem 19.3	373
Definition 17.1	331	Theorem 19.4	375
Definition 17.2	332	Theorem 19.5	376
Definition 17.3	332	Theorem 19.6	377
Definition 17.4	333	Theorem 19.7	377
Definition 17.5	333	Theorem 19.8	377
Lemma 17.6	333	Definition 22.1	419
Lemma 17.7	333	Definition 22.2	420
Lemma 17.8	333	Definition 22.3	420
Lemma 17.9	334	Definition 22.4	420
Theorem 17.10	337	Definition 22.5	421
Definition 17.11	337	Definition 22.6	422
Lemma 17.12	338	Theorem 22.7	423
Lemma 17.13	339	Theorem 22.8	423
Definition 17.14	339	Definition 23.1	435
Lemma 17.15	339	Theorem 23.2	437
Definition 17.16	340	Corollary 23.3	437
Definition 17.17	340	Theorem 23.4	437
Theorem 17.18	341	Example 23.5	438
Definition 18.1	354	Example 23.6	439
Theorem 18.2	361		

List of figures

Figure 3.1	28	Figure 17.2	327
Figure 3.2	29	Figure 17.3	327
Figure 3.3	35	Figure 17.4	329
Figure 3.4	36	Figure 17.5	329
Figure 3.5	37	Figure 17.6	330
Figure 4.1	53	Figure 17.7	343
Figure 5.1	75	Figure 18.1	354
Figure 5.2	76	Figure 18.2	357
Figure 5.3	77	Figure 18.3	361
Figure 5.4	82	Figure 19.1	373
Figure 9.1	148	Figure 19.2	374
Figure 9.2	156	Figure 19.3	375
Figure 9.3	162	Figure 20.1	390
Figure 11.1	190	Figure 20.2	391
Figure 11.2	193	Figure 20.3	392
Figure 11.3	198	Figure 20.4	393
Figure 12.1	206	Figure 21.1	406
Figure 12.2	207	Figure 21.2	407
Figure 12.3	211	Figure 21.3	408
Figure 12.4	211	Figure 22.1	418
Figure 12.5	212	Figure 22.2	424
Figure 12.6	212	Figure 22.3	424
Figure 12.7	214	Figure 22.4	425
Figure 12.8	215	Figure 22.5	426
Figure 12.9	217	Figure 22.6	427
Figure 12.10	217	Figure 22.7	428
Figure 12.11	219	Figure 22.8	428
Figure 14.1	256	Figure 22.9	429
Figure 14.2	259	Figure 23.1	435
Figure 14.3	263	Figure 23.2	436
Figure 15.1	295	Figure 23.3	438
Figure 15.2	299	Figure 23.4	439
Figure 17.1	326		

List of exercises

Exercise 1.1	11	Exercise 9.4	167
Exercise 1.2	11	Exercise 9.5	167
Exercise 2.1	22	Exercise 9.6	167
Exercise 3.1	44	Exercise 9.7	167
Exercise 3.2	44	Exercise 9.8	167
Exercise 4.1	65	Exercise 9.9	167
Exercise 4.2	66	Exercise 9.10	167
Exercise 4.3	66	Exercise 12.1	220
Exercise 4.4	66	Exercise 13.1	246
Exercise 5.1	94	Exercise 13.2	246
Exercise 5.2	94	Exercise 13.3	247
Exercise 5.3	94	Exercise 13.4	247
Exercise 5.4	95	Exercise 14.1	278
Exercise 5.5	96	Exercise 14.2	278
Exercise 5.6	96	Exercise 14.3	279
Exercise 5.7	96	Exercise 14.4	279
Exercise 6.1	103	Exercise 14.5	279
Exercise 7.1	127	Exercise 15.1	301
Exercise 7.2	127	Exercise 15.2	301
Exercise 7.3	127	Exercise 15.3	302
Exercise 7.4	127	Exercise 15.4	302
Exercise 7.5	127	Exercise 15.5	302
Exercise 8.1	142	Exercise 15.6	303
Exercise 8.2	142	Exercise 15.7	303
Exercise 8.3	142	Exercise 15.8	303
Exercise 8.4	142	Exercise 15.9	303
Exercise 8.5	143	Exercise 15.10	303
Exercise 8.6	143	Exercise 15.11	303
Exercise 8.7	143	Exercise 15.12	303
Exercise 8.8	143	Exercise 15.13	303
Exercise 8.9	143	Exercise 16.1	321
Exercise 8.10	143	Exercise 16.2	321
Exercise 8.11	143	Exercise 16.3	321
Exercise 8.12	143	Exercise 16.4	321
Exercise 9.1	167	Exercise 16.5	321
Exercise 9.2	167	Exercise 17.1	345
Exercise 9.3	167	Exercise 17.2	345

List of exercises

Exercise 17.3	345	Exercise 18.1	365
Exercise 17.4	345	Exercise 18.2	365
Exercise 17.5	345	Exercise 21.1	409
Exercise 17.6	345	Exercise 21.2	409
Exercise 17.7	345	Exercise 21.3	409
Exercise 17.8	345	Exercise 21.4	409
Exercise 17.9	346	Exercise 23.1	439
Exercise 17.10	346	Exercise 23.2	440
Exercise 17.11	346	Exercise 23.3	440
Exercise 17.12	346	Exercise 23.4	440
Exercise 17.13	346		

Part I

Motivation



Chapter 1

Introduction

Protecting sensitive information from improper disclosure or corruption is a long-standing, fundamental goal of computer security: but it is one that is not currently being achieved well at all, as is evident from continual news reports of large-scale data compromises. Such compromises result from many causes, of course. Sometimes the compromises are essentially *social*, as in “phishing” attacks: users are led to fraudulent websites and tricked into disclosing their authentication credentials there, enabling attackers to impersonate them and to gain access to their sensitive data. Other compromises are more technical, resulting from flaws (both malicious and inadvertent) in the design and implementation of computer systems that process sensitive data.

Whenever a compromise is discovered, it is natural and important to investigate its particular causes, so that whatever flaws were exploited can be corrected. But the approach of patching flaws as they are discovered is fundamentally limited — at best it brings us to a situation where all the flaws we know about have been corrected, but it leaves open the question of whether there might be other flaws yet to be discovered.

Achieving a secure and trustworthy cyber-infrastructure requires a more disciplined approach, one where the focus turns from particular attacks to a true *science of security*. This book aims to contribute to such a science by carefully studying one aspect of computer security, namely the *information flow* that occurs when a computer system processes sensitive information.

Imagine a system that is given some sensitive information as input and then, as it processes it, somehow produces some publicly observable output. The output might be the result of the computation, intentionally made public, or it might be some aspect of the computation itself, such as time taken, or energy used, or number of cache faults — such outputs, often unintentional, are called *side channels*. A basic question that we wish to answer is whether the system causes some of the sensitive information to “flow” from the input to the observable output, thereby causing “leakage” of that information — and of course we need to understand precisely what that means. Also, while our first thought might be that a “secure” system should have no such leakage, we find in fact that some leakage is unavoidable in many practical situations.

Consider for example a *password checker* that has been given a secret password, which of course it should not leak. But when a user tries to log in with some guessed password, the checker must reveal whether the guess is correct or not, and rejecting an incorrect guess leaks the fact that the secret password is different from the guess.

As a second example, consider an *election-tallying* program that takes as inputs the ballots of a group of voters. Typically we would demand that the ballots be kept secret, yet the election system needs to output the result of the election, usually

including the tally of votes for each candidate. That clearly leaks some information about the secret ballots — in the extreme case of an election that turns out to be unanimous, for instance, the tally reveals how everyone voted. What then is the best choice for publishing the election results so that the aims of both integrity and privacy are served?

A third example involves side channels. In typical implementations the *time* required to do an *RSA* decryption varies, depending on the secret key: that can lead to a timing attack. Similarly, the *cache* state that results from an *AES* decryption varies, depending on the secret key. In both those cases, there is then leakage from the secret key to system timing or caching behavior, which might be publicly observable and might indeed leak enough to allow an adversary to recover the secret key. If some defense against the side channel is employed, how do we determine its effectiveness?

Those last examples highlight an important challenge concerning the modeling of the computer systems we wish to analyze. To facilitate our analyses, we would prefer simple mathematical models that exhibit the essential features of the systems while abstracting from irrelevant details. But what is *essential* and what is *irrelevant*? As shown by side channels, many low-level system details (e.g. timing, caching behavior, power consumption), which a mathematical model might naturally ignore, turn out to give significant leaks of sensitive information. As a consequence, we need to choose system models with a skeptical eye, always being mindful of the abstractions that we make and sensitive to the issues that we might be overlooking. In the case of a password checker, for instance, we might model the checker as outputting only whether the guessed password is correct or not. But it might turn out that the implementation of the checker works by comparing the guess to the secret password character by character, rejecting as soon as a mismatch is found. In that case, the *time* taken by the checker would be proportional to the length of the maximum correct prefix of the guess. If an adversary could observe that running time precisely, the leakage would be far greater.

Returning to the issue of whether a system leaks sensitive information, we see from the above examples that “whether” is often not very useful, as all of those systems do in fact leak some information. But it is intuitively clear that many of those leaks are “small”, and that suggests that it would be more fruitful to ask instead *how much* sensitive information is leaked and *how useful* it is to the adversary: perhaps it’s “large”? In the case of an election system, for instance, we would expect to be able to show that if the number of voters is large, then the leakage caused by releasing the tally is somehow “small”, enabling us to show that a *quantitative* leakage policy is satisfied. But what, precisely, would such a policy mean, and what security guarantees would it provide?

To address such questions, in this book we develop a theory of *Quantitative Information Flow*, which aims to explain precisely what information leakage *is*, how it can be *assessed* quantitatively, and how systems can be constructed that satisfy rigorous information-flow *guarantees*. We begin in the following section with an informal discussion of information leakage, briefly introducing some key concepts of Quantitative Information Flow and trying to build intuition that will motivate the detailed discussion in later chapters.

1.1 A first discussion of information leakage

In this section we discuss information leakage in a “toy” system, chosen to make the analysis relatively simple.

1.1.1 Secrets

Let us consider a simple example. Suppose that a secret called X is generated by rolling a pair of distinguishable dice, one red and one white. Then the set \mathcal{X} of possible values of X has 36 elements, as shown here:

We can write each such value more compactly as a pair (r, w) , where r is the value of the red die and w is the value of the white die. (From a security perspective, we can think of X as a two-digit PIN, where the digits are limited to numbers from 1 to 6.)

What does it mean to say that X is a *secret* with respect to an adversary? We take that to mean that the adversary knows only a *probability distribution* π that specifies the probability π_x of each possible value x of X . (Secrets will be defined formally in Def. 2.1.) If we assume here that the two dice are fair and independent, then in fact π will be *uniform*, i.e. will be such that $\pi_x = \pi_{x'}$ for all x, x' in \mathcal{X} . As a mnemonic, we will usually write ϑ for the uniform distribution, generic in \mathcal{X} ,¹ so that in this case the probability $1/36$ is assigned to each outcome: that is,

$$\vartheta_{(1,1)} = \vartheta_{(1,2)} = \vartheta_{(1,3)} = \cdots = \vartheta_{(6,6)} = 1/36 \quad .$$

In general (i.e. whether it is uniform or not) we refer to the above as a *prior distribution*, because it reflects the adversary’s knowledge about X *before* observing the output of a system. Later we will discuss *posterior distributions*, which reflect the adversary’s knowledge *after* observing the output.

1.1.2 Bayes vulnerability

Assuming that the adversary’s knowledge of X is limited to π , we now wish to quantify the “threat” to X . As we will discuss subsequently, there are many reasonable ways of doing that. But for now we focus on a basic measure that we call *Bayes vulnerability*, and which is an adversary’s maximum probability of guessing the value of X correctly in one try — clearly the adversary should guess any value x whose probability π_x is *maximum*. Denoting the Bayes vulnerability of π by $V_1(\pi)$, we then have

$$V_1(\pi) := \max_{x \in \mathcal{X}} \pi_x \quad .$$

¹ Think of ϑ as an elaborate “ u ”.

(The “1” subscript in V_1 is chosen to reflect the “one try” nature of Bayes vulnerability, which is discussed in detail in §2.3.) In our dice example, where the prior π is the uniform distribution ϑ , we have $V_1(\pi) = V_1(\vartheta) = 1/36$, since $1/36$ is the maximum probability assigned by ϑ ; in that case note that all 36 of the possible values are equally good guesses.

1.1.3 Deterministic channels

Now we turn our attention to the information leakage caused by a *system C* that processes a secret X .² In the first part of this book, we consider systems that take a secret X as input and whose only publicly observable behavior is to produce an output Y ; such systems are known as *channels*. In this section we restrict our attention to *deterministic* channels, where each input value x leads to a unique output value y , which we describe as $C(x)=y$. (More general systems are considered in Parts III and IV.) What is the effect of such a C on the secrecy of X ? The key observation is that an adversary seeing an output value y learns that the value of X must be one of the values in \mathcal{X} that are mapped by C to y ; all other values for X are eliminated. (That conclusion depends on the worst-case assumption, which we make throughout this book, that the adversary *knows* how channel C works. That assumption is related to the slogan “No security through obscurity” and it is sometimes called *Kerckhoffs’ Principle*.)

Returning to our dice example, we suppose that C takes as input the value (r, w) of X and outputs the sum of the two dice, so that $C(r, w) = r+w$. (From a security perspective, we can think of C as a malicious program that leaks the sum of the digits of a PIN.)

Here the space \mathcal{Y} of possible output values is $\{2, 3, 4, \dots, 12\}$, and the effect is to *partition* the space \mathcal{X} into blocks consisting of the pairs of dice that sum to each of those values:

Y	Possible values of X
2	$\{(1, 1)\}$
3	$\{(1, 2), (2, 1)\}$
4	$\{(1, 3), (2, 2), (3, 1)\}$
5	$\{(1, 4), (2, 3), (3, 2), (4, 1)\}$
6	$\{(1, 5), (2, 4), (3, 3), (4, 2), (5, 1)\}$
7	$\{(1, 6), (2, 5), (3, 4), (4, 3), (5, 2), (6, 1)\}$
8	$\{(2, 6), (3, 5), (4, 4), (5, 3), (6, 2)\}$
9	$\{(3, 6), (4, 5), (5, 4), (6, 3)\}$
10	$\{(4, 6), (5, 5), (6, 4)\}$
11	$\{(5, 6), (6, 5)\}$
12	$\{(6, 6)\}$

That partition reflects the 11 possible states of knowledge, or “worlds”, that an adversary seeing the output of C can end up in. Note that those blocks are not equally good from the adversary’s perspective: the blocks when Y is 2 or 12 are singletons, meaning that the value of X is known exactly, while the block when Y is 7 has size 6, meaning that the value of X remains quite uncertain.

Thinking about deterministic channels in general, we note that many partitions are possible. At one extreme, the partition might consist of a single block, which contains all of \mathcal{X} . That happens when the channel is a constant function, giving the same output on all inputs, which means that there is no leakage at all. Later we will call it

² We use upper-case letters like X for the names of secrets, and lower-case letters like x for the actual values they might take.

the “identity” channel, because it does not change the adversary’s state of knowledge. At the other extreme, the blocks in the partition might all be singletons. That happens when the channel is a one-to-one function, which means that X is leaked completely. It will be called the “zero” channel because of its secrecy-annihilating property. But in general the partition consists of a number of blocks of varying sizes, as in our dice channel C .

1.1.4 Posterior distributions and hyper-distributions

To understand the effect of the partition more precisely, we must also pay attention to probabilities. First, note that each block in the partition gives rise to a *posterior distribution* on \mathcal{X} : the set of possible values of X is restricted to the elements of the block, and the relative likelihood of those values is the same as in the prior distribution. In our dice example we are assuming a uniform prior distribution ϑ , which means that each of the 11 posterior distributions is also uniform. Second, note that each posterior distribution itself has a *probability* of occurring, which comes from the probability of the output value from which that posterior was deduced. That is in effect the probability that the adversary will find herself in that world. The result is that channel C maps the prior distribution ϑ to a *distribution on posterior distributions*, which we call a *hyper-distribution* and denote by $[\vartheta \triangleright C]$. (Hyper-distributions are discussed in detail in §4.3.) Here is a representation of the hyper-distribution for our dice example:

Probability	Posterior distribution on \mathcal{X} (each one sub-uniform)
$1/36$	$\{(1, 1)\}$
$2/36$	$\{(1, 2), (2, 1)\}$
$3/36$	$\{(1, 3), (2, 2), (3, 1)\}$
$4/36$	$\{(1, 4), (2, 3), (3, 2), (4, 1)\}$
$5/36$	$\{(1, 5), (2, 4), (3, 3), (4, 2), (5, 1)\}$
$6/36$	$\{(1, 6), (2, 5), (3, 4), (4, 3), (5, 2), (6, 1)\}$
$5/36$	$\{(2, 6), (3, 5), (4, 4), (5, 3), (6, 2)\}$
$4/36$	$\{(3, 6), (4, 5), (5, 4), (6, 3)\}$
$3/36$	$\{(4, 6), (5, 5), (6, 4)\}$
$2/36$	$\{(5, 6), (6, 5)\}$
$1/36$	$\{(6, 6)\}$

We are writing the distributions at right temporarily as *sets*, meaning however the uniform distribution over that set in each case. In general a *sub-uniform* distribution on \mathcal{X} is a uniform distribution on some subset of \mathcal{X} . Later we introduce notation for more specific distributions.

In talking about hyper-distribution $[\vartheta \triangleright C]$, we refer to the 11 posterior distributions as its *inner distributions*, and to their probabilities (given in the left column above) as its *outer distribution*. The outer distribution is quite important for understanding the effect of channel C — note in particular that the posterior distributions that are *most* desirable to the adversary (those coming from $Y=2$ and $Y=12$) are *least* likely (having probability $1/36$), while the *least* desirable (that coming from $Y=7$) is the *most* likely (having probability $6/36$).

1.1.5 Posterior Bayes vulnerability

As we have seen, the Bayes vulnerability $V_1(\pi)$ is a measure of the threat to X based only on the prior distribution π ; we will now refer to $V_1(\pi)$ as the *prior Bayes vulnerability*. How should we assess *posterior* Bayes vulnerability?

One approach, which could be called *dynamic*, is to consider the posterior vulnerability after a particular output value y has been observed. But if we wish to assess the channel C as a whole (for example to decide whether we are willing to run it) then it seems more appropriate to use a *static* approach that considers the entire hyper-distribution $[\pi \triangleright C]$. That lets us consider *all at once* the possible posterior distributions (inners) that the adversary might learn, each with its respective (outer) probability. Each inner distribution describes a world (state of knowledge) the adversary might inhabit after running the system, and its associated outer is the probability of that world's occurring.

Under the static perspective, we can naturally calculate the “threat” associated with each of the posterior distributions separately by taking its Bayes vulnerability. But how should we combine those numbers into a single value?

If we are pessimistic, we could consider the *maximum* Bayes vulnerability over all the posterior distributions, since that is the maximum knowledge that the adversary could learn from C . In the dice example, that maximum is 1, corresponding to the case when the sum is 2 or 12. But using that criterion would mean that the dice channel would be considered to be as bad as a channel that leaks X totally (since there the maximum probability is 1 for *every* inner, each one being in fact a point distribution on some $x=(r, w)$ in \mathcal{X}). Even worse, note that exactly the same result would occur in the case of an election with a million voters: the posterior distribution corresponding to a *unanimous* tally (however unlikely) has Bayes vulnerability of 1, which means that the election tally would be considered to be as bad as leaking the secret ballots completely.

For that reason, we instead define the posterior Bayes vulnerability of $[\pi \triangleright C]$, denoted $V_1[\pi \triangleright C]$, to be the *weighted average* of the Bayes vulnerabilities of the inner distributions, with the weights determined by the outer distribution. (A detailed discussion of posterior vulnerability is given in §5.1, and alternative definitions are considered in §5.7 and §11.3.1.) That definition of posterior Bayes vulnerability can be understood operationally as the probability that a smart adversary will correctly guess the value of secret X , randomly chosen according to π , given the output of C .

As an example of taking the weighted average, on our dice channel C we calculate

$$\begin{aligned} V_1[\pi \triangleright C] &= 1/36 \cdot 1 + 2/36 \cdot 1/2 + 3/36 \cdot 1/3 + 4/36 \cdot 1/4 + 5/36 \cdot 1/5 + 6/36 \cdot 1/6 + \\ &\quad 5/36 \cdot 1/5 + 4/36 \cdot 1/4 + 3/36 \cdot 1/3 + 2/36 \cdot 1/2 + 1/36 \cdot 1 \\ &= 1/36 + 1/36 + 1/36 + 1/36 + 1/36 + 1/36 + 1/36 + 1/36 + 1/36 + 1/36 \\ &= 11/36 . \end{aligned}$$

Note that all of the terms in the sum turn out to be $1/36$. That is in fact an interesting reflection of a general property of deterministic channels under uniform prior distributions.

Theorem 1.1 Let ϑ be a uniform prior distribution on an N -element set \mathcal{X} and let C be a deterministic channel with M possible output values (which implies that $1 \leq M \leq N$). Then $V_1[\vartheta \triangleright C] = M/N$.

Proof. Let the possible output values be y_1, y_2, \dots, y_M , and let the corresponding blocks have sizes s_1, s_2, \dots, s_M . Now the Bayes vulnerability given output y_m is $1/s_m$, since the posterior distribution is uniform on the s_m block elements. And the probability of output y_m is s_m/N . Hence we have

$$V_1[\vartheta \triangleright C] = \sum_{m=1}^M s_m/N \cdot 1/s_m = \sum_{m=1}^M 1/N = M/N .$$

□

Remarkably, in this case the *sizes* of the blocks do not matter; all that matters is the *number* of possible output values. This theorem is a simple, yet interesting, example of how the scientific study of quantitative information flow can yield results that might be found surprising: it says that if we have two deterministic channels C and D taking input X , then to compare the posterior Bayes vulnerabilities under a uniform prior ϑ , it suffices to *count* the number of possible outputs of C and of D ; the “shapes” of the two partitions do not matter.

Finally, notice that Thm. 1.1 implies that C never causes Bayes vulnerability to *decrease*, since we have $V_1(\vartheta) = 1/N \leq M/N = V_1[\vartheta \triangleright C]$.

1.1.6 Quantifying leakage

Now having quantified vulnerability before and after the operation of C , we are ready to consider how we might quantify the *leakage* of X caused by C . It is natural to do that by comparing the prior Bayes vulnerability $V_1(\pi)$ and the posterior Bayes vulnerability $V_1[\pi \triangleright C]$. The comparison can be done either *multiplicatively* (considering the *factor* by which vulnerability is increased) or *additively* (considering the *absolute amount* by which vulnerability is increased). We therefore define *multiplicative Bayes leakage* $\mathcal{L}_1^\times(\pi, C)$ by

$$\mathcal{L}_1^\times(\pi, C) := \frac{V_1[\pi \triangleright C]}{V_1(\pi)}$$

and *additive Bayes leakage* $\mathcal{L}_1^+(\pi, C)$ by

$$\mathcal{L}_1^+(\pi, C) := V_1[\pi \triangleright C] - V_1(\pi) .$$

(Notice that we have not yet considered posterior Bayes vulnerability for non-uniform prior distributions or probabilistic channels. But, as will be seen in Chap. 5, the above definitions of leakage are valid in general.)

In our dice channel, for example, we have seen that the prior Bayes vulnerability $V_1(\vartheta)$ is $1/36$ and the posterior Bayes vulnerability $V_1[\vartheta \triangleright C]$ is $11/36$. Hence the multiplicative Bayes leakage is $\mathcal{L}_1^\times(\vartheta, C) = (11/36)/(1/36) = 11$ and the additive Bayes leakage is $\mathcal{L}_1^+(\vartheta, C) = 11/36 - 1/36 = 5/18$. Those particular results are again a reflection of a general property, which follows immediately from Thm. 1.1:

Corollary 1.2 Let ϑ be a uniform prior distribution on an N -element set \mathcal{X} and let C be a deterministic channel with M possible output values. Then $\mathcal{L}_1^\times(\vartheta, C) = M$ and $\mathcal{L}_1^+(\vartheta, C) = (M-1)/N$. □

Those results for the Bayes leakage of a deterministic channel under a uniform prior are especially useful, as they say that to calculate leakage we just need to count the *number of possible output values*. As an example, consider a (deterministic) channel D that (like our dice channel C) takes as input the value (r, w) of X , but instead returns

the *product* of the two dice, so that $D(r, w) = r \cdot w$. If we wish to compute the Bayes leakage of D , then by Cor. 1.2 it suffices to count the number of possible output values. A bit of thought reveals that there are 18 such values (namely 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, 20, 24, 25, 30, and 36), and hence $\mathcal{L}_1^X(\vartheta, D) = 18$ and $\mathcal{L}_1^+(\vartheta, D) = 17/36$.

1.2 Looking ahead

The discussion of information leakage in the previous section introduces many important concepts of quantitative information flow, but it also can be seen to raise many questions.

First, Bayes vulnerability clearly measures a basic security concern, namely the adversary's probability of correctly guessing the secret in one try. But maybe the adversary can succeed by guessing the secret within *three tries*, or by guessing the secret only *approximately*. Or maybe she is *penalized* for making an incorrect guess. How can we address the great multiplicity of possible operational scenarios? That question is addressed in Part II of the book, where we present the family of *g-vulnerability* measures $V_g(\pi)$, parameterizing vulnerability with a *gain function* g that models the operational scenario.

Also, our dice examples C and D are *deterministic* and we studied them under the assumption that the prior distribution was *uniform*. Can we analyze channels whose behavior is *probabilistic*? Can we deal with a non-uniform prior distribution π (as would result from biased dice)? Those questions are addressed in Part III, where in Chap. 4 we show that the view of channels as mappings from prior distributions to hyper-distributions remains valid in that more general setting. Moreover, we show in Chap. 5 that we can generalize from posterior Bayes vulnerability and Bayes leakage to *posterior g-vulnerability* and *g-leakage*, for an arbitrary gain function g , giving us a rich family of leakage measures.

But this richness also raises questions. Recall that we found that the multiplicative Bayes leakage of dice channel C is 11, while that of dice channel D is 18. Does that mean that C is *more secure* than D in general, in that replacing D with C can only decrease leakage? What if the prior distribution π were not uniform? Or what if we measured leakage using some *g*-vulnerability other than Bayes vulnerability? Would the leakage ordering between C and D always stay the same? It turns out that the answer is *no*: even just changing to a non-uniform π can cause C to leak more than D rather than less.

As a result, we are led to question the *robustness* of our leakage analyses. The key issue, which we discuss in Chap. 6, is that the prior distribution π models the adversary's prior knowledge about the secret X , and the gain function g models the adversary's goals and capabilities. But can we really be sure about those things? As a result, we seek ways of achieving leakage analyses that are less sensitive to assumptions about π and g .

One approach, considered in Chap. 7, is to study notions of *capacity*, by which we mean worst-case leakage over all priors and/or over all gain functions. Such analyses allow us to obtain upper bounds on the leakage that a channel could ever cause.

Another approach, considered in Chap. 9, studies a refinement order (\sqsubseteq) on channels, allowing us to understand when one channel can *never* leak more than another, regardless of the prior or gain function. Refinement turns out to have a structural characterization, which facilitates its use in developing secure systems through what is called *stepwise* refinement.

Yet another aspect of robustness considers the fact that different secrets might turn out to be *correlated*. For instance, a person's height and weight might both be viewed as secrets, but it is clear that they are not *independent*, since a tall person is likely to weigh more than a short person. As a result, a system that leaks information about a person's height can be seen also to leak information about the person's weight, even if the system has *no access* to the weight. We call that *Dalenius leakage*, and we show in Chap. 10 that it can be bounded, even if arbitrary correlations are possible.

In Part IV, we move from our Part III study of systems as channels, which are mostly viewed as being monolithic, to a more detailed study of systems as *programs* written in a simple probabilistic imperative programming language extended with information flow. A crucial goal there is that the language's information-flow features behave just as channels do when studied on their own, i.e. that we achieve an *embedding* of existing techniques rather than having to start afresh. An interesting aspect of systems modeled as imperative programs is that they require us to consider the possibility that the value of a secret can *change over time*, due to the assignments to variables that imperative programming languages allow. As a result, we find that richer semantic structures are needed, based on *Hidden Markov Models*.

Finally, in Part V we present a number of chapters describing *applications* of quantitative-information-flow analysis to a number of interesting realistic scenarios, including anonymity protocols, timing attacks on cryptography, voting protocols, and differential privacy. We have tried to make these chapters somewhat self-contained, so that an interested reader can study them profitably without first mastering the theory as presented in Parts II, III, and IV. To facilitate that, these chapters try to give pointers to the particular parts of the theory on which they depend, allowing the reader to look back at those parts as necessary.

1.3 Exercises

Exercise 1.1 Recall dice channel C from §1.1, defined by $C(r, w) = r + w$. Now consider a channel E that instead outputs the *maximum* of the two dice, so that $E(r, w) = \max\{r, w\}$. Assuming a uniform prior distribution ϑ , find the additive and multiplicative Bayes leakage of E . What partition of \mathcal{X} does E give? \square

Exercise 1.2 Consider an election in which k voters choose between candidates A and B . Ballots are supposed to be secret, of course, so we can take the sequence of votes cast to be the secret input X . (For example, if $k = 3$ then the set \mathcal{X} of possible values for X is $\{AAA, AAB, ABA, ABB, BAA, BAB, BBA, BBB\}$.) Assuming a uniform prior ϑ , the prior Bayes vulnerability $V_1(\vartheta) = 2^{-k}$, since there are 2^k possible sequences of votes, each occurring with probability 2^{-k} .

When the election is tallied, the number of votes for each candidate is made public. (For example, when $k = 8$ we might get the vote sequence $AABABAAB$, whose tally is 5 votes for A and 3 votes for B .) Note that election tabulation can be seen as a *deterministic channel* T from X to Y , where Y is the tally of votes.

- (a) Given k , what is the multiplicative Bayes leakage $\mathcal{L}_1^\times(\vartheta, T)$ of the election tabulation channel?
- (b) Suppose we want the *posterior Bayes vulnerability* $V_1[\vartheta \triangleright T]$ to be at most $1/8$. Determine the minimum value of k that achieves that bound.

\square

1.4 Chapter notes

The side-channel attacks mentioned at the start of this chapter have turned out to be a real danger. Some important historical results include timing attacks on *RSA* and *DSS* [5] and remote timing attacks [3]; caching attacks on *AES* [4, 7, 2], on *RSA* [8], and on *DSA* [1]; and the (infamous) Dutch *RFID* public-transit–card attacks [9, 6].

Bibliography

- [1] Aciçmez, O., Brumley, B.B., Grabher, P.: New results on instruction cache attacks. In: S. Mangard, F.X. Standaert (eds.) Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2010), *Lecture Notes in Computer Science*, vol. 6225, pp. 110–124. Springer, Berlin (2010)
- [2] Bernstein, D.J.: Cache-timing attacks on AES (2005). <http://cr.yp.to/antiforgery/cachetiming-20050414.pdf>
- [3] Brumley, D., Boneh, D.: Remote timing attacks are practical. *Computer Networks* **48**(5), 701–716 (2005)
- [4] Grund, D.: Static Cache Analysis for Real-Time Systems: LRU, FIFO, PLRU. epubli GmbH (2012)
- [5] Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: N. Koblitz (ed.) Proceedings of Advances in Cryptology (CRYPTO 1996), *Lecture Notes in Computer Science*, vol. 1109, pp. 104–113. Springer, Berlin (1996)
- [6] Nohl, K., Evans, D., Starbug, Plötz, H.: Reverse-engineering a cryptographic RFID tag. In: Proceedings of the 17th USENIX Security Symposium, pp. 185–193. USENIX Association, Berkley (2008)
- [7] Osvik, D.A., Shamir, A., Tromer, E.: Cache attacks and countermeasures: The case of AES. In: D. Pointcheval (ed.) Topics in Cryptology - CT-RSA 2006, Proceedings of The Cryptographer’s Track at the RSA Conference, *Lecture Notes in Computer Science*, vol. 3860, pp. 1–20. Springer, Berlin (2006)
- [8] Percival, C.: Cache missing for fun and profit. In: Proceedings of BSDCan 2005 (2005)
- [9] Siekerman, P., van der Schee, M.: Security evaluation of the disposable OV-chipkaart v1.7 (2007)

Part II

Secrets and How to Measure Them



Chapter 2

Modeling secrets

We begin our detailed study of quantitative information flow by asking ourselves what a secret really is. There are many non-controversial examples, such as a user's password, social security number, or current location — but what does it mean for us to treat them as *secrets*?

2.1 Secrets and probability distributions

Our starting point will be to say that a secret is something for which the adversary knows the *probability distribution*, but not the value itself. Knowing the former, and by observing information flow out of some mechanism, she wants to improve her knowledge of the latter.

Definition 2.1

A *secret*, called X , has a set \mathcal{X} of possible values, which we call the *type* of X and which we assume to be *finite*; moreover we assume that the knowledge that some adversary has about X is given by a *probability distribution* π on \mathcal{X} that specifies the probability π_x of each possible value x of X . (Thus the secrecy of X is defined relative to a *particular* adversary, leaving open the possibility that a *different* adversary might have different knowledge of X , and so require a different distribution to reflect that.)¹

In general, a probability distribution δ on a finite set \mathcal{X} is a function from \mathcal{X} to the interval $[0, 1]$ (which we write $\delta: \mathcal{X} \rightarrow [0, 1]$) such that $\sum_{x \in \mathcal{X}} \delta_x = 1$. We write $\mathbb{D}\mathcal{X}$ for the set of all distributions on \mathcal{X} . When \mathcal{X} is small and ordered, we sometimes denote a distribution simply as a tuple of its probabilities. With $\mathcal{X} = \{x_1, x_2, x_3, x_4\}$, for example, we might write $\delta = (1/2, 1/6, 0, 1/3)$. The *support* of a distribution, denoted by $[\delta]$, is the set of values to which it gives nonzero probability. For example, the δ above has support $\{x_1, x_2, x_4\}$. A distribution δ in $\mathbb{D}\mathcal{X}$ is said to be *full support* if $[\delta] = \mathcal{X}$.

We refer to π as the *prior* distribution, because it represents the adversary's knowledge about the system *before* any output is observed. Later we will consider *posterior* distributions, which represent the adversary's knowledge *after* observing the output of a system. We now consider some examples of prior distributions.

¹ In §15.2 varying adversarial points of view are indicated explicitly.

If X is generated by a known probabilistic process, then π is clear. For instance, if X is generated by 3 flips of a fair coin, then X has 8 equally probable values, giving a *uniform* prior π^{coin} , as here:

HHH	HHT	HTH	HTT	THH	THT	TTH	TTT
$1/8$	$1/8$	$1/8$	$1/8$	$1/8$	$1/8$	$1/8$	$1/8$

If however X is generated as the sum of a roll of two fair dice, then we get a *non-uniform* prior π^{dice} , as here:

2	3	4	5	6	7	8	9	10	11	12
$1/36$	$2/36$	$3/36$	$4/36$	$5/36$	$6/36$	$5/36$	$4/36$	$3/36$	$2/36$	$1/36$

But the appropriate prior distribution for secrets in general can be more doubtful. If for instance X is the *mother's maiden name* of a certain user, then X is not generated by a clear random process. In that case, it seems more appropriate for the prior distribution to reflect the adversary's knowledge of the population that the user comes from. For instance, if the adversary knows only that the user is an American born in the second half of the twentieth century, then the prior distribution π^{mother} can be determined based on data from the 2000 United States census about the prevalence of different surnames. In this case the space \mathcal{X} of possible values is very large: indeed the census reports 151,671 different surnames occurring at least 100 times. If those values are ordered by decreasing probability, the distribution π^{mother} begins as follows.

Smith	Johnson	Williams	Brown	Jones	Miller	Davis	Garcia	...
0.00881	0.00688	0.00569	0.00512	0.00505	0.00418	0.00398	0.00318	...

As a final example, if X is a certain young person's *location on Saturday night*, then an appropriate prior distribution would be quite hard to determine, but it could in principle be based on the popularity of various bars, clubs, and other activities in the population the person comes from.

Having just illustrated the possible variety in *kinds* of secrets, we now turn our attention to the *quantification of secrecy*. Given a secret X with (prior) distribution π , it is natural to say that the “amount” of secrecy X has is determined by π . Intuitively, a uniform π should mean “more” secrecy, and a skewed π should mean “less” secrecy. In particular, if π is a *point distribution* on some x , written $[x]$ and meaning that $\pi_x = 1$, then there is no secrecy at all, since in that case the value of X is certainly x . But how, precisely, should secrecy be quantified?

Before considering answers to that question, we note first that there is a basic design choice that we have. We could measure the adversary's “uncertainty” about the secret (the lower the better for the adversary) or we could measure the “threat” to the secret (the higher the better for the adversary). While those two perspectives are completely complementary, we will find that sometimes one or the other is more convenient. For this reason, we will sometimes work with measures of *uncertainty* and other times with measures of *threat*.

2.2 Shannon entropy

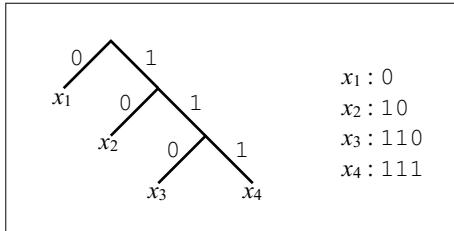
From the “uncertainty” perspective, it is tempting (at least for readers familiar with information theory) to measure the secrecy of an X having distribution π as the *Shannon entropy* $H(\pi)$, defined by

$$H(\pi) := - \sum_{x \in \mathcal{X}} \pi_x \log_2 \pi_x .$$

(Note that we use “:=” in definitions, reserving “=” for assertions of equality.)

An indication that Shannon entropy might be a reasonable choice can be seen by noting that if $|\mathcal{X}| = n$ then the possible values of $H(\pi)$ range from 0, in the case of a point distribution, up to $\log_2 n$, in the case of a uniform distribution. While those values are encouraging, they do not in themselves imply that different values of Shannon entropy are associated with definite security guarantees. Indeed the *operational significance* of particular measures of “uncertainty” or “threat” will be an important question throughout this book, and we now consider it for Shannon entropy.

In fact, *Shannon’s source-coding theorem* gives an interesting operational significance to Shannon entropy: it says that $H(\pi)$ is the average number of bits required to transmit X under an optimal encoding. We illustrate this result through an example. Suppose that $\mathcal{X} = \{x_1, x_2, x_3, x_4\}$ and $\pi = (1/2, 1/4, 1/8, 1/8)$. Then we can encode values of X using a variable-length *Huffman code*, which uses shorter codes for more likely values, and longer codes for less likely values:



Recall that the Huffman tree is formed by starting with a size-one weighted tree for each element of \mathcal{X} , where tree x has weight π_x , and then repeatedly joining two trees of minimal weight under a new root, giving the resulting tree a weight equal to the sum of the weights of the joined trees comprising it. (For instance, above we first join the trivial trees for x_3 and x_4 into a new tree with weight $1/8 + 1/8 = 1/4$. That tree is then joined with x_2 .)

Observe that under this code, the expected number of bits required to transmit X is

$$\frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \frac{1}{8} \cdot 3 = \frac{7}{4},$$

which is exactly the same² as $H(\pi)$ calculated

$$H(\pi) = \frac{1}{2} \log_2 2 + \frac{1}{4} \log_2 4 + \frac{1}{8} \log_2 8 + \frac{1}{8} \log_2 8 = \frac{7}{4}.$$

This result tells us that if $H(\pi)$ is large, then many bits are required on average to transmit the value of X . We might then be tempted to conclude that the adversary has a great deal of “uncertainty” about X , and that there is little “threat” to X .

More precisely, if we recall that Shannon entropy is 0 on a point distribution and $\log_2 n$ on a uniform distribution on n values, and that the adversary’s chance of correctly guessing the secret in one try is 1 in the former case, and $1/n$ in the latter case, then we might be led to conjecture that Shannon entropy has the following operational significance for confidentiality.

² In fact this perfect correspondence holds only for distributions (like π here) that are *dyadic*, meaning that all probabilities are negative powers of 2. For general distributions, one must actually transmit *blocks* resulting from sampling X repeatedly to get the claimed correspondence with $H(\pi)$.

Conjecture 2.2 If a secret X has distribution π , then an adversary's probability of guessing the value of X correctly in one try is at most $2^{-H(\pi)}$. \square

But it turns out that such a conjecture fails badly. To see this, consider a secret X with $\mathcal{X} = \{x_1, x_2, \dots, x_n\}$. With a uniform π we have $H(\pi) = \log_2 n$ — and so a smart adversary's probability of correctly guessing X in one try will indeed be $1/n = 2^{-\log_2 n} = 2^{-H(\pi)}$. But now suppose that we shift half of the mass from x_2 through x_n to x_1 . The result is a distribution π' where $\pi'_{x_1} = (n+1)/2n$ and $\pi'_{x_2} = \dots = \pi'_{x_n} = 1/2n$. This change increases the adversary's chance of correctly guessing X in one try to over $1/2$, but we find that $H(\pi')$ is still about half as big as $H(\pi)$: we calculate that

$$\begin{aligned} H(\pi') &= \frac{n+1}{2n} \log_2 \frac{2n}{n+1} + \frac{n-1}{2n} \log_2(2n) \\ &= \left(\frac{n+1}{2n} + \frac{n-1}{2n} \right) \log_2(2n) - \frac{n+1}{2n} \log_2(n+1) \\ &= \log_2 n - \frac{n+1}{2n} \log_2(n+1) + 1 \\ &\approx \frac{1}{2} \log_2 n + 1 \end{aligned}$$

for large n . Concretely, if X is a 128-bit secret, so that $n = 2^{128}$, then we find that $H(\pi') \approx 65$ — but the adversary's guessing probability is 2^{-1} , not the 2^{-65} that the conjecture would claim.

Thus we see that Shannon entropy can be quite misleading with respect to confidentiality; and that is why we will consider other measures of secrecy — ones that have better operational significance.

2.3 Bayes vulnerability

Motivated by the failed Conjecture 2.2 about Shannon entropy, we now refocus on a more basic security question: what is the adversary's probability of guessing X correctly in one try? (Note that we are now switching our perspective from the adversary's “uncertainty” about X to the “threat” to X .) It is immediate that this probability is simply the maximum probability that π assigns to any of the values in \mathcal{X} , since the best that the adversary can do is to guess any value with maximum probability. This leads to the definition of what we call *Bayes vulnerability*.

Definition 2.3 (Bayes vulnerability)

Given a distribution π on a finite set \mathcal{X} (viewed as the distribution of possible values of a secret X) the *Bayes vulnerability* of π is defined by

$$V_1(\pi) := \max_{x \in \mathcal{X}} \pi_x .$$

To see some examples of Bayes vulnerability, recall the *coin flip*, *dice*, and *mother's maiden name* examples above. We see that

- $V_1(\pi^{\text{coin}}) = 1/8$ (all guesses are equally good),
- $V_1(\pi^{\text{dice}}) = 6/36$ (7 is the best guess), and
- $V_1(\pi^{\text{mother}}) = 0.00881$ (Smith is the best guess).

A strength of Bayes vulnerability is its clear operational significance with respect to confidentiality. Indeed we can understand $V_1(\pi)$ as an adversary \mathcal{A} 's maximum probability of winning the following game, in which X is sampled according to π and \mathcal{A} , knowing only π , tries to guess it:

Prior Bayes-vulnerability Game

<ul style="list-style-type: none"> - <i>Defender</i> $X \in \pi$ 	<ul style="list-style-type: none"> - <i>Choose X according to prior.</i>
<ul style="list-style-type: none"> - <i>and then Adversary</i> 	
<ul style="list-style-type: none"> $W \in \mathcal{A}(\pi)$ if $W=X$ then “win” else “lose” 	<ul style="list-style-type: none"> - <i>Choose guess, and...</i> - ... win if correct.

In this pseudocode, the assignment operation $X \in \dots$ indicates a *probabilistic* choice from its right-hand side, whose type is distribution over \mathcal{X} .

There is at this point an important clarification to be made about the adversaries that we will consider in this book. In the game we have just shown, we wrote $\mathcal{A}(\pi)$ to indicate that the adversary has access to π in determining its action, to guess W . But in what form is π given to \mathcal{A} ? We would assume, at a minimum, that \mathcal{A} has the ability to compute π_x for any x in \mathcal{X} . That would indeed be the case if \mathcal{A} were given a table of probabilities, as in the examples above. But notice that to achieve the Bayes vulnerability in the game above, \mathcal{A} actually needs to *find* a value x such that π_x is maximal. It is of course possible for \mathcal{A} to do that by searching through the table of probabilities, but it might require (in effect) “finding a needle in a haystack”. In this book, we ignore such questions of efficiency, focusing only on the *information* available to the adversary.

Assumption

Throughout this book, we assume that adversaries are *information theoretic* — that is, without computational resource bounds.

2.4 A more general view

The definite operational significance of Bayes vulnerability is a strength, but it can also be seen as a *limitation*. There are of course many possible *operational scenarios*, by which we mean not only the “rules” about what the adversary is allowed to do but also the adversary’s particular goals. And, depending on the scenario, Bayes vulnerability might or might not do a good job of measuring the threat to the secret; in that sense it could be considered to be insufficiently general.

Our more general approach is therefore to imagine an adversary’s trying to decide among some possible *actions* to take in order to exploit her³ knowledge about X . For each possible action and each possible secret value, there will be some “reward” that she would achieve. In that framework, we can see that Bayes vulnerability corresponds to a scenario where she is *required* to make a guess (perhaps by entering it into a keypad) and is rewarded only if that *one guess* is exactly correct.

³ Recall that, as explained in the Preface, we follow the convention of referring to the *adversary* as “she” or “her” and to the *defender* as “he” or “him”.

But now consider a login scenario where a user is allowed *three tries* to enter the correct password before she is locked out. In that scenario, an adversary's possible actions would consist of *sets of three passwords* to be tried, and she would be rewarded if *any* of the three were correct. Here therefore it would be more accurate to model the threat using “three tries” Bayes vulnerability, which we can define as the sum of the three largest probabilities assigned by π . Denoting that by V_3 , we have

- $V_3(\pi^{\text{coin}}) = \pi_{HHH}^{\text{coin}} + \pi_{HHT}^{\text{coin}} + \pi_{HTH}^{\text{coin}} = 3/8$,
- $V_3(\pi^{\text{dice}}) = \pi_7^{\text{dice}} + \pi_6^{\text{dice}} + \pi_8^{\text{dice}} = 6/36 + 5/36 + 5/36 = 4/9$, and
- $V_3(\pi^{\text{mother}}) = \pi_{\text{Smith}}^{\text{mother}} + \pi_{\text{Johnson}}^{\text{mother}} + \pi_{\text{Williams}}^{\text{mother}} = 0.02138$.

The important insight here is that there are *many ways* to measure “uncertainty” or “threat” for a secret X , and they depend on the operational scenario — and so we might wonder whether there is a single framework that can encompass them all.

Indeed there is — as we will see in the next chapter when we develop the *g-vulnerability* family of vulnerability measures.

2.5 Exercises

Exercise 2.1 Recall that 3 flips of a fair coin results in a uniform prior π^{coin} on the 8 values $HHH, HHT, HTH, HTT, THH, THT, TTH$, and TTT . What is the prior π^{bent} that results from 3 flips of a *bent* coin that gives heads with probability $2/3$ and tails with probability $1/3$? Compare the Shannon entropies $H(\pi^{\text{coin}})$ and $H(\pi^{\text{bent}})$ and the Bayes vulnerabilities $V_1(\pi^{\text{coin}})$ and $V_1(\pi^{\text{bent}})$. \square

2.6 Chapter notes

The modeling of secrets using probability distributions touches on deep philosophical issues. Such distributions can be given a *frequentist* (or *generative*) interpretation, and also a *Bayesian* (or *degree of knowledge*) interpretation. Discussion of such issues can be found in Halpern [3]. In the context of quantitative information flow, Clarkson, Myers, and Schneider [2] consider adversaries with possibly incorrect *beliefs*, rather than *knowledge*, about secrets.

Census data about surname frequency in the United States can be found at <https://www.census.gov/topics/population/genealogy.html>.

Shannon entropy and the source-coding theorem appear in his 1948 paper [5].

The use of Bayes vulnerability $V_1(\pi)$, originally called just “vulnerability”, was advocated by Smith [6]. But the idea of measuring confidentiality based on an adversary’s maximum probability of guessing the secret correctly in one try has a long history, going back at least to Rényi’s *min-entropy* $H_\infty(\pi)$ [4], which is the negative logarithm of Bayes vulnerability: $H_\infty(\pi) = -\log_2 V_1(\pi)$. We will see in Chap. 5 that the corresponding posterior vulnerability is *Bayes risk*, whose use was advocated by Chatzikokolakis, Palamidessi, and Panangaden [1].

Bibliography

- [1] Chatzikokolakis, K., Palamidessi, C., Panangaden, P.: On the Bayes risk in information-hiding protocols. *Journal of Computer Security* **16**(5), 531–571 (2008)
- [2] Clarkson, M.R., Myers, A.C., Schneider, F.B.: Quantifying information flow with beliefs. *Journal of Computer Security* **17**(5), 655–701 (2009)
- [3] Halpern, J.Y.: *Reasoning about Uncertainty*. MIT Press (2003)
- [4] Rényi, A.: On measures of entropy and information. In: *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*, pp. 547–561. University of California Press, Berkeley (1961)
- [5] Shannon, C.E.: A mathematical theory of communication. *Bell System Technical Journal* **27**(3), 379–423, 623–656 (1948)
- [6] Smith, G.: On the foundations of quantitative information flow. In: L. de Alfaro (ed.) *Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS 2009)*, *Lecture Notes in Computer Science*, vol. 5504, pp. 288–302. Springer, Berlin (2009)



Chapter 3

On g -vulnerability

As we have just seen in Chap. 2, Bayes vulnerability measures one basic threat to a secret X , namely the risk that the adversary could correctly guess it in one try. But of course there are many other operational scenarios that we could be worried about. For instance, she might benefit from guessing only *part* of X , an *approximate* value of X , a *property* of X , or from guessing X correctly within a *fixed number of tries*. Also she might be *penalized* for making an incorrect guess. To accommodate this multiplicity of possible operational scenarios, in this chapter we develop a decision-theoretic framework that we call *g-vulnerability*.

3.1 Basic definitions

The perspective of *g-vulnerability* is that knowledge about a secret X is important only to the extent that it can be *exploited* by an adversary, enabling her to take some *action* that rewards her. For instance, knowing a user’s password enables an adversary to log in as that user; knowing a combination enables her to open a safe; and knowing a sentry’s location enables her to enter a facility without being detected. We therefore model her operational scenario by specifying a set \mathcal{W} of possible actions that she could take.

We sometimes think of those actions as being *guesses* that the adversary could make about the secret, in which case it is natural that $\mathcal{W}=\mathcal{X}$ — but there are other possibilities. For instance, an action might be to opt *not* to make a guess, which might be her choice if the situation were too risky. As a result, we allow \mathcal{W} to be any nonempty set of *actions*.

To complete the model of the operational scenario, we need a way of specifying the “reward” to the adversary of taking a certain action, given a certain value of the secret X . For this we use what we call a *gain function*.

Definition 3.1

Given a finite, nonempty set \mathcal{X} (of possible secret values) and a nonempty set \mathcal{W} (of possible actions), a *gain function* is a function $g: \mathcal{W} \times \mathcal{X} \rightarrow \mathbb{R}$.

The idea is that $g(w, x)$ specifies the gain that the adversary achieves by taking action w when the value of the secret is x . It is often the case that a gain function returns values in the interval $[0, 1]$, so that $g(w, x) = 0$ when w is a “worthless” action and $g(w, x) = 1$ when w is a “best” action. But in general we may want gain values

to represent the *economic value* to the adversary of different actions: in this case it makes sense to allow arbitrary gain values, both positive and negative.

As we have said, the actions \mathcal{W} are intended to correspond to the actual adversarial actions possible in some operational scenario (e.g. “Enter 12913 on the keypad.” or “Try to enter the fort through the south gate.”), but note that a gain function $g: \mathcal{W} \times \mathcal{X} \rightarrow \mathbb{R}$ abstracts from such details, as they are irrelevant in assessing the magnitude of the threat that they imply. Thus in Example 3.3 below we will consider a gain function with actions $\{w_1, w_2, w_3, w_4, w_5\}$ without any thought of what exactly those actions might be. Still, it is important to note that a particular gain function g is relevant to a particular operational scenario only if there really are actual actions corresponding to the elements of \mathcal{W} and whose effectiveness is correctly modeled by g .¹

In the examples that we present below, we see that usually it is enough to have a *finite* set \mathcal{W} of actions.² In such cases, we sometimes find it convenient to represent a gain function g as a matrix \mathbf{G} indexed by $\mathcal{W} \times \mathcal{X}$ so that $\mathbf{G}_{w,x} = g(w, x)$ and the rows of \mathbf{G} correspond to actions, the columns to secrets.

Now we are ready to define g -vulnerability. The key idea is that when X has distribution π , a smart adversary should choose an action w that maximizes her expected gain $\sum_{x \in \mathcal{X}} \pi_x g(w, x)$ with respect to π . Hence we have this definition:

Definition 3.2

The g -vulnerability of π is defined as

$$V_g(\pi) := \max_{w \in \mathcal{W}} \sum_{x \in \mathcal{X}} \pi_x g(w, x) .$$

If \mathcal{W} is infinite then the max should be replaced by sup.

Example 3.3 With $\mathcal{X} = \{x_1, x_2\}$ and $\mathcal{W} = \{w_1, w_2, w_3, w_4, w_5\}$, let gain function g have the (rather arbitrarily chosen) values shown in the following matrix:

\mathbf{G}	x_1	x_2
w_1	-1.0	1.0
w_2	0.0	0.5
w_3	0.4	0.1
w_4	0.8	-0.9
w_5	0.1	0.2

To compute the value of V_g on (say) $\pi = (0.3, 0.7)$, we must compute the expected gain for each possible action w in \mathcal{W} , given by the expression $\sum_{x \in \mathcal{X}} \pi_x g(w, x)$ for each one, to see which of them is best. The results are as follows.

$$\begin{aligned} \pi_{x_1} g(w_1, x_1) + \pi_{x_2} g(w_1, x_2) &= 0.3 \cdot (-1.0) + 0.7 \cdot 1.0 &= 0.40 \\ \pi_{x_1} g(w_2, x_1) + \pi_{x_2} g(w_2, x_2) &= 0.3 \cdot 0.0 + 0.7 \cdot 0.5 &= 0.35 \\ \pi_{x_1} g(w_3, x_1) + \pi_{x_2} g(w_3, x_2) &= 0.3 \cdot 0.4 + 0.7 \cdot 0.1 &= 0.19 \\ \pi_{x_1} g(w_4, x_1) + \pi_{x_2} g(w_4, x_2) &= 0.3 \cdot 0.8 + 0.7 \cdot (-0.9) &= -0.39 \\ \pi_{x_1} g(w_5, x_1) + \pi_{x_2} g(w_5, x_2) &= 0.3 \cdot 0.1 + 0.7 \cdot 0.2 &= 0.17 \end{aligned}$$

Thus we find that w_1 is the best action and $V_g(\pi) = 0.4$. \square

¹ We will, however, see further justifications for the relevance of gain (and loss-) functions, in §9.8 (and particularly the “compositional closure” of §9.8.3), and in Thm. 10.4 which shows their “Dalenius necessity”. There is a general discussion in §9.11.

² In the special case where there is only *one* action, the gain functions become equivalent in distinguishing power to the “post-expectations” in a purely probabilistic language like *pGCL*, which has *QIF* features. See the Chapter Notes.

Negative values for $g(w, x)$ are useful for expressing the fact that the adversary “loses” when she performs w and the secret is x ; the smaller $g(w, x)$ is (including “more negative”), the higher her incentive becomes for choosing some other action when the probability of that x increases. However, we find it convenient to require that the adversary’s *maximum expected gain* V_g should always be non-negative, so that 0 represents the best case of “no vulnerability”. Hence throughout this book we restrict our attention to the class of gain functions g such that V_g is always non-negative. We denote this class by $\mathbb{G}\mathcal{X}$, and discuss it (and other classes of gain functions) in §3.3.

Another useful representation of a gain function consists of viewing each action as a vector w in $\mathbb{R}^{|\mathcal{X}|}$ whose elements are the corresponding gains, that is $w_x = g(w, x)$. (In other words, action w is the row $\mathbf{G}_{w,-}$ in the matrix representation.) The expected gain of w under prior π is then given by the dot product $w \cdot \pi$, while g can be seen as a set of such action vectors. This “geometric” view is discussed in Chap. 12.

As its name suggests, V_g is a measure of the *vulnerability* of, or “threat” to, the secret. A complementary approach is to define a measure of the adversary’s *uncertainty* about the secret, using a *loss function* $\ell: \mathcal{W} \times \mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$ instead of a gain function, and assuming that the adversary wants to minimize her loss. We can then define ℓ -*uncertainty* U_ℓ ³ dually as follows.

Definition 3.4

The ℓ -uncertainty of π is defined as

$$U_\ell(\pi) := \min_{w \in \mathcal{W}} \sum_{x \in \mathcal{X}} \pi_x \ell(w, x) .$$

If \mathcal{W} is infinite then the min should be replaced by inf.

Note that, although gain functions are allowed to take negative values, loss functions need to be non-negative (since otherwise U_ℓ itself becomes negative). Note also that uncertainties can be converted to vulnerabilities (and vice versa) by complementing, setting $g = B - \ell$, where B is some upper bound of U_ℓ , we get $V_g = B - U_\ell$.

To avoid confusion between gains and losses, we will (mostly) limit our attention to g -vulnerability in the rest of this chapter.

3.1.1 Graphing g -vulnerability

To understand g -vulnerability better, it is helpful to graph the vulnerability V_g as a function of the prior π . Drawing such graphs is easiest in the case where the set \mathcal{X} of possible secret values has size 2, say $\mathcal{X} = \{x_1, x_2\}$. In that case a distribution π on \mathcal{X} is completely determined by the single number π_{x_1} as here:

$$\pi = (\pi_{x_1}, \pi_{x_2}) = (\pi_{x_1}, 1 - \pi_{x_1}) .$$

This lets us draw a two-dimensional graph of $V_g(\pi)$, where the x -axis represents π (specified by π_{x_1}) and the y -axis represents the corresponding g -vulnerability.

For instance, the graph of the V_g defined in Example 3.3 is shown in Fig. 3.1. It is interesting to note that the graph of V_g is *convex* (\smile) and *continuous*. And it turns out that those two properties are fundamental to g -vulnerability: they hold of V_g no matter what gain function g is used to define it. To understand why the graph has the shape that it does, it is instructive to consider the graphs of the expected gains provided by each of the five actions w_1, w_2, w_3, w_4 , and w_5 as shown in Fig. 3.2, where the lines are determined quite simply. For instance, the line corresponding to action

³ $U_\ell(\pi)$ is often called *Bayes Risk* in the area of decision theory [4, Section 8.2].

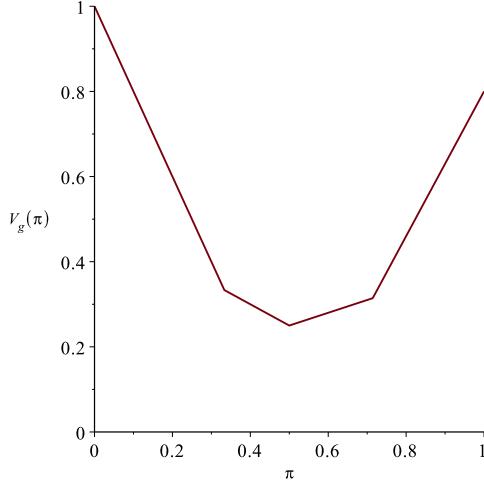


Figure 3.1 Graph of V_g from Example 3.3

w_4 connects the points $(0, -0.9)$ and $(1, 0.8)$. The reason is that the x -coordinate represents π_{x_1} , which means that if $x=0$ then π is the point distribution $(0, 1)$ and the expected gain from action w_4 is just $g(w_4, x_2) = -0.9$. Similarly, if $x=1$ then π is the point distribution $(1, 0)$ and the expected gain from action w_4 is just $g(w_4, x_1) = 0.8$. (In higher dimensions, i.e. larger state-spaces \mathcal{X} , the lines become hyperplanes — and they are determined by their passing through points on the verticals above the point distributions in a barycentric representation. In the case where \mathcal{X} has three elements, that is illustrated geometrically in Fig. 12.2 where lines passing through two points (as just above) become planes passing through three points.)

Turning now to the relationship between the two graphs, we notice that the graph of V_g in Fig. 3.1 lies along the *maximum* of all of the lines in Fig. 3.2. Moreover, V_g 's four line segments, from left to right, correspond to the four sets of distributions π for which actions w_1 , w_2 , w_3 , and w_4 are respectively optimal for the adversary. Also note that the three interior vertices represent the places where the adversary changes her mind about the best action. At the leftmost vertex, for instance, there is a *tie* between actions w_1 and w_2 , with w_1 being optimal to the left of the vertex and w_2 being optimal to the right of the vertex. Finally, notice that while action w_5 is sometimes better than each of the other four actions, it is never better than *all* of them, in that there is always some other action that gives a better expected gain: hence an optimal adversary will never choose it.

As we noted above, it is easiest to graph V_g in the case when \mathcal{X} has size 2. But V_g can also be graphed usefully when \mathcal{X} has size 3, using a *three-dimensional* graph. For then a prior $\pi = (\pi_{x_1}, \pi_{x_2}, \pi_{x_3})$ is completely determined by the pair (π_{x_1}, π_{x_2}) , which can be plotted on the xy -plane, and then the value of $V_g(\pi)$ can be plotted in the vertical z dimension. Fig. 3.3(b) below gives an example of such a graph. (It is also possible to plot three-dimensional gain functions in barycentric coordinates, as is done in Fig. 3.4 and later in Chap. 12; that gives a more symmetric presentation.) Similarly, when \mathcal{X} has size 4 we could plot prior π in three-dimensional space, but then the value of $V_g(\pi)$ would have to be shown in a fourth dimension, making visualization difficult.

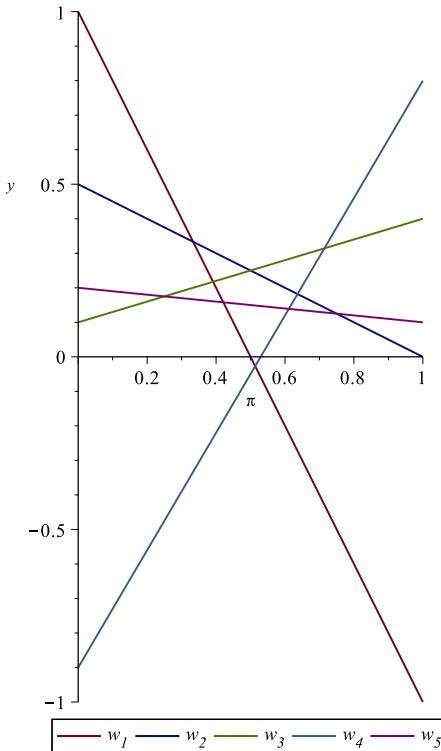


Figure 3.2 Graph of the expected gains from actions w_1, w_2, w_3, w_4 , and w_5 of Example 3.3

3.2 A catalog of gain functions

The gain function of Example 3.3 was chosen somewhat randomly, just to illustrate the basic definitions of gain functions and g -vulnerability. In this section, we present a number of gain functions chosen more deliberately, so that they correspond to realistic scenarios interesting in practice and illustrate how g -vulnerability lets us model the threat to a secret in operationally significant ways.

3.2.1 The identity gain function

One obvious (and often appropriate) gain function is the one that corresponds to a scenario where the adversary benefits only by guessing the value of the secret exactly, and is allowed only one try. Here the actions can simply be the values that can be guessed, and the gain function specifies that a correct guess is worth 1 and an incorrect guess is worth 0.

Definition 3.5 (Identity gain function) The *identity* gain function $g_{\text{id}}: \mathcal{X} \times \mathcal{X} \rightarrow \{0, 1\}$ is given by

$$g_{\text{id}}(w, x) := \begin{cases} 1, & \text{if } w = x, \\ 0, & \text{if } w \neq x \end{cases}.$$

□

Note that g_{id} takes $\mathcal{W} = \mathcal{X}$, meaning that the adversary is *required* to make a guess — she does not have the option of *not guessing*.⁴ In terms of representing a gain function as a matrix, g_{id} corresponds to the $\mathcal{X} \times \mathcal{X}$ -indexed identity matrix.

Now we can show that g -vulnerability is a generalization of Bayes vulnerability.

Theorem 3.6 Vulnerability under g_{id} coincides with Bayes vulnerability: we have

$$V_{g_{\text{id}}}(\pi) = V_1(\pi).$$

Proof. Since $\sum_x \pi_x g_{\text{id}}(w, x) = \pi_w$, we have $V_{g_{\text{id}}}(\pi) = \max_w \pi_w = V_1(\pi)$. □

A natural extension of g_{id} is to consider an adversary who still benefits only by guessing the secret in one try, but whose gain from a correct guess depends on the actual guessed secret, instead of its always being 1. A particularly interesting case is when the gain is defined as the reciprocal of the probability that some distribution σ (i.e. not necessarily the prior) assigns to that secret.

Definition 3.7 For $\sigma: \mathbb{D}\mathcal{X}$, the *distribution-reciprocal* gain function $g_{\sigma^{-1}}$ is given by $\mathcal{W} = [\sigma]$ and

$$g_{\sigma^{-1}}(w, x) := \begin{cases} 1/\sigma_x & \text{if } w = x \\ 0 & \text{otherwise} \end{cases}.$$

□

In matrix form, $g_{\sigma^{-1}}$ is a square (assuming a full-support σ) diagonal matrix, with the reciprocals of σ along its main diagonal. If σ is uniform, then $g_{\sigma^{-1}}$ is simply g_{id} scaled by $|\mathcal{X}|$. (Scaling is discussed in §3.4.1.)

Note that $V_{g_{\sigma^{-1}}}(\pi) = \max_x \pi_x / \sigma_x$. Since a guess with small σ_x is rewarded more by $g_{\sigma^{-1}}$, the best guess might not be the value with the maximum probability π_x — instead it is the one that maximizes the ratio π_x / σ_x . Note also that $V_{g_{\sigma^{-1}}}(\pi)$ is minimized when π is σ itself; that is, for any π we have $V_{g_{\sigma^{-1}}}(\pi) \geq V_{g_{\sigma^{-1}}}(\sigma) = 1$. That is because all that $g_{\sigma^{-1}}$ does on σ is to completely counterbalance all differences in probability: each guess is then equally good.

3.2.2 Gain functions induced from distance functions

Exploring other gain functions, we find that one quite natural kind of structure that \mathcal{X} can exhibit is a notion of *distance* between its secret values. That is, the set \mathcal{X} might come equipped with a *distance function* $d: (\mathcal{X} \times \mathcal{X}) \rightarrow \mathbb{R}_{\geq 0}$, possibly (but not necessarily) satisfying the properties of some variant of *metrics*. Then with an additional function $h: \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}$, of our choice, we can define a gain function that is h applied to that distance, i.e.

$$g_{d,h}(w, x) := h(d(w, x)).$$

(Note that here we are taking $\mathcal{W} = \mathcal{X}$.) In that case we say that $g_{d,h}$ is the gain function *induced* by d and h .

⁴ *Oui, mais il faut parler. Cela n'est pas volontaire, vous êtes embarqué. Lequel prendrez-vous donc?* (Blaise Pascal)

Given some distance r , a natural choice for $h(r)$ is $R - r$, where R is the maximum distance between any two secret values in \mathcal{X} ; others include $1/r$ and e^{-r} . Those are all decreasing functions, expressing the idea that the adversary gains more by selecting a w that is “closer” to the actual x . But increasing functions can also make sense: for example $h(r) = r$ expresses an adversary’s trying to *avoid* x (as for the complement of g_{id}).

Distances induce a large class of gain functions — and we note in particular that the identity gain function (i.e. Bayes vulnerability) is induced by the *discrete metric* (with $h(r) = 1 - r$), because it assigns distance 1 to any two distinct values. However, there are several reasons why it is useful to allow more generality.

For one thing, it might make sense to generalize to a distance on a set \mathcal{W} that is a *superset* of \mathcal{X} . To see why, we suppose that the space of secrets is the set of corner points of a unit square: thus $\mathcal{X} = \{(0,0), (0,1), (1,0), (1,1)\}$. Suppose further that we use the gain function $g_{d,h}(w,x) = \sqrt{2} - d(w,x)$, where $d(w,x) = \|w-x\|_2$ is the Euclidean distance, and $h(r) = \sqrt{2} - r$. For a uniform prior, it is easy to see that any of the four corner points are equally good guesses, giving

$$V_{g_{d,h}}(\vartheta) = \frac{1}{4} (\sqrt{2} + 2(\sqrt{2} - 1) + 0) \approx 0.5606 .$$

But the adversary could actually do better by guessing $(1/2, 1/2)$, a value that is *not* in \mathcal{X} , since that guess has distance $\sqrt{1/2}$ from each of the four corner points, giving $V_{g_{d,h}}(\vartheta) = \sqrt{1/2} \approx 0.7071$ — and that is larger than the previous vulnerability.

Moreover, *non-symmetric* distances can be sometimes appropriate. Suppose that the secret is the time (rounded to the nearest minute) that the last RER-B train will depart from Lozère back to Paris.⁵ The adversary (i.e. the weary traveler) wants to guess this time as accurately as possible, but note that guessing 23h44 when the actual time is 23h47 is completely different from guessing 23h47 when the actual time is 23h44. If we normalize so that a wait of an hour or more is considered intolerable, then we would want the distance function

$$d(w,x) := \begin{cases} \frac{x-w}{60} & \text{if } x-60 < w \leq x \\ 1 & \text{otherwise} \end{cases} ,$$

and that is not symmetric.

3.2.3 Binary gain functions

The family of gain functions that return either 0 or 1 (and no other values) are of particular interest, since we can characterize them concretely. For such a gain function, each action exactly corresponds to the *subset* of \mathcal{X} for which that action gives gain 1. (Moreover we can assume without loss of generality that no two actions correspond to the *same* subset of \mathcal{X} , since such actions might as well be merged into one.) Hence we can use the subsets *themselves* as the actions, leading to the following definition.

Definition 3.8 (Binary gain function) Given $\mathcal{W} \subseteq 2^{\mathcal{X}}$ with \mathcal{W} nonempty, the *binary* gain function $g_{\mathcal{W}}$ is given by

$$g_{\mathcal{W}}(W, x) := \begin{cases} 1, & \text{if } x \in W \\ 0, & \text{otherwise} \end{cases} .$$

□

And now we can identify a number of interesting gain functions by considering different choices of \mathcal{W} .

⁵ It appears that RATP uses sophisticated techniques to make this time as unpredictable as possible.

3.2.3.1 Two-block gain functions

If $\mathcal{W} = \{W, \mathcal{X} - W\}$ then we can see W as a *property* that the secret X might or might not satisfy, and $g_{\mathcal{W}}$ is the gain function corresponding to an adversary that merely wants to decide whether or not X satisfies that property.⁶

3.2.3.2 Partition gain functions

More generally, \mathcal{W} could be any *partition* of \mathcal{X} into one or more disjoint *blocks*, where the adversary wants to determine which block the secret belongs to. That is equivalent to saying that $\mathcal{W} = \mathcal{X}/\sim$, by which we mean the set of blocks induced by the quotient with respect to an equivalence relation (\sim) on \mathcal{X} .

There are two extremes. If (\sim) is the identity relation, then the elements of \mathcal{W} are all singletons, which means that g_{\sim} is essentially the same as g_{id} .⁷ And if (\sim) is the universal relation, then \mathcal{W} consists of a single block: thus $\mathcal{W} = \{\mathcal{X}\}$. And so $g_{\sim} = g_{\odot}$, the “happy” gain function having the property that $g_{\odot}(\mathcal{X}, x) = 1$ for every x .

3.2.3.3 The k -tries gain function

We can also express *k -tries Bayes vulnerability*, in which the adversary is allowed to make k guesses, rather than just 1. For if we define

$$\mathcal{W}_k := \{W \mid W \subseteq \mathcal{X} \wedge |W| \leq k\} ,$$

then $V_{g_{\mathcal{W}_k}}(\pi)$ is exactly the probability that the adversary could guess the value of X correctly within k tries. This gain function is particularly appropriate for a login program that allows the user only k tries before locking her out. Notice that $g_{\mathcal{W}_k}$ is not a partition gain function if $k > 1$, since in that case its blocks overlap.

By analogy with Bayes vulnerability’s being written V_1 , we write $V_{g_{\mathcal{W}_k}}$ as V_k so that for example the vulnerability with respect to 3 guesses would be V_3 .

3.2.3.4 General binary gain functions

In a general binary gain function, the actions \mathcal{W} can be any nonempty subset of $2^{\mathcal{X}}$. In that case, each element of \mathcal{W} can be understood as a property that X might satisfy, and $g_{\mathcal{W}}$ is the gain function of an adversary who wants to guess *any* of those properties that X satisfies.

One extreme is the case $\mathcal{W} = \{\emptyset\}$, which gives $g_{\mathcal{W}} = g_{\odot}$, the “sad” gain function such that $g_{\odot}(\emptyset, x) = 0$, for every x .

Given an arbitrary binary gain function g , we can define another binary gain function g^c by *complementation*: thus $g^c(w, x) = 1 - g(w, x)$. (Because g is binary, also g^c is binary and, in particular, cannot be negative.) It is interesting to notice that if $\mathcal{W} \subseteq 2^{\mathcal{X}}$, then $g_{\mathcal{W}}^c$ is essentially the same as $g_{\mathcal{W}'}$, where $\mathcal{W}' = \{W^c \mid W \in \mathcal{W}\}$. For example, for g_{id}^c we have that \mathcal{W}' is the set of complements of singletons. This means that g_{id}^c is the gain function of an adversary who just wants to guess some value *different* from the actual value of the secret; in the context of anonymity, this corresponds to wanting to guess an *innocent* user.

⁶ Such two-block gain functions are reminiscent of the cryptographic notion of *indistinguishability*, which demands that from a ciphertext an adversary should not be able to decide any *property* of the corresponding plaintext.

⁷ While the two \mathcal{W} ’s do not actually have the same set of actions, they nevertheless generate the same vulnerability function.

3.2.4 Gain functions for a password database

We now consider a more concrete operational scenario. Suppose that the secret is an array X containing 10-bit passwords for a set U of 1000 users, so that $X[u]$ is user u 's password. What gain functions might be suitable here?

One possibility is to use the identity gain function g_{id} , which as we have seen corresponds to Bayes vulnerability. But that specifies that the adversary gains only by correctly guessing the *entire* array X , which is not appropriate if we view X as consisting of 1000 secrets, each of which is individually valuable.

We can instead define a gain function g_{pw} that describes an adversary who wants to guess *some* user's password, with no preference as to whose it is. That suggests

$$\mathcal{W} := \{(u, x) \mid u \in U \wedge 0 \leq x < 1024\}$$

and

$$g_{\text{pw}}((u, x), X) := \begin{cases} 1 & \text{if } X[u] = x \\ 0 & \text{otherwise} \end{cases}.$$

To see the effect of g_{pw} , suppose that the prior distribution on X is uniform, which means that the 1000 passwords are independent and uniformly distributed over those 10 bits. In this case we see that the expected gain of every element (u, x) of \mathcal{W} is 2^{-10} , which means that $V_{g_{\text{pw}}}(\pi) = 2^{-10}$. In contrast, $V_{g_{\text{id}}}(\pi) = 2^{-10,000}$. Those values match our intuition that under g_{pw} the adversary just needs to guess any single 10-bit password, while under g_{id} she needs to guess 1000 such passwords.

It is interesting to consider a variant of g_{pw} that allows the adversary to guess just a password x , without specifying *whose* it is, and that gives a gain of 1 if x is correct for *any* of the users: that suggests

$$\mathcal{W}' := \{x \mid 0 \leq x < 1024\}$$

and

$$g_{\text{pw}'}(x, X) := \begin{cases} 1 & \text{if } X[u] = x \text{ for some } u \in U \\ 0 & \text{otherwise} \end{cases}.$$

Again using the uniform prior π , we now find that

$$V_{g_{\text{pw}'}}(\pi) = 1 - (1023/1024)^{1000} \approx 0.6236,$$

since that is the probability that any given x occurs somewhere in X . But $g_{\text{pw}'}$ does not seem a very reasonable gain function, since really a password is valuable only with respect to a particular user.

As yet another possibility, we can consider a gain function that specifies that some passwords are more valuable than others. In particular, suppose that one of the users is Bill Gates, whose password is vastly more valuable than all the other passwords. In that scenario, it would be appropriate to replace g_{pw} with a gain function like

$$g_{\text{Gates}}((u, x), X) := \begin{cases} 1 & \text{if } u = \text{Bill Gates and } x = X[u] \\ 0.01 & \text{if } u \neq \text{Bill Gates and } x = X[u] \\ 0 & \text{otherwise} \end{cases}.$$

With respect to a uniform prior π , we find that $V_{g_{\text{Gates}}}(\pi) = 2^{-10}$, since that is the expected gain for any guess $(\text{Bill Gates}, x)$ where $0 \leq x < 1024$. (Making a guess about any u other than Bill Gates has an expected gain only one-hundredth as large.)

An interesting aspect of $V_{g_{\text{Gates}}}$ is that it is not invariant with respect to *permutations* of the probabilities in π . Consider a distribution $\pi^{u,x}$ where some user u 's password is known to be x , and all others are uniform and independent:

$$\pi_X^{u,x} := \begin{cases} 2^{-9990} & \text{if } X[u] = x \\ 0 & \text{otherwise} \end{cases}.$$

Now notice that choosing different values of u and x simply permutes the probabilities assigned by $\pi^{u,x}$. This has no effect on the Shannon entropy $H(\pi^{u,x})$ or the Bayes vulnerability $V(\pi^{u,x})$, which are equal to 9990 bits or 2^{-9990} respectively, regardless of the values of u and x . But it has a big effect on $V_{g_{\text{Gates}}}$ since when $u = \text{Bill Gates}$, it is Bill Gates' password that is known. Hence we have $V_{g_{\text{Gates}}}(\pi^{\text{Bill Gates},x}) = 1$, while for $u \neq \text{Bill Gates}$ we have $V_{g_{\text{Gates}}}(\pi^{u,x}) = 0.01$.

Note more generally that also $V_{g_{\text{pw}}}$ is not invariant with respect to permutations of probabilities. Consider a distribution π in which 10 of the bits in X are known. The choice of which bits are known simply permutes the probabilities in π . But notice that if the 10 known bits all come from the *same* password, then $V_{g_{\text{pw}}}(\pi) = 1$. In contrast, if the 10 known bits come from 10 *different* passwords, then $V_{g_{\text{pw}}}(\pi) = 2^{-9}$, since the adversary's best action is to try to guess the unknown 9 bits from any one of those 10 passwords. This means that g_{pw} captures the *structure* of the password database, where certain sets of bits are worth more than others.

3.2.5 A gain function that penalizes wrong guesses

Now imagine a scenario where the adversary tries to input an access code X into a keypad outside a locked door. Inputting the correct code unlocks the door, but inputting the wrong code triggers a penalty (say, opening a trap door to a pit of tigers).

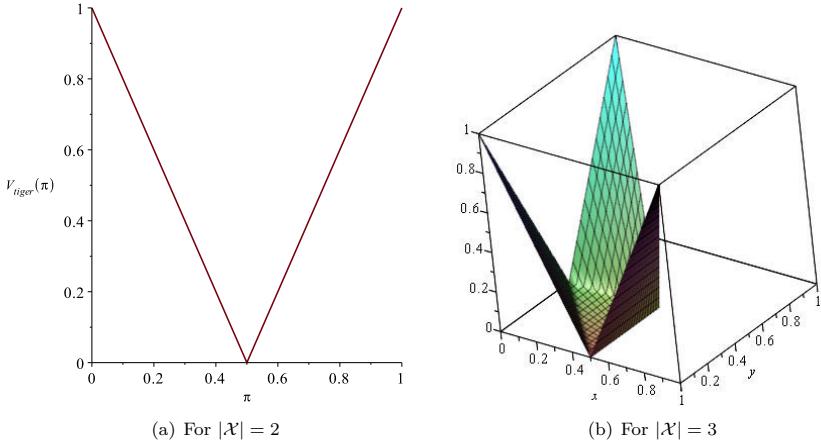
This scenario can be modeled with a gain function g_{tiger} using $\mathcal{W} = \mathcal{X} \cup \{\perp\}$, where the special action \perp is used to opt *not* to input a guess, and

$$g_{\text{tiger}}(w, x) := \begin{cases} 1 & \text{if } w = x \\ 0 & \text{if } w = \perp \\ -1 & \text{otherwise} \end{cases}.$$

The graphs of $V_{g_{\text{tiger}}}$ in the cases where \mathcal{X} has size 2 or 3 are shown in Figures 3.3(a) and 3.3(b), respectively. In the three-dimensional graph, we specify π on $\mathcal{X} = \{x_1, x_2, x_3\}$ by letting the x -axis represent π_{x_1} and the y -axis represent π_{x_2} , and of course $\pi_{x_3} = 1 - \pi_{x_1} - \pi_{x_2}$. (That implies that $0 \leq x, y$ and $x+y \leq 1$, which is why the plot is limited to just half of the unit square.)

Notice that the shape of the graph is again *convex* (\smile). Also notice that there is a triangular region in the center where $V_{g_{\text{tiger}}}(\pi) = 0$. That is because for the adversary to benefit from making a guess, at least one of the three probabilities must exceed $1/2$ — otherwise her expected gain from any guess is negative, making action \perp optimal.

An alternative way of presenting the three-dimensional graph is shown in Fig. 3.4. It uses what are called *barycentric coordinates*, in which the isosceles right triangle at the base of Fig. 3.3(b) becomes instead an equilateral triangle. (More discussion of barycentric coordinates is given in Chap. 12.)

Figure 3.3 Graph of $V_{g_{\text{tiger}}}$

3.2.6 A gain function for a medical diagnosis scenario

Related to g_{tiger} is a gain function for a scenario in which the adversary is trying to diagnose a certain disease. Here the set \mathcal{X} of possible values of the secret is $\{\text{disease}, \text{no disease}\}$. Of course she would like to guess correctly whether or not the disease is present, but she is more interested in what *action* to take. Thus she might have $\mathcal{W} = \{\text{treat}, \text{don't treat}\}$. Or she might have a *set* of possible treatments, some aggressive and some conservative.⁸ What is interesting here is that, unlike in the case of g_{tiger} , different errors would bring different penalties. Thus we might want a gain function something like

$g_{\text{diagnosis}}$	<i>disease</i>	<i>no disease</i>
<i>treat</i>	5	-2
<i>don't treat</i>	-5	2

to specify that leaving the disease untreated is worse than performing an unnecessary treatment (though of course the actual values would depend on the severity of the disease and the unpleasantness of the treatment).

3.2.7 A loss function that gives guessing entropy

To get a sense of the expressiveness of the g -vulnerability framework, we now consider how we can express *guessing entropy*. The guessing entropy of a secret X with distribution π is defined as the expected number of brute-force tries needed to guess the secret. The adversary's best strategy is to try possible values in non-increasing order of probability. If x_n is an indexing of \mathcal{X} in such an order, and $|\mathcal{X}|=N$, then the guessing entropy is defined by

$$G(\pi) := \sum_{n=1}^N n\pi_{x_n} .$$

⁸ Notice that in such scenarios, we are actually inclined to view the “adversary” as benevolent.

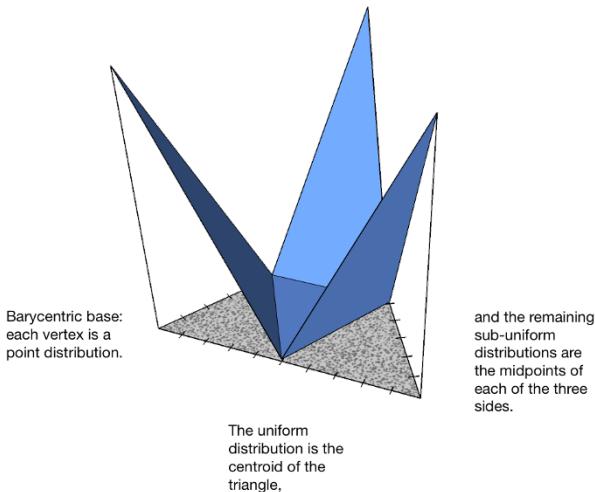


Figure 3.4 Barycentric graph of $V_{g_{\text{tiger}}}$

Because guessing entropy is an *uncertainty* measure, rather than a vulnerability measure, we wish to express it as an ℓ -uncertainty, for some loss function ℓ .

To do this, we let the set of actions \mathcal{W} be the set of all *permutations* f of \mathcal{X} , and define loss function ℓ_G by

$$\ell_G(f, x) := n ,$$

where n is the unique *index* of x within permutation f , assumed to range from 1 to N . (Hence the loss values for ℓ_G range from 1 to N .) Now observe that the expected loss from permutation f is

$$\sum_{n=1}^N \pi_{f_n} \cdot n ,$$

which is minimized if f orders \mathcal{X} in non-increasing probability order — in which case the expected loss is $G(\pi)$. Hence we have

$$U_{\ell_G}(\pi) = G(\pi) .$$

While loss function ℓ_G indeed gives guessing entropy, we might wonder (as discussed in §3.1) what operational scenario corresponds to ℓ_G and what action corresponds to a permutation f . Here it is easy to see that the scenario is one where the adversary must make brute-force guesses of the form “Is the secret equal to x_n ?” and the loss is the number of guesses needed to find the correct value; moreover the actual action corresponding to a permutation f is the “strategy” of guessing the values of \mathcal{X} in the particular *order* specified by f .

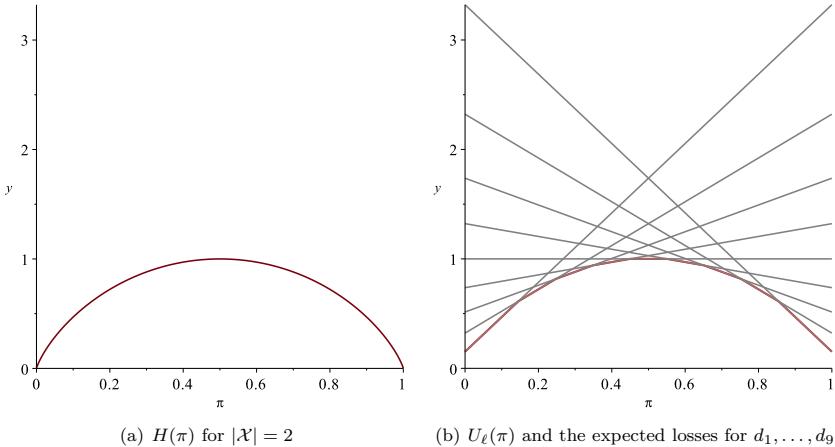


Figure 3.5 Shannon entropy

3.2.8 A loss function that gives Shannon entropy

We now show that we can also express *Shannon entropy*, defined by

$$H(\pi) := - \sum_{x \in \mathcal{X}} \pi_x \log_2 \pi_x ,$$

with a loss function: like guessing entropy, it is an *uncertainty* measure — and that is why we express it as an ℓ -uncertainty for some loss function ℓ . Unlike in all our previous examples, here it is necessary for us to use an *infinite* set \mathcal{W} of actions.

To get intuition about the construction, let us start with the case when \mathcal{X} has size 2. Figure 3.5(a) shows the plot of Shannon entropy in that case.⁹ Notice that the graph is *concave* (\curvearrowright) rather than *convex* (\curvearrowleft).

Now consider the following loss function ℓ :

ℓ	x_1	x_2
d_1	$-\log_2 0.1$	$-\log_2 0.9$
d_2	$-\log_2 0.2$	$-\log_2 0.8$
d_3	$-\log_2 0.3$	$-\log_2 0.7$
d_4	$-\log_2 0.4$	$-\log_2 0.6$
d_5	$-\log_2 0.5$	$-\log_2 0.5$
d_6	$-\log_2 0.6$	$-\log_2 0.4$
d_7	$-\log_2 0.7$	$-\log_2 0.3$
d_8	$-\log_2 0.8$	$-\log_2 0.2$
d_9	$-\log_2 0.9$	$-\log_2 0.1$

Figure 3.5(b) shows the graph of $U_\ell(\pi)$, along with the graphs of the expected losses for each of the actions d_1, d_2, \dots, d_9 . Note that $U_\ell(\pi)$ is called the *envelope* of the expected loss graphs.

As can be seen, the graph of U_ℓ closely approximates that of Shannon entropy, but lies slightly above it.

⁹ For comparison, Fig. 12.2 in Chap. 12 shows the three-dimensional version.

3 On g -vulnerability

To achieve Shannon entropy exactly, we simply *extend* the loss function shown in the table above by including a row for *every* probability distribution ω on \mathcal{X} — this of course makes the set \mathcal{W} uncountably infinite. In particular we include an action d_0 with loss values $(-\log_2 0, -\log_2 1) = (\infty, 0)$ and an action d_{10} with loss values $(-\log_2 1, -\log_2 0) = (0, \infty)$. In general, we define our loss function ℓ_H for Shannon entropy by

$$\ell_H(\omega, x) := -\log_2 \omega_x .$$

Since ω_x is in $[0, 1]$, this loss function has range $[0, \infty]$. The expected loss with action ω is then

$$\sum_{x \in \mathcal{X}} \pi_x (-\log_2 \omega_x) ,$$

which, it turns out, is minimized when $\omega = \pi$, thanks to Gibbs' inequality.¹⁰ Hence we have

$$U_{\ell_H}(\pi) = \inf_{\omega \in \mathcal{W}} \sum_{x \in \mathcal{X}} \pi_x (-\log_2 \omega_x) = -\sum_{x \in \mathcal{X}} \pi_x \log_2 \pi_x = H(\pi) .$$

Recall, however, that previously we defined loss functions to have type $\mathcal{W} \times \mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$, which actually *disallows* our constructed ℓ_H , since it sometimes gives loss values of ∞ . Of course one option is to relax the definition of loss functions so that infinite loss values *are* allowed. But it turns out that we can modify our ℓ_H to avoid this issue. We can simply restrict the set \mathcal{W} to include only *full-support* distributions ω , for then $\ell_H(\omega, x) = -\log_2 \omega_x$ is never infinite. But now what happens to $U_{\ell_H}(\pi)$ when π is not full support? It turns out that $U_{\ell_H}(\pi)$ has the same value as before, but that value is now achieved as a proper *infimum*.

Consider the case where $\pi = (0, 1)$. Previously the adversary would have wanted to choose action $\omega = (0, 1)$, giving expected loss

$$\pi_{x_1} (-\log_2 \omega_{x_1}) + \pi_{x_2} (-\log_2 \omega_{x_2}) = 0 \cdot \infty + 1 \cdot 0 = 0 ,$$

but now that action is unavailable to her. She can, however, choose actions that are very close to $(0, 1)$. Action $(0.01, 0.99)$, for instance, gives expected loss

$$0 \cdot (-\log_2 0.01) + 1 \cdot (-\log_2 0.99) \approx 0 \cdot 6.6439 + 1 \cdot 0.0145 = 0.0145$$

and action $(0.0001, 0.9999)$ gives expected loss

$$0 \cdot (-\log_2 0.0001) + 1 \cdot (-\log_2 0.9999) \approx 0 \cdot 13.2877 + 1 \cdot 0.0001 = 0.0001 ,$$

which makes it clear that the infimum over all full-support actions is 0, as desired.¹¹

Turning finally to the question of operational significance, we again might wonder (as we did in the case of guessing entropy) what operational scenario corresponds to ℓ_H and what actual action corresponds to a distribution ω . Here we can get an answer by reinterpreting *Shannon's source-coding theorem*, discussed in §2.2. The operational scenario allows the adversary to ask arbitrary *yes/no* questions about the secret X , and the loss is the number of questions required to learn the value of X . Now the meaning of the action ω is to construct a *Huffman tree* corresponding to ω and to determine the value of X by starting at the root and repeatedly asking whether X is

¹⁰ Gibbs' inequality states that, for any distributions π and ω , $-\sum_x \pi_x \log_2 \pi_x \leq -\sum_x \pi_x \log_2 \omega_x$ with equality iff $\pi = \omega$. This is equivalent to the fact that *Kullback-Leibler distance* $D_{KL}(\pi || \omega) = \sum_x \pi_x \log_2 \frac{\pi_x}{\omega_x}$ is non-negative and is equal to 0 iff $\pi = \omega$. See [3, Theorem 2.6.3].

¹¹ Relatedly, we can also use continuity to restrict \mathcal{W} to *rational-valued* distributions ω , thereby making \mathcal{W} countably infinite.

among the values in the left subtree. In the case of the optimal action where $\omega = \pi$, and in the special case where π is dyadic (meaning that all probabilities are negative powers of 2), then the expected loss is exactly the Shannon entropy $H(\pi)$.

But that nice correspondence does *not* hold in the case where π isn't dyadic. In particular consider the case when $\mathcal{X} = \{x_1, x_2\}$ and π is a skewed distribution like $(0.999, 0.001)$. Here $H(\pi)$ is quite small (approximately 0.0114), but of course the adversary needs to ask one question in order to determine the value of X . In this case, the exact correspondence with Shannon entropy can be achieved only in a (rather contrived) scenario where a *sequence* of values of X are independently sampled according to π , and she tries to learn the value of the entire sequence via yes/no questions. In this case, by making the sequence long enough, we can make the *expected number of questions per value* arbitrarily close to $H(\pi)$.

We conclude this section by thinking intuitively about the construction illustrated in Fig. 3.5(b) (and, for three dimensions, in Fig. 12.2). As can be seen, each action w in \mathcal{W} corresponds to one of the *tangents* of the graph of Shannon entropy. Note moreover that each such w (except when the tangent is *vertical*) is determined by two *finite, non-negative* loss values. So it is plausible that (dealing somehow with vertical tangents) we can represent *any* non-negative, concave, continuous function U in $\mathbb{D}\mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$ as an ℓ -uncertainty, for some non-negative loss function ℓ . And plausibly the same sort of result should hold for vulnerabilities and gain functions as well. Indeed such results do hold, and we investigate them rigorously in §11.2.1.

3.3 Classes of gain functions

As just seen in §3.2, we can use gain functions of type $g: \mathcal{W} \times \mathcal{X} \rightarrow \mathbb{R}$ to define a rich variety of operationally significant g -vulnerabilities. In this section, we turn our attention from *particular* gain functions to *classes* of gain functions with properties that we will find useful subsequently. There are four classes that we consider here: they are $\mathbb{G}\mathcal{X}$, $\mathbb{G}^{\text{fin}}\mathcal{X}$, $\mathbb{G}^+\mathcal{X}$, and $\mathbb{G}^\dagger\mathcal{X}$.

3.3.1 Finite-valued, non-negative vulnerabilities: the class $\mathbb{G}\mathcal{X}$

The most general class that we consider is $\mathbb{G}\mathcal{X}$, which is the set of all gain functions g such that the induced V_g gives *finite* and *non-negative* values.

Definition 3.9 (Gain-function class) The class $\mathbb{G}\mathcal{X}$ consists of all gain functions g such that V_g is in $\mathbb{D}\mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$. □

The finiteness condition simply requires that the gain function g be *bounded from above*. For if g 's gain values never exceed some bound b , then V_g cannot exceed b either. And if g can give arbitrarily large gain values (which is possible only if \mathcal{W} is infinite), then (since \mathcal{X} is finite) there must be an x in \mathcal{X} such that $g(-, x)$ can give arbitrarily large gain values — and that would mean that on the point distribution $[x]$ we would have $V_g([x]) = \infty$.¹²

The non-negativity condition is more subtle. It is of course sufficient for g itself to be non-negative,¹³ but this is not *necessary*, as can be seen by Example 3.3, where g is

¹² To see a non-example with $\mathcal{X} = \{x_1, x_2\}$, let \mathcal{W} be the set of all integers and let g be given by $g(w, x_1) = w$ and $g(w, x_2) = -w$. Here g is not bounded from above, and we find that V_g is 0 on the uniform prior (for which the gain on one secret is exactly counterbalanced by the loss on the other), and ∞ for all other priors. Hence V_g fails finiteness, and moreover it is *discontinuous*.

¹³ It is in fact sufficient for $g(w, -)$ to be non-negative for a single action w : for instance, an action \perp giving 0 gain for any secret value, as in the case of g_{tiger} from §3.2.5.

sometimes negative but (as seen in Fig. 3.1) the induced vulnerability V_g is everywhere positive.

The class $\mathbb{G}\mathcal{X}$ is quite expressive; indeed it can be verified that *all* of the gain functions considered in §3.2 belong to it.

Moreover, we can precisely characterize the vulnerability functions V_g induced by gain functions g in $\mathbb{G}\mathcal{X}$: as will be shown in §11.2.1, they are exactly the functions of type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$ that are *continuous* and *convex*.¹⁴

Assumption

We restrict our attention exclusively to gain functions in $\mathbb{G}\mathcal{X}$ throughout this book (although we often restrict further to subsets of it).

We conclude this subsection with a brief discussion of the analogous class of *loss functions*.

Definition 3.10 (Loss-function class) The class $\mathbb{L}\mathcal{X}$ consists of all loss functions ℓ such that U_ℓ is in $\mathbb{D}\mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$. \square

Somewhat surprisingly, although a V_g can be seen as a U_ℓ “turned upside-down” and vice versa, there are important asymmetries between the two cases.¹⁵ In particular, it is easy to see that $\mathbb{L}\mathcal{X}$ is simply the set of all *non-negative loss functions*. Since U_ℓ involves *minimization*, if for some w and x we have $\ell(w, x) < 0$, then necessarily $U_\ell[x] < 0$ as well. Moreover, an unbounded U_ℓ would have to produce $-\infty$ (not $+\infty$), but (as we have required ℓ to be non-negative) that is impossible.

3.3.2 Finitely many actions: $\mathbb{G}^{\text{fin}}\mathcal{X}$

To simplify the presentation, in many parts of this book we restrict to the class $\mathbb{G}^{\text{fin}}\mathcal{X} \subset \mathbb{G}\mathcal{X}$ of gain functions having a *finite* set of actions \mathcal{W} . All of the gain- and loss functions discussed in §3.2, with the exception of the one for Shannon entropy, have finitely many actions. Note that any such gain function g is automatically bounded, making V_g finite-valued; but we still require V_g to be non-negative (which is not implied by finiteness).

The use of $\mathbb{G}^{\text{fin}}\mathcal{X}$ is implied whenever a gain function is expressed as a matrix G , or whenever V_g is expressed as a max instead of a sup. Note that this restriction is merely for presentation purposes; unless otherwise stated, all results about $\mathbb{G}^{\text{fin}}\mathcal{X}$ should be assumed to extend to $\mathbb{G}\mathcal{X}$.

3.3.3 Non-negative gain functions: $\mathbb{G}^+\mathcal{X}$

The next restriction that we consider here is to require the gain function g itself to be non-negative.

Definition 3.11 The class $\mathbb{G}^+\mathcal{X}$ consists of all non-negative gain functions g in $\mathbb{G}\mathcal{X}$. \square

This restriction ensures that V_g is also non-negative, but we still need g to be bounded in order for V_g to be finite-valued.

¹⁴ The continuity is with respect to the “usual” metric on discrete distributions, equivalently Euclidean, Manhattan, or Kantorovich (the last over the discrete metric on \mathcal{X}).

¹⁵ The asymmetry –where one might have expected a full duality– is because we did not turn the “non-negative” into “non-positive”.

3.3.4 One-bounded gain functions: $\mathbb{G}^\dagger \mathcal{X}$

Finally we consider gain functions g that are bounded from above by 1.

Definition 3.12 The class $\mathbb{G}^\dagger \mathcal{X}$ consists of all gain functions $g: \mathbb{G}\mathcal{X}$ such that $g \leq 1$. \square

Note that g need not be bounded from below (except that, as required by $\mathbb{G}\mathcal{X}$, the induced V_g must be non-negative).

Notice also that if g is one-bounded then V_g is also one-bounded. And the converse holds as well: if $g(w, x) > 1$ for some w and x , then $V_g([x]) > 1$. Hence $\mathbb{G}^\dagger \mathcal{X}$ can equivalently be defined as the set of all gain functions g such that the induced vulnerability V_g is always between 0 and 1.

3.4 Mathematical properties

In this section we explore some general mathematical properties of g -vulnerability.

First we show that every V_g is a *convex function* from distributions to reals. A vector $\sum_n a_n x^n$, where x^1, \dots, x^N are vectors and a_1, \dots, a_N are non-negative reals adding up to 1, is called a *convex combination* (and the a_n 's are called convex coefficients). A set is convex iff it contains all convex combinations of its elements. It is easy to see that $\mathbb{D}\mathcal{X}$ is convex. A function f , defined on a convex set, is called convex iff $f(\sum_n a_n x^n) \leq \sum_n a_n f(x^n)$, and *linear* iff $f(\sum_n a_n x^n) = \sum_n a_n f(x^n)$.¹⁶ We have the following theorem.

Theorem 3.13 (Convexity of V_g) For any $g: \mathbb{G}\mathcal{X}$, the induced $V_g(\pi)$ is a convex function of π .

Proof. Note that V_g can be expressed as the sup of the convex (and in fact linear) functions $\pi \mapsto \sum_x \pi_x g(w, x)$; and it is well known that the sup of convex functions is convex. To show the theorem directly in the case $g: \mathbb{G}^{\text{fin}}\mathcal{X}$, we have

$$\begin{aligned} & V_g\left(\sum_i a_i \pi^i\right) \\ &= \max_w \sum_x \left(\sum_i a_i \pi^i\right)_x g(w, x) \\ &= \max_w \sum_x \sum_i a_i \pi_x^i g(w, x) \\ &\leq \sum_i a_i \max_w \sum_x \pi_x^i g(w, x) \\ &= \sum_i a_i V_g(\pi^i). \end{aligned}$$

\square

The intuition behind the convexity of V_g is that in $V_g(\sum_i a_i \pi^i)$ the adversary is forced to select a *single* action for the “composite” distribution $\sum_i a_i \pi^i$, while in $\sum_i a_i V_g(\pi^i)$ the adversary is able to select a *different*, “custom” action for each π^i , potentially enabling a better expected gain. This is exactly the source of the crucial “ \leq ” step in the above proof.¹⁷

Note that, for a finite \mathcal{W} , vulnerability V_g can be immediately shown to be *continuous* as the max of finitely many continuous functions. The (infinitary) sup of continuous

¹⁶ In vector spaces, f is called linear if $f(\sum_n a_n x^n) = \sum_n a_n f(x^n)$ for arbitrary real a_n (hence $f(0)$ must be 0), and *affine* if this holds for real $a_{1..N}$ summing up to 1. In our case, $\mathbb{D}\mathcal{X}$ is not a vector space (only a convex subset of $\mathbb{R}^{|\mathcal{X}|}$) and $\sum_n a_n x^n$ only makes sense if $a_{1..N}$ are non-negative and sum to 1. Still we use the term “linear”.

¹⁷ In elementary terms, it’s the same reason that $(a \max b) + (c \max d) \geq (a+b) \max (c+d)$. On the left the max is bespoke but, on the right, it’s “one size fits all”.

functions, however, is not necessarily continuous, so such an immediate proof fails. Still, in §11.2.1 it is shown that V_g is in fact continuous for all g in $\mathbb{G}\mathcal{X}$.

A similar proof shows that ℓ -uncertainties are concave.

3.4.1 Gain function algebra

Given a gain function $g: \mathcal{W} \times \mathcal{X} \rightarrow \mathbb{R}$, the *particular values* returned by g are not very important; what really matters is how the various gain values compare with one another. In consequence, *modifying* the gain values in certain ways leads to predictable modifications of g -vulnerabilities.

The first modification we consider is *scaling*, where we multiply all gain values by a *non-negative* constant k . That is, we define a *scaled* gain function $g_{\times k}$ by

$$g_{\times k}(w, x) := k \times g(w, x) .$$

The second modification we consider is *shifting*, where we add a value κ_x , possibly depending on x , to all gain values. That is, we define a *shifted* gain function $g_{+ \kappa}$ by

$$g_{+ \kappa}(w, x) := g(w, x) + \kappa_x .$$

The choice of the vector κ gives us flexibility in the way the shifting is performed: for instance, we could shift all gain values by the same amount k by choosing $\kappa = k\mathbf{1}$, or shift the gain values of a single secret by choosing $\kappa = k[x]$.¹⁸

Viewing g as a set of gain vectors w , we see that scaling and shifting are simply the scalar multiplication kw and vector addition $w + \kappa$ of all gain vectors respectively. Similarly, when g is represented as a matrix G , scaling consists of multiplying G by k , while shifting consists of adding κ to all rows of G .

In the following theorem, we show how the vulnerabilities of a modified gain function can be derived from the vulnerabilities of the original gain function.

Theorem 3.14 For any $g: \mathbb{G}\mathcal{X}$, prior π , $k \geq 0$ and $\kappa \in \mathbb{R}^{|\mathcal{X}|}$ the following hold:

$$\begin{aligned} V_{g_{\times k}}(\pi) &= kV_g(\pi) , \\ V_{g_{+ \kappa}}(\pi) &= V_g(\pi) + \kappa \cdot \pi . \end{aligned}$$

Proof. Using the representation of g as a set of action vectors, we reason as follows.

$$\begin{aligned} & V_{g_{\times k}}(\pi) \\ = & \sup_{w \in g_{\times k}} w \cdot \pi && \text{“Def. 3.2, for action vectors”} \\ = & \sup_{w \in g} kw \cdot \pi && \text{“}g_{\times k} = \{kw \mid w \in g\}\text{”} \\ = & \sup_{w \in g} k(w \cdot \pi) \\ = & k \sup_{w \in g} w \cdot \pi && \text{“}\sup_i kx_i = k \sup_i x_i \text{ for } k \geq 0\text{”} \\ = & kV_g(\pi) \end{aligned}$$

$$\begin{aligned} \text{and } & V_{g_{+ \kappa}}(\pi) \\ = & \sup_{w \in g_{+ \kappa}} w \cdot \pi && \text{“Def. 3.2”} \\ = & \sup_{w \in g} (w + \kappa) \cdot \pi && \text{“}g_{+ \kappa} = \{w + \kappa \mid w \in g\}\text{”} \\ = & \sup_{w \in g} (w \cdot \pi + \kappa \cdot \pi) \\ = & (\sup_{w \in g} w \cdot \pi) + \kappa \cdot \pi && \text{“}\sup_i (x_i + c) = (\sup_i x_i) + c\text{”} \\ = & V_g(\pi) + \kappa \cdot \pi . \end{aligned}$$

□

¹⁸ Note that the point prior $[x]$ is a vector with 1 at x 's position and 0 elsewhere.

An issue that must be considered, however, is the effect of scaling and shifting on membership in $\mathbb{G}\mathcal{X}$. For any g in $\mathbb{G}\mathcal{X}$, it is easy to see that $g_{\times k}$ is also in $\mathbb{G}\mathcal{X}$ for any $k \geq 0$. But $g_{+\kappa}$ might *not* be in $\mathbb{G}\mathcal{X}$, since negative entries of κ could make $V_{g+\kappa}$ negative on some priors.

3.5 On “absolute” versus “relative” security

As we have seen, gain functions and g -vulnerability enable us to measure security in a rich variety of ways. But it is often hard to know precisely which gain function is most appropriate. An important insight, however, is that often we don’t care about the precise value of $V_g(\pi)$ (“absolute” security), but only about how the expected gains of the various actions in \mathcal{W} *compare* with each other (“relative” security).

A striking feature of the proof of Thm. 3.14, for instance, is that the modified gain functions $g_{\times k}$ and $g_{+\kappa}$ always have the *same* optimal actions as does the original gain function g . Thus the modifications affect the precise vulnerability values, but they do *not* affect what action the adversary should take.

Consider the case of a neighborhood under threat of burglary. The secret X here is the contents of the various houses (e.g. jewelry, cash, electronics) and the security measures protecting them (e.g. locks, safes, alarms, surveillance cameras), along with time-dependent information (who is at home, awake, or sleeping, in the various houses). The prior distribution π models the adversary’s (partial) knowledge of X , and the gain function g models the actions \mathcal{W} that the adversary could take (e.g. break the glass door at the back of a certain house at a certain time) and the gain values $g(w, x)$ model their result (e.g. successful theft of a diamond necklace worth \$10,000 –good for her– or detection and arrest by police — bad). Clearly here the complete scenario (X , π , and g) could be very complex, and not knowable precisely. But if we are a homeowner in the neighborhood, our own goal is considerably more modest. When we think about what security devices we might buy for our house, our goal is not to make our house *absolutely secure* (which is in fact impossible), but only to make our house less tempting to the burglar than our neighbors’ houses are — so that the burglar’s optimal action is to go to one of their houses rather than ours. The particular gain values that lead to that don’t really matter to us. ¹⁹

Looking ahead, in Chap. 4 we will turn our attention to *channels*, systems that can leak information about secrets, and in Chap. 5 we will see how to measure their impact on the g -vulnerability of a secret. One possibility is again to consider absolute security, the precise g -vulnerability values obtained. But here in fact there is a particularly fruitful approach to relative security, where we wish just to *compare* the security of two channels A and B . In Chap. 9 we will develop a *refinement* relation on channels, whose significance is that if A is refined by B (denoted $A \sqsubseteq B$), then the g -vulnerability resulting from B never exceeds that resulting from A , regardless of the prior π and gain function g . In such a situation, we are freed from having to think about precise g -vulnerability values — in particular, if A is our *specification* of what leakage we can tolerate, and B is an *implementation* satisfying $A \sqsubseteq B$, then we don’t need to worry about numbers at all; we can be content that B is secure enough.

¹⁹ A nice example is the advertisement for sports gear where one of two men threatened by a lion pulls running shoes out of his bag and puts them on. The other wonders why he thinks the shoes will allow him to outrun the lion. He replies “I don’t have to outrun the lion: I just have to outrun you.”

3.6 Exercises

Exercise 3.1 Let us explore g -vulnerability in the context of a hypothetical lottery. Suppose that the lottery sells tickets for \$2 each, in which the purchaser marks 6 choices out of the numbers from 1 to 40. (For example, the purchaser might mark 3, 23, 24, 31, 33, and 40.)

Then a drawing is held in which 6 such numbers are selected randomly. Suppose that the rules are that a ticket wins only if it exactly matches the 6 selected numbers, in which case the player gets \$5 million; otherwise, the player gets nothing. (Real lotteries have more complicated rules!)

- (a) Design a gain function g suitable for modeling this lottery. The set \mathcal{X} of possible secret values x is the set of all sets of 6 numbers from 1 to 40. Let the set \mathcal{W} of possible actions include “buy t ” for each set t of 6 numbers from 1 to 40, along with the action “don’t play”.
- (b) Calculate the g -vulnerability $V_g(\vartheta)$, assuming that ϑ is uniform. Which action is optimal?

□

Exercise 3.2 Let g be a gain function with the following matrix representation:

G	x_1	x_2
w_1	3	-1
w_2	-8	2

Give a prior π such that $V_g(\pi) < 0$, which implies that $g \notin \mathbb{G}\mathcal{X}$.

□

3.7 Chapter notes

The g -vulnerability family was proposed by Alvim et al. [2] and further developed by McIver et al. [9] and Alvim et al. [1]. Its use of gain functions and loss functions to determine optimal actions is also fundamental in *decision theory*, discussed for example by DeGroot [4].

The observation that there are many reasonable (and incomparable) ways to measure uncertainty (or vulnerability) has a long history, with important contributions from Rényi [12], Massey [7], and Pliam [11].

The fact that the sup of convex functions is convex (used in the proof of Thm. 3.13) can be found in Rockafellar [13, Thm. 5.5].

We will see in Part IV, where channels are embedded into a “QIF-aware” probabilistic programming language, that loss functions can play a role analogous to Kozen’s generalization of Hoare-style assertions [5], and a nice hierarchy is revealed: if your loss function ℓ is based on a single action, and takes values 0,1 only (effectively Boolean), then it is equivalent in power to a Hoare-style postcondition, suitable for reasoning when you have neither probability nor leakage; if however you allow real values, in the style of Kozen [6] and further developed in *pGCL* [10, 8], but still restrict to a single action, then you can reason about probabilistic behavior but not yet information flow. Only when you allow *multiple* real-valued actions, as we have explained in this chapter, does the semantics become “QIF-aware”.

Bibliography

- [1] Alvim, M.S., Chatzikokolakis, K., McIver, A., Morgan, C., Palamidessi, C., Smith, G.: Additive and multiplicative notions of leakage, and their capacities. In: Proceedings of the 2014 IEEE 27th Computer Security Foundations Symposium (CSF 2014), pp. 308–322. IEEE, Los Alamitos (2014)
- [2] Alvim, M.S., Chatzikokolakis, K., Palamidessi, C., Smith, G.: Measuring information leakage using generalized gain functions. In: Proceedings of the 2012 IEEE 25th Computer Security Foundations Symposium (CSF 2012), pp. 265–279. IEEE, Los Alamitos (2012)
- [3] Cover, T.M., Thomas, J.A.: Elements of Information Theory. Wiley Series in Telecommunications and Signal Processing. Wiley-Interscience (2006)
- [4] DeGroot, M.H.: Optimal Statistical Decisions. Wiley-Interscience (2004)
- [5] Hoare, C.A.R.: An axiomatic basis for computer programming. Communications of the ACM **12**(10), 576–580 (1969)
- [6] Kozen, D.: A probabilistic PDL. In: Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC 1983), pp. 291–297. ACM, New York (1983)
- [7] Massey, J.L.: Guessing and entropy. In: Proceedings of the 1994 IEEE International Symposium on Information Theory, p. 204 (1994)
- [8] McIver, A., Morgan, C.: Abstraction, Refinement and Proof for Probabilistic Systems. Monographs in Computer Science. Springer, Berlin (2005)
- [9] McIver, A., Morgan, C., Smith, G., Espinoza, B., Meinicke, L.: Abstract channels and their robust information-leakage ordering. In: M. Abadi, S. Kremer (eds.) Proceedings of the 3rd Conference on Principles of Security and Trust (POST 2014), *Lecture Notes in Computer Science*, vol. 8414, pp. 83–102. Springer, Berlin (2014)
- [10] Morgan, C., McIver, A., Seidel, K.: Probabilistic predicate transformers. ACM Transactions on Programming Languages and Systems **18**(3), 325–353 (1996)
- [11] Pliam, J.O.: On the incomparability of entropy and marginal guesswork in brute-force attacks. In: B. Ray, E. Okamoto (eds.) Proceedings of the First International Conference on Cryptology in India, INDOCRYPT 2000, *Lecture Notes in Computer Science*, vol. 1977, pp. 67–79. Springer, Berlin (2000)

Bibliography

- [12] Rényi, A.: On measures of entropy and information. In: Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics, pp. 547–561. University of California Press, Berkeley (1961)
- [13] Rockafellar, R.T.: Convex analysis. Princeton Mathematical Series. Princeton University Press, Princeton (1970)

Part III

Channels and Information Leakage



Chapter 4

Channels

We now turn our attention to *systems* that process secret information. In this part of the book, we model such systems as *information-theoretic channels*, which are (possibly probabilistic) functions from inputs to outputs.¹

A channel is a system that takes as input a secret X , whose possible values come from a finite set \mathcal{X} , and whose only observable behavior is to produce an output Y , whose possible values come from a finite set \mathcal{Y} . Because we wish to allow *probabilistic* systems, we model such a system as a function C of type $\mathcal{X} \rightarrow \mathbb{D}\mathcal{Y}$, so that for each x in \mathcal{X} , we have $C(x)$ as a *distribution* on \mathcal{Y} , giving the probability of each possible output value y . An important special case is a *deterministic* system, where each possible input gives rise to a unique output; in that case the channel can be described more simply as a function C in $\mathcal{X} \rightarrow \mathcal{Y}$.

In thinking about channels, it is important to understand that the set \mathcal{Y} of observable output values should model *everything* that an adversary can observe about the system's behavior. Hence the choice of \mathcal{Y} is crucial — and it can be subtle. For instance, given a secret input X a system might output a value Y . But it might be that the system is implemented in such a way that the amount of *time* needed to calculate Y depends on the value of X . In that case it might be more appropriate to model the space of outputs as $\mathcal{Y} \times \mathcal{T}$, where \mathcal{T} is the set of possible calculation times. In general, therefore, it is important to remember that a channel is only a *model* — and it might or might not be faithful enough to encompass all actual threats in a real-life scenario.

4.1 Channel matrices

Above we defined a channel to have type $\mathcal{X} \rightarrow \mathbb{D}\mathcal{Y}$, where \mathcal{X} is the type of its input and \mathcal{Y} is the type of its outputs/observables. But channels can of course be *described* in a variety of ways, and for that purpose we will usually use information-theoretic *channel matrices*, which show the input-output behavior explicitly.

¹ Later, in Part IV, we will consider a richer class of systems that can be written as sequential programs, and there we will distinguish *outputs* from *observables*: a program's output is its final state, and its observables are the information that leaks from it as it executes. In this chapter however we use “output” and “observable” synonymously.

Definition 4.1 (Channel matrix)

Let \mathcal{X} and \mathcal{Y} be finite sets, intuitively representing secret input values and observable output values. A *channel matrix C from \mathcal{X} to \mathcal{Y}* is a matrix, indexed by $\mathcal{X} \times \mathcal{Y}$, whose rows give the distribution on outputs corresponding to each possible input. That is, entry $C_{x,y}$ denotes $p(y|x)$, the conditional probability of getting output y given input x .² Note that all entries in a channel matrix are between 0 and 1, and each row sums to 1, which property is called *stochastic* (and is how we will describe such matrices even if they might not correspond to actual channels).

The x 'th row of C is written $C_{x,-}$, and similarly the y 'th column is $C_{-,y}$.

Mathematically, a channel matrix has type $\mathcal{X} \times \mathcal{Y} \rightarrow [0, 1]$; but that is of course isomorphic to the type $\mathcal{X} \rightarrow \mathbb{D}\mathcal{Y}$ that we gave previously, provided the matrix is *stochastic*, i.e. has one-summing rows. But we will often prefer to write the type of a channel more concisely as $\mathcal{X} \rightarrow \mathcal{Y}$, suppressing the “syntactic noise” of the \times and the $[0, 1]$, and thinking of it as a “probabilistic function” from \mathcal{X} to \mathcal{Y} .

For example, here is a channel matrix C :

C	y_1	y_2	y_3	y_4
x_1	1/2	1/2	0	0
x_2	0	1/4	1/2	1/4
x_3	1/2	1/3	1/6	0

In general, channel matrices (like C here) are (properly) *probabilistic*, in that the output is not always uniquely determined by the input: for example, in C we have $p(y_3|x_3) = 1/6$. As we said above, however, an important special case is the *deterministic* channel matrix, in which each row contains a single 1, identifying the unique possible output for that input. For example, here is a deterministic channel matrix D :

D	y_1	y_2
x_1	1	0
x_2	1	0
x_3	0	1

However, we will see later, in Def. 4.13, that a properly probabilistic channel matrix can nonetheless be seen as “essentially” deterministic.³

A disadvantage of channel matrices however is that, when \mathcal{X} or \mathcal{Y} are large, they can be too big to write down explicitly. For that reason, we sometimes find it convenient to represent channels as *pseudocode*. For example, a certain deterministic channel taking as input a 64-bit unsigned X (whose channel matrix hence has 2^{64} rows) can be described by the following pseudocode:

$$Y := \text{if } (X \bmod 8) = 0 \text{ then } X \text{ else } 1 \quad (4.1)$$

² Here it might be objected that $p(y|x)$ is meaningless without a joint distribution. However, we agree with Rényi that “the basic notion of probability theory should be the notion of the conditional probability of A under the condition B ”.

³ Additional discussion of deterministic channels is given in Chap. 17, where a different generalization is explored: demonic rather than probabilistic choice, corresponding to *qualitative* (or possibilistic) rather than quantitative information flow.

In this part of the book, pseudocode fragments like the above are kept informal: we use them simply as convenient abbreviations for channel matrices. (In contrast, in Part IV we will treat programs formally, carefully developing in §13.4 a programming language whose semantics incorporates an observation model aimed specifically at supporting the development of secure programs by stepwise refinement.)

4.2 The effect of a channel on the adversary's knowledge

Now we turn to the question of how a channel affects the adversary's knowledge about a secret X with prior distribution π .

Suppose that channel matrix C shown above produces output y_4 . In that case, the adversary can deduce that X must be x_2 , since that is the only input that can have led to that output. But note that here we are relying on the assumption that she knows how C works — and we do assume that throughout.

Assumption

Throughout this book, we make the worst-case assumption that the adversary *knows how the channel works*, in the sense of knowing the channel matrix C .⁴ At a minimum, that means that she can compute $C_{x,y}$ for any $x:\mathcal{X}$ and $y:\mathcal{Y}$.

That assumption is related to the common security slogan “no security through obscurity”, which is also known as *Kerckhoffs' Principle*.⁵

In general, observing an output y allows the adversary to update her knowledge about X from the prior distribution π to a *posterior distribution*, which we denote $p_{X|y}$. The relevant probabilities can be computed using Bayes' Theorem

$$p(x|y) = \frac{p(y|x) p(x)}{p(y)} .$$

More precisely, if we fix a prior distribution π and channel matrix C then we can describe very compactly a variety of important probabilities and distributions, writing them using a conventional “ p ” notation in which the π and $C:\mathcal{X}\rightarrow\mathcal{Y}$ are implicit, i.e. taken from context. We illustrate that in the next several paragraphs.

First, we can define the joint distribution p_{XY} (which gives the joint probability of each input-output pair) by

$$p_{XY}(x,y) := \pi_x C_{x,y} .$$

Then by marginalization we obtain random variables X and Y with distributions

$$p_X(x) = \sum_{y:\mathcal{Y}} p_{XY}(x,y)$$

and

$$p_Y(y) = \sum_{x:\mathcal{X}} p_{XY}(x,y) .$$

⁴ When we consider programs in Part IV, that is tantamount to saying that the adversary knows the program's source code.

⁵ Auguste Kerckhoffs' 1883 article “La cryptographie militaire” set forth six desiderata for military ciphers. The second desideratum can be translated as “The system must not require secrecy, and must be able to fall into the hands of the enemy without disadvantage.”

However it is customary and convenient to *omit the subscripts* from such distributions whenever they can be deduced unambiguously from the arguments used; and we will follow that convention, writing for example $p(x, y)$, $p(x)$, and $p(y)$.

Now we can compute the following conditional probabilities, provided of course that the denominators $p(x)$ or $p(y)$ are nonzero:

$$p(y|x) = \frac{p(x,y)}{p(x)}$$

and

$$p(x|y) = \frac{p(x,y)}{p(y)} .$$

At this point, we can justify the definition above of p_{XY} : it turns out that p_{XY} is the *unique* joint distribution that recovers π and C , in that $p(x) = \pi_x$ and $p(y|x) = C_{x,y}$.

For a given y , the conditional probabilities $p(x|y)$ for each x in \mathcal{X} form the *posterior distribution* $p_{X|y}$, which represents the posterior knowledge that the adversary has about input X after observing output y . If we have $\mathcal{X} = \{x_1, x_2, \dots, x_n\}$, then we can write $p_{X|y}$ explicitly as a tuple:

$$p_{X|y} = (p(x_1|y), p(x_2|y), \dots, p(x_n|y)) .$$

The different posterior distributions $p_{X|y}$ for y in \mathcal{Y} represent the different states of knowledge, or “worlds”, that an adversary seeing the output of C can end up in. But there is more to the story than just the *set* of posterior distributions; we also must consider that posterior distribution $p_{X|y}$ occurs when the output is y , and that output y *itself* occurs with probability $p(y)$. Thus a posterior distribution *as a whole* has a probability of occurring; and that probability is quite important in understanding the situation, because although a posterior distribution might be favorable for the adversary, it might not be very likely to occur.

Example 4.2 Let us compute the posterior distributions and their probabilities for the channel matrix C from above, viz.

C	y_1	y_2	y_3	y_4
x_1	$1/2$	$1/2$	0	0
x_2	0	$1/4$	$1/2$	$1/4$
x_3	$1/2$	$1/3$	$1/6$	0

under the prior $\pi = (1/4, 1/2, 1/4)$. We can conveniently write the joint distribution p_{XY} that C defines as a *joint matrix* J where $J_{x,y} = p(x,y)$. Note that J is computed by multiplying row x of C by the prior probability π_x , giving

J	y_1	y_2	y_3	y_4
x_1	$1/8$	$1/8$	0	0
x_2	0	$1/8$	$1/4$	$1/8$
x_3	$1/8$	$1/12$	$1/24$	0

The marginal distribution p_Y can be found by summing the columns of J , since $p_Y(y) = \sum_{x \in \mathcal{X}} p(x,y)$; thus here we get $p_Y = (1/4, 1/3, 7/24, 1/8)$. Now each possible value of Y gives a posterior distribution on X , by Bayesian updating. Since $p(x|y) = p(x,y)/p(y)$, those posterior distributions can be calculated by normalizing the columns of J , giving

	$p_{X y_1}$	$p_{X y_2}$	$p_{X y_3}$	$p_{X y_4}$
x_1	$1/2$	$3/8$	0	0
x_2	0	$3/8$	$6/7$	1
x_3	$1/2$	$1/4$	$1/7$	0

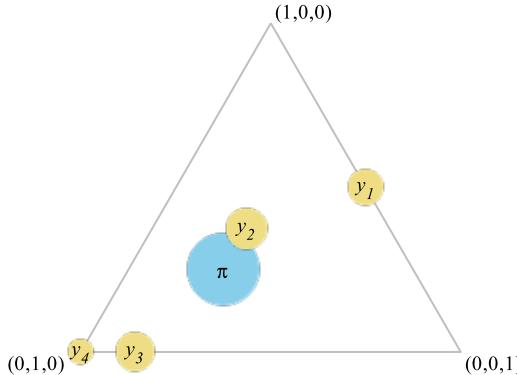


Figure 4.1 Geometric representation of prior π and the four posterior distributions induced by channel C

It is interesting to note here that $p_{X|y_4}$ appears most favorable for the adversary, since it is a point distribution; but it is the least likely to occur, since further up we see that $p(y_4) = 1/8$. In contrast, $p_{X|y_2}$ appears least favorable to the adversary, since it is fairly uniform — but it is the most likely to occur, since $p(y_2) = 1/3$.

Figure 4.1 shows a geometric representation, using barycentric coordinates,⁶ of how channel C “explodes” the prior π into four posterior distributions; note that the sizes of the various circles are proportional to their probabilities. \square

There is a fundamental property satisfied by the posterior distributions $p_{X|y}$ and their respective marginal probabilities $p(y)$.

Theorem 4.3 For any $\pi: \mathbb{D}\mathcal{X}$ and channel matrix $C: \mathcal{X} \rightarrow \mathcal{Y}$, the average (i.e. expected value) of the posterior distributions $p_{X|y}$, weighted by their probabilities $p(y)$, is equal to the prior π :

$$\sum_{y \in \mathcal{Y}} p(y) p_{X|y} = \pi ,$$

where within the summation we are multiplying a distribution (on the right) by a scalar (on the left). It is understood pointwise: thus more explicitly we could write $\sum_{y \in \mathcal{Y}} p(y) p_{X|y}(x) = \pi_x$.

Proof. Weighting a posterior distribution $p_{X|y}$ with its probability $p(y)$ recovers the corresponding column of the joint matrix J . Hence summing those weighted posterior distributions gives the marginal distribution p_X , which is equal to the prior π . (See also Exercise 4.3.) \square

That theorem can be understood geometrically: in Figure 4.1, if the four posterior distributions are weighted according to their probabilities $(1/4, 1/3, 7/24, 1/8)$, then the *center of mass* is located at the prior π .

⁶ Barycentric coordinates are discussed more extensively in Chap. 12.

4.3 From joint distributions to hyper-distributions

The discussion in the previous section is fundamentally based on the joint distribution p_{XY} . However, that turns out not to be entirely satisfactory.

A first issue is that the particular output set \mathcal{Y} is an *inessential detail* of a channel matrix C , as far as leakage is concerned. The reason is that the *information* available to the adversary does not at all depend on the fact that C 's outputs belong to any particular set \mathcal{Y} ; renaming the outputs would make no difference to the leakage caused by C , since the adversary's deductions depend only on the *correlation* between input values and output values, and not on the particular output values themselves. As an example, consider a channel C taking as input a user's medical record X , assumed to contain both non-sensitive data (e.g. the state where the patient resides, say Florida) and sensitive data (e.g. the patient's diagnoses). Suppose that C is malicious and outputs *diabetes* if the patient has been diagnosed with diabetes, and *no diabetes* if not. Remembering that the adversary is assumed to know the channel matrix C , we see that it would make absolutely *no difference* to the sensitive information leaked if the two outputs *diabetes* and *no diabetes* were renamed to *Florida* and *FL*, respectively, writing the patient's state of residence in either unabbreviated or abbreviated form according to whether or not he has diabetes.⁷ But of course that change might make the leak less obvious to a casual observer.

A second issue is that two complications can arise in going from a prior π and channel matrix C to the posterior distributions $p_{X|y}$ and their (\mathcal{Y} -marginal) probabilities $p(y)$. First, an output value y can be *impossible*, in that $p(y) = 0$. That happens trivially when column y of C is all zeroes, but it also can happen when prior π is not full support and the nonzero entries of column y occur only for inputs x that, according to the prior, can never occur: they are on rows x where $\pi_x = 0$. In that case, column y of the joint matrix is all zeroes, preventing our normalizing it, and therefore it does not contribute a posterior distribution at all. Second, two output values y and y' can give rise to the *same* posterior distribution, meaning that the adversary gets no benefit from distinguishing those two outputs from each other.

As an illustration, suppose that prior π is $(1/3, 1/3, 0, 1/3)$ and that C is

C	y_1	y_2	y_3	y_4
x_1	$1/2$	$1/6$	$1/3$	0
x_2	0	$1/3$	$2/3$	0
x_3	0	$1/2$	0	$1/2$
x_4	$1/4$	$1/4$	$1/2$	0

⁷ It should however be noted that the unimportance of the particular output values relies on our assumption (discussed at the end of Section 2.3) that adversaries are *information-theoretic*, without computational resource bounds. If we were considering computationally bounded adversaries, then the particular output values could well matter, because an output y might or might not allow the adversary to compute information about the input *efficiently*, without having to scan the entire column y of C . As an example, consider a channel matrix whose input is a 1000-bit prime p . From an information-theoretic perspective, it would make no difference if the output were p^2 or pq , for a randomly-chosen 1001-bit prime q , since in either case there would be only one input value corresponding to each output value. But the output p^2 would let the adversary recover p efficiently by computing a square root, while the output pq would require the adversary to solve a difficult factorization problem. Interestingly, however, that distinction does *not* hold in the setting of *quantum computers*, where Shor's algorithm enables efficient integer factorization. And so relying too much on computational limits now could lead to security issues later.

Here we find that the joint matrix is

J	y_1	y_2	y_3	y_4
x_1	$1/6$	$1/18$	$1/9$	0
x_2	0	$1/9$	$2/9$	0
x_3	0	0	0	0
x_4	$1/12$	$1/12$	$1/6$	0

so that $p_Y = (1/4, 1/4, 1/2, 0)$ and therefore $p_{X|y_4}$ is undefined. Because $p(y_4)$ is zero, we can reduce the matrix by dropping that column, and the posterior distributions that we get are these:

	$p_{X y_1}$	$p_{X y_2}$	$p_{X y_3}$
x_1	$2/3$	$2/9$	$2/9$
x_2	0	$4/9$	$4/9$
x_3	0	0	0
x_4	$1/3$	$1/3$	$1/3$

But now we find that $p_{X|y_2} = p_{X|y_3}$, and hence there are only *two* possible adversary “worlds”, rather than *three*, as it makes no difference to the adversary whether the output is y_2 or y_3 . Moreover, the probability of the “world” $(2/9, 4/9, 0, 1/3)$ is actually $p(y_2) + p(y_3) = 1/4 + 1/2 = 3/4$.

The above suggests that we could reduce the matrix again, this time merging columns y_2 and y_3 together: the result is

	$p_{X y_1}$	$p_{X “y_2 \text{ or } y_3”}$
x_1	$2/3$	$2/9$
x_2	0	$4/9$
x_3	0	0
x_4	$1/3$	$1/3$

Later, we will see the same “reduction” process applied to channel matrices, for similar reasons.

In light of those two issues, we choose to forget about the particular output values $\{y_1, y_2, y_3, y_4\}$, and to model the effect of channel C on prior π simply as a *distribution on posterior distributions*, which we call a *hyper-distribution* and denote by $[\pi \triangleright C]$ and pronounce “ π through C”. Here is the hyper-distribution for our example above:

$[\pi \triangleright C]$	$1/4$	$3/4$
x_1	$2/3$	$2/9$
x_2	0	$4/9$
x_3	0	0
x_4	$1/3$	$1/3$

It should be noted that a hyper-distribution abstracts from the “bookkeeping” that the adversary needs to remember in order to know which outputs correspond to which posterior distributions. In the case above, the bookkeeping would have been remembering that output y_1 corresponds to posterior distribution $(2/3, 0, 0, 1/3)$, while outputs y_2 and y_3 both correspond to posterior distribution $(2/9, 4/9, 0, 1/3)$. But –as we have shown– those details are irrelevant to an assessment of what information leakage about X is caused by C under prior π .

Furthermore, it turns out that the abstraction afforded by the hyper-distribution perspective is particularly important when we are interested in *comparing* channels to try to decide whether one is more secure than the other. Here is a striking example.

Example 4.4 Given $\mathcal{X} = \{x_1, x_2, x_3\}$, consider these channel matrices C and D:

C	y_1	y_2	y_3
x_1	1	0	0
x_2	$1/4$	$1/2$	$1/4$
x_3	$1/2$	$1/3$	$1/6$

D	z_1	z_2	z_3
x_1	$2/5$	0	$3/5$
x_2	$1/10$	$3/4$	$3/20$
x_3	$1/5$	$1/2$	$3/10$

While C and D look completely different, it turns out that they are actually *identical* with respect to the leakage of X that they cause. Indeed both map an arbitrary prior distribution $\pi = (p_1, p_2, p_3)$ to the very same hyper-distribution:

	$\frac{4p_1+p_2+2p_3}{4}$	$\frac{3p_2+2p_3}{4}$
x_1	$\frac{4p_1}{4p_1+p_2+2p_3}$	0
x_2	$\frac{p_2}{4p_1+p_2+2p_3}$	$\frac{3p_2}{3p_2+2p_3}$
x_3	$\frac{2p_3}{4p_1+p_2+2p_3}$	$\frac{2p_3}{3p_2+2p_3}$

Hence there is no reason for any adversary to prefer C to D, or vice versa, leading us to the view that C and D should be seen, at least in abstract terms, as the same. Indeed by defining “abstract channels” in §4.4, we will make that precise; there we will also look more carefully at C and D to explain why that happens. \square

We now define hyper-distributions formally, and introduce some convenient terminology for talking about them.

Definition 4.5 (Hyper-distribution)

If \mathcal{X} is a finite set (of possible secret values), a *hyper-distribution* (or just a *hyper*) Δ is a distribution on distributions on \mathcal{X} , so that Δ has type $\mathbb{D}(\mathbb{D}\mathcal{X})$, which we abbreviate to $\mathbb{D}^2\mathcal{X}$. We recall that the support $[\Delta]$ of Δ is the set of distributions to which Δ gives positive probability (i.e. $[\Delta] = \{\delta: \mathbb{D}\mathcal{X} \mid \Delta_\delta > 0\}$) and we assume that set to be *finite*;⁸ they are the set of possible “worlds” under Δ and we call them the *inner distributions*, or just the *inners*, of Δ . We refer to the distribution on the inners themselves as the *outer distribution* of Δ .

As mentioned above, the hyper-distribution that results from a channel matrix C on prior π is written $[\pi \triangleright C]$, and pronounced “ π through C”.

When it is convenient to have *names* for the inners and their outer probabilities, we can write $[\pi \triangleright C] = \sum_i a_i [\delta^i]$ to indicate that $[\pi \triangleright C]$ has inners δ^i and outer probabilities a_i . (See §4.6.1 for further explanation of that notation.)

For example, the hyper-distribution for Example 4.2 is

$[\pi \triangleright C]$	$1/4$	$1/3$	$7/24$	$1/8$
x_1	$1/2$	$3/8$	0	0
x_2	0	$3/8$	$6/7$	1
x_3	$1/2$	$1/4$	$1/7$	0

Here the *outer distribution* is $(1/4, 1/3, 7/24, 1/8)$ on the *inner distributions* $(1/2, 0, 1/2)$ and $(3/8, 3/8, 1/4)$ and $(0, 6/7, 1/7)$ and $(0, 1, 0)$ respectively.

⁸ In Part IV we will actually relax our assumption of finitely many inners when we consider *sequential programs*, which can give rise to (countably) infinitely many inners due to looping, for example.

It is convenient to have a more explicit notation for the joint matrix J resulting from prior π and channel matrix C ; for that we simply drop the outer brackets $[-]$ from our notation for hypers, writing just $\pi \triangleright C$. In general, when we write J we will mean $\pi \triangleright C$ for some π and C in the text nearby.

That usage is supported by the following definition.

Definition 4.6 (Joint matrix and bracket notation) For prior π in $\mathbb{D}\mathcal{X}$ and channel matrix C in $\mathcal{X} \rightarrow \mathcal{Y}$, we write $\pi \triangleright C$ for the joint matrix of type $\mathbb{D}(\mathcal{X} \times \mathcal{Y})$ defined by

$$(\pi \triangleright C)_{x,y} := \pi_x C_{x,y} .$$

As for hypers, we call the operation “pushing π through C ”. Moreover, we define that the brackets $[-]$ on their own act as the function that takes a joint matrix of type $\mathbb{D}(\mathcal{X} \times \mathcal{Y})$ to the corresponding hyper-distribution of type $\mathbb{D}^2\mathcal{X}$; note that this two-step process for forming $[\pi \triangleright C]$ is consistent with Def. 4.5 just above.⁹ \square

We conclude this section with the following fundamental principle.

Principle

The information-theoretic essence of a channel matrix C is a mapping from priors π to hyper-distributions $[\pi \triangleright C]$.

4.4 Abstract channels

As discussed in the previous section, the information-theoretic leakage caused by a channel matrix C is determined exactly by the mapping that C defines from prior distributions to hyper-distributions. We call that mapping the *abstract channel* denoted by C .

Definition 4.7 (Abstract channel)

The *abstract channel* C denoted by channel matrix C is the mapping of type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$ that C gives, namely $\pi \mapsto [\pi \triangleright C]$. We use semantic brackets for that denotation, writing $C = [\![C]\!]$.

Note that we write channel matrices using a sans-serif font (C) and abstract channels using a math font (C). So we have two views of channels: *concrete channels* $C: \mathcal{X} \rightarrow \mathcal{Y}$ (typically represented by channel matrices) and *abstract channels* $C: \mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$.

Notation

As already noted, the hyper-distribution that results from a channel matrix C on a prior π is denoted $[\pi \triangleright C]$. Since an abstract channel C is a mapping from priors to hyper-distributions, we could denote the hyper-distribution produced by C on prior π as an ordinary function application, that is $C(\pi)$, and occasionally we will. Usually, however, we will overload the notation $[\pi \triangleright -]$ by applying it to both abstract and concrete channels, so that $[\pi \triangleright C] = [\![C]\!(\pi)] = [\pi \triangleright C]$.

⁹ A further step in this direction is taken in Def. 16.1 where $[-]$ is extended to act on “sub-” joint distributions. That is not needed here, but it turns out to be important for proving general properties about how hypers interact with each other.

The abstract-channel perspective is of central importance in our theory. For, as already seen in Example 4.4, apparently different channel matrices can actually be identical as far as information leakage is concerned, which implies that channel matrices contain detail that is extraneous to their fundamental meaning.¹⁰

Indeed there are aspects of a channel matrix that are somehow “accidental”, even if it was designed maliciously: the order of its columns, and the labels (in \mathcal{Y}) assigned to them, have no effect on the leakage it induces. (Neither does the order of the rows: but of course the row *labels* do matter.) We might even go so far as to say, quoting Jean-Yves Girard, that a channel matrix is “an unaccommodating object of study, without true structure, a piece of *soft camembert*”.

As a result, the theory becomes cleaner when focused on abstract channels. For instance, in Chap. 9 we develop a *refinement* relation (\sqsubseteq) to say that one channel never leaks more than another. Over channel matrices, we find that refinement fails to be *antisymmetric* — for instance, on the channel matrices C and D of Example 4.4 we have both $C \sqsubseteq D$ and $D \sqsubseteq C$. Over abstract channels, in contrast, refinement is antisymmetric, thus a *partial order* — that is, $C \sqsubseteq D \sqsubseteq C$ only when $C=D$.

Another crucial benefit of the abstract-channel perspective will become clear in Part IV, when we generalize from channels to *sequential programs*. The semantics of such programs, based on *Hidden Markov Models*, turn out, remarkably, *also* to be mappings of type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$, and that allows abstract channels to be incorporated seamlessly into that more general setting.

We should however clarify that our abstract-channel perspective is concerned solely with understanding the *information leakage* caused by some C , not with any functional specification it might be satisfying. (In effect, we are viewing C as a “covert channel”.) That is why abstracting from the actual output values is possible, and indeed is an advantage of our perspective. But what about a system that is supposed to perform a useful computation? In that case, we could of course use the concrete channel C to model the functional behavior. A better answer, however, will be developed in our study of *programs* in Part IV; there our use of Hidden Markov Models allows us to distinguish between leakage (via channels) and computation (via state updates), and to study how they interact.

We now develop some important properties of abstract channels. First, the fundamental property given in Thm. 4.3 carries over to abstract channels.¹¹

Corollary 4.8

For any abstract channel C and prior π , the average of the inner distributions of $[\pi \triangleright C]$, weighted by their outer probabilities, is equal to the prior π . That is,

$$\pi = \sum_i a_i \delta^i, \quad \text{where} \quad [\pi \triangleright C] = \sum_i a_i [\delta^i].$$

Proof. The only difference from Thm. 4.3 is that $[\pi \triangleright C]$ merges any posterior distributions that are equal, summing their outer probabilities — and that has no effect on the expectation. \square

Note that this weighted average is simply the *expected value* (or *mean*) of the hyper-distribution $[\pi \triangleright C]$. Writing $\mu\Delta$ for the expected value of a distribution Δ (note: a

¹⁰ Joseph Goguen referred to that principle as “no junk”. The complementary principle of an abstraction’s not being *too abstract* is called “no confusion”.

¹¹ Interestingly, while the programs considered in Part IV do have type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$, they do *not* satisfy that property: channels satisfy it because, even if generalized to programs, they do not update the state. They update only *what is known* about the state. (See however §16.3.2, where it is argued that a “nonterminating” channel can in fact alter the state.)

distribution of distributions, in this case), so that $\mu\Delta = \sum_{\delta} \Delta_{\delta} \times \delta$, the corollary above can be written as $\mu[\pi \triangleright C] = \pi$, or more revealingly as just $\mu \circ C = \text{id}$, where (\circ) is the usual function composition and id the identity function.

Turning next to the relationship between channel matrices and the abstract channels they denote, we recall from the discussion above that channel matrices contain *extraneous structure*, namely

1. *labels* on columns,
2. columns that are *all zero*, representing outputs that can never occur, and
3. *similar* columns, by which we mean columns that are scalar multiples of each other and therefore yield the same posterior distributions on all priors.¹²

At the level of syntax, we can eliminate the extraneous structure from a channel matrix to get a well defined *reduced matrix*.

Definition 4.9 (Reduced matrix) The *reduced matrix* C^r of a channel matrix C is formed by deleting output labels and all-zero columns, adding similar columns together, and finally ordering the resulting columns lexicographically.¹³ \square

The reduced matrix is analogous to a *normal form*, also a syntactic concept.

The significance of the reduction process is given by the following theorem and corollary, that meaning is preserved.

Theorem 4.10 Any channel matrix C denotes the same abstract channel as its reduced matrix C^r . That is, $\llbracket C \rrbracket = \llbracket C^r \rrbracket$.

Proof. Output labels, all-zero columns, and column ordering all have no effect on the hyper-distribution. And similar columns each contribute weight to the same posterior distribution; hence merging them leaves the hyper-distribution unchanged. \square

Corollary 4.11 Two channel matrices C and D denote the same abstract channel iff their reduced matrices are equal: that is, $\llbracket C \rrbracket = \llbracket D \rrbracket$ iff $C^r = D^r$.

Proof. From Thm. 4.10 we have $\llbracket C \rrbracket = \llbracket D \rrbracket$ iff $\llbracket C^r \rrbracket = \llbracket D^r \rrbracket$. And it is easy to see that $\llbracket C^r \rrbracket = \llbracket D^r \rrbracket$ iff $C^r = D^r$: the backward implication is immediate, and the forward implication follows because distinct reduced matrices cannot map the uniform prior (for instance) to the same hyper-distribution. \square

The result is that a reduced matrix (like a normal form) serves as a *canonical syntactic representation* of an abstract channel. Another way to say this is that reduced matrices serve as normal forms that are unique and respect the semantics.

Example 4.12 Consider again the channel matrices C and D from Example 4.4:

C	y_1	y_2	y_3
x_1	1	0	0
x_2	$1/4$	$1/2$	$1/4$
x_3	$1/2$	$1/3$	$1/6$

D	z_1	z_2	z_3
x_1	$2/5$	0	$3/5$
x_2	$1/10$	$3/4$	$3/20$
x_3	$1/5$	$1/2$	$3/10$

¹² They can be seen as analogous to redundant information in computer programs, like the names of local variables, dead code, and if-statements with identical branches. Case 2 could be seen as an instance of Case 3 with a scaling factor of zero; but then similarity would not be symmetric.

¹³ Ordering the columns lexicographically is just for the sake of uniqueness, and it should be understood that the particular order is of no significance, exactly as with sets we have $\{1, 2\} = \{2, 1\}$.

Let us find their corresponding reduced matrices. In C the columns y_2 and y_3 are similar: column y_2 is two times column y_3 . And in D the columns z_1 and z_3 are similar: column z_1 is two-thirds times column z_3 . Merging those similar columns, we find that C and D have the *same* reduced matrix:

$$C^r = D^r = \left[\begin{array}{c|cc} x_1 & 1 & 0 \\ x_2 & 1/4 & 3/4 \\ x_3 & 1/2 & 1/2 \end{array} \right].$$

That shows that C and D denote the *same* abstract channel. \square

Recall that in §4.1 we defined *determinism* as a property of *channel matrices*. However, it is meaningful to apply that concept to abstract channels as well.

Definition 4.13 (Deterministic abstract channel) We say that an abstract channel C is deterministic just when, for every π , the inners of $[\pi \triangleright C]$ have pairwise-disjoint supports. (The corresponding concept for concrete channels is that their reductions contain only zeroes and ones.) \square

We earlier defined “deterministic” for channel matrices to be “only one nonzero entry per row”. But we saw later that when a channel matrix is reduced, several (nonzero) fractions could combine to a single nonzero entry for the whole row (which entry therefore must be 1). In that case we might say that while the channel matrix appears not to be deterministic, “in essence” it is. That idea is captured exactly by saying that a channel matrix C is *essentially* deterministic iff its corresponding abstract channel C is deterministic according to Def. 4.13.

The two most extreme channels, \mathbb{O} and $\mathbb{1}$

There are two extreme channels that deserve consideration: they are the channel that leaks *nothing*, and the channel that leaks *everything*. If we recall the way that Fig. 4.1 shows graphically how a channel “explodes” a prior into a hyper-distribution, then we can see that the channel that leaks nothing is a “dud” that causes no explosion at all, while the channel that leaks everything blows the prior “to bits”, making all the posterior distributions sit on the *vertices* of the triangle.

The abstract channel that leaks nothing is (using the notation for hyper-distributions from Def. 4.5) the mapping $\pi \mapsto [\pi]$; the corresponding reduced channel matrix is a single column of 1’s. We choose to call that channel $\mathbb{1}$, which can be remembered by noting that it *preserves* secrecy.

The abstract channel that leaks everything is the mapping $\pi \mapsto \sum_x \pi_x [[x]]$; the corresponding reduced channel matrix is the identity matrix \mathbb{I} . We choose to call that channel \mathbb{O} , which can be remembered by noting that it *annihilates* secrecy.

Further intuition for those names will become clear in Chap. 8, when we consider the parallel composition of channels: channel $C \parallel D$ is the channel that maps the prior to the hyper-distribution resulting from getting the outputs of *both* C and D . There we will find that for every C , we have the nice algebraic laws $C \parallel \mathbb{1} = C$ and $C \parallel \mathbb{O} = \mathbb{O}$. Moreover, when we consider *refinement* in Chap. 9, we will find that $\mathbb{O} \sqsubseteq \mathbb{1}$, which is what we would expect.

A traditional name for the “no leakage” property of $\mathbb{1}$ is *noninterference*; on channel matrices this is the property that the output is completely independent of the input.

Definition 4.14 (Noninterference) A channel matrix C satisfies *noninterference* just when its rows are all the same, which means that its reduction C' consists of a single column of 1’s. Equivalently, its abstract channel is the mapping $\pi \mapsto [\pi]$. (Channel $\mathbb{1}$ is hence the canonical noninterfering channel.) \square

4.5 More on abstract channels

From Def. 4.7, we know that an abstract channel C is a mapping $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$ denoted by some channel matrix C , so that $C = [\mathbb{C}]$. In this section, we explore such mappings a bit further, trying to understand the special properties that they satisfy.

There are in fact two special properties of abstract channels that we have already noted. First, as reflected in Def. 4.5, any abstract channel C always produces hyper-distributions with only *finitely many inners*, because any channel matrix has only finitely many columns. Second, as proved in Cor. 4.8, for any abstract channel C and prior π , the *expected value* of the hyper-distribution $[\pi \triangleright C]$ is always the prior π . (Recall that after Fig. 4.1, we interpreted that property graphically as saying that the *center of mass* of the hyper-distribution $[\pi \triangleright C]$ is always located at the prior π .)

We might wonder whether those two properties are enough to characterize the abstract channels that are denoted by channel matrices; but the following counterexample shows that they are not.

Example 4.15 Let $\mathcal{X} = \{x_1, x_2\}$ and let $C : \mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$ map prior $\pi = (a, 1-a)$, for $0 \leq a \leq 1$, to the hyper-distribution

$[\pi \triangleright C]$	a^2	$1 - a^2$
x_1	1	$\frac{a}{1+a}$
x_2	0	$\frac{1}{1+a}$

Note that this hyper-distribution indeed has finitely many inners, and its expectation is indeed the prior π . (Also notice that if $a=0$ or $a=1$ then the hyper-distribution actually has just one inner.)

But it is easy to see that C is not the denotation of any channel matrix. For if $0 < a < 1$, then we can work backwards to recover a unique reduced channel matrix C such that $[\pi \triangleright C] = [\pi \triangleright C]$. First we multiply each inner by its outer probability to recover the joint matrix

$$J = \begin{bmatrix} x_1 & a^2 & a(1-a) \\ x_2 & 0 & 1-a \end{bmatrix},$$

and then by normalizing the rows we obtain the reduced channel matrix

$$C = \begin{bmatrix} x_1 & a & 1-a \\ x_2 & 0 & 1 \end{bmatrix}.$$

But now we see that no single C will do the job, since this C varies with the prior $\pi = (a, 1-a)$. \square

The same reasoning used in Example 4.15 lets us prove the following theorem, which shows that the abstract channels denoted by channel matrices are very tightly constrained indeed.

Theorem 4.16 (Decomposition of hyper-distribution) Let Δ be a hyper-distribution with finitely many inners. Then there exists an abstract channel C and a unique prior π such that $[\pi \triangleright C] = \Delta$. And if π is full support, then also C is unique.

Proof. Let π be the expectation of Δ ; by Cor. 4.8 it is the unique possible prior. By scaling each of the inners of Δ with its outer probability, we recover a joint matrix J with finitely many columns and whose marginal distribution on \mathcal{X} is π . Moreover, since the inners of Δ are distinct and the outer probabilities are nonzero, none of the columns of J are similar.

To prove existence, form a channel matrix C by normalizing each nonzero row of J and by choosing an arbitrary one-summing row for each all-zero row of J . Then $[\pi \triangleright C] = \Delta$, so we can choose $C = [\![C]\!]$.

Now suppose that π is full support. To prove uniqueness, suppose that R is any reduced channel matrix such that $[\pi \triangleright R] = \Delta$. Because none of the columns of R are similar and π is full support, also none of the columns of the joint matrix $\pi \triangleright R$ are similar. Hence the set of columns of $\pi \triangleright R$ must be equal to the set of columns of J . Hence the entries of R are uniquely determined — we must have $R_{x,y} = J_{x,y}/\pi_x$. And the uniqueness of R implies the uniqueness of C , by Cor. 4.11. \square

We remark that Thm. 4.16 tells us that we can also represent an abstract channel canonically by the hyper-distribution Δ that it produces on (for instance) the uniform prior ϑ .¹⁴

Interestingly, an abstract channel is *not* necessarily uniquely determined by the *set* of inners (i.e. the *support* of the hyper) that it produces on a full-support prior. Here is an example.

Example 4.17 For $0 < a < 1$, let channel matrix C^a be defined by

C^a	y_1	y_2	y_3	
x_1	0	a	$1-a$	
x_2	$1-a$	a	0	

On the uniform prior $\vartheta = (1/2, 1/2)$, the inners of $[\vartheta \triangleright C^a]$ are always $(0, 1)$, $(1/2, 1/2)$, and $(1, 0)$, regardless of a . But different values of a do affect the outer distribution, and hence the abstract channel denoted by C^a . \square

An interesting topic for future study is to determine what properties characterize abstract channels precisely, viewing them simply as functions of type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$, independent of the channel matrices that realize them. Such properties are sometimes called “healthiness conditions”.

¹⁴ In the setting of *Hidden Markov Models* considered in Part IV, however, such functions from priors to hyper-distributions do not have that property, because they are (at least potentially) modeling state *updates* as well as leakage. Thus they are strictly more general.

4.6 A first look at channel compositions

So far we have viewed channels as being *monolithic*. But, from the perspective of *modularity*, it is valuable to consider channels that are *composed* in some way from smaller channels. We briefly consider some channel compositions here, postponing a more detailed discussion until Chap. 8. Further, in Part IV, we show how to compose channels not only with themselves but with *programs*, thus building larger systems that not only (possibly) leak their state but also change it.

4.6.1 Convex combinations of channels

We begin with some general remarks about convex combinations. A convex combination is the averaging of a set of elements using coefficients c_i that are non-negative and sum to one. Such combinations are natural for most objects that we study; for instance the convex combination of distributions δ^i produces a new distribution $\delta = \sum_i c_i \delta^i$, as we saw earlier in Cor. 4.8 where we took a convex combination of a hyper's inners, with coefficients given by the hyper's outer, to recover the prior that generated the hyper.

Hypers, in $\mathbb{D}^2\mathcal{X}$, are thus distributions over elements in $\mathbb{D}\mathcal{X}$ — and hence they too support convex combination. For example $\Delta = \sum_i c_i \Delta^i$ is a new hyper, assigning to each inner δ the new outer probability $\Delta_\delta = \sum_i c_i \Delta_\delta^i$. Note that *only the outer probabilities* are affected by that operation; the inners of Δ are simply the *union* of the inners of all Δ^i 's (excluding the degenerate case where some c_i is 0). The fact that hypers can be combined that way is the reason we write $[\pi \triangleright C] = \sum_i a_i [\delta^i]$ (as mentioned in Def. 4.5) to denote that $[\pi \triangleright C]$ has inners δ^i and outer probabilities a_i . Each $[\delta^i]$ is a point hyper, with a single inner δ^i , hence their convex combination with the outer probabilities a_i forms the hyper $[\pi \triangleright C]$.

As a consequence, *abstract* channels can be also constructed by convex combination of their produced hypers. Namely $C := \sum_i c_i C^i$ is the abstract channel such that $[\pi \triangleright C] = \sum_i c_i [\pi \triangleright C^i]$. Here C can be viewed as a channel that probabilistically chooses some C^i , *revealing* which one was chosen, and then behaves like C^i . That operation is called *external choice* and it is studied further in Chap. 8.

To see external choice more intuitively, imagine a scenario where the adversary probabilistically gets a report from one of two spies: Boris (with probability p) or Natasha (with probability $1-p$).¹⁵ The reports are *handwritten*, so that the adversary can tell who was the author. Hence, writing B and N for Boris's and Natasha's abstract channels, the hyper-distribution that the adversary gets is the convex combination $p[\pi \triangleright B] + (1-p)[\pi \triangleright N]$.

Notice that external choice can also be done on the *channel matrices* B and N , provided that we rename output labels as necessary to make the output sets of B and N *disjoint* (corresponding to distinguishable handwriting). Then the set of columns of the resulting channel matrix $pB + (1-p)N$ is simply the *union* of the columns of B and N , and the entries in columns from B are scaled with p while those from N are scaled with $1-p$.

But now suppose that Boris and Natasha write their reports on a *typewriter*, making them indistinguishable, and moreover that they use the very same set \mathcal{Y} of possible outputs. That corresponds to *another* way of combining channel matrices, again convex: if C^i are channel matrices from \mathcal{X} to \mathcal{Y} , then $C = \sum_i c_i C^i$ is simply the convex combination of the corresponding matrices. (Note how that differs from external choice on abstract channels, where each C^i could have had its *own* \mathcal{Y}^i when formulated

¹⁵ Those characters are the two canonical spies from the television series “The Adventures of Rocky and Bullwinkle”. We relax our gender convention here, to allow Boris to be male.

as a channel matrix C^i .) Here C can be viewed as a channel that probabilistically chooses some C^i , *without revealing* which one was chosen, and then behaves like C^i . That operation is called *internal choice* in Chapters 8 and 17, and it is fundamentally different from external choice.¹⁶

The effect of internal choice depends critically on the “accidental” detail of how X is correlated with the *values* in Y for each C^i . In particular, *permuting* the output labels of some C^i has *no effect* on the abstract channel that it denotes, but it could greatly affect the resulting internal choice.

To see that, suppose for example that Boris and Natasha use the following channel matrices:

B	y_1	y_2
x_1	1	0
x_2	0	1

N	y_1	y_2
x_1	0	1
x_2	1	0

Although each of those channel matrices leaks X completely, both denoting the leak-everything channel O , their output labels have opposite meanings. As a result, their convex combination $\frac{1}{2}B + \frac{1}{2}N$ as matrices leaks *nothing*. (It is the matrix with $\frac{1}{2}$ in every position, which reduces to the single-column matrix of all 1’s, i.e. the channel 1). Yet, if we take the convex combination of the corresponding abstract channels, we have $B = N$, and hence $C = \frac{1}{2}B + \frac{1}{2}N$ also leaks X completely. Moreover, *swapping* the output labels y_1 and y_2 in B has no effect on B , but it causes the convex combination $\frac{1}{2}B + \frac{1}{2}N$ also to leak X completely.

Note finally that internal choice cannot even be *defined* on abstract channels, because (since they have abstracted from the output labels) there is no way to know which inners should be combined.

4.6.2 Cascading and the Data-Processing Inequality

We conclude with a discussion of an important and natural way of composing channel matrices known as *cascading*.

Definition 4.18 (Cascade) —

Given channel matrices $C: \mathcal{X} \rightarrow \mathcal{Y}$ and $D: \mathcal{Y} \rightarrow \mathcal{Z}$, the *cascade* of C and D is the channel matrix CD of type $\mathcal{X} \rightarrow \mathcal{Z}$, where CD is given by ordinary matrix multiplication.

Informally, we can see C as specifying the correlation $p(y|x)$ between X and Y , and D as specifying the correlation $p(z|y)$ between Y and Z ; the cascade CD is then the resulting correlation $p(z|x)$ between X and Z .

More precisely, given prior $\pi: \mathbb{D}\mathcal{X}$ and channel matrices C and D , the joint distribution

$$p_{XYZ}(x, y, z) := \pi_x C_{x,y} D_{y,z}$$

can be proven to be the *unique* joint distribution satisfying the following four conditions, where (as usual) we assume that the relevant denominators are nonzero and we omit the subscripts from p whenever they are unambiguous:

1. p_{XYZ} recovers π , so that $p(x) = \pi_x$,
2. p_{XYZ} recovers C , so that $p(y|x) = C_{x,y}$,
3. p_{XYZ} recovers D , so that $p(z|y) = D_{y,z}$, and
4. $p(z|x, y)$ does *not* depend on x , so that $p(z|x, y) = p(z|y)$.

¹⁶ It is further illustrated by Warner’s protocol in §8.1.5, and the spies Ms. Vowel and Mrs. Early using a radio at (17.3) in §17.4.

Condition 4 captures the fact that D has access only to Y , so its behavior cannot be affected by X .¹⁷

Our above informal statement about cascading can now be made precise:

$$p(z|x) = (\text{CD})_{x,z} .$$

We mention here that this property will be used in §10.1; it is justified as follows.

$$p(z|x) = \frac{p(x,z)}{p(x)} = \frac{\sum_y p(x,y,z)}{p(x)} = \frac{\sum_y \pi_x C_{x,y} D_{y,z}}{\pi_x} = (\text{CD})_{x,z} .$$

Let us now consider cascade CD from the perspective of *information leakage*. Here D can best be understood as a *sanitization policy* that specifies how the output labels \mathcal{Y} of C should be mapped to the output labels \mathcal{Z} of D . The idea is that CD *suppresses* the release of Y (which would maybe leak too much about X) and instead releases the “sanitized” Z , thereby (perhaps) leaking less. Notice that the *weakest* possible sanitization policy is the channel O that leaks everything, so that $CO = C$, while the *strongest* policy is the channel $\mathbf{1}$ that leaks nothing, so that $C\mathbf{1} = \mathbf{1}$.¹⁸

In fact we will see later that for any “sanitation policy” D the leakage can *never* be increased. That is known as the *Data-Processing Inequality*, and we will return to it at Thm. 9.11 below. But it should be obvious that different sanitization policies could be more- or less effective at limiting the leakage caused by C .

And here we run into the issue that the effectiveness of a sanitization policy D depends critically on the “accidental” detail of how, precisely, X is correlated with Y under C . In particular, if we *permute* the output labels \mathcal{Y} , it has no effect at all on the leakage caused by C –that is, C still denotes the same abstract channel– but it obviously could have a huge impact on the leakage caused by the sanitized channel CD .

Moreover we note that cascading cannot even be *defined* on the abstract channels C and D denoted by \mathbf{C} and \mathbf{D} , because C , which has type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$, has abstracted from \mathcal{Y} and hence has lost the specific “wiring” needed to connect the outputs of C to the inputs of D .

Thus we find that cascading, while very natural on channel matrices, sits uneasily with respect to the abstract-channel perspective. We will see nevertheless, in Part IV, that it *is* possible to define cascade at the abstract level, i.e. using $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$, when we consider mechanisms that can *update* secrets rather than merely leak them (§14.4.8).

A detailed discussion of channel compositions, on both channel matrices and abstract channels, is given in Chap. 8.

4.7 Exercises

Exercise 4.1 Compute the hyper $[\vartheta \triangleright C]$ when $\vartheta = (1/4, 1/4, 1/4, 1/4)$ and C is

C	y_1	y_2	y_3	y_4
x_1	$1/2$	$1/2$	0	0
x_2	0	0	1	0
x_3	$1/2$	$1/4$	0	$1/4$
x_4	$1/8$	$1/8$	$1/4$	$1/2$

□

¹⁷ Interestingly, without condition 4 it turns out that p_{XYZ} is *not* uniquely determined.

¹⁸ The names O and $\mathbf{1}$ are chosen not because of their matrix-multiplication properties, clearly: for here they behave exactly the opposite of what one would expect. However (as mentioned on p. 60) for parallel composition of channels, indeed $\mathbf{1}$ is the identity and O is the zero.

Exercise 4.2 A password checker tests whether a guessed password is correct or not, outputting “accept” or “reject”. This can be modeled as a *family* of channel matrices C^g , parameterized by the guess g , and whose secret input X is the correct password.¹⁹ For instance, with 3-bit passwords and guess 110, we get the following channel matrix:

C^{110}	reject	accept
000	1	0
001	1	0
010	1	0
011	1	0
100	1	0
101	1	0
110	0	1
111	1	0

Channel matrix C^g models the unavoidable information leakage inherent in password checking. (Recall the remarks about **Access Denied**, in the Preface on p. vii.) But an *implementation* of the checker might work by comparing the guess and the correct password bit by bit, and rejecting as soon as a mismatch is found.²⁰ In that case, the running time of the implementation is proportional to the length of the maximum correct prefix of the guess, resulting in a *timing side-channel*. Assuming that the adversary can observe that time precisely, the implementation is then more accurately modeled as a family of channels D^g whose set of possible outputs (still assuming 3-bit passwords) is $\{(reject, 1), (reject, 2), (reject, 3), (accept)\}$, reflecting the fact that the first mismatch can occur at the first, second, or third bit of g .

- (a) Show the channel matrix D^{110} .
- (b) Compute the two hyper-distributions $[\vartheta \triangleright C^{110}]$ and $[\vartheta \triangleright D^{110}]$ for the uniform prior $\vartheta = (1/8, 1/8, 1/8, 1/8, 1/8, 1/8, 1/8, 1/8)$.

□

Exercise 4.3 Prove Thm. 4.3 rigorously, with careful calculational steps and using the $p()$ notation.

□

Exercise 4.4 Suppose that C is a channel matrix whose rows are all the same. What does its reduced matrix C^r look like? What abstract channel does it denote?

□

4.8 Chapter notes

The quote from Alfréd Rényi in Footnote 2 on p. 50 comes from Rényi [16, p. 35]. Auguste Kerckhoffs’ 1883 article “La cryptographie militaire” [10], mentioned in Footnote 5, is currently available at

http://www.petitcolas.net/kerckhoffs/crypto_militaire_1.pdf.

The quote from Jean-Yves Girard in §4.4 (p. 58) comes from a discussion of *sense* versus *denotation* in logic, in Girard, Taylor, and Lafont [8, Section 1.1]:

¹⁹ To clarify, the *correct password* X is the secret input to the checker; it is input when that password is created. The *guess* g is *not* a secret input to the checker, but instead a *parameter* that selects which channel matrix in the family is to be used.

²⁰ See §16.7 for three password-checker implementations written as *programs* in the language developed in Part IV.

On the other hand, it is impossible to say “forget completely the denotation and concentrate on the sense”, for the simple reason that the sense contains the denotation, at least implicitly. So it is not a matter of symmetry. In fact there is hardly any unified syntactic point of view, because we have never been able to give an operational meaning to this mysterious sense. The only tangible reality about sense is the way it is written, the formalism; but the formalism remains an unaccommodating object of study, without true structure, a piece of *soft camembert*.

Channel matrices are a basic object of study in the area of information theory, and many interesting examples can be seen in textbooks such as that of Cover and Thomas [5]. Examples of the use of channel matrices in quantitative information flow include works by Köpf and Basin [11] and Chatzikokolakis et al. [3]. Modeling of interactive systems using the richer model of channels with memory and feedback is considered by Alvim et al. [1].

The fundamental principle that (from the perspective of information flow) channels should be seen as mappings from priors to hyper-distributions was, we believe, first formulated clearly by McIver, Meinicke, and Morgan [13]. Abstract channels were first proposed and studied by McIver et al. [14]. The characterization of cascading presented in §4.6.2 is from Espinoza and Smith [7].

Early work in information flow was usually non-quantitative, in the sense that it aimed to prevent *any* leakage of sensitive information. In the setting of programs taking “high” and “low” inputs and producing “high” and “low” outputs, the goal is to prevent information from flowing from “high” inputs to “low” outputs. Important examples include the work of Cohen [4], who called leakage “strong dependency”, the Dennings [6], Volpano et al. [18, 17], and Bhargava and Palamidessi [2]. (It should be noted however that the definition of “no leakage” is quite subtle in contexts allowing nontermination and concurrency.) There is a related but substantially different body of work involving instead high and low *users* interacting with a system; there the question is whether the high users can communicate information to the low users. Important works in that setting include that of Goguen and Meseguer [9], McCullough [12], Wittbold and Johnson [19], and McLean [15]. In particular, Goguen and Meseguer defined “noninterference” to mean that *purging* all high user actions has no effect on what the low users observe. Nevertheless, the name “noninterference” has come to be the standard name for the “no leakage” property, in any setting.

Bibliography

- [1] Alvim, M.S., Andrés, M.E., Palamidessi, C.: Quantitative information flow in interactive systems. *Journal of Computer Security* **20**(1), 3–50 (2012)
- [2] Bhargava, M., Palamidessi, C.: Probabilistic anonymity. In: M. Abadi, L. de Alfaro (eds.) *Proceedings of the 16th International Conference on Concurrency Theory (CONCUR 2005)*, *Lecture Notes in Computer Science*, vol. 3653, pp. 171–185. Springer, Berlin (2005)
- [3] Chatzikokolakis, K., Palamidessi, C., Panangaden, P.: Anonymity protocols as noisy channels. *Information and Computation* **206**(2-4), 378–401 (2008)
- [4] Cohen, E.: Information transmission in computational systems. In: *Proceedings of the 6th ACM Symposium on Operating Systems Principles (SOSP 1977)*, pp. 133–139. ACM, New York (1977)
- [5] Cover, T.M., Thomas, J.A.: *Elements of Information Theory*. Wiley Series in Telecommunications and Signal Processing. Wiley-Interscience (2006)
- [6] Denning, D.E., Denning, P.J.: Certification of programs for secure information flow. *Communications of the ACM* **20**(7), 504–513 (1977)
- [7] Espinoza, B., Smith, G.: Min-entropy leakage of channels in cascade. In: G. Barthe, A. Datta, S. Etalle (eds.) *Proceedings of the 7th International Workshop on Formal Aspects of Security and Trust (FAST 2011)*, *Lecture Notes in Computer Science*, vol. 7140, pp. 70–84. Springer, Berlin (2012)
- [8] Girard, J.Y., Taylor, P., Lafont, Y.: *Proofs and Types*. Cambridge University Press, New York (1989)
- [9] Goguen, J.A., Meseguer, J.: Security policies and security models. In: *Proceedings of the 1982 IEEE Symposium on Security and Privacy*, pp. 11–20. IEEE, Los Alamitos (1982)
- [10] Kerckhoffs, A.: La cryptographie militaire. *Journal des Sciences Militaires* **9**, 5–38 (1883)
- [11] Köpf, B., Basin, D.: An information-theoretic model for adaptive side-channel attacks. In: *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS 2007)*, pp. 286–296. ACM, New York (2007)
- [12] McCullough, D.: Noninterference and the composability of security properties. In: *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, pp. 177–186. IEEE, Los Alamitos (1988)

Bibliography

- [13] McIver, A., Meinicke, L., Morgan, C.: Compositional closure for Bayes risk in probabilistic noninterference. In: S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide, P.G. Spirakis (eds.) Proceedings of the 37th International Colloquium on Automata, Languages, and Programming (ICALP 2010), *Lecture Notes in Computer Science*, vol. 6199, pp. 223–235. Springer, Berlin (2010)
- [14] McIver, A., Morgan, C., Smith, G., Espinoza, B., Meinicke, L.: Abstract channels and their robust information-leakage ordering. In: M. Abadi, S. Kremer (eds.) Proceedings of the 3rd Conference on Principles of Security and Trust (POST 2014), *Lecture Notes in Computer Science*, vol. 8414, pp. 83–102. Springer, Berlin (2014)
- [15] McLean, J.: Security models and information flow. In: Proceedings of the 1990 IEEE Symposium on Security and Privacy, pp. 180–187. IEEE, Los Alamitos (1990)
- [16] Rényi, A.: Foundations of Probability. Holden-Day (1970)
- [17] Smith, G., Volpano, D.M.: Secure information flow in a multi-threaded imperative language. In: Proceedings of the 25th ACM Symposium on Principles of Programming Languages (POPL 2008), pp. 355–364. ACM, New York (1998)
- [18] Volpano, D.M., Irvine, C.E., Smith, G.: A sound type system for secure flow analysis. *Journal of Computer Security* **4**(2-3), 167–187 (1996)
- [19] Wittbold, J.T., Johnson, D.M.: Information flow in nondeterministic systems. In: Proceedings of the 1990 IEEE Symposium on Security and Privacy, pp. 144–161. IEEE, Los Alamitos (1990)



Chapter 5

Posterior vulnerability and leakage

We recall from Chap. 3 that, given a secret X with prior distribution π , the “threat” to X can be measured as its g -vulnerability $V_g(\pi)$ for a suitably chosen gain function g — where the choice of g reflects the circumstances, and the importance to us of the secret’s *remaining* undisclosed, and might vary depending on who “we” are and what adversary we are facing. Suppose therefore that there is a channel matrix $C: \mathcal{X} \rightarrow \mathcal{Y}$, which might leak some information about X . In this chapter we consider how to quantify the g -vulnerability of X after C is run.

5.1 Posterior g -vulnerability and its basic properties

There are two perspectives we can take on “after C is run”.

One perspective, which we call *dynamic*, considers the effect of the adversary’s observing a *particular* channel output y . As discussed in §4.2, that allows the adversary to update her knowledge about X , changing it from the prior π to the posterior distribution $p_{X|y}$. If we were to focus for simplicity on Bayes vulnerability, then it would be natural to say that this run of C has *changed* the Bayes vulnerability (to the adversary) from $V_1(\pi)$ to $V_1(p_{X|y})$. And then we might measure the amount of leakage as the difference $V_1(p_{X|y}) - V_1(\pi)$.

While the dynamic view of leakage is natural, it does suffer from some significant drawbacks. For one thing, under the dynamic view it turns out that Bayes vulnerability can actually *decrease* as a result of an observation, as we illustrate in this example.

Example 5.1 Consider the following channel matrix C :

C	y_1	y_2
x_1	1	0
x_2	0	1
x_3	0	1
x_4	0	1
x_5	0	1

Suppose that the prior π is $(9/10, 1/40, 1/40, 1/40, 1/40)$, whose Bayes vulnerability $V_1(\pi)$ clearly is $9/10$. If output y_2 is observed, however, the posterior distribution $p_{X|y_2}$ is $(0, 1/4, 1/4, 1/4, 1/4)$, which means that the Bayes vulnerability has *decreased* from $9/10$ to $1/4$. Hence we would have the counter-intuitive situation of a *negative leakage* of $1/4 - 9/10 = -13/20$. Running the channel seems to have *decreased* the threat posed by the adversary. \square

A real-world scenario corresponding to this example is the case of a doctor's trying to diagnose an unknown disease. Based on the symptoms, there might be only one likely diagnosis, making the prior "vulnerability" large. But if a medical test refutes that diagnosis, then the doctor is left with no idea of the disease, making the posterior "vulnerability" small. (Here again we have a benevolent adversary.)

An additional issue for the dynamic approach concerns policy enforcement. We might imagine using an execution monitor to track the Bayes vulnerability of the secret, verifying that it stays below some threshold. But if a run produces an output that leaks too much, what could the monitor do? It might try to respond by aborting the execution, but the very act of aborting might in itself leak a lot of information to the adversary. For example, in the case of a password checker, aborting the execution when the adversary enters the correct password would reveal the entire password to the adversary (and also makes the password checker useless).

Thus we consider a different perspective: the *static* view considers *all* the possible outputs of a channel C , independent of any particular execution.

From the static perspective, we have seen that a channel C maps the prior π to a hyper-distribution $[\pi \triangleright C]$, which represents all the possible "worlds" of adversary knowledge about X (the *inners*), together with their probabilities (the *outer*). It is natural to calculate the g -vulnerability of each of the inners, to see how vulnerable the secret is in that "world". But how should that set of vulnerabilities –they are just numbers– be combined?

One possibility is to consider the *maximum* of those vulnerabilities, as this represents the maximum threat to the secret. (That is, we statically compute the worst "world" that could occur in the dynamic perspective.) That approach is quite pessimistic, however, because it pays no attention to the *outer* probability of the worst inner, which may indeed be highly unlikely. Consider the case of a password checker when the user has tried to log in with some guess g . There are two possible outputs: *accept* and *reject*, depending on whether g is correct or not. But notice that in the case of output *accept*, the posterior distribution is a point distribution on g , which means that the max-case posterior Bayes vulnerability is $V_1(p_{X|accept}) = 1$. Hence a password checker is judged to be as bad as a channel that leaks the password completely.

We therefore find it more useful to define posterior g -vulnerability as the *expected value of the g -vulnerability over the hyper-distribution*, meaning that the g -vulnerabilities of the inner distributions are weighted with their outer probabilities.

Definition 5.2

Given prior π , gain function $g: \mathbb{G}\mathcal{X}$ and channel C , the *posterior g -vulnerability* $V_g[\pi \triangleright C]$ is defined as the expected value of V_g over $[\pi \triangleright C]$, that is

$$V_g[\pi \triangleright C] := \sum_i a_i V_g(\delta^i), \quad \text{where} \quad [\pi \triangleright C] = \sum_i a_i [\delta^i].$$

(We are thus *overloading* V_g , allowing it to take as argument either a *distribution* in $\mathbb{D}\mathcal{X}$ or a *hyper-distribution* in $\mathbb{D}^2\mathcal{X}$, in the latter case omitting function-application parentheses if the argument has brackets already.)

Recalling that $\mathcal{E}_\alpha F$ denotes the expected value of some function $F: \mathcal{A} \rightarrow \mathbb{R}$ over a distribution $\alpha: \mathbb{D}\mathcal{A}$, we see that with Def. 5.2 the posterior g -vulnerability can be written as $V_g[\pi \triangleright C] = \mathcal{E}_{[\pi \triangleright C]} V_g$. Note that the definition depends only on the mapping that C gives from priors to hyper-distributions, meaning that it is *well defined* on abstract channels.

To understand posterior g -vulnerability better, let us look at some examples. We start with an example of posterior *Bayes* vulnerability specifically (which of course is the same as posterior g_{id} -vulnerability).

Example 5.3 Recall that Example 4.2 presents a channel C that maps prior distribution $\pi = (1/4, 1/2, 1/4)$ to hyper-distribution

$[\pi \triangleright C]$	$1/4$	$1/3$	$7/24$	$1/8$.
x_1	$1/2$	$3/8$	0	0	
x_2	0	$3/8$	$6/7$	1	
x_3	$1/2$	$1/4$	$1/7$	0	

(5.1)

In this case the prior Bayes vulnerability is $1/2$, as the adversary's best action *a priori* is to guess x_2 . From (5.1) we find that the posterior Bayes vulnerability is

$$\begin{aligned} V_1[\pi \triangleright C] &= 1/4 \cdot V_1(1/2, 0, 1/2) + 1/3 \cdot V_1(3/8, 3/8, 1/4) + 7/24 \cdot V_1(0, 6/7, 1/7) + 1/8 \cdot V_1(0, 1, 0) \\ &= 1/4 \cdot 1/2 + 1/3 \cdot 3/8 + 7/24 \cdot 6/7 + 1/8 \cdot 1 \\ &= 5/8 , \end{aligned}$$

which is larger than the prior Bayes vulnerability; and that reflects the fact that the channel outputs help the adversary to choose actions that are better for her. In particular, the first posterior distribution in $[\pi \triangleright C]$ directs her not to guess x_2 but instead x_1 or x_3 , which are equally good. (On the other three posterior distributions, guessing x_2 remains optimal.) Thus her probability of guessing X correctly increases from $1/2$ to $5/8$. \square

Example 5.4 Next let us consider posterior g -vulnerability for the gain function g from Example 3.3, whose matrix representation is

G	x_1	x_2	.
w_1	-1.0	1.0	
w_2	0.0	0.5	
w_3	0.4	0.1	
w_4	0.8	-0.9	
w_5	0.1	0.2	

and suppose that channel C is

C	y_1	y_2	.
x_1	0.75	0.25	
x_2	0.25	0.75	

Notice that C reveals rather a lot of information about the secret: if the input is x_i (where i is 1 or 2), then three-fourths of the time the output is y_i .

Now let us compute $V_g[\pi \triangleright C]$ directly from Def. 5.2 in the case when $\pi = (0.3, 0.7)$. We first find hyper-distribution $[\pi \triangleright C]$ using the process shown earlier in Example 4.2. Scaling the rows of C with π , we get joint matrix J as here:

J	y_1	y_2
x_1	0.225	0.075
x_2	0.175	0.525

5 Posterior vulnerability and leakage

And by summing and normalizing the columns of J , we find that $[\pi \triangleright C]$ has inners $\delta^1 = (0.5625, 0.4375)$ and $\delta^2 = (0.125, 0.875)$, with outer probabilities 0.4 and 0.6, respectively.

Now we must compute $V_g(\delta^1)$ and $V_g(\delta^2)$. For each inner, this requires computing the expected gain for each possible w in \mathcal{W} , to see which action is best. For δ^1 we get the following results:

$$\begin{aligned}\delta_{x_1}^1 g(w_1, x_1) + \delta_{x_2}^1 g(w_1, x_2) &= 0.5625 \cdot (-1.0) + 0.4375 \cdot 1.0 = -0.12500 \\ \delta_{x_1}^1 g(w_2, x_1) + \delta_{x_2}^1 g(w_2, x_2) &= 0.5625 \cdot 0.0 + 0.4375 \cdot 0.5 = 0.21875 \\ \delta_{x_1}^1 g(w_3, x_1) + \delta_{x_2}^1 g(w_3, x_2) &= 0.5625 \cdot 0.4 + 0.4375 \cdot 0.1 = 0.26875 \\ \delta_{x_1}^1 g(w_4, x_1) + \delta_{x_2}^1 g(w_4, x_2) &= 0.5625 \cdot 0.8 + 0.4375 \cdot (-0.9) = 0.05625 \\ \delta_{x_1}^1 g(w_5, x_1) + \delta_{x_2}^1 g(w_5, x_2) &= 0.5625 \cdot 0.1 + 0.4375 \cdot 0.2 = 0.14375\end{aligned}$$

Here we find that w_3 is the best action and $V_g(\delta^1) = 0.26875$. For δ^2 we get

$$\begin{aligned}\delta_{x_1}^2 g(w_1, x_1) + \delta_{x_2}^2 g(w_1, x_2) &= 0.125 \cdot (-1.0) + 0.875 \cdot 1.0 = 0.7500 \\ \delta_{x_1}^2 g(w_2, x_1) + \delta_{x_2}^2 g(w_2, x_2) &= 0.125 \cdot 0.0 + 0.875 \cdot 0.5 = 0.4375 \\ \delta_{x_1}^2 g(w_3, x_1) + \delta_{x_2}^2 g(w_3, x_2) &= 0.125 \cdot 0.4 + 0.875 \cdot 0.1 = 0.1375 \\ \delta_{x_1}^2 g(w_4, x_1) + \delta_{x_2}^2 g(w_4, x_2) &= 0.125 \cdot 0.8 + 0.875 \cdot (-0.9) = -0.6875 \\ \delta_{x_1}^2 g(w_5, x_1) + \delta_{x_2}^2 g(w_5, x_2) &= 0.125 \cdot 0.1 + 0.875 \cdot 0.2 = 0.1875\end{aligned}$$

Here we find instead that w_1 is the best action and $V_g(\delta^2) = 0.75$.

Finally, we compute $V_g[\pi \triangleright C]$ overall by weighting the g -vulnerabilities of the inners with their respective outer probabilities:

$$V_g[\pi \triangleright C] = 0.4 \cdot V_g(\delta^1) + 0.6 \cdot V_g(\delta^2) = 0.4 \cdot 0.26875 + 0.6 \cdot 0.75 = 0.5575.$$

We remark that, although the process that we have used here to compute $V_g[\pi \triangleright C]$ is conceptually clear, it is in detail quite laborious. In fact, since this vulnerability is constructed directly from a gain function, its effect $V_g[\pi \triangleright C]$ can be computed with substantially less effort, thanks to the following observation: the *normalization* needed to convert the columns of J to the inners of $[\pi \triangleright C]$ is exactly *canceled out* by the *weighting* of the g -vulnerabilities of the inners in the computation of $V_g[\pi \triangleright C]$. That observation is formalized in Thm. 5.7 below, and later given a matrix-based formulation in Thm. 5.18.¹

We conclude our example by considering the operational significance of the fact that $V_g[\pi \triangleright C] = 0.5575$. Recall that in Example 3.3 we found that $V_g(\pi) = 0.4$, which is less than 0.5575. This reflects the fact that C allows the adversary to use a better *strategy* for selecting actions: on output y_1 (corresponding to δ^1) her best action is w_3 and on output y_2 (corresponding to δ^2) her best action is w_1 . In contrast, if she knew only π , her only reasonable strategy would be to choose action w_1 every time. \square

¹ However with vulnerabilities and uncertainties for which a reasonable g/ℓ -function is not available, such as Shannon Entropy (which needs an infinite \mathcal{W} , if it is to be expressed as U_ℓ), the short-cut offered by Thm. 5.18 does not apply: in that case one must work from the hyper directly.

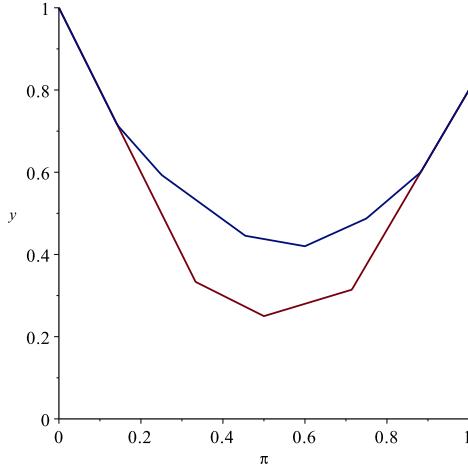


Figure 5.1 Comparison of $V_g(\pi)$ (red) and $V_g[\pi \triangleright C]$ (blue) for Example 5.5

Example 5.5 To get a fuller understanding of posterior g -vulnerability for the gain function g and channel matrix C considered in Example 5.4, let us now graph $V_g[\pi \triangleright C]$ as a function of a general prior $\pi = (x, 1-x)$, where $0 \leq x \leq 1$. Here we get the hyper-distribution

$[\pi \triangleright C]$	$\frac{1+2x}{4}$	$\frac{3-2x}{4}$
x_1	$\frac{3x}{1+2x}$	$\frac{x}{3-2x}$
x_2	$\frac{1-x}{1+2x}$	$\frac{3-3x}{3-2x}$

Figure 5.1 compares the graph of $V_g(\pi)$ (which was seen already in Fig. 3.1) and $V_g[\pi \triangleright C]$ with $[\pi \triangleright C]$ from just above. As can be seen, $V_g[\pi \triangleright C]$ is often (but not always) greater than $V_g(\pi)$. This is to be expected intuitively, since channel C increases the adversary's knowledge about X , enabling a better choice of actions. But the particular line segments in the graph of $V_g[\pi \triangleright C]$ might need some clarification.

Recall that in the prior situation, the adversary's choice of which action to take is guided only by π itself — she does not consider the channel. But in the posterior situation, her choice can be guided by both π and the output of channel C , so that she can choose one action if the output is y_1 and another if the output is y_2 . If we let $w_i w_j$ denote the strategy of choosing action w_i on output y_1 and w_j on output y_2 , then we see that she has $|\mathcal{W}|^{|\mathcal{Y}|} = 5^2 = 25$ possible strategies. Note that those of the form $w_i w_i$ are “degenerate” strategies, since they choose action w_i regardless of the output: we therefore write them simply as w_i .

Figure 5.2 plots the expected gain for seven of these strategies: they are w_1 , $w_2 w_1$, $w_3 w_1$, $w_4 w_1$, $w_4 w_3$, $w_4 w_2$, and w_4 . (Note that the graphs of the “degenerate” strategies w_1 and w_4 were seen already in Fig. 3.2.) The expected gains for the remaining $25 - 7 = 18$ strategies are not shown, because each of them is *dominated* by the other seven, in the sense that on any π at least one of the seven gives an expected gain that is at least as large. (Recall that in Fig. 3.2 the strategy w_5 is dominated in the same way.)

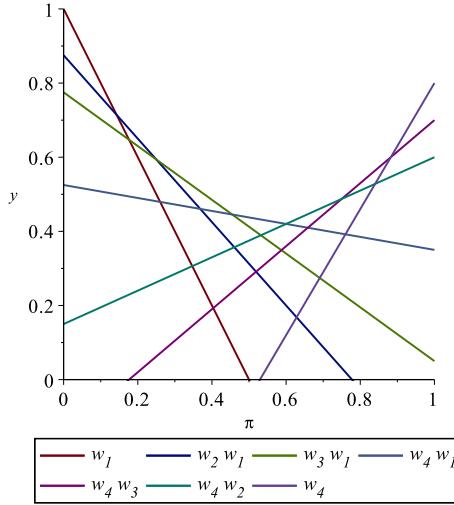


Figure 5.2 Graph of the expected gains from seven strategies for Example 5.5

Comparing Fig. 5.1 and Fig. 5.2, we see that $V_g[\pi \triangleright C]$ lies along the *maximum* of those seven strategies, each of which turns out to be optimal for certain π . It can be considered to be the envelope (from above) of all the individual strategy lines and, in general, the envelope of hyperplanes. (For uncertainties we would take the envelope from below.) In particular, notice that strategies w_1 and w_4 are optimal in the portions of the graph where $V_g(\pi) = V_g[\pi \triangleright C]$, which makes sense because if the optimal strategy for π ignores the output of C , then C is of no help to the adversary on that π .

Finally, it is interesting to compare the posterior g -vulnerability of C with that of the channel that leaks *everything*, which we have called O (p. 60). Expressed concretely, and reduced, the leak-everything O is the identity matrix

O	x_1	x_2
x_1	1	0
x_2	0	1

which copies its input directly to its output. Figure 5.3 compares the graphs of $V_g(\pi)$ and $V_g[\pi \triangleright C]$ and $V_g[\pi \triangleright O]$.

The reason that the graph of $V_g[\pi \triangleright O]$ has the shape it does is that the output of O reveals whether the secret is x_1 (in which case the adversary should take action w_4 , giving gain 0.8) or x_2 (in which case she should take action w_1 , giving gain 1.0). Hence on prior $(x, 1-x)$ her expected gain is

$$x \cdot 0.8 + (1-x) \cdot 1.0 = 1 - 0.2 \cdot x .$$

Observe that in Fig. 5.3 we have for every π that $V_g(\pi) \leq V_g[\pi \triangleright C] \leq V_g[\pi \triangleright O]$. Those inequalities turn out to hold in general, as we will prove in Thm. 5.8 and Example 5.26 below. In the meantime, however, it is interesting to recall (from our definitions) what $V_g(\pi)$ and $V_g[\pi \triangleright O]$ actually mean. In both cases, the gain function g determines for the adversary the value (to her) of taking a particular action. However in the case of

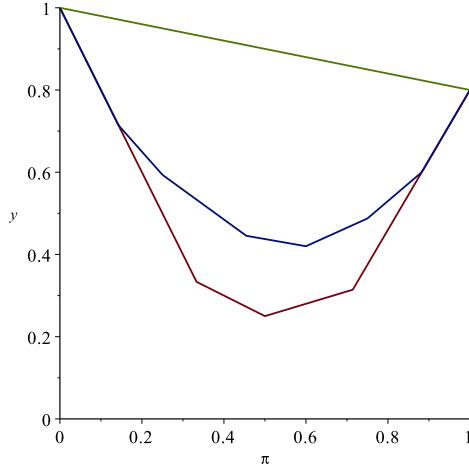


Figure 5.3 Comparison of $V_g(\pi)$ (red) and $V_g[\pi \triangleright C]$ (blue) and $V_g[\pi \triangleright O]$ (green) for Example 5.5

$V_g(\pi)$, as we know, she chooses the action that has the greatest expected value over the prior π — she has to take the *expected* value, because she does not know what the actual input value is. But in the case of $V_g[\pi \triangleright O]$, she *does* know what the input is (or was) — and so her strategy is to take the action that gives the greatest *actual* value for that input. This is reminiscent of the dynamic view of leakage, mentioned above. But still an expected value must be taken — because she does not know beforehand *which* of X 's values she will turn out to have learned from channel O . That expected value is taken in effect over the prior — because in the case of O the prior and the outer are *isomorphic*.² \square

In the remainder of this section, we establish some important properties satisfied by posterior g -vulnerability. To begin with, recall that Def. 5.2 defines $V_g[\pi \triangleright C]$ in terms of the hyper-distribution $[\pi \triangleright C]$. But it is also useful to establish formulae that allow convenient calculation of posterior g -vulnerability directly from a channel matrix $C: \mathcal{X} \rightarrow \mathcal{Y}$ that realizes C . We first show that $V_g[\pi \triangleright C]$ can be characterized in terms of the posterior distributions $p_{X|y}$ for $y \in \mathcal{Y}$. (Recall that $p_{X|y}$ is an instance of the “ p ” notation used in §4.2, and that the $p_{X|y}$'s are closely related to the inners of the hyper $[\pi \triangleright C]$.)

Theorem 5.6 Given prior π , gain function $g: \mathbb{G}\mathcal{X}$, and channel matrix C from \mathcal{X} to \mathcal{Y} , we have

$$V_g[\pi \triangleright C] = \sum_{\substack{y \in \mathcal{Y} \\ p(y) \neq 0}} p(y) V_g(p_{X|y}) .$$

² This is the static view again. Also, by *isomorphic* distributions we mean those whose base type can be placed in 1-1 correspondence in a way that preserves the structure of the distribution.

Proof. Recall that the support of hyper-distribution $[\pi \triangleright C]$ consists of the posterior distributions $p_{X|y}$ such that $p(y) \neq 0$. But recall that if outputs y_1, \dots, y_k should give rise to the *same* posterior distribution,

$$p_{X|y_1} = \dots = p_{X|y_k},$$

then the hyper-distribution coalesces them into a single inner distribution $p_{X|y_1}$ with outer probability $p(y_1) + \dots + p(y_k)$. Since then we have

$$(p(y_1) + \dots + p(y_k))V_g(p_{X|y_1}) = p(y_1)V_g(p_{X|y_1}) + \dots + p(y_k)V_g(p_{X|y_k}),$$

the desired equality follows. \square

The next theorem (whose proof is left as Exercise 5.1) shows that $V_g[\pi \triangleright C]$ can be calculated directly from π and C and g , with no need to calculate the posterior distributions.

Theorem 5.7 Given prior π , gain function $g: \mathbb{G}^{\text{fin}}\mathcal{X}$, and channel matrix C from \mathcal{X} to \mathcal{Y} , we have

$$V_g[\pi \triangleright C] = \sum_{y \in \mathcal{Y}} \max_{w \in \mathcal{W}} \sum_{x \in \mathcal{X}} \pi_x C_{x,y} g(w, x).$$

For generic $g: \mathbb{G}\mathcal{X}$, the max should be changed to sup. \square

Now we consider properties of the *behavior* of posterior g -vulnerability. Looking at Fig. 5.1, we see that the graph of $V_g[\pi \triangleright C]$ never falls below that of $V_g(\pi)$. That turns out to be a fundamental property of g -vulnerability.

Theorem 5.8 (Monotonicity)

Posterior g -vulnerability is always greater than or equal to prior g -vulnerability: for any prior π , channel C and gain function $g: \mathbb{G}\mathcal{X}$, we have $V_g[\pi \triangleright C] \geq V_g(\pi)$.

Proof. The property follows from two key facts: that V_g is convex (Thm. 3.13), and that the expectation of a hyper-distribution $[\pi \triangleright C]$ is the prior π (Cor. 4.8). So, letting $[\pi \triangleright C] = \sum_i a_i [\delta^i]$, we can reason as follows:

$$V_g[\pi \triangleright C] = \sum_i a_i V_g(\delta^i) \geq V_g(\sum_i a_i \delta^i) = V_g(\pi).$$

The convexity is what justifies the inequality in the middle. \square

A geometric illustration of Thm. 5.8 is given in §12.3.

We noted just above that $V_g(\pi)$ is a convex function of π . In Fig. 5.1, we see that $V_g[\pi \triangleright C]$ is *also* a convex function of π , which is another fundamental property of posterior g -vulnerability. Moreover, it turns out that for fixed π , $V_g[\pi \triangleright C]$ is convex on C for concrete channels, while $V_g[\pi \triangleright C]$ is linear on C for abstract channels. That difference comes from the fact that the convex combination of channels, discussed in §4.6.1, is a fundamentally different operation for abstract channels C than it is for abstract channels C .

Theorem 5.9 (Convexity of posterior V_g)

For any $g: \mathbb{G}\mathcal{X}$, posterior g -vulnerability is a convex function of priors: thus $V_g[\sum_i a_i \delta^i \triangleright C] \leq \sum_i a_i V_g[\delta^i \triangleright C]$.

Posterior g -vulnerability is as well a convex function of concrete channels: thus $V_g[\pi \triangleright \sum_i a_i \mathbf{C}^i] \leq \sum_i a_i V_g[\pi \triangleright \mathbf{C}^i]$.

But on abstract channels, posterior g -vulnerability is actually linear, not merely convex: thus $V_g[\pi \triangleright \sum_i a_i \mathbf{C}^i] = \sum_i a_i V_g[\pi \triangleright \mathbf{C}^i]$.

Proof. For convexity on priors, if \mathbf{C} is any channel matrix realizing C then

$$\begin{aligned}
 & V_g[\sum_i a_i \delta^i \triangleright C] \\
 = & V_g[\sum_i a_i \delta^i \triangleright \mathbf{C}] && \text{"change from } C \text{ to } \mathbf{C}" \\
 = & \sum_y \max_w \sum_x (\sum_i a_i \delta^i)_x \mathbf{C}_{x,y} g(w, x) && \text{"Thm. 5.7"} \\
 = & \sum_y \max_w \sum_x \sum_i a_i \delta^i_x \mathbf{C}_{x,y} g(w, x) && \text{"convex combination of distributions"} \\
 = & \sum_y \max_w \sum_i a_i \sum_x \delta^i_x \mathbf{C}_{x,y} g(w, x) && \text{"reorganizing sum"} \\
 \leq & \sum_y \sum_i a_i \max_w \sum_x \delta^i_x \mathbf{C}_{x,y} g(w, x) && \text{"maximum of sum } \leq \text{ sum of maximums"} \\
 = & \sum_i a_i \sum_y \max_w \sum_x \delta^i_x \mathbf{C}_{x,y} g(w, x) && \text{"reorganizing sum"} \\
 = & \sum_i a_i V_g[\delta^i \triangleright \mathbf{C}] && \text{"Thm. 5.7"} \\
 = & \sum_i a_i V_g[\delta^i \triangleright C] .
 \end{aligned}$$

For generic $g: \mathbb{G}\mathcal{X}$, the max's should be changed to sup's.

Similar arguments prove convexity on concrete channels (for which we appeal to $(\sum_i a_i \mathbf{C}^i)_{x,y} = \sum_i a_i \mathbf{C}_{x,y}^i$) and linearity on abstract channels (appealing instead to $[\pi \triangleright \sum_i a_i \mathbf{C}^i] = \sum_i a_i [\pi \triangleright \mathbf{C}^i]$). \square

Finally, we show that the *gain-function–algebra* properties that were proved in Thm. 3.14 (p. 42) also carry over to posterior g -vulnerability.

Theorem 5.10 For any $g: \mathbb{G}\mathcal{X}$, prior π , channel C , $k \geq 0$ and $\kappa \in \mathbb{R}^{|\mathcal{X}|}$, we have

$$\begin{aligned}
 V_{g \times k}[\pi \triangleright C] &= k V_g[\pi \triangleright C] , \\
 V_{g+\kappa}[\pi \triangleright C] &= V_g[\pi \triangleright C] + \kappa \cdot \pi .
 \end{aligned}$$

Proof. These follow quite easily from Thm. 3.14. Letting $[\pi \triangleright C] = \sum_i a_i [\delta^i]$, we can reason

$$\begin{aligned}
 & V_{g \times k}[\pi \triangleright C] \\
 = & \sum_i a_i V_{g \times k}(\delta^i) \\
 = & \sum_i a_i k V_g(\delta^i) && \text{"Thm. 3.14"} \\
 = & k \sum_i a_i V_g(\delta^i) \\
 = & k V_g[\pi \triangleright C] ,
 \end{aligned}$$

and similarly for $V_{g+\kappa}[\pi \triangleright C]$. \square

5.2 Multiplicative and additive g -leakage

Now we are ready to define g -leakage. It is natural to quantify leakage in terms of the prior and posterior g -vulnerabilities $V_g(\pi)$ and $V_g[\pi \triangleright C]$. In fact the comparison can be done either “multiplicatively” (focusing on the *relative* difference) or “additively” (focusing on the *absolute* difference).

Definition 5.11 (g -leakage)

Given prior distribution π , gain function $g: \mathbb{G}\mathcal{X}$, and channel C , the *multiplicative g -leakage* is

$$\mathcal{L}_g^{\times}(\pi, C) := \frac{V_g[\pi \triangleright C]}{V_g(\pi)} ,$$

and the *additive g -leakage* is

$$\mathcal{L}_g^{+}(\pi, C) := V_g[\pi \triangleright C] - V_g(\pi) .$$

Notice that multiplicative leakage does not really make sense if vulnerability can be both positive and negative — and that was the main motivation for our interest in non-negative vulnerabilities in §3.3.

By Thm. 5.8, the smallest possible leakage occurs when the prior- and posterior vulnerabilities are *equal* — then the *additive g -leakage is 0* and the *multiplicative g -leakage is 1*. In both those cases, we say that there is *no leakage*.

There is another issue with multiplicative g -leakage $\mathcal{L}_g^{\times}(\pi, C)$ that is more complicated: if the prior vulnerability $V_g(\pi)$ is *zero*, the quotient will be undefined. In this case we define $\mathcal{L}_g^{\times}(\pi, C)$ to be 1 if $V_g[\pi \triangleright C]$ is also 0, and $+\infty$ otherwise. But it turns out that $+\infty$ is never needed for *non-negative* gain functions ($\mathbb{G}^+\mathcal{X}$), as the following result shows.

Theorem 5.12 For any non-negative gain function $g: \mathbb{G}^+\mathcal{X}$, prior π , and abstract channel C , if $V_g(\pi) = 0$ then also $V_g[\pi \triangleright C] = 0$.

Proof. If g is a non-negative gain function and $\sum_w \pi_x g(w, x) = 0$, then for each x in the support of π we must have $g(w, x) = 0$ for every w . Since every inner of $[\pi \triangleright C]$ has support that is a subset of the support of π , it follows that the posterior g -vulnerability must also be 0. \square

Non-negative gain functions are of particular importance for multiplicative leakage; the above result shows that it is always finite for this class, while in §7.1 we see that we can further obtain tight bounds on $\mathcal{L}_g^{\times}(\pi, C)$ for $\mathbb{G}^+\mathcal{X}$.

We next observe that the gain-function–algebra Thm. 3.14 and Thm. 5.10 together immediately imply that each variant of g -leakage is *invariant* with respect to one of the gain-function operators, while possibly affected by the other.

Theorem 5.13 For all $g: \mathbb{G}\mathcal{X}$, prior π , channel C , scalar constant $k \geq 0$, and tuple $\kappa: \mathbb{R}^{|\mathcal{X}|}$, the following hold (provided of course that $g_{+\kappa}$ is in $\mathbb{G}\mathcal{X}$):

1. Multiplicative leakage is invariant under scaling but affected by shifting:

$$\begin{aligned}\mathcal{L}_{g_{\times k}}^{\times}(\pi, C) &= \mathcal{L}_g^{\times}(\pi, C), \\ \mathcal{L}_{g_{+\kappa}}^{\times}(\pi, C) &= (1-\lambda)\mathcal{L}_g^{\times}(\pi, C) + \lambda \quad \text{where} \quad \lambda := \frac{\kappa \cdot \pi}{V_{g_{+\kappa}}(\pi)}.\end{aligned}$$

2. Additive leakage is invariant under shifting but affected by scaling:

$$\begin{aligned}\mathcal{L}_{g_{+\kappa}}^{+}(\pi, C) &= \mathcal{L}_g^{+}(\pi, C), \\ \mathcal{L}_{g_{\times k}}^{+}(\pi, C) &= k \mathcal{L}_g^{+}(\pi, C).\end{aligned}$$

Proof. All cases are direct consequences of Thm. 3.14 and Thm. 5.10. We showcase here the case of shifting for multiplicative leakage, which is the most challenging. Assume that both $V_g(\pi)$ and $V_{g_{+\kappa}}(\pi)$ are nonzero, and let $a = V_g[\pi \triangleright C]$, $b = V_g(\pi)$, and $k = \kappa \cdot \pi$. We have that

$$\begin{aligned}&(1-\lambda)\mathcal{L}_g^{\times}(\pi, C) + \lambda \\ &= (1 - \frac{k}{b+k}) \frac{a}{b} + \frac{k}{b+k} \quad \text{“def. of } \mathcal{L}_g^{\times}(\pi, C), b \neq 0, b+k \neq 0\text{”} \\ &= \frac{a(b+k)}{b(b+k)} - \frac{ak}{b(b+k)} + \frac{bk}{b(b+k)} \quad \text{“algebra”} \\ &= \frac{b(a+k)}{b(b+k)} \quad \text{“algebra”} \\ &= V_{g_{+\kappa}}[\pi \triangleright C] / V_{g_{+\kappa}}(\pi) \quad \text{“Thm. 3.14, Thm. 5.10”} \\ &= \mathcal{L}_{g_{+\kappa}}^{\times}(\pi, C) \quad \text{“def. of } \mathcal{L}_{g_{+\kappa}}^{\times}(\pi, C)\text{”}\end{aligned}$$

□

Example 5.14 To gain more insight into additive and multiplicative leakage, let us graph the leakage for Example 5.5. To further demonstrate the effect of *shifting*, we use the shifted gain function g_{+1} , which adds 1 to all the gain values of g (which in fact makes g non-negative). As shown in Thm. 3.14 and Thm. 5.10, that simply has the effect of increasing all vulnerabilities by 1. And, as shown in Thm. 5.13, it has no effect on the additive leakage (but it does affect the multiplicative leakage).

Figure 5.4 shows the additive and multiplicative g_{+1} -leakage of C . As can be seen, the additive leakage ranges between 0 and about 0.2, while the multiplicative leakage ranges between 1 and about 1.15. In particular, for heavily skewed priors, we see that the additive leakage is 0 and the multiplicative leakage is 1, both of which mean *no leakage*; as can be seen from Fig. 5.1, that happens exactly when the prior- and posterior vulnerabilities are equal, meaning that the output of C does not help the adversary to choose a better action.

It is also interesting to observe that the additive and multiplicative leakages are *neither* convex nor concave functions of π , and both have several local minimums and local maximums. From the two dotted horizontal lines, it appears that additive leakage is maximized to 0.2 when $\pi = (1/3, 2/3)$, while multiplicative leakage is maximized to 1.15 when either $\pi = (1/3, 2/3)$ or $\pi = (1/2, 1/2)$. □

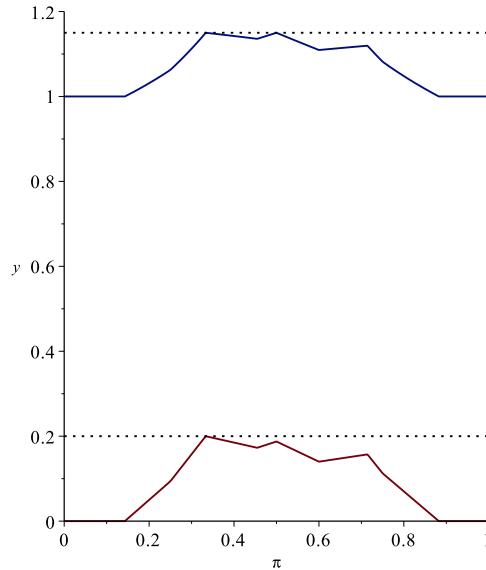


Figure 5.4 Comparison of $\mathcal{L}_{g+1}^+(\pi, C)$ (red) and $\mathcal{L}_{g+1}^\times(\pi, C)$ (blue) for Example 5.5

5.3 A closer look at posterior Bayes vulnerability and Bayes leakage

In this section, we look more closely at posterior Bayes vulnerability and Bayes leakage (which corresponds to g_{1d}), establishing some important properties.

First we note that posterior Bayes vulnerability has clear operational significance with respect to confidentiality. Indeed we can understand $V_1[\pi \triangleright C]$ as a smart adversary \mathcal{A} 's *probability of winning* the following game, in which X is sampled according to π , channel output Y is sampled according to row X of C , and \mathcal{A} (knowing π , C , and Y) tries to guess X :

Posterior Bayes-vulnerability Game

- | | |
|--|---|
| <ul style="list-style-type: none"> - <i>Defender</i> $X \in \pi$ $Y \in C_{X,-}$ | <ul style="list-style-type: none"> - <i>Choose X according to prior.</i> - <i>Output Y according to C.</i> |
| <ul style="list-style-type: none"> - <i>and then Adversary</i> $W \in \mathcal{A}(\pi, C, Y)$ $\text{if } W=X \text{ then "win" else "lose"}$ | <ul style="list-style-type: none"> - <i>Choose guess, and...</i> - <i>...win if correct.</i> |

This game should be contrasted with the Prior Bayes-vulnerability Game on p. 21, where \mathcal{A} knows only π .

Recall that, as discussed in Section 4.2, joint matrix J is defined by $J = \pi \triangleright C$, so that $J_{x,y} = \pi_x C_{x,y}$. There is a particularly easy way to calculate posterior Bayes vulnerability from J .

Theorem 5.15 Posterior Bayes vulnerability is the sum of the column maximums of the joint matrix J :

$$V_1[\pi \triangleright C] = \sum_{y \in \mathcal{Y}} \max_{x \in \mathcal{X}} J_{x,y} .$$

Proof. Given channel matrix C and prior π , we reason

$$\begin{aligned} & V_1[\pi \triangleright C] \\ = & V_{g_{\text{id}}}[\pi \triangleright C] && \text{"Thm. 3.6"} \\ = & \sum_y \max_w \sum_x \pi_x C_{x,y} g_{\text{id}}(w, x) && \text{"Thm. 5.7"} \\ = & \sum_y \max_w \pi_w C_{w,y} && "g_{\text{id}}(w, x) = (1 \text{ if } w=x \text{ else } 0); \text{one-point rule}" \\ = & \sum_y \max_x J_{x,y} && "\text{rename } w \text{ to } x; \text{definition of } J" \end{aligned}$$

□

Notice that prior Bayes vulnerability can also be expressed in terms of J , i.e. that

$$V_1(\pi) = \max_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} J_{x,y} ,$$

which differs from the expression for posterior Bayes vulnerability only in having swapped the order of summation and maximization. Thus posterior Bayes vulnerability is the *sum of the column maximums of J* , while prior Bayes vulnerability is the *maximum of the row sums of J* .

A consequence of that characterization is that posterior Bayes vulnerability is greater than prior Bayes vulnerability only if the column maximums of J do not all occur in the same row. For if they do all occur in some row x , then the posterior Bayes vulnerability is the sum of that row, which must also be the prior Bayes vulnerability.

Another way of saying the above is to observe that there is *no* Bayes leakage if the adversary's best guess about X is unaffected by the output Y . That can sometimes be surprising, as in the following example, which illustrates the so-called *base-rate fallacy*.

Example 5.16 Suppose that C is the channel matrix of a good, but imperfect, test for a certain disease:

C	<i>positive</i>	<i>negative</i>
<i>disease</i>	9/10	1/10
<i>no disease</i>	1/10	9/10

Moreover, suppose that for the population under consideration (say, age 40–50, no symptoms, no family history) the prior π is heavily biased towards “no disease”:

$$\pi = (1/100, 99/100) .$$

Then, although the channel might appear to be quite reliable, we find that there is *no* Bayes leakage. For we find that the hyper-distribution $[\pi \triangleright C]$ is

$[\pi \triangleright C]$	27/250	223/250
<i>disease</i>	1/12	1/892
<i>no disease</i>	11/12	891/892

5 Posterior vulnerability and leakage

reflecting the fact that a positive test result (corresponding to the first inner) increases the probability of disease from $1/100$ to $1/12$, while a negative test result (corresponding to the second inner) decreases it to $1/892$. Nevertheless, C 's output is useless in winning the Posterior Bayes-vulnerability Game on p. 82, since the doctor's best action is always to guess "no disease". And indeed we see that

$$V_1[\pi \triangleright C] = 27/250 \cdot 11/12 + 223/250 \cdot 891/892 = 99/100 .$$

We can in fact calculate $V_1[\pi \triangleright C]$ more directly, using Thm. 5.15. Since J is

J	<i>positive</i>	<i>negative</i>	
<i>disease</i>	$9/1000$	$1/1000$	
<i>no disease</i>	$99/1000$	$891/1000$	

we find that

$$V_1[\pi \triangleright C] = 99/1000 + 891/1000 = 99/100 .$$

Since $V_1[\pi \triangleright C] = 99/100 = V_1(\pi)$, we see that the prior- and posterior vulnerabilities are equal, and hence that there is no Bayes leakage for this π and C . \square

We conclude this section by showing that *multiplicative Bayes leakage on a uniform prior* can be calculated very easily.

Theorem 5.17 For any channel matrix C , if ϑ is uniform then the multiplicative Bayes leakage is equal to the sum of the column maximums of C :

$$\mathcal{L}_1^\times(\vartheta, C) = \sum_{y \in \mathcal{Y}} \max_{x \in \mathcal{X}} C_{x,y} .$$

Proof. It is a simple corollary of Thm. 5.15. If \mathcal{X} has size n and ϑ is uniform, then we have

$$\mathcal{L}_1^\times(\vartheta, C) = \frac{V_1[\vartheta \triangleright C]}{V_1(\vartheta)} = \frac{\sum_y \max_x J_{x,y}}{1/n} = \frac{\sum_y \max_x \frac{C_{x,y}}{n}}{1/n} = \sum_y \max_x C_{x,y} .$$

\square

We will see (in Thm. 7.2) that multiplicative Bayes leakage on a *uniform* prior is particularly interesting, in that it is always the *maximum* multiplicative Bayes leakage over *all* priors π .

5.4 Measuring leakage with Shannon entropy

Shannon entropy, defined by

$$H(\pi) := - \sum_{x \in \mathcal{X}} \pi_x \log_2 \pi_x$$

and discussed previously in §2.2 and §3.2.8, is at the heart of *information theory*, a rich and well-developed area of mathematics with a remarkable breadth of fruitful applications.

Moreover, Shannon entropy gives an obvious way to measure information leakage (from the *uncertainty* rather than the *vulnerability* perspective): prior uncertainty is $H(\pi)$, traditionally denoted $H(X)$; posterior uncertainty is $H[\pi \triangleright C]$, traditionally denoted $H(X|Y)$; and information leakage is the difference $H(\pi) - H[\pi \triangleright C]$, which is called *mutual information* and traditionally denoted $I(X; Y)$. Indeed because it is so natural to do this, and because of the stature of Shannon entropy, early researchers in quantitative information flow were led to adopt this as the definition of information leakage.

But by now we are accustomed to ask “What is the *operational significance* of mutual information with respect to confidentiality?” As discussed in §3.2.8, *Shannon’s source-coding theorem* gives such a significance in terms of the expected number of arbitrary yes/no questions about X needed to determine its value.³ But this scenario is perhaps not particularly common in practice and, as we found in the failed Conjecture 2.2, high Shannon entropy does not guarantee low Bayes vulnerability.

Here is an example that shows the problem clearly. Suppose that X is a 64-bit unsigned integer (so that $\mathcal{X} = \{0, 1, 2, \dots, 2^{64} - 1\}$) and suppose that ϑ is uniform. Now consider the following channel C (previously mentioned as (4.1)):

$$Y := \text{if } (X \bmod 8) = 0 \text{ then } X \text{ else } 1 .$$

Let us calculate the leakage as measured by mutual information $I(X; Y)$. First, we have prior uncertainty $H(\vartheta) = \log_2 2^{64} = 64$. Turning to posterior uncertainty $H[\vartheta \triangleright C]$, we first need to calculate $H(p_{X|y})$ for each possible value y .

Looking at the `then` branch, we see that Y can be any of the 2^{61} multiples of 8 in the given range. For each such y , the distribution $p_{X|y}$ is the *point distribution* that gives X probability 1 of having value y and probability 0 of having any other value; as a result we have $H(p_{X|y}) = 0$ for every such y . Moreover $p_Y(y) = 2^{-64}$ for each such y .

Looking at the `else` branch, we see that Y can also be 1, in which case X can be any value in the given range that is *not* a multiple of 8. Hence $p_{X|1}$ is a uniform distribution on $7/8 \cdot 2^{64}$ or equivalently $7 \cdot 2^{61}$ values of X . Hence

$$H(p_{X|1}) = \log_2(7 \cdot 2^{61}) = \log_2 7 + 61 \approx 2.807 + 61 \approx 63.807 .$$

Moreover $p_Y(1) = 7/8$.

Finally, by putting those together we obtain

$$H[\vartheta \triangleright C] \approx 2^{61} \cdot 2^{-64} \cdot 0 + 7/8 \cdot 63.807 \approx 55.831$$

and

$$I(X; Y) = H(\vartheta) - H[\vartheta \triangleright C] \approx 64 - 55.831 \approx 8.169 .$$

The result is that, if we measure leakage using Shannon entropy and mutual information, we conclude that channel C leaks just 8.169 bits out of 64, leaving a posterior uncertainty of 55.831 bits — which suggests that the channel does not harm the confidentiality of X very much. But notice that whenever Y is not equal to 1, which happens with probability $1/8$, then the exact value of X is revealed, showing that C is very bad indeed for the confidentiality of X ! Thus we see that mutual-information leakage can be quite misleading with respect to confidentiality.

³ As discussed in §3.2.8, this precise correspondence actually holds only when π is a *dyadic distribution*, meaning that all its probabilities are negative powers of 2.

In contrast, consider the multiplicative Bayes leakage $\mathcal{L}_1^{\times}(\vartheta, C)$. Using Thm. 5.17, that can be calculated very easily as the sum of the column maximums of C , which (since C is deterministic) is just the number of possible output values of C . It is easy to see that the `then` branch gives 2^{61} possible values and the `else` branch gives 1 more. Hence we have $\mathcal{L}_1^{\times}(\vartheta, C) = 2^{61} + 1$, which implies that the posterior Bayes vulnerability is

$$V_1[\vartheta \triangleright C] = V_1(\vartheta) \cdot \mathcal{L}_1^{\times}(\vartheta, C) = 2^{-64} \cdot (2^{61} + 1) = 1/8 + 2^{-64},$$

which is very high indeed if compared to the prior vulnerability. We remark that we can get the same result directly by observing that the adversary's chance of guessing X correctly given the output of C is 1 in the case when $Y \neq 1$, which happens with probability $1/8$, and $8/7 \cdot 2^{-64}$ when $Y = 1$, which happens with probability $7/8$. Hence we get

$$V_1[\vartheta \triangleright C] = 1/8 \cdot 1 + 7/8 \cdot 8/7 \cdot 2^{-64} = 1/8 + 2^{-64},$$

which agrees with the previous calculation.

5.5 More properties of posterior g -vulnerability and g -leakage

Now we return our attention to general posterior g -vulnerability, establishing some additional important properties.

5.5.1 A matrix-based formulation of posterior g -vulnerability

Theorem 5.15 formulates posterior Bayes vulnerability $V_1[\pi \triangleright C]$ in terms of the joint matrix J . Here we show that this result generalizes to posterior g -vulnerability.

Recall that a gain function $g: \mathbb{G}^{\text{fin}}\mathcal{X} \rightarrow \mathbb{R}$ can be represented as a $\mathcal{W} \times \mathcal{X}$ -indexed matrix G , where $G_{w,x} = g(w, x)$; here the rows of G correspond to actions and the columns to secrets. It turns out that the posterior g -vulnerability can be calculated easily from the $\mathcal{W} \times \mathcal{Y}$ -indexed matrix GJ .⁴

Theorem 5.18 For $g: \mathbb{G}^{\text{fin}}\mathcal{X} \rightarrow \mathbb{R}$, posterior g -vulnerability is the sum of the column maximums of the matrix GJ :

$$V_g[\pi \triangleright C] = \sum_{y \in \mathcal{Y}} \max_{w \in \mathcal{W}} (GJ)_{w,y}.$$

Proof. This follows easily from Thm. 5.7:

$$\begin{aligned} & V_g[\pi \triangleright C] \\ &= \sum_y \max_w \sum_x \pi_x C_{x,y} g(w, x) && \text{"Thm. 5.7"} \\ &= \sum_y \max_w \sum_x J_{x,y} G_{w,x} && \text{"definitions of } G \text{ and } J\text{"} \\ &= \sum_y \max_w (GJ)_{w,y}. && \text{"definition of matrix multiplication"} \end{aligned}$$

□

⁴ Recall that we mean matrix multiplication here.

There is an important operational meaning to the column maximums of \mathbf{GJ} : if the maximum of column y occurs at row w , it means that action w is optimal for the adversary, given output y .

Also note that, since the matrix representation of g_{id} is the $\mathcal{X} \times \mathcal{X}$ -indexed identity matrix \mathbf{I} and $\mathbf{I} \cdot \mathbf{J} = \mathbf{J}$, we can see that Thm. 5.15 is hence a simple corollary to Thm. 5.18.

Example 5.19 Recall that Example 5.4 included a lengthy calculation showing that $V_g[\pi \triangleright \mathbf{C}] = 0.5575$ when $\pi = (0.3, 0.7)$. Theorem 5.18 lets us calculate that result much more easily. First we compute the matrix product \mathbf{GJ} as follows:

$$\begin{array}{|c|cc|} \hline \mathbf{G} & x_1 & x_2 \\ \hline w_1 & -1.0 & 1.0 \\ w_2 & 0.0 & 0.5 \\ w_3 & 0.4 & 0.1 \\ w_4 & 0.8 & -0.9 \\ w_5 & 0.1 & 0.2 \\ \hline \end{array} \quad \begin{array}{|c|cc|} \hline \mathbf{J} & y_1 & y_2 \\ \hline x_1 & 0.225 & 0.075 \\ x_2 & 0.175 & 0.525 \\ \hline \end{array} = \begin{array}{|c|cc|} \hline \mathbf{GJ} & y_1 & y_2 \\ \hline w_1 & -0.0500 & 0.4500 \\ w_2 & 0.0875 & 0.2625 \\ w_3 & 0.1075 & 0.0825 \\ w_4 & 0.0225 & -0.4125 \\ w_5 & 0.0575 & 0.1125 \\ \hline \end{array} .$$

Then, looking at the column maximums of \mathbf{GJ} , we obtain

$$V_g[\pi \triangleright \mathbf{C}] = 0.1075 + 0.4500 = 0.5575 .$$

□

It is interesting to observe that we can also formulate prior g -vulnerability in terms of \mathbf{GJ} :

Theorem 5.20 For $g: \mathbb{G}^{\text{fin}} \mathcal{X}$, prior g -vulnerability is the maximum of the row sums of the matrix \mathbf{GJ} :

$$V_g(\pi) = \max_{w \in \mathcal{W}} \sum_{y \in \mathcal{Y}} (\mathbf{GJ})_{w,y} .$$

Proof. We can reason as follows:

$$\begin{aligned} & V_g(\pi) \\ &= \max_w \sum_x \pi_x g(w, x) \\ &= \max_w \sum_x \pi_x g(w, x) \sum_y C_{x,y} && \text{“$\sum_y C_{x,y} = 1$”} \\ &= \max_w \sum_y \sum_x \pi_x C_{x,y} g(w, x) && \text{“reorganizing sum”} \\ &= \max_w \sum_y \sum_x J_{x,y} G_{w,x} && \text{“definitions of J and G”} \\ &= \max_w \sum_y (\mathbf{GJ})_{w,y} && \text{“definition of matrix multiplication”} \end{aligned}$$

□

Note that Thm. 5.18 and Thm. 5.20 imply that $V_g[\pi \triangleright \mathbf{C}] = V_g(\pi)$ just when all the column maximums of \mathbf{GJ} occur in the same row w , say. And so the adversary's best action is always that w , i.e. is the *same*, regardless of the output of \mathbf{C} . Thus (at least for this g and π), channel \mathbf{C} is useless to her.

5.5.2 A trace-based formulation of posterior g -vulnerability

We now build on Thm. 5.18 to develop an important formulation of posterior g -vulnerability as a *matrix trace*, whose definition we first recall.

Definition 5.21 (Trace of square matrix) If \mathbf{M} is a square matrix, then its *trace* is the sum of its diagonal entries: thus $\text{tr}(\mathbf{M}) = \sum_i M_{i,i}$. □

5 Posterior vulnerability and leakage

Now note that both Thm. 5.7 and Thm. 5.18 formulate posterior g -vulnerability as a *summation over \mathcal{Y} of a maximization over \mathcal{W}* , which means that for each y in \mathcal{Y} , a best w in \mathcal{W} must be selected. This selection can be *reified* as a *strategy* S , and expressed as a channel matrix from \mathcal{Y} to \mathcal{W} that tells which action w is best, given output y .⁵ While one might expect that S should be deterministic, it is in fact reasonable to allow it to be probabilistic, since there could be more than one w that is optimal for a given y . That is, row y of S can give some distribution to the actions w that are optimal given output y .

Now we show that taking the sum of the column maximums of a matrix can be expressed neatly as a maximization over strategies of a matrix trace.

Lemma 5.22 If M is any $\mathcal{W} \times \mathcal{Y}$ -indexed matrix, then

$$\max_S \text{tr}(SM) = \sum_{y \in \mathcal{Y}} \max_{w \in \mathcal{W}} M_{w,y},$$

where S ranges over strategies from \mathcal{Y} to \mathcal{W} . Moreover, the maximum is always realized on some *deterministic* strategy S .

Proof. By the definition of matrix multiplication, $(SM)_{y,y} = \sum_w S_{y,w} M_{w,y}$, which means that $(SM)_{y,y}$ is a convex combination of column y of M . But any convex combination must be less than or equal to the maximum, with equality holding if S satisfies the condition that $S_{y,w} > 0$ only if $M_{w,y}$ is a maximum in column y . In particular, this condition is satisfied by a *deterministic* S that selects one best w for each y .

So if S is any matrix satisfying the condition, then SM contains the column maximums of M on its diagonal, and the lemma follows. \square

A benefit of trace-based formulations is that trace satisfies a remarkable *cyclic property*.

Theorem 5.23 If matrix product AB is square, than $\text{tr}(AB) = \text{tr}(BA)$.

Proof. If A is indexed by $\mathcal{X} \times \mathcal{Y}$ and B is indexed by $\mathcal{Y} \times \mathcal{X}$, then we have

$$\begin{aligned} & \text{tr}(AB) \\ = & \sum_x (AB)_{x,x} \\ = & \sum_x \sum_y A_{x,y} B_{y,x} \\ = & \sum_y \sum_x B_{y,x} A_{x,y} && \text{"reorganizing sum"} \\ = & \sum_y (BA)_{y,y} \\ = & \text{tr}(BA). \end{aligned}$$

\square

Moreover, by the associativity of matrix multiplication, the cyclic property generalizes to a product of any number of matrices:

$$\text{tr}(ABCD) = \text{tr}(BCDA) = \text{tr}(CDAB) = \text{tr}(DABC).$$

With those results, we are now ready to show a trace-based formulation of posterior g -vulnerability.

⁵ This reification is reminiscent of Skolemization in predicate logic.

Theorem 5.24

For any $g: \mathbb{G}^{\text{fin}}\mathcal{X}$, prior π , and channel matrix $C: \mathcal{X} \rightarrow \mathcal{Y}$, we have

$$V_g[\pi \triangleright C] = \max_S \text{tr}(G^\top \pi_{\downarrow} CS) ,$$

where S ranges over strategies from \mathcal{Y} to \mathcal{W} and π_{\downarrow} denotes a diagonal matrix with π on its diagonal. Moreover, the maximum can always be realized by a deterministic strategy.

Proof. We reason as follows:

$$\begin{aligned} & V_g[\pi \triangleright C] \\ = & \sum_y \max_w (GJ)_{w,y} && \text{"Thm. 5.18"} \\ = & \max_S \text{tr}(SGJ) && \text{"Lem. 5.22"} \\ = & \max_S \text{tr}(SG \pi_{\downarrow} C) && \text{"} J = \pi \triangleright C = \pi_{\downarrow} C \text{"} \\ = & \max_S \text{tr}(G^\top \pi_{\downarrow} CS) && \text{"cyclic property of trace"} \end{aligned}$$

□

A benefit of Thm. 5.24's trace-based formulation of posterior g -vulnerability is that it lets us derive relationships among different leakage measures by exploiting two key properties: the associativity of matrix multiplication and the cyclic property of trace. The key idea is that we can regroup the matrices in $\text{tr}(G^\top \pi_{\downarrow} CS)$ to obtain new interpretations of $V_g[\pi \triangleright C]$. We illustrate with two examples.

Example 5.25 (Moving the prior into the gain function) For any prior π , observe that the matrix π_{\downarrow} can be factored into $\rho_{\downarrow} \vartheta_{\downarrow}$, where ϑ is the uniform distribution on \mathcal{X} and $\rho_x = \pi_x / \vartheta_x$. (Note that ρ is typically not a probability distribution.) Hence we have

$$\text{tr}(G^\top \pi_{\downarrow} CS) = \text{tr}(G(\rho_{\downarrow} \vartheta_{\downarrow}) CS) = \text{tr}((G^\top \rho_{\downarrow}) \vartheta_{\downarrow} CS) ,$$

which implies that $V_g[\pi \triangleright C] = V_{g'}[\vartheta \triangleright C]$, where g' is the gain function represented by $G^\top \rho_{\downarrow}$. (Notice in contrast that moving the gain function into the prior is not possible except in the special case where G is diagonal.) □

Example 5.26 (Moving the channel matrix into the strategy) Since any channel matrix C trivially factors into $I|C$, where I is the $\mathcal{X} \times \mathcal{X}$ -indexed identity matrix, we have

$$\text{tr}(G^\top \pi_{\downarrow} CS) = \text{tr}(G^\top \pi_{\downarrow} (I|C) S) = \text{tr}(G^\top \pi_{\downarrow} I(CS)) ,$$

where I is now the channel and CS can be seen as a strategy from \mathcal{X} to \mathcal{W} . There is no guarantee of course that CS is optimal but, in spite of that, we can still derive an interesting inequality:

$$\begin{aligned} & V_g[\pi \triangleright C] \\ = & \max_S \text{tr}(G^\top \pi_{\downarrow} CS) && \text{"Thm. 5.24"} \\ = & \max_S \text{tr}(G^\top \pi_{\downarrow} I(CS)) && \text{"as above: } I \text{ is identity matrix"} \\ \leq & \max_{S'} \text{tr}(G^\top \pi_{\downarrow} IS') && \text{"CS might not be optimal"} \\ = & V_g[\pi \triangleright I] . \end{aligned}$$

Thus we see that the posterior g -vulnerability on any channel C never exceeds that on the channel \mathbb{O} that leaks everything, which as a reduced channel matrix is I . □

5.5.3 A linear-programming formulation

Seeing the elements of strategy S as variables, we realize that $V_g[\pi \triangleright C]$ is the solution to the *linear optimization problem* (for fixed π and C and G):

Choose S to maximize $\text{tr}(G^\top \pi \triangleleft CS)$,
subject to all rows of S being probability distributions.

Note that the constraint “being a probability distribution” means just that the corresponding variables should be non-negative and sum up to 1. Note also that here S is not necessarily deterministic, meaning that the choice of action can be made probabilistically. However, from Thm. 5.24 we know that there is always an optimal solution where S is deterministic.

We should clarify that this linear-programming formulation is *not* particularly useful if we just wish to compute $V_g[\pi \triangleright C]$, since we can directly evaluate the formula in Thm. 5.7. Rather, the value of those linear-programming insights is that we can adapt them in order to solve some challenging algorithmic problems, as we now demonstrate.

Algorithm 5.27 Given channel matrices $A: \mathcal{X} \rightarrow \mathcal{Y}$ and $B: \mathcal{X} \rightarrow \mathcal{Z}$, and gain function g , decide whether there exists a prior π such that $V_g[\pi \triangleright A] < V_g[\pi \triangleright B]$ and, if so, find a prior π that maximizes the difference. (Notice that this is equivalent to finding a prior that maximizes the difference between B 's additive g -leakage and A 's.)

The obvious challenge here is that there are infinitely many priors π to consider. However, we can formulate that as an optimization problem for fixed A , B , and G (the matrix representation of g); and if we view the elements of π , S^A , and S^B as variables, that becomes

Choose π to maximize $(\max_{S^B} \text{tr}(G^\top \pi \triangleleft BS^B) - \max_{S^A} \text{tr}(G^\top \pi \triangleleft AS^A))$
subject to π and all rows of S^A, S^B being probability distributions.

There are however two issues with that formulation: first, $\text{tr}(G^\top \pi \triangleleft BS^B)$ is quadratic in the variables π and S^B , rather than linear, and second, it contains nested maximizations. But we can address those issues by first observing that we can assume without loss of generality that the strategies are deterministic, and then noting that there are only finitely many deterministic strategies S^A and S^B , namely $|\mathcal{W}|^{|\mathcal{Y}|}$ and $|\mathcal{W}|^{|\mathcal{Z}|}$ many, respectively. Moreover, for a fixed S^A we can express the property “ π is a prior such that S^A is optimal” via a set of linear constraints that say that, for every y , the expected gain from action $S^A(y)$ is at least as good as that from any action w . That is, we require for all y in \mathcal{Y} and w in \mathcal{W} that

$$\sum_x \pi_x A_{x,y} g(S^A(y), x) \geq \sum_x \pi_x A_{x,y} g(w, x) .$$

If we denote those constraints as $opt(S^A)$, then the solution to our non-linear optimization above will be the *maximum* of the solutions to the following linear problems, over all choices of S^A and S^B :

Choose π to maximize $(\text{tr}(G^\top \pi \triangleleft BS^B) - \text{tr}(G^\top \pi \triangleleft AS^A))$
subject to π being a probability distribution and $opt(S^A)$.⁶

⁶ It might appear that we should also include the constraints $opt(S^B)$, but they are not actually necessary — the optimization of the objective function automatically forces S^B to be optimal.

Thus we are able to solve our original problem by solving a total of $|\mathcal{W}|^{|\mathcal{Y}|+|\mathcal{Z}|}$ linear-programming problems. (Of course that will still be prohibitively expensive unless \mathcal{W} , \mathcal{Y} , and \mathcal{Z} are small.) \square

Algorithm 5.28 Given channel matrices $A: \mathcal{X} \rightarrow \mathcal{Y}$ and $B: \mathcal{X} \rightarrow \mathcal{Z}$, and prior π , decide whether there exists a gain function g such that $V_g[\pi \triangleright A] < V_g[\pi \triangleright B]$ and, if so, find a gain function g that maximizes the difference.

This problem is similar to that considered in Algorithm 5.27, with the difference that here π is fixed and the elements of G (the matrix representation of g) are variables.

One issue, however, is that maximizing over all of $\mathbb{G}\mathcal{X}$ will typically lead to an unbounded solution, since (by Thm. 5.10) scaling g by k will also scale the difference $V_g[\pi \triangleright B] - V_g[\pi \triangleright A]$ by k . For this reason, we instead will maximize over one-bounded gain functions $g: \mathbb{G}^{\dagger}\mathcal{X}$. To do this, we constrain the gain values of g to be at most 1 (but we do not bound them from below). And to ensure that g 's vulnerabilities are non-negative, we include a special action \perp whose gain values are all 0. A second issue is that the number of possible actions of g is not fixed. However it is easy to see that it suffices to have $|\mathcal{Y}| + |\mathcal{Z}| + 1$ possible actions, since $V_g[\pi \triangleright A]$ can use at most $|\mathcal{Y}|$ actions, $V_g[\pi \triangleright B]$ can use at most $|\mathcal{Z}|$, and we also need the \perp action.

At this point, we could proceed as in Algorithm 5.27 and try exponentially many strategies S^A and S^B . But it turns out that here we can do enormously better — it is sufficient to consider *just one* S^A and S^B !

To see this, observe that we can assume without loss of generality that the first $|\mathcal{Y}|$ rows of G contain, in order, optimal actions for the columns of A ; and the next $|\mathcal{Z}|$ rows of G contain, in order, optimal actions for the columns of B ; and finally the last row of G is all zero, for the \perp action. Achieving this just requires reordering the rows of G and possibly duplicating some actions (since the same action might be optimal for more than one column). Once this is done, we can use a fixed S^A that maps column j of A to row j of G , and a fixed S^B that maps column k of B to row $k+|\mathcal{Y}|$ of G .

Now we can solve a *single* linear programming problem for that S^A and S^B :

Choose G to maximize $(\text{tr}(G^\top \pi \triangleright B S^B) - \text{tr}(G^\top \pi \triangleright A S^A))$
 subject to G having elements of at most 1 and a final all-zero row, and $\text{opt}(S^A)$.

Remarkably, and in sharp contrast to Algorithm 5.27, this means that a maximizing gain function G can be found in *polynomial time*. \square

5.6 Example channels and their leakage

Let us now consider some example channels and their leakage under a variety of gain functions.

Recall the gain functions for a password database discussed in Section 3.2.4. Here the secret is an array X containing 10-bit passwords for a set U of 1000 users, so that for u in U the entry $X[u]$ is user u 's password.

One possibility is to use g_{id} , corresponding to Bayes vulnerability, which represents an adversary trying to guess the *entire* array X .

5 Posterior vulnerability and leakage

A second possibility is to use g_{pw} , which represents an adversary interested in guessing *some* user's password, with no preference as to whose it is. Here we have

$$\mathcal{W} = \{(u, x) \mid u \in U \text{ and } 0 \leq x \leq 1023\}$$

and

$$g_{\text{pw}}((u, x), X) = \begin{cases} 1 & \text{if } X[u] = x \\ 0 & \text{otherwise} \end{cases}.$$

A third possibility is to use g_{Gates} , which (we recall) specifies that one of the users is Bill Gates, whose password is much more valuable than the other passwords. Here we have the same \mathcal{W} as for g_{pw} and

$$g_{\text{Gates}}((u, x), X) = \begin{cases} 1 & \text{if } u = \text{Bill Gates and } x = X[u] \\ 0.01 & \text{if } u \neq \text{Bill Gates and } x = X[u] \\ 0 & \text{otherwise} \end{cases}.$$

Suppose now that the prior π is uniform, meaning that the 1000 passwords in X are independent and uniformly distributed on $[0..1023]$. Consider the following probabilistic channel C , which reveals *some* randomly chosen user's password:

$$\begin{aligned} u &:= \text{uniform}(U) && -\text{Choose } u \text{ uniformly from } U. \\ Y &:= (u, X[u]) && -\text{Leak password via the observable output } Y. \end{aligned}$$

If we analyze leakage using Bayes vulnerability (i.e. g_{id}), we find that the prior Bayes vulnerability $V_1(\pi) = 2^{-10,000}$, since there are 10,000 completely unknown bits in X . And the posterior Bayes vulnerability $V_1[\pi \triangleright C] = 2^{-9990}$, since now 10 of the 10,000 bits of X are known. Hence the multiplicative Bayes leakage $\mathcal{L}_1^X(\pi, C)$ is $2^{-9990}/2^{-10,000}$, that is 2^{10} .

In contrast, if we measure leakage using g_{pw} , then we find that the prior g_{pw} -vulnerability is 2^{-10} . This follows from the fact that the expected gain from every action (u, x) in \mathcal{W} is 2^{-10} since, for every u , the entry $X[u]$ is uniformly distributed on $[0..1023]$. And the posterior g_{pw} -vulnerability is $V_{g_{\text{pw}}}[\pi \triangleright C] = 1$, since given that $Y = (u, X[u])$ the adversary can perform action $(u, X[u])$ and be sure of getting gain 1. Hence the multiplicative g_{pw} -leakage $\mathcal{L}_{g_{\text{pw}}}^X(\pi, C) = 1/2^{-10} = 2^{10}$.

It is interesting that the multiplicative leakage is the same, that is 2^{10} , for both Bayes vulnerability and g_{pw} -vulnerability. On the other hand, if we look at *additive* leakage, then the story is very different. With Bayes vulnerability we get additive leakage $\mathcal{L}_1^+(\pi, C) = 2^{-9990} - 2^{-10,000}$, which is a negligible increase, while with g_{pw} -vulnerability we get additive leakage $\mathcal{L}_{g_{\text{pw}}}^+(\pi, C) = 1 - 2^{-10} \approx 0.999$, which is a huge difference.

In summary, we find with Bayes vulnerability that C 's multiplicative leakage is large while its additive leakage is negligible, while we find with g_{pw} -vulnerability that both C 's multiplicative and additive leakage are large. But with other channels and gain functions, the situation could be reversed. For example, if under gain function g_1 the vulnerability increases from 0.4 to 0.7 and under gain function g_2 the vulnerability increases from 0.000001 to 0.300001, then the additive leakage is 0.3 in both cases, but the multiplicative leakage under g_1 is modest (1.75) while under g_2 it is huge (300,001). Those examples suggest that perhaps leakage should be judged to be "significant" only if *both* the multiplicative- and additive leakages are "large".

It is also interesting to consider a variant of channel C that selects 10 random users and leaks just the last bit of each of their passwords. Because the variant still reveals 10 bits of X to the adversary, its multiplicative Bayes leakage remains 2^{10} . But the

multiplicative g_{pw} -leakage is now only 2, because the posterior g_{pw} -vulnerability is just 2^{-9} , since now at least 9 bits of each user's password remain unknown. Thus gain function g_{pw} captures the fact that some sets of 10 bits are worth more than others.

Let us now consider channel D, which leaks Bill Gates' password with probability 1/10 but otherwise leaks nothing:

```
n := uniform(0..9)
Y := if n=0 then X["Bill Gates"] else 0
```

If we consider channel D under multiplicative Bayes leakage, then we find that it leaks less than channel C. For D's posterior Bayes vulnerability is

$$V_1[\pi \triangleright D] = 1/10 \cdot 2^{-9990} + 9/10 \cdot 2^{-10,000}$$

which means that its multiplicative Bayes leakage is

$$\mathcal{L}_1^{\times}(\pi, D) = \frac{1/10 \cdot 2^{-9990} + 9/10 \cdot 2^{-10,000}}{2^{-10,000}} = 1/10 \cdot 2^{10} + 9/10 = 103.3 ,$$

which is much less than C's multiplicative Bayes leakage of 1024.

But if we analyze the multiplicative g_{Gates} -leakage of channels C and D, we find that the situation is reversed. First note that $V_{g_{\text{Gates}}}(\pi) = 2^{-10}$, since action (Bill Gates, x) has expected gain 2^{-10} for every x . Next note that channel C's posterior g_{Gates} -vulnerability is

$$V_{g_{\text{Gates}}}[\pi \triangleright C] = 1/1000 \cdot 1 + 999/1000 \cdot 0.01 = 0.01099$$

since the best action is always $(u, X[u])$, which gives gain 1 if $u = \text{Bill Gates}$ and gain 0.01 otherwise. Hence

$$\mathcal{L}_{g_{\text{Gates}}}^{\times}(\pi, C) = 0.01099/2^{-10} = 11.25376 .$$

In contrast, channel D's posterior g_{Gates} -vulnerability is

$$V_{g_{\text{Gates}}}[\pi \triangleright D] = 1/10 \cdot 1 + 9/10 \cdot 2^{-10} = 0.10087890625 ,$$

since the best action is always (Bill Gates, Y), which gives gain 1 if $n=0$ and gives gain 2^{-10} otherwise. Hence

$$\mathcal{L}_{g_{\text{Gates}}}^{\times}(\pi, D) = 0.10087890625/2^{-10} = 103.3 ,$$

which is much greater than C's leakage of 11.25376.

5.7 Max-case posterior g -vulnerability

As discussed in Section 5.1, an alternative way to define posterior g -vulnerability of $[\pi \triangleright C]$ is to consider the *maximum* g -vulnerability over all of the inners of $[\pi \triangleright C]$; this represents the *worst* possible “world” that C can give, ignoring the question of how *likely* that world might be.

Definition 5.29 (max-case posterior g-vulnerability) Given prior π , and g in $\mathbb{G}\mathcal{X}$ and channel C , let $[\pi \triangleright C] = \sum_i a_i [\delta^i]$ (recalling that the outer probabilities a_i are nonzero). Then the *max-case posterior g-vulnerability* $V_g^{\max}[\pi \triangleright C]$ is defined as the maximum value of V_g over all the inners:

$$V_g^{\max}[\pi \triangleright C] := \max_i V_g(\delta^i) .$$

(Notice that this definition pays no attention to the outer probabilities.) \square

As previously noted, $V_g^{\max}[\pi \triangleright C]$ is quite pessimistic, in that a single bad inner results in high max-case posterior g -vulnerability even if that inner is very unlikely.

On the other hand, for many channels the max-case posterior g -vulnerability is small, and in this case it is useful in offering stronger guarantees than those provided by the usual expectation-based posterior g -vulnerability. This is captured in the following theorem.

Theorem 5.30 For any $g: \mathbb{G}\mathcal{X}$ and π and C , we have $V_g[\pi \triangleright C] \leq V_g^{\max}[\pi \triangleright C]$.

Proof. Let $[\pi \triangleright C] = \sum a_i [\delta^i]$. Then we have

$$V_g[\pi \triangleright C] = \sum_i a_i V_g(\delta^i) \leq \sum_i a_i \max_j V_g(\delta^j) = V_g^{\max}[\pi \triangleright C].$$

□

Thus we see that it is sometimes desirable to analyze the max-case posterior g -vulnerability of a channel. Quite interestingly, if we are just interested in establishing that max-case posterior g -vulnerability does not exceed some threshold, then we can achieve this by calculating the expected-value-based posterior vulnerability for a *modified* gain function g_t . The following theorem makes this precise:

Theorem 5.31 Given $g: \mathbb{G}\mathcal{X}$, define g_t for $t \in \mathbb{R}$ as g with an extra action \perp such that $g_t(\perp, x) = t$ for all x in \mathcal{X} . Then for all π and C we have

$$V_{g_t}[\pi \triangleright C] = t \quad \text{iff} \quad V_g^{\max}[\pi \triangleright C] \leq t.$$

Proof. As usual, let $[\pi \triangleright C] = \sum_i a_i [\delta^i]$. By the construction of g_t , we have for every i that

$$V_{g_t}(\delta^i) = \max\{V_g(\delta^i), t\}.$$

For the forward direction, suppose that $V_{g_t}[\pi \triangleright C] = \sum_i a_i V_{g_t}(\delta^i) = t$. Then, since $V_{g_t}(\delta^i) \geq t$ and the a_i 's are convex coefficients, we must have $V_{g_t}(\delta^i) = t$ for all i . Hence we have $V_g(\delta^i) \leq t$ for all i , which implies that $V_g^{\max}[\pi \triangleright C] \leq t$.

For the backwards direction, suppose that $V_g^{\max}[\pi \triangleright C] = \max_i V_g(\delta^i) \leq t$. Then we must have $V_{g_t}(\delta^i) = t$ for all i , hence $V_{g_t}[\pi \triangleright C] = \sum_i a_i V_{g_t}(\delta^i) = t$. □

Thus we see that, to some extent, expected-value-based posterior vulnerability analysis can subsume max-case posterior vulnerability analysis.

5.8 Exercises

Exercise 5.1 Prove Theorem 5.7. □

Exercise 5.2 Suppose that the prior π is a *point distribution*, i.e. that some x has probability 1 and all others have probability 0. Show that, regardless of the channel C and gain function g , there is no g -leakage from π as prior: that is, $\mathcal{L}_g^\times(\pi, C) = 1$ and $\mathcal{L}_g^+(\pi, C) = 0$. □

Exercise 5.3 It is noted before Def. 4.14 that *noninterference* is a traditional name for the “no leakage” property. Show that noninterference indeed implies “no g -leakage”—that is, show that if C satisfies noninterference, then for any prior π and gain function g , we have $\mathcal{L}_g^\times(\pi, C) = 1$ and $\mathcal{L}_g^+(\pi, C) = 0$. □

Exercise 5.4 The *Monty Hall problem* is a famous brain teaser, based loosely on the old game show *Let's Make a Deal*, whose host was Monty Hall. In 1990, the problem appeared in the “Ask Marilyn” column of *Parade* magazine, formulated as follows:

Suppose you're on a game show, and you're given the choice of three doors: behind one door is a car; behind the other two are goats. You pick a door, say Door 1, and the host, who knows what's behind the doors, opens another door, say Door 3, which has a goat. He then says to you, “Do you want to pick Door 2 instead?”

Is it to your advantage to switch your choice?

This formulation is actually a bit imprecise about Monty Hall's behavior. What is intended is that Monty *always* opens a door that you did not choose and that contains a goat. (Since there are two doors containing goats, it is always possible for him to do that.) Also, he *always* gives you the option of switching.⁷

To solve this puzzle, let us formulate Monty Hall as a probabilistic channel M (for “Monty”) whose secret input X is the door containing the car, and whose output Y is the door that Monty opens after your initial choice, which we assume is Door 1. In the case when the car is behind Door 1, note that *both* Doors 2 and 3 contain goats, giving Monty a choice of which door to open. Here we assume that he makes this choice by flipping a fair coin, opening Door 2 if he gets *heads* and Door 3 if he gets *tails*. Hence the channel matrix is as follows:

M	2	3
1	1/2	1/2
2	0	1
3	1	0

Also, we assume a uniform prior $\vartheta = (1/3, 1/3, 1/3)$, so that the car is equally likely to be behind each of the doors.

- (a) Calculate the hyper-distribution $[\vartheta \triangleright M]$.
- (b) Calculate the posterior Bayes vulnerability $V_1[\vartheta \triangleright M]$ and the multiplicative Bayes leakage $L_1^\times(\vartheta, M)$. Based on your calculations, should you stick with Door 1 or should you switch?
- (c) Now assume that when the car is behind Door 1 (more generally, behind the door you choose), Monty uses a *biased* coin that gives *heads* with probability p and *tails* with probability $1-p$, for some p such that $0 \leq p \leq 1$. This changes the channel matrix to the following:

M	2	3
1	p	$1-p$
2	0	1
3	1	0

How does that change the results?

⁷ A notable advantage of setting Quantitative Information Flow on a rigorous foundation is that it allows tricky problems (such as this one, Monty Hall) to be solved more or less *mechanically* — that is, without the need for deep thought. It calls to mind a wonderful quote from Alfred Whitehead:

It is a profoundly erroneous truism, repeated by all copy-books and by eminent people when they are making speeches, that we should cultivate the habit of thinking of what we are doing. The precise opposite is the case. Civilization advances by extending the number of important operations which we can perform without thinking about them. Operations of thought are like cavalry charges in a battle — they are strictly limited in number, they require fresh horses, and must only be made at decisive moments.

□

Exercise 5.5 The following is known as the “Three Prisoners problem”. Three prisoners, A , B , and C are sentenced to death, but one of them (uniformly chosen at random) is selected to be pardoned, so that just two out of the three prisoners will be executed. The warden knows which one will be pardoned, but he is not allowed to tell the prisoners.

Prisoner A begs the warden to let her know the identity of *one of the others* who will be executed: “If B is pardoned, give me C ’s name, and vice versa. If I’m pardoned, choose randomly to name B or C . ”

- (a) Model the problem as a channel and compute its multiplicative Bayes capacity. Using the capacity, compute the probability of correctly guessing the pardoned prisoner after receiving the warden’s answer. Also, justify why the capacity is relevant for this task.
- (b) Prisoner A is of course only interested in finding information *about herself*, i.e. whether *she* is going to be executed or not. Is the warden’s answer useful for A ? Justify your answer by an information-flow analysis of an appropriate channel (either the same channel as in the first part, or a different one).

□

Exercise 5.6 Example 5.16 discussed the following channel matrix of a good, but imperfect, test for a certain disease:

C	<i>positive</i>	<i>negative</i>
<i>disease</i>	$\frac{9}{10}$	$\frac{1}{10}$
<i>no disease</i>	$\frac{1}{10}$	$\frac{9}{10}$

There we found that if $\pi = (1/100, 99/100)$, meaning that the disease is very unlikely, then there is *no* Bayes leakage.

Suppose however that we use a gain function like the one considered in Section 3.2.6, except that we have increased the magnitude of the gains and losses when the disease is present (perhaps because the disease is a serious one):

g_d	<i>disease</i>	<i>no disease</i>
<i>treat</i>	50	-2
<i>don’t treat</i>	-50	2

- (a) Calculate the prior and posterior g_d -vulnerabilities $V_{g_d}(\pi)$ and $V_{g_d}[\pi \triangleright C]$ and the additive g_d -leakage $\mathcal{L}_{g_d}^+(\pi, C)$.
- (b) Based on your results, what is the best action in the case of a *positive* output? What about in the case of a *negative* output?

□

Exercise 5.7 Recall Exercise 4.2, which considers a password checker implementation with a timing side channel. Here we continue that topic in the more realistic setting of a 4-digit PIN ranging from 0000 to 9999. As before, an ideal password checker is modeled by a family of channels C^g with output set {reject, accept}. And a flawed implementation that compares the guess with the correct PIN digit by digit, rejecting as soon as a mismatch is found, is modeled by a family of channels D^g with output set {(reject, 1), (reject, 2), (reject, 3), (reject, 4), accept}.

- (a) Assuming a uniform prior $\vartheta = (1/10,000, 1/10,000, \dots, 1/10,000)$, show that, for any guess \mathbf{g} , the posterior Bayes vulnerabilities under $C^{\mathbf{g}}$ and $D^{\mathbf{g}}$ are $V_1[\vartheta \triangleright C^{\mathbf{g}}] = 1/5000$ and $V_1[\vartheta \triangleright D^{\mathbf{g}}] = 1/2000$.

- (b) Given that, one might be tempted to conclude that the difference in the posterior Bayes vulnerabilities is small enough that the side channel in $D^{\mathbf{g}}$ is not worth bothering about.

But consider that in a typical PIN scenario, the adversary can enter a *sequence* of guesses $\mathbf{g}_1, \mathbf{g}_2, \dots, \mathbf{g}_k$, thus running channels $D^{\mathbf{g}_1}, D^{\mathbf{g}_2}, \dots, D^{\mathbf{g}_k}$ for some moderate value of k . (Too many consecutive incorrect guesses might get her locked out.) Note that she can choose her guesses *adaptively*, which means that she can choose each guess \mathbf{g}_i based on the outputs from the previous guesses $\mathbf{g}_1, \mathbf{g}_2, \dots, \mathbf{g}_{i-1}$. How many adaptively chosen guesses to $D^{\mathbf{g}}$ does she need to determine the correct PIN in the worst case? In the average case?

- (c) In contrast, how many adaptively chosen guesses to the ideal password checker $C^{\mathbf{g}}$ does she need in the worst case? In the average case?

□

5.9 Chapter notes

Many of the results in this chapter first appeared in Alvim et al. [1].

The properties of convex functions used in the proof of Thm. 5.9 are standard; see for example Rockafellar [11].

The quote from Alfred North Whitehead in Footnote 7 comes from his discussion in Chapter V of Whitehead [13] of the value of symbols in mathematics. It appears just after his comparison of the symbolic law “ $x + y = y + x$ ” to its expression without symbols “If a second number be added to any given number the result is the same as if the first given number had been added to the second number”, and his conclusion that “by the aid of symbolism, we can make transitions in reasoning almost mechanically by the eye, which otherwise would call into play the higher faculties of the brain.”

The *logarithm* (to base 2) of what we are calling *multiplicative Bayes leakage* $\mathcal{L}_1^\times(\pi, C)$ has been called *min-entropy leakage* in the previous quantitative-information-flow literature (e.g. Espinoza and Smith [5]). The outer logarithm just changes the *scale* of the leakage values, for example changing multiplicative Bayes leakage of 1 to min-entropy leakage of 0. The former terminology was motivated by the fact that the negative logarithm of prior Bayes vulnerability is Rényi’s *min-entropy* [10]. But the negative logarithm of posterior Bayes vulnerability ($-\log_2 V_1[\pi \triangleright C]$) is *not* the expected min-entropy over the hyper-distribution $[\pi \triangleright C]$ – for which see §11.4 – and for this reason the name “Bayes leakage” seems preferable to “min-entropy leakage”.

The example program in Section 5.4, which shows that mutual information can be very misleading with respect to confidentiality, is from Smith [12].

There has been much interest in automated analysis of leakage in systems. Here we just mention that a variety of static-analysis techniques have been explored, including those based on type systems (e.g. Clark, Hunt, and Malacaria [3]), statistical sampling (e.g. Köpf and Rybalchenko [7]), model checking (e.g. Backes, Köpf, and Rybalchenko [2] and Newsome, McCamant, and Song [9]), and abstract interpretation (e.g. Mardziel et al. [8] and Doychev et al. [4]).

The adaptive adversaries considered in Exercise 5.7 have been investigated by Köpf and Basin [6].

Bibliography

- [1] Alvim, M.S., Chatzikokolakis, K., Palamidessi, C., Smith, G.: Measuring information leakage using generalized gain functions. In: Proceedings of the 2012 IEEE 25th Computer Security Foundations Symposium (CSF 2012), pp. 265–279. IEEE, Los Alamitos (2012)
- [2] Backes, M., Köpf, B., Rybalchenko, A.: Automatic discovery and quantification of information leaks. In: Proceedings of the 2009 IEEE Symposium on Security and Privacy, pp. 141–153. IEEE, Los Alamitos (2009)
- [3] Clark, D., Hunt, S., Malacaria, P.: A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security* **15**(3), 321–371 (2007)
- [4] Doychev, G., Feld, D., Köpf, B., Mauborgne, L., Reineke, J.: Cacheaudit: A tool for the static analysis of cache side channels. In: Proceedings of the 22nd USENIX Security Symposium, pp. 431–446. USENIX Association, Berkeley (2013)
- [5] Espinoza, B., Smith, G.: Min-entropy as a resource. *Information and Computation* **226**, 57–75 (2013)
- [6] Köpf, B., Basin, D.: An information-theoretic model for adaptive side-channel attacks. In: Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS 2007), pp. 286–296. ACM, New York (2007)
- [7] Köpf, B., Rybalchenko, A.: Approximation and randomization for quantitative information-flow analysis. In: Proceedings of the 2010 IEEE 23rd Computer Security Foundations Symposium (CSF 2010), pp. 3–14. IEEE, Los Alamitos (2010)
- [8] Mardziel, P., Magill, S., Hicks, M., Srivatsa, M.: Dynamic enforcement of knowledge-based security policies. In: Proceedings of the 2011 IEEE 24th Computer Security Foundations Symposium (CSF 2011), pp. 114–128. IEEE, Los Alamitos (2011)
- [9] Newsome, J., McCamant, S., Song, D.: Measuring channel capacity to distinguish undue influence. In: Proceedings of the 4th Workshop on Programming Languages and Analysis for Security (PLAS 2009), pp. 73–85. ACM, New York (2009)
- [10] Rényi, A.: On measures of entropy and information. In: Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics, pp. 547–561. University of California Press, Berkeley (1961)

Bibliography

- [11] Rockafellar, R.T.: Convex analysis. Princeton Mathematical Series. Princeton University Press, Princeton (1970)
- [12] Smith, G.: On the foundations of quantitative information flow. In: L. de Alfaro (ed.) Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS 2009), *Lecture Notes in Computer Science*, vol. 5504, pp. 288–302. Springer, Berlin (2009)
- [13] Whitehead, A.N.: An Introduction to Mathematics. Williams and Norgate, London (1911)



Chapter 6

Robustness

Given a channel with input X , we have now seen that g -leakage provides us with a rich variety of ways to measure the information leakage of X that the channel causes. The prior models the adversary's *prior knowledge* about X ; the gain function g models the *operational scenario*, which encompasses both the set of *actions* that the adversary can take and also the *worth* to the adversary of each such action, for each possible value of X ; and the choice of *multiplicative*- or *additive* leakage allows us to measure either the relative or the absolute increase in g -vulnerability.

There is clear benefit in having such an extensive vocabulary, as it allows us to achieve precise *operational significance* in our leakage assessments. But this same richness brings worries about the *robustness* of our leakage assessments, and that is what we consider in this chapter.

6.1 The need for robustness

What do we know about the adversary's prior knowledge about X ? What actions might she take, and how valuable might they be to her? It is often said that some adversarial behavior is “unlikely in practice” — but in the context of security it is precisely that kind of thinking that can be dangerous. Indeed, anything that we think is “unlikely in practice” is arguably *more* likely, since adversaries are *thinking about what we are thinking*, and will be trying to exploit it. Another issue is that a channel might be developed with a certain operating context in mind, but (as is typical of software) it might later be migrated to a different operating context where the original assumptions do not hold. As a result, if we calculate say multiplicative leakage $\mathcal{L}_g^{\times}(\pi, C)$ for a certain g and π and C , and decide that it is acceptably small, how confident can we be that C really is safe to deploy? With respect to the prior π in particular, the adversary might have knowledge of which we are unaware. (In the context of passwords, for example, large-scale studies have shown that user-selected passwords are not at all uniform.) Hence analyzing channels with respect to (say) a uniform prior might well give a misleading view of the risks that they pose.

As an example, suppose that $\mathcal{X} = \{0, 1, 2, \dots, 9999\}$ and let channel A be

```
Y := if X=0 then 1 else X
```

and let channel B be

```
Y := if X=0 then 0 else 1 .
```

6 Robustness

If we assume a uniform prior ϑ , then we can easily compute the multiplicative Bayes leakages of A and B using Thm. 5.17: they are

$$\mathcal{L}_1^{\times}(\vartheta, A) = 9999 \quad \text{and} \quad \mathcal{L}_1^{\times}(\vartheta, B) = 2 \quad .$$

As a result, we might decide that A leaks much more than B and that replacing A with B would be a good idea.

But suppose that the adversary somehow knows that the value of X is actually either 0 or 1, each with probability $1/2$, so that the prior is *not* uniform; rather it is $\pi^n = (1/2, 1/2, 0, 0, \dots, 0)$. Now observe that A 's output is always 1, which means that it leaks *nothing* about X , while B reveals the value of X exactly: thus we now have

$$\mathcal{L}_1^{\times}(\vartheta, A) = 1 \quad \text{and} \quad \mathcal{L}_1^{\times}(\vartheta, B) = 2 \quad .$$

Hence replacing A with B might actually be a decision that we would regret.

The question of robustness arises also for the gain function g — suppose now that X is a 64-bit unsigned integer (so that $\mathcal{X} = \{0, 1, 2, \dots, 2^{64}-1\}$) and let C be the channel discussed in §5.4:

```
 $Y := \text{if } (X \bmod 8) = 0 \text{ then } X \text{ else } 1$ 
```

whereas channel D is

```
 $Y := X \mid 0x7 \quad ,$ 
```

with “ $\mid 0x7$ ” denoting *bitwise or* with 7 (written as a hexadecimal constant). Then D copies the first 61 bits of X into Y , but “masks out” the last 3 bits.

Assuming a uniform prior ϑ , we can again use Thm. 5.17 to compute the multiplicative Bayes leakage by counting the number of possible output values, and we find that

$$\mathcal{L}_1^{\times}(\vartheta, C) = 2^{61} + 1 \quad \text{and} \quad \mathcal{L}_1^{\times}(\vartheta, D) = 2^{61} \quad .$$

Hence C and D have almost exactly the same multiplicative Bayes leakage. But are they (almost) equally secure?

One might have the impression that channel D is less secure, since it *always* leaks the first 61 bits of X , while channel C usually leaks almost nothing: whenever X is not a multiple of 8, channel C outputs 1, which reveals only that X is not a multiple of 8 — meaning that $7/8 \cdot 2^{64}$ values remain possible. Indeed if we evaluate leakage using the *three-tries* gain function V_3 (instead of the “one-try” V_1) discussed in §3.2.3.3, we find that the posterior vulnerability for D increases from $1/8$ to $3/8$, while the posterior vulnerability for C is hardly increased at all — in the case when $Y \neq 1$, the adversary already knows the value of X and does not benefit from three guesses, while in the case when $Y=1$ she just gets “three stabs in the dark” among the $7/8 \cdot 2^{64}$ possible values, giving a negligible increase in posterior vulnerability. Hence the three-tries gain function makes D leak more than C .

But with gain function g_{tiger} from §3.2.5, the situation is reversed. With channel D , the posterior vulnerability is 0, since a $1/8$ probability of guessing correctly is not high enough to overcome the risk of being eaten by tigers. But with channel C , the posterior vulnerability is greater than 0: whenever $Y \neq 1$, the adversary *knows* the value of X , making it safe for her to guess then.¹ Hence g_{tiger} makes channel C leak more than channel D .

¹ Unless she is prone to typos!

6.2 Approaches to robustness

The previous section makes clear that analyzing a channel with respect to a particular prior and a particular gain function is *not enough* to understand its information leakage in general. For this reason, there has been considerable interest in achieving robustness in quantitative-information-flow analysis, and several fruitful approaches have been developed.

One important approach is to consider *capacity*, by which we mean the *maximum* leakage (of either kind) over all priors π and/or over all gain functions g ; such a capacity analysis enables us to conclude that a channel's leakage is *no worse* than some amount, over some range of operational scenarios. In Chap. 7 we will consider a total of six capacity scenarios, based on whether we maximize over just the prior, over just the gain function, or over both the prior and the gain function; and on whether we measure leakage multiplicatively or additively.

A second approach to robustness concerns the *comparison* of channels, aimed at showing that one channel *never* leaks more than another, regardless of the operational scenario. We find that there is a structural *refinement* order (\sqsubseteq) that coincides with this strong leakage ordering. Moreover (\sqsubseteq) turns out to be a *partial order* on abstract channels. The theory of refinement is developed in Chap. 9, preceded by a preparatory discussion of channel compositions in Chap. 8.

A third perspective on robustness considers that a channel that leaks information about a secret X may have the surprising “collateral” effect of leaking information about a *different* secret Z , and that can happen because X and Z may turn out to be *correlated*. Such correlations might be discovered at any time, as when medical research determines a correlation between diet and susceptibility to disease — hence while learning someone’s favorite dish might be relatively harmless today, it might not be so in the future. We refer to such leakage as *Dalenius leakage*, and we find that it is possible to prove upper bounds on the Dalenius leakage of a channel, regardless of any correlations that might be discovered later. The theory of Dalenius leakage is developed in Chap. 10.

6.3 Exercises

Exercise 6.1 Recall the dice channels C and D from §1.1, whose input is the value (r, w) resulting from throwing a red die and a white die and defined by $C(r, w) := r + w$ and $D(r, w) := r \cdot w$. Recall that with *fair* dice, C ’s multiplicative Bayes leakage is 11, while D ’s is 18. Show that with *biased* dice, it is possible to make C ’s multiplicative Bayes leakage *exceed* D ’s. \square

6.4 Chapter notes

An analysis showing that user-selected passwords are highly non-uniform appears in Bonneau [1].

Bibliography

- [1] Bonneau, J.: The science of guessing: analyzing an anonymized corpus of 70 million passwords. In: Proceedings of the 2012 IEEE Symposium on Security and Privacy, pp. 538–552. IEEE, Los Alamitos (2012)



Chapter 7

Capacity

In this chapter we develop the theory of *channel capacity*.

In Def. 5.11 we considered for a fixed prior π and gain function g the idea of comparing prior and posterior vulnerabilities. As we noted in §6.2 however, in general one cannot be sure of the π 's and g 's our channels might face in practice; and so it is worth asking “What is the *worst* leakage that we, as defenders, might have to deal with if we use this channel?”

There are two dimensions over which “worst” might vary. The first is the priors: rather than just one, i.e. some particular π , we might want to consider them all — the whole set $\mathbb{D}\mathcal{X}$. Or we might consider not all of them, but still more than just the one: some subset \mathcal{D} of $\mathbb{D}\mathcal{X}$. The second dimension of variation is the gain functions: rather than just some particular g , we might consider all gain functions $\mathbb{G}\mathcal{X}$, or some subset \mathcal{G} of them. We will see below that there are interesting limits to the leakage, whether multiplicative or additive, that are induced by the variations we allow in the priors, or in the gain functions, or both. Moreover, questions about how to *compute* the various capacities of a given channel matrix C turn out to be interesting and subtle in themselves — as we will see, certain capacities can be computed efficiently, while others cannot.

7.1 Multiplicative Bayes capacity

We begin by developing the theory of *multiplicative Bayes capacity*, by which we mean the maximum multiplicative Bayes leakage over all prior distributions π . To notate that, we add “ \mathcal{M} ” to our usual notation “ \mathcal{L}_1 ” to indicate that we are maximizing, and write \mathbb{D} instead of π to indicate that the prior varies over all distributions in $\mathbb{D}\mathcal{X}$. Note that both the channel and the gain function are fixed, the latter because it is *Bayes capacity* we are examining: only the prior varies.

Definition 7.1

The *multiplicative Bayes capacity* of channel C , denoted $\mathcal{MC}_1^{\times}(\mathbb{D}, C)$, is the maximum multiplicative Bayes leakage over all priors:

$$\mathcal{MC}_1^{\times}(\mathbb{D}, C) := \sup_{\pi \in \mathbb{D}\mathcal{X}} \mathcal{L}_1^{\times}(\pi, C) .$$

Of course, it is not immediately clear how $\mathcal{ML}_1^{\times}(\mathbb{D}, C)$ could be computed, since its definition involves a supremum over the infinite set $\mathbb{D}\mathcal{X}$. But it turns out that multiplicative Bayes capacity is always realized on a *uniform prior*, and hence (by Thm. 5.17) it is easy to compute it by taking any channel matrix C for C , and then summing the column maximums of that.¹

Theorem 7.2

For any channel C , the multiplicative Bayes capacity $\mathcal{ML}_1^{\times}(\mathbb{D}, C)$ is always realized on a uniform prior π . Moreover, for channel matrices we have that $\mathcal{ML}_1^{\times}(\mathbb{D}, C) = \sum_{y \in \mathcal{Y}} \max_{x \in \mathcal{X}} C_{x,y}$, where C is a concrete channel corresponding to the abstract channel C .

Proof. Consider an arbitrary channel matrix C such that $\llbracket C \rrbracket = C$. Observe that for any prior π , we have

$$\begin{aligned} & \mathcal{L}_1^{\times}(\pi, C) \\ = & V_1[\pi \triangleright C] / V_1(\pi) \\ = & \sum_{y \in \mathcal{Y}} \max_{x \in \mathcal{X}} (\pi_x C_{x,y}) / \max_{x \in \mathcal{X}} \pi_x \\ \leq & \sum_{y \in \mathcal{Y}} \max_{x \in \mathcal{X}} (\max_{x \in \mathcal{X}} \pi_x) C_{x,y} / \max_{x \in \mathcal{X}} \pi_x \\ = & \sum_{y \in \mathcal{Y}} \max_{x \in \mathcal{X}} C_{x,y} . \end{aligned} \quad \text{“Thm. 5.15”}$$

The upper bound is clearly realized when π is uniform, because in that case all π_x 's are equal and the third step above becomes an equality. (Note however that the upper bound can also be realized on a non-uniform π , provided that some proper subset of the rows of C includes at least one maximum from each column.) \square

Theorem 7.2 has two useful corollaries.

Corollary 7.3 If C is deterministic, then $\mathcal{ML}_1^{\times}(\mathbb{D}, C)$ is the number of possible output values of C , where an output value is *possible* just when its corresponding column contains at least one nonzero entry.

Proof. If C is deterministic, then all of its entries are either 0 or 1. Hence the sum of its column maximums is simply the number of columns containing at least one nonzero entry, which is the number of possible output values. \square

This corollary is quite useful for automated leakage analysis of a deterministic program, because it says that in that case the multiplicative Bayes capacity can be computed simply by counting the number of distinct outputs that the program can produce; and that can be done using a variety of static-analysis techniques.

Corollary 7.4 $\mathcal{ML}_1^{\times}(\mathbb{D}, C) = 1$ iff the rows of C are identical.

Proof. By Thm. 7.2, the multiplicative Bayes capacity of C is 1 iff the sum of its column maximums is 1. But each row of C sums to 1, so after considering the first row of C the sum of the column maximums is already 1. If any subsequent row differs at all from the earlier rows, it increases the sum of the column maximums above 1. On the other hand, if the rows of C are identical, then the sum of the column maximums is 1 exactly. \square

¹ Recall that abstract channels are written as C and concrete channels, i.e. stochastic matrices, as C . Generally in this chapter we will switch freely between the abstract and concrete formulations for channels, using concrete when aspects of a matrix are appealed to (e.g. columns) and abstract when the matrix form is not necessary.

Note that if the rows of C are identical then the output Y of C is completely independent of the input X , meaning that C satisfies *noninterference*, and reducing C yields a matrix with a single column all of whose entries are 1. That is, C is the channel $\mathbf{1}$ which leaks nothing. The “if” direction of Cor. 7.4 is therefore unsurprising. But the “only if” direction is more interesting: it says the *only* way for a channel to have no Bayes leakage on a uniform prior is for it to satisfy noninterference.

Multiplicative Bayes capacity $\mathcal{ML}_1^{\times}(\mathbb{D}, C)$ also has very important relationships with *other* leakage measures, relationships that make it a very robust measure of the worst-case leakage of C . Most remarkably, it turns out that it is an *upper bound* on the multiplicative g -leakage of C , regardless of the prior π and *non-negative* gain function g .

Theorem 7.5 (“Miracle”)

For any channel C , prior π , and non-negative gain function $g: \mathbb{G}^+ \times \mathcal{X}$, we have

$$\mathcal{L}_g^{\times}(\pi, C) \leq \sum_{y: \mathcal{Y}} \max_{x: \mathcal{X}} C_{x,y} = \mathcal{ML}_1^{\times}(\mathbb{D}, C).$$

Proof. In the following, variables x, y, w range over $\mathcal{X}, \mathcal{Y}, \mathcal{W}$ in the corresponding sums, max’s, and sup’s; the sets themselves are omitted for brevity. We have that

$$\begin{aligned} & V_g[\pi \triangleright C] \\ = & \sum_y \sup_w \sum_x g(w, x) \pi_x C_{x,y} && \text{“Thm. 5.7, for infinite } \mathcal{W} \text{”} \\ \leq & \sum_y \sup_w \sum_x g(w, x) \pi_x (\max_x C_{x,y}) && \text{“} g(w, x) \pi_x C_{x,y} \leq g(w, x) \pi_x (\max_x C_{x,y}) \\ & && \text{since } g(w, x) \pi_x \geq 0 \text{”} \\ = & (\sup_w \sum_x \pi_x g(w, x)) \left(\sum_y \max_x C_{x,y} \right) && \text{“} \sum_i c x_i = c \sum_i x_i \text{ and } \sup_i c x_i = c \sup_i x_i \text{ for } c \geq 0 \text{”} \\ = & V_g(\pi) \mathcal{ML}_1^{\times}(\mathbb{D}, C) && \text{“Thm. 7.2”} \end{aligned}$$

Note that the “ \leq ” step, above, uses crucially the assumption that $g(w, x) \geq 0$, i.e. that g is a non-negative gain function. Finally, we have

$$\mathcal{L}_g^{\times}(\pi, C) = \frac{V_g[\pi \triangleright C]}{V_g(\pi)} \leq \frac{V_g(\pi) \mathcal{ML}_1^{\times}(\mathbb{D}, C)}{V_g(\pi)} = \mathcal{ML}_1^{\times}(\mathbb{D}, C).$$

□

Note that a generic gain function $g: \mathbb{G} \times \mathcal{X}$ is allowed to take negative gain values (as long as the induced V_g is always non-negative). It is interesting to observe that the Miracle theorem can actually *fail* if g is somewhere negative, as this example shows.

Example 7.6 Recall from §3.2.5 the gain function $g_{\text{tiger}}: \mathbb{G} \times \mathcal{X}$ for $\mathcal{X} = \{x_1, x_2\}$ on an “almost” uniform prior distribution $\pi = (0.5 + \epsilon, 0.5 - \epsilon)$ for some $\epsilon > 0$. In this case the best action is to guess x_1 , and the expected gain is then $(0.5 + \epsilon) \cdot 1 + (0.5 - \epsilon) \cdot (-1) = 2\epsilon$, so that $V_{g_{\text{tiger}}}(\pi) = 2\epsilon$.

Now let C be the following channel matrix, which gives rather good information about the secret:

C	y_1	y_2
x_1	0.8	0.2
x_2	0.2	0.8

7 Capacity

Using Thm. 5.18 we calculate the posterior g_{tiger} -vulnerability as the sum of the column maximums of the matrix $\mathbf{G} \cdot \mathbf{J}$ as given below:²

$$\begin{array}{|c|cc|} \hline \mathbf{G} & x_1 & x_2 \\ \hline x_1 & 1 & -1 \\ \perp & 0 & 0 \\ x_2 & -1 & 1 \\ \hline \end{array} \cdot \begin{array}{|c|cc|} \hline \mathbf{J} & y_1 & y_2 \\ \hline x_1 & 0.8(0.5 + \epsilon) & 0.2(0.5 + \epsilon) \\ x_2 & 0.2(0.5 - \epsilon) & 0.8(0.5 - \epsilon) \\ \hline \end{array} = \begin{array}{|c|cc|} \hline \mathbf{GJ} & y_1 & y_2 \\ \hline x_1 & 0.3 + \epsilon & -0.3 + \epsilon \\ \perp & 0 & 0 \\ x_2 & -0.3 - \epsilon & 0.3 - \epsilon \\ \hline \end{array} .$$

Hence $V_{g_{\text{tiger}}}[\pi \triangleright \mathbf{C}] = 0.3 + \epsilon + 0.3 - \epsilon = 0.6$ and

$$\mathcal{L}_{g_{\text{tiger}}}^{\times}(\pi, \mathbf{C}) = \frac{0.6}{2\epsilon},$$

which can be arbitrarily large as ϵ approaches 0. Hence it can be arbitrarily larger than the multiplicative Bayes capacity, which is equal to

$$\mathcal{ML}_1^{\times}(\mathbb{D}, \mathbf{C}) = 0.8 + 0.8 = 1.6.$$

□

Yet another interesting aspect of multiplicative Bayes capacity is its relationship with *Shannon capacity*, by which we mean the maximum mutual information over all priors.

Theorem 7.7 If \mathbf{C} is deterministic, then its Shannon capacity is equal to the logarithm of its multiplicative Bayes capacity.

Proof. Given a channel matrix \mathbf{C} from \mathcal{X} to \mathcal{Y} and a prior π , the mutual information $I(X; Y)$ is given by

$$I(X; Y) = H(X) - H(X|Y).$$

But we know that mutual information is symmetric, which means that also

$$I(X; Y) = I(Y; X) = H(Y) - H(Y|X).$$

Now observe that in the case when \mathbf{C} is deterministic, because $H(Y|X) = 0$ we can conclude $I(X; Y) = H(Y)$.

Suppose further that deterministic \mathbf{C} has m columns (after deleting any all-zero columns). In this case, recalling Cor. 7.3 above, we know that $\mathcal{ML}_1^{\times}(\mathbb{D}, \mathbf{C}) = m$. And here the Shannon capacity of \mathbf{C} is the maximum value of $H(Y)$ over all priors π . That maximum is just $\log_2 m$, since Y has m possible values and we can construct a π that makes them all equally likely. Hence the Shannon capacity is equal to $\log_2 \mathcal{ML}_1^{\times}(\mathbb{D}, \mathbf{C})$. □

In the general case of probabilistic channels, we still find that the logarithm of multiplicative Bayes capacity gives at least an *upper bound* on Shannon capacity.

Theorem 7.8 For any channel C , the logarithm of C 's multiplicative Bayes capacity is at least as great as its Shannon capacity.

² Recall that, as discussed in §4.2, the joint matrix \mathbf{J} is defined by $\mathbf{J} = \pi \triangleright \mathbf{C}$, so that $J_{x,y} = \pi_x C_{x,y}$. Also, the special action \perp is used in g_{tiger} to indicate “Don't make a guess at all.”

Proof. The argument makes crucial use of *Jensen's inequality*, which says that if f is a concave (\curvearrowright) function, and a_1, a_2, \dots, a_n are convex coefficients (non-negative reals that sum to 1), and x_1, x_2, \dots, x_n are arbitrary, then

$$\sum_i a_i f(x_i) \leq f\left(\sum_i a_i x_i\right).$$

Let $C: \mathcal{X} \rightarrow \mathcal{Y}$ be a channel matrix for C , and let prior π on \mathcal{X} be arbitrary. As usual we get joint distribution $p(x, y) := \pi_x C_{x,y}$, and we reason as follows:

$$\begin{aligned} & I(X; Y) \\ = & H(Y) - H(Y|X) \\ = & -\sum_y p(y) \log_2 p(y) + \sum_x p(x) \sum_y p(y|x) \log_2 p(y|x) \\ = & -\sum_y \sum_x p(x, y) \log_2 p(y) + \sum_x \sum_y p(x, y) \log_2 p(y|x) \\ = & \sum_{x,y} p(x, y) \log_2 p(y|x)/p(y) \\ \leq & \log_2 \sum_{x,y} p(x, y) \log_2 p(y|x)/p(y) && \text{"by Jensen's inequality and the concavity of } \log_2 \text{"} \\ = & \log_2 \sum_{x,y} p(x|y)p(y|x) \\ \leq & \log_2 \sum_{x,y} p(x|y)(\max_x p(y|x)) \\ = & \log_2 \sum_y (\max_x p(y|x)) \sum_x p(x|y) \\ = & \log_2 \max_x \sum_y C_{x,y} && \text{"since } \sum_x p(x|y) = 1 \text{ for any } y \text{"} \\ = & \log_2 \mathcal{ML}_1^{\times}(\mathbb{D}, C). \end{aligned}$$

Because this inequality holds for every prior π , it follows that

$$\text{Shannon capacity of } C = \sup_{\pi \in \mathbb{D}^{\mathcal{X}}} I(X; Y) \leq \log_2 \mathcal{ML}_1^{\times}(\mathbb{D}, C).$$

□

7.2 Additive Bayes capacity

Now we turn our attention to additive Bayes capacity.

Definition 7.9

The *additive Bayes capacity* of channel C , denoted $\mathcal{ML}_1^+(\mathbb{D}, C)$, is the maximum additive Bayes leakage over all priors π :

$$\mathcal{ML}_1^+(\mathbb{D}, C) := \sup_{\pi \in \mathbb{D}^{\mathcal{X}}} \mathcal{L}_1^+(\pi, C).$$

Interestingly, it turns out that additive Bayes capacity is far harder to compute than multiplicative Bayes capacity (which by Thm. 7.2 is just the sum of the column maximums of the channel matrix). The key challenge is that additive Bayes capacity need not be realized on a uniform prior, as the following example shows.

Example 7.10 On channel matrix

C	y ₁	y ₂	y ₃
x ₁	0	1/2	1/2
x ₂	1/2	0	1/2
x ₃	1/2	1/2	0

the additive Bayes leakage on the uniform prior $\vartheta = (1/3, 1/3, 1/3)$ is 1/6, as shown here:

$$\mathcal{L}_1^+(\vartheta, C) = V_1[\vartheta \triangleright C] - V_1(\vartheta) = 1/2 - 1/3 = 1/6.$$

But the additive Bayes leakage on the non-uniform prior $\pi = (1/2, 1/2, 0)$ is higher, at $1/4$ as shown here:

$$\mathcal{L}_1^+(\pi, C) = V_1[\pi \triangleright C] - V_1(\pi) = 3/4 - 1/2 = 1/4 .$$

(In fact we can verify, using Thm. 7.11 below, that no prior can give greater additive Bayes leakage than π does — and so $\mathcal{ML}_1^+(\mathbb{D}, C)$ is $1/4$.) \square

However, it turns out that additive Bayes capacity is always realized on some *sub-uniform distribution*, meaning a distribution that is uniform on its support. (Note that π in Example 7.10 above is a sub-uniform distribution.)

Theorem 7.11 —

For any channel C , the additive Bayes capacity $\mathcal{ML}_1^+(\mathbb{D}, C)$ is always realized on a sub-uniform distribution.

Proof. Recall that, as proved in Thm. 3.13 and Thm. 5.9, both prior and posterior Bayes vulnerability $V_1(\pi)$ and $V_1[\pi \triangleright C]$ are *convex* functions of π . Hence if we express π as a convex combination of distributions $\delta^1, \delta^2, \dots, \delta^m$, so that $\pi = \sum_{i=1}^m a_i \delta^i$, where a_1, a_2, \dots, a_m are non-negative reals that sum to 1, then the two convexity properties imply that

$$V_1(\pi) \leq \sum_{i=1}^m a_i V_1(\delta^i)$$

and

$$V_1[\pi \triangleright C] \leq \sum_{i=1}^m a_i V_1[\delta^i \triangleright C] .$$

Now note that if the first inequality went the *other way*, then the inequalities would give us bounds on $V_1[\pi \triangleright C] - V_1(\pi)$. But, as it is, they do not.

To make progress, we now argue that any π can be represented as a convex combination of *sub-uniform distributions* $\delta^1, \delta^2, \dots, \delta^m$ in such a way that the first inequality becomes an *equality*:

$$V_1(\pi) = \sum_{i=1}^m a_i V_1(\delta^i) .$$

We start with a non-example. If $\pi = (7/10, 1/10, 0, 2/10)$, then we can trivially represent it as a convex combination of point distributions (which are of course sub-uniform):

$$\pi = 7/10 \cdot (1, 0, 0, 0) + 1/10 \cdot (0, 1, 0, 0) + 2/10 \cdot (0, 0, 0, 1) .$$

But that does not work, because

$$7/10 \cdot V_1(1, 0, 0, 0) + 1/10 \cdot V_1(0, 1, 0, 0) + 2/10 \cdot V_1(0, 0, 0, 1) = 7/10 + 1/10 + 2/10 = 1 ,$$

while in fact $V_1(\pi) = 7/10$.

Instead the desired representation is found by first selecting a sub-uniform distribution with support as large as possible, and scaling it to eliminate as many of the nonzero entries in π as possible. The process then repeats with the remaining part of π , until no nonzero entries are left.

For example, on $\pi = (7/10, 1/10, 0, 2/10)$ we first select the sub-uniform distribution $(1/3, 1/3, 0, 1/3)$ and scale it with $3/10$ to eliminate the smallest nonzero entry of π :

$$(7/10, 1/10, 0, 2/10) - 3/10 \cdot (1/3, 1/3, 0, 1/3) = (6/10, 0, 0, 1/10) .$$

Next we select the sub-uniform distribution $(1/2, 0, 0, 1/2)$ and scale it with $2/10$ to eliminate the smallest remaining nonzero entry:

$$(6/10, 0, 0, 1/10) - 2/10 \cdot (1/2, 0, 0, 1/2) = (5/10, 0, 0, 0) .$$

Finally we select the sub-uniform distribution $(1, 0, 0, 0)$ and scale it with $5/10$ to eliminate the last nonzero entry:

$$(5/10, 0, 0, 0) - 5/10 \cdot (1, 0, 0, 0) = (0, 0, 0, 0) .$$

The result is that

$$(7/10, 1/10, 0, 2/10) = 3/10 \cdot (1/3, 1/3, 0, 1/3) + 2/10 \cdot (1/2, 0, 0, 1/2) + 5/10 \cdot (1, 0, 0, 0) .$$

The key point is that in each step the scaling factor times the Bayes vulnerability of the sub-uniform distribution is exactly the amount that is eliminated in that step from each of the remaining nonzero entries of π . Thus in the example we have

$$3/10 \cdot V_1(1/3, 1/3, 0, 1/3) + 2/10 \cdot V_1(1/2, 0, 0, 1/2) + 5/10 \cdot V_1(1, 0, 0, 0) = 7/10 ,$$

and therefore in general $V_1(\pi) = \sum_{i=1}^m a_i V_1(\delta^i)$ if the sub-uniform distributions δ^i are selected that way.

We now can reason as follows:

$$\begin{aligned} & \mathcal{L}_1^+(\pi, C) \\ = & V_1[\pi \triangleright C] - V_1(\pi) \\ = & V_1\left[\sum_i a_i \delta^i \triangleright C\right] - V_1\left(\sum_i a_i \delta^i\right) && \text{"representation of } \pi\text{"} \\ \leq & \sum_i a_i V_1[\delta^i \triangleright C] - V_1\left(\sum_i a_i \delta^i\right) && \text{"convexity of posterior Bayes vulnerability"} \\ = & \sum_i a_i V_1[\delta^i \triangleright C] - \sum_i a_i V_1(\delta^i) && \text{"special property of chosen representation"} \\ = & \sum_i a_i (V_1[\delta^i \triangleright C] - V_1(\delta^i)) \\ = & \sum_i a_i \mathcal{L}_1^+(\delta^i, C) \\ \leq & \sum_i a_i \mathcal{L}_1^+(\delta^k, C) && \text{"choosing } k \text{ so that } \mathcal{L}_1^+(\delta^k, C) \text{ is maximal"} \\ = & \mathcal{L}_1^+(\delta^k, C) . \end{aligned}$$

Hence we have shown that for any prior π , there exists a sub-uniform distribution δ^k such that $\mathcal{L}_1^+(\pi, C) \leq \mathcal{L}_1^+(\delta^k, C)$. Since there are only *finitely many* sub-uniform distributions, it follows that $\mathcal{ML}_1^+(\mathbb{D}, C)$ is realized on a sub-uniform distribution. \square

Theorem 7.11 implies that the additive Bayes capacity of C is computable. For if $|\mathcal{X}| = n$, then there are $2^n - 1$ sub-uniform distributions (we just need to consider the nonempty subsets of \mathcal{X} for the support), and we can simply compute the additive Bayes leakage on *all* of them. But because $2^n - 1$ is exponential in the size of C , that is not an efficient algorithm. And the following theorem shows that an efficient algorithm probably does not exist: it is NP-complete to decide whether C 's additive Bayes capacity exceeds a given threshold.³

Theorem 7.12

Given a channel matrix C and a threshold t , it is NP-complete to decide whether $\mathcal{ML}_1^+(\mathbb{D}, C) \geq t$.

³ It is worth noting that this theorem is fundamentally different from similar results for channels represented as *programs*, which inevitably leads to intractability of analysis because of the difficulty of figuring out what the program's behavior is in the first place. In contrast, this theorem is based on the *channel matrix*, which gives the channel's behavior explicitly.

Proof. Before beginning the proof, we establish some basic properties and notation for sub-uniform distributions. Given a nonempty subset \mathcal{S} of \mathcal{X} , the sub-uniform distribution $\pi^{\mathcal{S}}$ gives probability $1/|\mathcal{S}|$ to the values in \mathcal{S} and (therefore) gives probability 0 to the values in $\mathcal{X} - \mathcal{S}$. Now observe that

$$\begin{aligned} & \mathcal{L}_1^+(\pi^{\mathcal{S}}, \mathbf{C}) \\ = & V_1[\pi^{\mathcal{S}} \triangleright \mathbf{C}] - V_1(\pi^{\mathcal{S}}) \\ = & \sum_{y \in \mathcal{Y}} \max_{x \in \mathcal{X}} \pi_x^{\mathcal{S}} C_{x,y} - \max_{x' \in \mathcal{X}} \pi_{x'}^{\mathcal{S}} \\ = & \sum_{y \in \mathcal{Y}} \max_{x \in \mathcal{S}} \frac{1}{|\mathcal{S}|} C_{x,y} - \frac{1}{|\mathcal{S}|} \\ = & \frac{1}{|\mathcal{S}|} \left(\sum_{y \in \mathcal{Y}} \max_{x \in \mathcal{S}} C_{x,y} - 1 \right) . \end{aligned}$$

Notice that $\sum_{y \in \mathcal{Y}} \max_{x \in \mathcal{S}} C_{x,y}$ is simply the *sum of the column maximums of \mathbf{C} , restricted to the rows in \mathcal{S}* . To simplify notation we let $\sigma_{\mathcal{S}}$ abbreviate that sum, allowing us to write

$$\mathcal{L}_1^+(\pi^{\mathcal{S}}, \mathbf{C}) = \frac{\sigma_{\mathcal{S}} - 1}{|\mathcal{S}|} . \quad (7.1)$$

Now we proceed with the proof. By the above result that additive Bayes capacity is realized on a sub-uniform distribution, it is easy to see that the decision problem “Is $\mathcal{ML}_1^+(\mathbb{D}, \mathbf{C}) \geq t$? ” is in NP. For we can simply guess a nonempty subset \mathcal{S} and verify that $\mathcal{L}_1^+(\pi^{\mathcal{S}}, \mathbf{C}) \geq t$.

We complete the NP-completeness proof by showing a reduction from the classical *Set-Packing* problem, which is to decide whether a collection of sets $\mathcal{U} = \{U_1, \dots, U_n\}$ contains k pairwise disjoint sets.

Suppose that the elements of the U_i are drawn from universe $\{a_1, \dots, a_m\}$. We begin our reduction by constructing an $n \times m$ channel matrix $\mathbf{C}^{\mathcal{U}}$ where

$$C_{i,j}^{\mathcal{U}} := \begin{cases} \frac{1}{|U_i|} & \text{if } a_j \in U_i \\ 0 & \text{otherwise} \end{cases} . \quad ^4$$

Now notice that if \mathcal{U} contains k pairwise disjoint sets, $k \geq 1$, then $\mathbf{C}^{\mathcal{U}}$ contains k pairwise non-overlapping rows. This means that $\mathbf{C}^{\mathcal{U}}$ leaks *everything* about the corresponding k inputs, since any possible output from those inputs will uniquely determine the input. Hence the additive Bayes leakage on the sub-uniform distribution on those k inputs is k^{-1}/k , since the Bayes vulnerability is increased from $1/k$ to 1.

We can see this more formally from (7.1) above. For if we let \mathcal{S} be a set consisting of the indices of k pairwise disjoint sets in \mathcal{U} , then we have

$$\mathcal{L}_1^+(\pi^{\mathcal{S}}, \mathbf{C}^{\mathcal{U}}) = \frac{\sigma_{\mathcal{S}} - 1}{k} ,$$

where $\sigma_{\mathcal{S}}$ is the sum of the column maximums of $\mathbf{C}^{\mathcal{U}}$, restricted to the rows in \mathcal{S} . But, since the rows in \mathcal{S} are non-overlapping, we know that each nonzero entry in each of those rows is the *only* nonzero entry in its column (restricted to the rows in \mathcal{S}), and hence is included in the sum $\sigma_{\mathcal{S}}$. So, since each row sums to 1, we get that $\sigma_{\mathcal{S}} = k$, which gives

$$\mathcal{L}_1^+(\pi^{\mathcal{S}}, \mathbf{C}^{\mathcal{U}}) = \frac{k - 1}{k} .$$

⁴ Note that if $\emptyset \in \mathcal{U}$, we do not actually get a channel matrix, since \emptyset leads to an all-zero row. But in this case notice that \mathcal{U} contains k pairwise disjoint sets iff $\mathcal{U} - \{\emptyset\}$ contains $k-1$ pairwise disjoint sets.

So far we have shown that if \mathcal{U} contains k pairwise disjoint sets, then we have $\mathcal{ML}_1^+(\mathbb{D}, \mathbf{C}^{\mathcal{U}}) \geq k^{-1}/k$. But does the converse hold? Not necessarily, as shown by the counterexample

$\mathbf{C}^{\mathcal{U}}$	y_1	y_2	y_3	y_4	y_5	y_6	y_7
x_1	1	0	0	0	0	0	0
x_2	$1/4$	$1/4$	$1/4$	$1/4$	0	0	0
x_3	$1/4$	0	0	0	$1/4$	$1/4$	$1/4$

In this case, set \mathcal{U} does *not* contain 2 disjoint sets, but still $\mathcal{ML}_1^+(\mathbb{D}, \mathbf{C}^{\mathcal{U}}) \geq \frac{2-1}{2} = \frac{1}{2}$, as a uniform distribution on all 3 rows gives additive Bayes leakage of $(5/2-1)/3 = 1/2$.

To correct this difficulty, we need to convert \mathcal{U} into a channel matrix in a slightly more complicated way, one that boosts the “penalty” for including an overlapping row in a sub-uniform distribution.

To that end, let $p \geq 1$ be the maximum cardinality of any of the sets in \mathcal{U} .⁵ Let $\mathbf{C}^{\mathcal{U}}$ be embedded into a block matrix $\overline{\mathbf{C}^{\mathcal{U}}}$ with p new rows and p new columns:

$$\overline{\mathbf{C}^{\mathcal{U}}} := \begin{array}{|c|c|} \hline \mathbf{I}_{p \times p} & \mathbf{0}_{p \times m} \\ \hline \mathbf{0}_{n \times p} & \mathbf{C}^{\mathcal{U}} \\ \hline \end{array} .$$

(Here \mathbf{I} denotes the identity matrix, and $\mathbf{0}$ the all-zero matrix.)

Extending our previous calculation, we note that if \mathcal{U} contains k pairwise disjoint sets, then we have

$$\mathcal{ML}_1^+(\mathbb{D}, \overline{\mathbf{C}^{\mathcal{U}}}) \geq \frac{p+k-1}{p+k} ,$$

since the p new rows are all non-overlapping.

But we can now argue that the converse also holds. For if $\mathcal{ML}_1^+(\mathbb{D}, \overline{\mathbf{C}^{\mathcal{U}}}) \geq \frac{p+k-1}{p+k}$, then there exists a set \mathcal{S} of indices such that

$$\mathcal{L}_1^+(\pi^{\mathcal{S}}, \overline{\mathbf{C}^{\mathcal{U}}}) \geq p+k-1/p+k .$$

Assume further that we choose \mathcal{S} of *minimum size* that can achieve that additive Bayes leakage threshold. Then we can argue by contradiction that the rows in \mathcal{S} must be non-overlapping, because if \mathcal{S} contained a row that overlaps some other row of \mathcal{S} , then we could *delete* it without decreasing the additive leakage.

For suppose that $|\mathcal{S}| = s$ for $s \geq 2$, and that \mathcal{S}' is formed by deleting a row with some overlap with the other rows of \mathcal{S} . Recall that

$$\mathcal{L}_1^+(\pi^{\mathcal{S}}, \overline{\mathbf{C}^{\mathcal{U}}}) = \frac{\sigma_{\mathcal{S}} - 1}{s} ,$$

where $\sigma_{\mathcal{S}}$ is the sum of the column maximums of $\overline{\mathbf{C}^{\mathcal{U}}}$, restricted to the rows in \mathcal{S} . Now observe that when we pass from \mathcal{S} to \mathcal{S}' , we lose the contribution to $\sigma_{\mathcal{S}}$ from the deleted row; this loss is at most $p-1/p$, since the deleted row must have overlap of at least $1/p$ with the other rows of \mathcal{S} . Hence we have

$$\begin{aligned} & \mathcal{L}_1^+(\pi^{\mathcal{S}'}, \overline{\mathbf{C}^{\mathcal{U}}}) \\ = & (\sigma_{\mathcal{S}'} - 1)/(s-1) \\ \geq & (\sigma_{\mathcal{S}} - p-1/p - 1)/(s-1) \\ \geq & (\sigma_{\mathcal{S}} - p+k-1/p+k - 1)/(s-1) \end{aligned} \quad "k \geq 0"$$

⁵ In fact, the Set-Pack problem remains NP-complete even if all sets in \mathcal{U} have cardinality at most 3.

$$\begin{aligned}
 &\geq (\sigma_S - \sigma_{S-1}/s - 1)/(s-1) \\
 &= ((\sigma_{S-1})(s-1)/s)/(s-1) \\
 &= (\sigma_S - 1)/s \\
 &= \mathcal{L}_1^+(\pi^S, \overline{\mathcal{C}^U}) \quad .
 \end{aligned}
 \quad \text{“} \mathcal{L}_1^+(\pi^S, \overline{\mathcal{C}^U}) \geq p+k-1/p+k \text{”}$$

Having proved that the rows in S are non-overlapping, we now note that S must contain at least $p+k$ rows, since a non-overlapping set of size u gives additive leakage $(u-1)/u$, which is less than $p+k-1/p+k$ unless $u \geq p+k$. And, finally, at least k of the non-overlapping rows must come from \mathcal{C}^U , which implies that U contains k pairwise disjoint sets.

We have thus shown that

$$\mathcal{U} \text{ contains } k \text{ pairwise disjoint sets iff } \mathcal{ML}_1^+(\mathbb{D}, \overline{\mathcal{C}^U}) \geq \frac{p+k-1}{p+k} .$$

Hence indeed the Set-Packing problem polynomial-time reduces to the Additive Bayes-Capacity Threshold problem. \square

7.3 General capacities

We now consider capacity more generally. Since the leakage of a channel C depends on both the prior and the gain function, we can maximize it over a set of gain functions in $\mathbb{G}\mathcal{X}$ and a set of priors in $\mathbb{D}\mathcal{X}$. Thus we use \mathcal{G} for a set of gain functions and \mathcal{D} for a set of priors.

Definition 7.13 ($(\mathcal{G}, \mathcal{D})$ -capacity)

For classes $\mathcal{G} \subseteq \mathbb{G}\mathcal{X}$, $\mathcal{D} \subseteq \mathbb{D}\mathcal{X}$ and channel C , the multiplicative and additive $(\mathcal{G}, \mathcal{D})$ -capacities of C are given by

$$\begin{aligned}
 \mathcal{ML}_{\mathcal{G}}^{\times}(\mathcal{D}, C) &:= \sup_{g: \mathcal{G}, \pi: \mathcal{D}} \mathcal{L}_g^{\times}(\pi, C) \quad \text{and} \\
 \mathcal{ML}_{\mathcal{G}}^{+}(\mathcal{D}, C) &:= \sup_{g: \mathcal{G}, \pi: \mathcal{D}} \mathcal{L}_g^{+}(\pi, C) \quad .
 \end{aligned}$$

When we want to maximize only over π , for a fixed g , we can simply take $\mathcal{G} = \{g\}$, in which case we write $\mathcal{ML}_g^{\times}(\mathcal{D}, C)$ instead of $\mathcal{ML}_{\{g\}}^{\times}(\mathcal{D}, C)$ for brevity; similarly, when π is fixed we write $\mathcal{ML}_{\mathcal{G}}^{\times}(\pi, C)$. For specific classes, say $\mathcal{G} := \mathbb{G}\mathcal{X}$ and $\mathcal{D} := \mathbb{D}\mathcal{X}$, we write $\mathcal{ML}_{\mathbb{G}}^{\times}(\mathbb{D}, C)$ instead of $\mathcal{ML}_{\mathbb{G}\mathcal{X}}^{\times}(\mathbb{D}\mathcal{X}, C)$, again for brevity (since \mathcal{X} is always taken to be the input domain of C).

In the following sections we study several instantiations of $(\mathcal{G}, \mathcal{D})$ -capacity. We first fix g and maximize over π , then we fix π and maximize over g , and finally we maximize over both π and g . Measuring leakage both multiplicatively and additively, we thus get a total of six capacity scenarios. For \mathcal{D} we use a singleton $\{\pi\}$ or the whole of $\mathbb{D}\mathcal{X}$,⁶ while for \mathcal{G} we use either a singleton $\{g\}$ or one of the classes that we introduced in §3.3.

The various scenarios are briefly introduced in the remainder of this section. Then the three multiplicative capacities are discussed in detail in §7.4, and finally the three additive ones in §7.5.

⁶ While we do not consider them here, other subsets of $\mathbb{D}\mathcal{X}$ could also be of interest. For instance, we could write $\mathbb{D}_{\theta}\mathcal{X}$ for the set of sub-uniform distributions on \mathcal{X} , allowing Thm. 7.11 to be formulated elegantly as “ $\mathcal{ML}_1^+(\mathbb{D}, C) = \mathcal{ML}_1^+(\mathbb{D}_{\theta}, C)$ ”.

(g, \mathbb{D}) -capacities: fixed g , maximize over π

When g is g_{id} , this scenario corresponds to the Bayes capacities already discussed in §7.1 and §7.2. Also, we can construct a gain function $g_H: \mathbb{G}\mathcal{X}$ such that $\mathcal{ML}_{g_H}^+(\mathbb{D}, C)$ is the Shannon capacity,⁷ well known from information theory for expressing the maximum transmission rate achievable by the communication channel C — this is the reason why we generally refer to the maximization of leakage as “capacity”.

(\mathcal{G}, π) -capacities: fixed π , maximize over g

For this scenario, an unrestricted maximization over $\mathbb{G}\mathcal{X}$ — i.e. considering (\mathbb{G}, π) -capacities— would result in unbounded leakage. To see that, recall how leakage is affected by shifting and scaling g (Thm. 5.13). Multiplicative leakage is invariant under scaling but is affected by shifting. More precisely, shifting g down (i.e. shifting by a vector κ with negative values), always *increases* multiplicative leakage, hence by arbitrarily shifting down g we can make the leakage arbitrarily large. To cope with this issue, we can restrict to the class $\mathbb{G}^+\mathcal{X}$ of non-negative gain functions (§3.3.3), essentially preventing g from being arbitrarily shifted down.

Additive leakage, on the other hand, is invariant under shifting, but is affected by scaling. More precisely, by scaling g up (i.e. by a positive k) the additive leakage always increases, hence an arbitrary scaling can lead to arbitrarily large leakage. To deal with this issue, we can restrict to the class $\mathbb{G}^\dagger\mathcal{X}$ of 1-bounded gain functions (§3.3.4), essentially preventing g from being arbitrarily scaled up.

$(\mathcal{G}, \mathbb{D})$ -capacities: maximize over both g and π

Given the discussion about (\mathcal{G}, π) -capacities, in this final scenario the leakage clearly also becomes unbounded when maximizing over $\mathbb{G}\mathcal{X}$. As a consequence, we will again restrict to $\mathbb{G}^+\mathcal{X}$ in the multiplicative case, and $\mathbb{G}^\dagger\mathcal{X}$ in the additive case.

7.4 Multiplicative capacities

We start with the three scenarios involving multiplicative leakage.

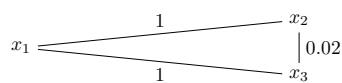
7.4.1 Fixed g , maximize over π

The key question here is how to calculate the maximum leakage efficiently, since we are again dealing with an infinite set $\mathbb{D}\mathcal{X}$ of distributions. For the identity gain function g_{id} (i.e. Bayes leakage), we know from Thm. 7.2 that the capacity is always realized on a *uniform prior*. That is not true in general, however, which is what makes this scenario challenging.

Consider, for instance, the channel and the metric-based gain function given below:

C	y_1	y_2
x_1	0.6	0.4
x_2	0	1
x_3	0	1

G _d	x_1	x_2	x_3
x_1	1	0	0
x_2	0	1	0.98
x_3	0	0.98	1



Under a uniform prior ϑ , we calculate that $V_{g_d}(\vartheta) = 0.66$ and $V_{g_d}[\vartheta \triangleright C] = 0.86$, giving $\mathcal{L}_{g_d}^X(\vartheta, C) \approx 1.3030$. Now if we consider the prior $\pi = (0.5, 0.5, 0)$, we find that

⁷ We can define $g_H(\omega, x) := \log_2(|\mathcal{X}|) - \ell_H(\omega, x)$, where ℓ_H is the loss function from §3.2.8 giving Shannon entropy. The result is that $V_{g_H}(\pi) = \log_2(|\mathcal{X}|) - H(\pi)$. See §7.6.3 for further discussion.

$V_{g_d}(\pi) = 0.5$ and $V_{g_d}[\pi \triangleright C] = 0.8$, which gives $\mathcal{L}_{g_d}^{\times}(\pi, C) = 1.6$. Hence the (g_d, \mathbb{D}) -capacity of C is *not* realized on a uniform distribution. Notice here that 1.6 is also C 's multiplicative Bayes capacity. Hence, by Thm. 7.5, we know that 1.6 must in fact be its (g_d, \mathbb{D}) -capacity, realized on π .

The existence of an efficient algorithm for computing $\mathcal{ML}_g^{\times}(\mathbb{D}, C)$ is unclear.

7.4.2 Fixed π , maximize over g

We now turn our attention to the case when the prior is fixed and we maximize over all gain functions. As already discussed in §7.3, an unrestricted maximization over the whole $\mathbb{G}\mathcal{X}$ leads to unbounded leakage. Later in this section we give an explicit construction for achieving infinite multiplicative leakage.

As a consequence, we restrict to the class $\mathbb{G}^+\mathcal{X}$ of non-negative gain functions, introduced in §3.3.3. Note that this class is quite expressive, because g is still allowed to have infinitely many actions and to take arbitrary non-negative values. A complete solution for this capacity for $\mathbb{G}^+\mathcal{X}$ is given by the following result, a consequence of the Miracle theorem 7.5.

Theorem 7.14 Given a channel C and prior π , we have

$$\mathcal{ML}_{\mathbb{G}^+}^{\times}(\pi, C) = \mathcal{L}_{g_{\pi^{-1}}}^{\times}(\pi, C) = \sum_{y \in \mathcal{Y}} \max_{x: [\pi]} C_{x,y} ,$$

where $g_{\pi^{-1}}$ is the distribution-reciprocal gain function for π (Def. 3.7) and C is any channel matrix realizing C .

Proof. Assume for the moment that π is full support. Recall that $\lceil \pi \rfloor$ denotes the diagonal matrix with π on its diagonal, and that from the discussion after Def. 3.7 we know that $g_{\pi^{-1}}$ can be expressed as a matrix $\lceil \pi^{-1} \rfloor$, and that $V_{g_{\pi^{-1}}}(\pi) = 1$. Since the joint matrix can be written as $J = \lceil \pi \rfloor \cdot C$, from Thm. 5.18 we get that

$$V_{g_{\pi^{-1}}}[\pi \triangleright C] = \sum_{y \in \mathcal{Y}} \max_{w \in \mathcal{W}} (\lceil \pi^{-1} \rfloor \cdot \lceil \pi \rfloor \cdot C)_{w,y} .$$

But $\lceil \pi^{-1} \rfloor \cdot \lceil \pi \rfloor = I$, hence $\mathcal{L}_{g_{\pi^{-1}}}^{\times}(\pi, C)$ is the sum of the column maximums of C . By the Thm. 7.5 again, that is an upper bound on multiplicative leakage over $\mathbb{G}^+\mathcal{X}$, and therefore $\mathcal{L}_{g_{\pi^{-1}}}^{\times}(\pi, C) = \mathcal{ML}_{\mathbb{G}^+}^{\times}(\pi, C)$.

Finally, if π is not full support, the leakage of C under any g is equal to that of the channel C' obtained by deleting the rows corresponding to secret values outside the support. Applying the result to C' gives capacity $\sum_{y \in \mathcal{Y}} \max_{x: [\pi]} C_{x,y}$. \square

Note that for any full-support π , the (\mathbb{G}^+, π) -capacity is exactly equal to $\mathcal{ML}_1^{\times}(\mathbb{D}, C)$.

Example 7.15 Consider again the imperfect disease test from Example 5.16; it had the channel matrix

C	positive	negative
disease	9/10	1/10
no disease	1/10	9/10

For prior $\pi = (1/100, 99/100)$ that channel has no Bayes leakage, since the best guess is “no disease”, regardless of what the test result might be. On the other hand, it turns out that with gain function

$g_{\pi^{-1}}$	disease	no disease	
disease	100	0	
no disease	0	100/99	

we find the $g_{\pi^{-1}}$ -leakage to be $1.8 = \mathcal{ML}_1^{\times}(\mathbb{D}, C)$. \square

To conclude this section, we give an explicit construction showing that unrestricted maximization over $\mathbb{G}\mathcal{X}$ yields $\mathcal{ML}_{\mathbb{G}}^{\times}(\pi, C) = +\infty$ whenever π is full support and C does not satisfy noninterference (p. 60). Define a gain function g by shifting the distribution-reciprocal gain function $g_{\pi^{-1}}$ (Def. 3.7) down by 1 everywhere. (Note that g is then in $\mathbb{G}\mathcal{X}$ but not in $\mathbb{G}^+\mathcal{X}$.) By the discussion after Def. 3.7, we see that $V_g(\pi) = 0$ and $V_g(\sigma) > 0$ for any distribution other than π . Hence if C does not satisfy noninterference we have $V_g[\pi \triangleright C] > 0$, since at least one of the inners of $[\pi \triangleright C]$ must differ from π . That implies that $\mathcal{L}_g^{\times}(\pi, C) = +\infty$.

7.4.3 Maximize over both g and π

Here we also have a complete solution for $\mathbb{G}^+\mathcal{X}$. By Thm. 7.5, multiplicative leakage is maximized by g_{id} and the uniform prior ϑ , which gives $\mathcal{ML}_{\mathbb{G}^+}^{\times}(\mathbb{D}, C) = \mathcal{ML}_1^{\times}(\mathbb{D}, C)$. In fact, by Thm. 7.14 we can let π be any full-support prior and let g be the distribution-reciprocal gain function $g_{\pi^{-1}}$.

For $\mathbb{G}\mathcal{X}$, on the other hand, (\mathbb{G}, \mathbb{D}) -capacity becomes infinite for all interfering C , since (\mathbb{G}, π) -capacity is already infinite for any full-support π .

7.5 Additive capacities

We now turn our attention to the efficiency of calculating capacities in the three additive scenarios.

7.5.1 Fixed g , maximize over π

Here we know a number of interesting things. For g_{id} it is shown in Thm. 7.11 that the additive leakage is maximized by some sub-uniform distribution, i.e. one that is uniform on some *subset* of \mathcal{X} . Since the set of sub-uniform distributions for a fixed $|\mathcal{X}|$ is finite, $\mathcal{ML}_{g_{\text{id}}}^+(\mathbb{D}, C)$ is computable. But because the number of sub-uniform distributions is exponential in the size of C 's matrix representation \mathbf{C} , that does not give an efficient algorithm. In fact, from Thm. 7.12 we know that it is NP-complete to decide whether C 's additive ($g_{\text{id}}, \mathbb{D}$)-capacity exceeds a given threshold.

For *some* gain functions, however, $\mathcal{ML}_g^+(\mathbb{D}, C)$ can be efficiently computed. For instance, for the gain function g_H giving the complement of Shannon entropy, the additive capacity $\mathcal{ML}_{g_H}^+(\mathbb{D}, C)$ is the Shannon capacity, which can be computed using the iterative Blahut-Arimoto algorithm.

Still, given the result for g_{id} specifically, we cannot of course expect the problem to be efficiently solvable for g in general. But we observe that we can use the linear-programming-based Algorithm 5.27 to find a maximizing π : given concrete channels \mathbf{A} and \mathbf{B} , that algorithm finds a π that maximizes $V_g[\pi \triangleright \mathbf{B}] - V_g[\pi \triangleright \mathbf{A}]$.

(It is aimed at testing whether B 's g -leakage can ever exceed A 's.) Here we can just let B be C and let A be the channel $\mathbf{1}$ that leaks nothing which, we recall, is as a reduced channel matrix just a single column of 1's — thus giving $V_g[\pi \triangleright \mathbf{1}] = V_g(\pi)$. Hence we get $V_g[\pi \triangleright B] - V_g[\pi \triangleright A] = V_g[\pi \triangleright C] - V_g(\pi) = \mathcal{L}_g^+(\pi, C)$.

To use linear programming, Algorithm 5.27 applies the traced-based formulation $V_g[\pi \triangleright C] = \max_S \text{tr}(G \cdot \pi \downarrow \cdot C \cdot S)$ (§5.5.2) of posterior g -vulnerability, which relies on a maximization over all strategies S . But notice that if we then try to compute the maximization

$$\max_{\pi} (V_g[\pi \triangleright B] - V_g[\pi \triangleright A]) \quad ,$$

we have *nested maximums* and a *quadratic* objective function, since the entries of both π and S are variables.

However, we can assume without loss of generality that the strategy S is *deterministic*, which allows the algorithm to try explicitly *all* strategies S^A for A and S^B for B , adding linear constraints to express that π is a prior for which S^A is optimal. In that way, it is able to find a π that maximizes additive g -leakage by solving $|\mathcal{W}|^{|\mathcal{Y}|+1}$ linear-programming problems. Of course, solving exponentially many linear-programming problems is not going to be feasible, except on very small channels.

A better solution is to reformulate our problem as a single *Quadratic Programming* problem, where the objective function is quadratic but the constraints are still linear. Quadratic programming cannot be solved efficiently in general, but there exist many mature quadratic-programming tools, and it appears likely that they could be applied fruitfully to the computation of $\mathcal{ML}_g^+(\mathbb{D}, C)$.

7.5.2 Fixed π , maximize over g

Our final two scenarios concern maximizing over all gain functions with respect to some class. As with multiplicative leakage we find that there is no useful answer when we allow gain functions to be unrestricted. For example, suppose for a given channel C and prior π that there is some gain function such that $\mathcal{L}_g^+(\pi, C) > 0$; in this case we reason that the maximum over all gain functions in $\mathbb{G}\mathcal{X}$ is unbounded:

$$\begin{aligned} & \mathcal{ML}_G^+(\pi, C) \\ \geq & \sup_{k \geq 0} \mathcal{L}_{g \times k}^+(\pi, C) \\ = & \sup_{k \geq 0} \mathcal{L}_g^+(\pi, C) \times k && \text{“Thm. 5.13”} \\ = & +\infty && \text{“}\mathcal{L}_g^+(\pi, C) > 0\text{”} \end{aligned}$$

Just as for multiplicative capacity, we can however restrict to an expressive class of gain functions that provide a useful upper bound and also a “miracle theorem” for additive leakage. This is the class $\mathbb{G}^\dagger\mathcal{X}$ (§3.3.4), defined either by requiring that g be bounded from above by 1, or equivalently that V_g be bounded from above by 1.⁸ A crucial remark about the expressiveness of this class is that, although it is clearly stricter than $\mathbb{G}\mathcal{X}$, *any* gain function g in $\mathbb{G}\mathcal{X}$ is upper-bounded by some $k > 0$. This means that, by scaling, we can use the capacity results for $\mathbb{G}^\dagger\mathcal{X}$ to provide additive leakage bounds for any g (e.g. for Shannon leakage). See §7.6.3 for examples.⁹

⁸ Note that g can still take negative values, while V_g is always assumed to be non-negative.

⁹ Note that, since additive leakage is invariant under shifting (Thm. 5.13), the capacity results of this section also hold for the slightly larger class requiring that the “span” of V_g (the size of its range) is at most 1. On the other hand, restricting the *span* of g itself leads to the strictly less expressive class $\mathbb{G}^1\mathcal{X}$; for instance the gain function g_H for the complement of Shannon entropy has infinite span. The class $\mathbb{G}^1\mathcal{X}$ is not discussed in this book: see the Chapter Notes.

The upper-bound condition means that we are unable to scale gain functions arbitrarily as in the calculation above but, more interestingly, the upper bound implies a very useful property reminiscent of the “1-Lipschitz” condition for continuous functions. In this case it provides a relationship between g -leakage and a “distance” between distributions in $\mathbb{D}\mathcal{X}$, where a *distance* measure between distributions is a non-negative function of type $\mathbb{D}\mathcal{X} \times \mathbb{D}\mathcal{X} \rightarrow \mathbb{R}$.¹⁰

Definition 7.16 (d -Lipschitz) Take g in $\mathbb{G}\mathcal{X}$ and let d in $\mathbb{D}\mathcal{X} \times \mathbb{D}\mathcal{X} \rightarrow \mathbb{R}$ be a distance measure. Vulnerability V_g is said to be d -Lipschitz with respect to d if

$$V_g(\pi) - V_g(\pi') \leq d(\pi, \pi') \quad \text{for all distributions } \pi, \pi'.$$

□

The d -Lipschitz condition says that those vulnerabilities are “contracting” with respect to a distance measure; this is the case for the following distance measure and the class of vulnerabilities defined by $\mathbb{G}^\dagger\mathcal{X}$.

Definition 7.17 (quasiconvex separation) Let $\pi, \sigma \in \mathbb{D}\mathcal{X}$. Define the quasiconvex separation distance as

$$\mathbf{q}(\sigma, \pi) := \max_{x: \lceil \sigma \rceil} \left(1 - \frac{\pi_x}{\sigma_x} \right).$$

□

(Compare Def. 11.3 in Chap. 11.) It is clear that $\mathbf{q}(\sigma, \pi) \geq 0$, and that $\mathbf{q}(\pi, \pi) = 0$. Moreover, we can show that any V_g satisfying $0 \leq V_g \leq 1$ is \mathbf{q} -Lipschitz.

Lemma 7.18 For any gain function g in $\mathbb{G}^\dagger\mathcal{X}$, its associated vulnerability V_g is \mathbf{q} -Lipschitz.

Proof. Let π, σ be distributions in $\mathbb{D}\mathcal{X}$, and assume that $\pi \neq \sigma$. For $c \geq 1$, consider the definition

$$\pi^c := \pi + c(\sigma - \pi). \quad (7.2)$$

Observe that when $c = 1$, we obtain $\pi^c = \sigma$. Moreover, for some $c > 1$, we must have that $\pi^c \notin \mathbb{D}\mathcal{X}$. To see that, observe that since both $\pi, \sigma \in \mathbb{D}\mathcal{X}$, for some x in \mathcal{X} , it must be the case that $\sigma_x < \pi_x$. When that happens, for $c > \pi_x/(\pi_x - \sigma_x)$ it must be that $\pi_x^c < 0$, and so $\pi^c \notin \mathbb{D}\mathcal{X}$. Let \mathbf{c} be the maximum value of c such that $\pi^c \in \mathbb{D}\mathcal{X}$. In fact \mathbf{c} is equal to the minimum value, greater than 0, of $\pi_x/(\pi_x - \sigma_x)$, for all $x \in \mathcal{X}$ for which $\sigma_x < \pi_x$.

We now reason:

$$\begin{aligned} & V_g(\sigma) - V_g(\pi) \\ = & V_g(\pi^c/c + (1-1/c)\pi) - V_g(\pi) && \text{“(7.2) implies } \sigma = \pi^c/c + (1-1/c)\pi\text{”} \\ \leq & V_g(\pi^c)/c + (1-1/c)V_g(\pi) - V_g(\pi) && \text{“}V_g \text{ is convex”} \\ = & (V_g(\pi^c) - V_g(\pi))/c && \text{“arithmetic”} \\ \leq & 1/\mathbf{c} && \text{“}0 \leq V_g \leq 1 \text{ implies } V_g(\pi^c) - V_g(\pi) \leq 1; \mathbf{c} > 0\text{”} \\ = & \max_{x \in \lceil \pi \rceil} (\pi_x - \sigma_x)/\pi_x && \text{“definition of } \mathbf{c}; \text{ arithmetic”} \\ = & \max_{x \in \lceil \pi \rceil} (1 - \sigma_x/\pi_x) \\ = & \mathbf{q}(\sigma, \pi). && \text{“Def. 7.17”} \end{aligned}$$

□

¹⁰ Distance measures in this instance do not need to be symmetric.

We are now ready to prove an upper bound on $\mathcal{ML}_{\mathbb{G}^\dagger}^+(\pi, C)$.

Lemma 7.19 Let C be a channel, π a prior, and g a gain function in $\mathbb{G}^\dagger \mathcal{X}$. Then

$$\mathcal{L}_g^+(\pi, C) \leq \mathcal{E}_{[\pi \triangleright C]} \mathbf{q}_\pi ,$$

where $\mathbf{q}_\pi(\sigma) := \mathbf{q}(\sigma, \pi)$. (Recall that $\mathcal{E}_{[\pi \triangleright C]} \mathbf{q}_\pi$ denotes the expected value of \mathbf{q}_π over distribution $[\pi \triangleright C]$.)

Proof. We have

$$\begin{aligned} & \mathcal{L}_g^+(\pi, C) \\ = & V_g[\pi \triangleright C] - V_g(\pi) \\ = & \mathcal{E}_{[\pi \triangleright C]} V_g - V_g(\pi) \\ = & \mathcal{E}_{[\pi \triangleright C]} (V_g - V_g(\pi)) \\ \leq & \mathcal{E}_{[\pi \triangleright C]} \mathbf{q}_\pi . \end{aligned} \quad \text{“Lem. 7.18 and } \mathbf{q}(\sigma, \pi) \geq 0\text{.”}$$

□

Observe that $\mathcal{E}_{[\pi \triangleright C]} \mathbf{q}_\pi$ depends only on the channel and the prior; moreover, the upper bound is achieved because the function \mathbf{q}_π is a convex and continuous function of $\mathbb{D}\mathcal{X}$, and therefore by Thm. 11.5 can be expressed as a vulnerability. With that we can compute the maximal leakage for fixed π .

Theorem 7.20 Let C be a channel and π a prior. The maximal leakage of C with respect to π is

$$\mathcal{ML}_{\mathbb{G}^\dagger}^+(\pi, C) = \mathcal{E}_{[\pi \triangleright C]} \mathbf{q}_\pi .$$

Proof. From Lem. 7.19 we have that $\mathcal{L}_g^+(\pi, C) \leq \mathcal{E}_{[\pi \triangleright C]} \mathbf{q}_\pi$. But note that \mathbf{q}_π is the maximum over a set of linear functions of the input, and therefore it is convex and continuous — thus by Thm. 11.5 is expressible as V_g for some g in $\mathbb{G}\mathcal{X}$; but since $0 \leq \mathbf{q}_\pi \leq 1$ this gain function must also be in $\mathbb{G}^\dagger \mathcal{X}$. The result now follows since the leakage with respect to \mathbf{q}_π is

$$\mathcal{E}_{[\pi \triangleright C]} \mathbf{q}_\pi - \mathbf{q}_\pi(\pi) = \mathcal{E}_{[\pi \triangleright C]} \mathbf{q}_\pi - 0 = \mathcal{E}_{[\pi \triangleright C]} \mathbf{q}_\pi .$$

□

Finally we compute $\mathcal{E}_{[\pi \triangleright C]} \mathbf{q}_\pi$, which, remarkably, turns out to be dependent only on C and the *support* of π .

Theorem 7.21

Given a channel C and prior π , we have

$$\mathcal{ML}_{\mathbb{G}^\dagger}^+(\pi, C) = \mathcal{L}_{g_{\pi^{-1}}^c}^+(\pi, C) = 1 - \sum_{y: \mathcal{Y}} \min_{x: [\pi]} \mathsf{C}_{x,y} ,$$

where $g_{\pi^{-1}}^c$ is the complement of the distribution-reciprocal gain function (Def. 3.7) and C is any channel matrix realizing C .

Proof. By Thm. 7.20 we have that $\mathcal{ML}_{\mathbb{G}^\ddagger \mathcal{X}}^+(\pi, C) = \mathcal{E}_{[\pi \triangleright C]} \mathbf{q}_\pi$, which we can compute directly as follows. Let σ be any posterior of $[\pi \triangleright C]$ corresponding to some column y of C and let α be the (marginal) probability corresponding to σ . Then $\mathbf{q}_\pi(\sigma) = \max_{x \in [\pi]} (1 - \sigma_x / \pi_x)$. But $\sigma_x / \pi_x = C_{x,y} / \alpha$, and so we have

$$\mathcal{ML}_{\mathbb{G}^\ddagger}^+(\pi, C) = \mathcal{E}_{[\pi \triangleright C]} \mathbf{q}_\pi = 1 - \sum_{y: \mathcal{Y}} \min_{x \in [\pi]} C_{x,y} .$$

Finally, direct calculation confirms that $V_{g_{\pi^{-1}}^c}[\pi \triangleright C] = 1 - \sum_{y: \mathcal{Y}} \min_{x: [\pi]} C_{x,y}$. \square

The complement of the distribution-reciprocal gain function $g_{\pi^{-1}}$ (Def. 3.7) is

$$g_{\pi^{-1}}^c(w, x) := \begin{cases} 1 - \frac{1}{\pi_x} & \text{if } w = x \\ 1 & \text{if } w \neq x \end{cases} \quad \text{for } \mathcal{W} = [\pi] .$$

Remarkably, $g_{\pi^{-1}}$ was used in the multiplicative case to “cancel out” the effect of π and show that the multiplicative (\mathbb{G}^+, π) -capacity is independent of π (§7.4.2). The same gain function (but now complemented) is used again here for the same reason, canceling the effect of the prior and making $\mathcal{ML}_{\mathbb{G}^\ddagger}^+(\pi, C)$ independent of π .

To get some operational intuition about Thm. 7.21, note that in the context of anonymity we view secret X as being “guilty”, and then gain function $g_{\pi^{-1}}^c$ models an adversary who wants to guess an “innocent” value. If correct, she gets a gain of 1. But if she guesses the “guilty” x , then she gets a gain of $1 - 1/\pi_x$, which is negative (unless $\pi_x = 1$). Notice what this says about prior vulnerability: if she guesses x , she gains 1 with probability $1 - \pi_x$, and gains $1 - 1/\pi_x$ with probability π_x , making her expected gain $(1 - \pi_x) + (\pi_x - 1) = 0$. Hence the prior vulnerability is 0.

Now suppose that C is a channel in which every column contains a 0 somewhere. Then for every output y , she has an x that is guaranteed to be innocent, giving her a posterior vulnerability of 1, which is then indeed 1 minus the sum of the column minimums. On the other hand, for any channel with a column without any 0’s, on that output she has no safe guess, and that makes her posterior vulnerability strictly less than 1.

Example 7.22 Recall the imperfect disease test of Example 5.16, also revisited in Example 7.15. For prior $\pi = (1/100, 99/100)$ the channel has no Bayes leakage, since the best guess is “no disease”, regardless of the test result. But for gain function

$g_{\pi^{-1}}^c$	disease	no disease
disease	-99	1
no disease	1	$-1/99$

we find that the additive $g_{\pi^{-1}}^c$ -leakage is $4/5 = \mathcal{ML}_{\mathbb{G}^\ddagger \mathcal{X}}^+(\pi, C)$. \square

7.5.3 Maximize over both g and π

Similarly to the multiplicative case (§7.4.3), we have a complete solution here for $\mathbb{G}^\ddagger \mathcal{X}$. By Thm. 7.21 we get that the capacity is given by any full-support π and the complement of the distribution-reciprocal gain function $g_{\pi^{-1}}^c$, giving

$$\mathcal{ML}_{\mathbb{G}^\ddagger}^+(\mathbb{D}, C) = 1 - \sum_{y: \mathcal{Y}} \min_{x: \mathcal{X}} C_{x,y} .$$

For $\mathbb{G} \mathcal{X}$, again similarly to the multiplicative case, (\mathbb{G}, \mathbb{D}) -capacity becomes infinite for all interfering C , since (\mathbb{G}, π) -capacity is already infinite for any full-support π .

7.6 Obtaining bounds on leakage

The motivation for introducing capacity is to address cases in which we do not want to protect against a single adversary, but instead we wish to be robust with respect to a class of operational scenarios. However, even when the adversary is fixed, expressed by some gain function g and prior knowledge π , the results of this chapter can be useful for obtaining *bounds* on g -leakage (or on g -vulnerability). Of course, for fixed g and π one could compute the corresponding leakage directly. However, such a computation might be challenging if, for instance, the size of the channel is big or the gain function g is hard to represent.

In this section, we start by formulating an “Additive miracle” theorem, providing an additive leakage bound for the class $\mathbb{G}^\dagger \mathcal{X}$, similar to the multiplicative one for $\mathbb{G}^+ \mathcal{X}$. Then, we show how we can use gain-function algebra to extend those bounds to (almost) any gain function in $\mathbb{G} \mathcal{X}$. We conclude this section with two examples illustrating the bounds.

7.6.1 The additive miracle theorem

In the multiplicative case we used the Miracle theorem (Thm. 7.5), which provides an upper bound for g -leakage, to compute the multiplicative (\mathbb{G}^+, π) - and $(\mathbb{G}^+, \mathbb{D})$ -capacities. In the additive case, having already a solution for capacity, we can go in the opposite direction and obtain a corresponding “Additive miracle” theorem, giving a tight upper bound for additive g -leakage.

Theorem 7.23 (“Additive Miracle”)

For any channel C , prior π , and gain function $g: \mathbb{G}^\dagger \mathcal{X}$, we have

$$\mathcal{L}_g^+(\pi, C) \leq 1 - \sum_{y: \mathcal{Y}} \min_{x: \mathcal{X}} C_{x,y} .$$

Proof. The inequality is a direct consequence of Thm. 7.21; note that it holds for any prior since

$$1 - \sum_{y: \mathcal{Y}} \min_{x: \mathcal{X}} C_{x,y} \geq 1 - \sum_{y: \mathcal{Y}} \min_{x: [\pi]} C_{x,y} .$$

□

We note that in the multiplicative Miracle theorem, the upper bound is given by $\mathcal{ML}_1^\times(\mathbb{D}, C)$, i.e. by the channel’s $(g_{\text{id}}, \mathbb{D})$ -capacity. That is no longer true in the additive case; however, the upper bound in Thm. 7.23 can be shown to be equal to the $(g_{\text{id}}^c \times |\mathcal{X}|, \mathbb{D})$ -capacity of C — that is, a complemented and scaled version of g_{id} is needed. (See Exercise 7.5.)

7.6.2 Improved miracle bounds

If g belongs to either $\mathbb{G}^+ \mathcal{X}$ or $\mathbb{G}^\dagger \mathcal{X}$ then the Miracle theorems can be directly applied. For a generic $g: \mathbb{G} \mathcal{X}$, however, we can still obtain bounds by first constructing a scaled or shifted version g' that falls inside one of those classes, obtaining a bound for g' , and then transforming it into a bound for g using Thm. 5.13. That technique is employed in the following theorem, providing generic bounds for (almost) any $g: \mathbb{G} \mathcal{X}$.

Theorem 7.24 For any channel C , prior π , and gain function g in $\mathbb{G}\mathcal{X}$, let k be the supremum of g . Then we have

$$\mathcal{L}_g^+(\pi, C) \leq k \left(1 - \sum_{y: \mathcal{Y}} \min_{x: \mathcal{X}} C_{x,y}\right) .$$

Now assume that g is bounded from below and $V_g(\pi) \neq 0$. Letting $\kappa: \mathbb{R}^{|\mathcal{X}|}$ be the vector of per-secret gain infimums (i.e. each κ_x is the infimum of $g(\cdot, x)$), we have

$$\mathcal{L}_g^\times(\pi, C) \leq \left(\sum_{y: \mathcal{Y}} \max_{x: \mathcal{X}} C_{x,y} \right) (1 - c) + c \quad \text{where } c := \frac{\kappa \cdot \pi}{V_g(\pi)} .$$

Proof. Letting $k' = 1/k$, it is easy to see that $g_{\times k'} \in \mathbb{G}^\dagger \mathcal{X}$. We can hence apply the additive Miracle theorem to $\mathcal{L}_{g_{\times k'}}^+(\pi, C)$ and then use Thm. 5.13. Similarly, letting $\kappa' = -\kappa$, we have $g_{+\kappa'} \in \mathbb{G}^+ \mathcal{X}$. We can hence apply the multiplicative Miracle theorem to $\mathcal{L}_{g_{+\kappa'}}^\times(\pi, C)$ and then again use Thm. 5.13. \square

Note that all gain functions $g: \mathbb{G}\mathcal{X}$ are bounded from above, so the additive bound of the above theorem is always applicable. The multiplicative bound, however, is not applicable when a gain function is not bounded from below, or when the prior g -vulnerability is 0.

Note also that these improved bounds might be of interest also in the case when g does belong to either $\mathbb{G}^+ \mathcal{X}$ or $\mathbb{G}^\dagger \mathcal{X}$. That is because the bounds might be smaller than those provided directly by the Miracle theorems, since the extrema of g are taken into account. For instance, consider a gain function with values in $[0, 1/2]$. Since its supremum is $k = 1/2$, the bound from Thm. 7.24 will be half the bound from the additive Miracle theorem. A similar improvement to the multiplicative–Miracle-theorem bound happens with gain functions $g: \mathbb{G}^+ \mathcal{X}$ that are *strictly* positive.

7.6.3 Examples

We illustrate our leakage bounds in the following two examples.

Tiger leakage Let us revisit the channel of Example 7.6, which was

C	y_1	y_2
x_1	0.8	0.2
x_2	0.2	0.8

and assume that we are interested in its g_{tiger} -vulnerability on either a uniform prior ϑ , or a quasi-uniform $\pi = (0.51, 0.49)$. The corresponding prior and posterior vulnerabilities are

$$\begin{aligned} V_{g_{\text{tiger}}}(\vartheta) &= 0 & V_{g_{\text{tiger}}}[\vartheta \triangleright C] &= 0.6 \\ V_{g_{\text{tiger}}}(\pi) &= 0.02 & V_{g_{\text{tiger}}}[\pi \triangleright C] &= 0.6 \end{aligned} ,$$

but for the sake of this example assume that we cannot compute the posterior vulnerabilities directly, and are therefore interested in bounding them.

A vulnerability bound can be obtained directly from a corresponding leakage bound. But since g_{tiger} is not non-negative, the Miracle theorem does not apply. (Indeed Example 7.6 shows that the Miracle theorem fails completely here.) For π , however,

we can apply the improved multiplicative bound of Thm. 7.24. The vector of gain infimums is $\kappa = (-1, -1)$, from which we compute

$$c = \kappa \cdot \pi / V_{g_{\text{tiger}}}(\pi) = -1 / 0.02 = -50 .$$

Applying the bound, we get

$$\mathcal{L}_{g_{\text{tiger}}}^+(\pi, C) \leq \left(\sum_{y: \mathcal{Y}} \max_{x: \mathcal{X}} C_{x,y} \right) (1-c) + c = (0.8 + 0.8) \cdot 51 - 50 = 31.6 ,$$

from which we conclude that $V_{g_{\text{tiger}}}[\pi \triangleright C] \leq 31.6 \cdot V_{g_{\text{tiger}}}(\pi) = 0.632$.

For ϑ , however, the bound of Thm. 7.24 is not applicable, since $V_{g_{\text{tiger}}}(\vartheta) = 0$.

We can also employ the *additive* bounds for our purpose. In this case, the additive Miracle theorem is directly applicable since $V_{g_{\text{tiger}}}$ is clearly bounded by 1, that is $g_{\text{tiger}}: \mathbb{G}^\dagger \mathcal{X}$. From the additive Miracle theorem we get that

$$\mathcal{L}_{g_{\text{tiger}}}^+(\pi, C) \leq 1 - \sum_y \min_x C_{x,y} = 1 - 0.2 - 0.2 = 0.6 ,$$

from which we conclude that $V_{g_{\text{tiger}}}[\pi \triangleright C] \leq 0.6 + V_{g_{\text{tiger}}}(\pi) = 0.62$, a bound slightly better than the one obtained via multiplicative leakage.

But the additive Miracle theorem is also applicable for ϑ ; in fact the *leakage* bound itself does not depend on the prior, hence we get $\mathcal{L}_{g_{\text{tiger}}}^+(\vartheta, C) \leq 0.6$, from which we conclude that $V_{g_{\text{tiger}}}[\vartheta \triangleright C] \leq 0.6 + V_{g_{\text{tiger}}}(\vartheta) = 0.6$. Remarkably, we obtain a *tight* bound, equal to the actual posterior vulnerability.

Shannon leakage Consider the channel

C	y_1	y_2	y_3
x_1	0.5	0.39	0.11
x_2	0.5	0.40	0.10
x_3	0.5	0.41	0.09

,

and assume that we are interested in its Shannon leakage (i.e. its Shannon mutual information) for some *unknown* prior π . Notice that the channel is *almost* noninterfering — its rows differ only slightly. Because of this, we intuitively expect its Shannon leakage to be small; but it is not zero, so we wish to establish a concrete bound. Of course *Shannon capacity* provides such a bound, but computing it requires that we apply the iterative Blahut-Arimoto algorithm. Is there a way to obtain a simple bound faster?

For this purpose, recall that there is a gain function g_H that gives the *complement* of Shannon entropy; that is, $V_{g_H}(\pi) = \log_2(|\mathcal{X}|) - H(\pi)$. This g_H can be constructed as the complement of the *loss* function ℓ_H described in §3.2.8, but its exact definition is not important for our purpose. Note also that $\mathcal{L}_{g_H}^+(\pi, C)$ is exactly equal to the Shannon mutual information of C ,¹¹ so our goal is simply to bound C 's additive g_H -leakage.

Since V_{g_H} takes values in $[0, \log_2 3]$, gain function g_H does not belong to $\mathbb{G}^\dagger \mathcal{X}$ and hence the additive Miracle theorem is not directly applicable. But we can still apply Thm. 7.24; the supremum of g_H is $k = \log_2 3$,¹² hence we get that

$$\mathcal{L}_{g_H}^+(\pi, C) \leq \log_2 3 \cdot (1 - \sum_y \min_x C_{x,y}) = \log_2 3 \cdot 0.02 \approx 0.032 .$$

As expected, we have established that C 's Shannon leakage is necessarily small. (Note that C 's actual Shannon capacity is approximately 0.001, so our bound is not tight but still useful.)

¹¹ The complement is applied to both the prior- and the posterior case, so it cancels out.

¹² Note that the supremum of g always coincides with that of V_g .

7.7 Exercises

Exercise 7.1 Let C be a channel matrix from \mathcal{X} to \mathcal{Y} . Show that for any $g: \mathbb{G}^+ \mathcal{X}$ and any prior, its multiplicative g -leakage is bounded by both $|\mathcal{X}|$ and $|\mathcal{Y}|$. Does the result necessarily hold if g is not in $\mathbb{G}^+ \mathcal{X}$? \square

Exercise 7.2 Recall that the logarithm of multiplicative Bayes capacity coincides with Shannon capacity in the case of deterministic channels (Thm. 7.7), and is an upper bound in general (Thm. 7.8). In this exercise we explore the question of how much they can differ.

Let C be a square channel matrix with $2^{64} - 2^{54} + 1$ rows and columns, whose entries are all 2^{-64} except on the main diagonal, where the entries are all 2^{-10} :

C	y_1	y_2	y_3	\dots
x_1	2^{-10}	2^{-64}	2^{-64}	\dots
x_2	2^{-64}	2^{-10}	2^{-64}	\dots
x_3	2^{-64}	2^{-64}	2^{-10}	\dots
\vdots	\vdots	\vdots	\vdots	\ddots

It turns out that the Shannon capacity of C is approximately 0.05132, or about one-twentieth of a bit. How does this compare with the logarithm of C 's multiplicative Bayes capacity? \square

Exercise 7.3 Show that for any π , the gain function $g_{\pi^{-1}}^c$ is in \mathbb{G}^\ddagger . \square

Exercise 7.4 Suppose that C is a deterministic channel matrix, meaning that all its entries are either 0 or 1. Show that $\mathcal{ML}_{\mathbb{G}^\ddagger}^+(\mathbb{D}, C)$, that is C 's additive capacity over 1-bounded gain functions and all priors, has only *two* possible values. \square

Exercise 7.5 Show that for any channel C and gain function $g: \mathbb{G}^\ddagger \mathcal{X}$, the additive g -leakage of C under any prior never exceeds its $g_{\text{id}}^c \times |\mathcal{X}|$ -leakage under the uniform prior. \square

7.8 Chapter notes

The idea of capacity of channels originates from Shannon's discussion of how to measure the correlation between two "random variables". The capacities studied in this chapter are, not surprisingly, related to other definitions for generalizing Shannon's mutual information. For example, the negative logarithm of Bayes vulnerability is the same as Rényi's min-entropy [9], implying similar relationships between capacities. We discuss axiomatic notions further in Chap. 11. Interestingly, the logarithm of multiplicative Bayes capacity is the same as Sibson's "Information Radius" [10], which was (originally) defined for studying "discrepancy", specifically between a finite number of probability distributions. Capacities have also been studied for covert channels [8] and in richer contexts such as feedback and memory [12].

Bayes capacity is of particular interest because of the operational interpretation in terms of guessing. Theorems 7.2 and 7.11 are originally due to Braun et al. [5], though the proof here of the latter is new. Theorems 7.5 and 7.8 are due to Alvim et al. [2]. Theorem 7.7 is due to Smith [11]. Theorem 7.12 is due to Alvim et al. [1]; note that it is fundamentally different from the intractability results of Yasuoka and Terauchi [13] — those results assume that the channel is represented as a *program*,

which inevitably leads to intractability of analysis. In contrast, Thm. 7.12 is based on a *channel matrix*, which effectively gives the program's behavior explicitly.

A more detailed account of the additive capacity results presented here has been given by Chatzikokolakis [6]. Additive capacity results for the class $\mathbb{G}^1\mathcal{X}$ have been given by Alvim et al. [1].

Algorithmic techniques have been developed to estimate information leaks for Shannon capacity in particular by Arimoto [3], and with respect to other entropies [4].

Finally the set-packing problem referred to in Thm. 7.12 was one of Karp's “21 NP-complete problems”, and is discussed at length (Problem [SP3]) by Garey and Johnson [7].

Bibliography

- [1] Alvim, M.S., Chatzikokolakis, K., McIver, A., Morgan, C., Palamidessi, C., Smith, G.: Additive and multiplicative notions of leakage, and their capacities. In: Proceedings of the 2014 IEEE 27th Computer Security Foundations Symposium (CSF 2014), pp. 308–322. IEEE, Los Alamitos (2014)
- [2] Alvim, M.S., Chatzikokolakis, K., Palamidessi, C., Smith, G.: Measuring information leakage using generalized gain functions. In: Proceedings of the 2012 IEEE 25th Computer Security Foundations Symposium (CSF 2012), pp. 265–279. IEEE, Los Alamitos (2012)
- [3] Arimoto, S.: An algorithm for computing the capacity of arbitrary discrete memoryless channels. *IEEE Transactions on Information Theory* **18**(1), 14–20 (1972)
- [4] Backes, M., Köpf, B., Rybalchenko, A.: Automatic discovery and quantification of information leaks. In: Proceedings of the 2009 IEEE Symposium on Security and Privacy, pp. 141–153. IEEE, Los Alamitos (2009)
- [5] Braun, C., Chatzikokolakis, K., Palamidessi, C.: Quantitative notions of leakage for one-try attacks. In: Proceedings of the 25th Conference on Mathematical Foundations of Programming Semantics (MFPS 2009), *Electronic Notes in Theoretical Computer Science*, vol. 249, pp. 75–91. Elsevier, Amsterdam (2009)
- [6] Chatzikokolakis, K.: On the additive capacity problem for quantitative information flow. In: A. McIver, A. Horvath (eds.) *Proceedings of the 15th International Conference on Quantitative Evaluation of Systems (QEST 2018)*, *Lecture Notes in Computer Science*, vol. 11024, pp. 1–19. Springer, Berlin (2018)
- [7] Garey, M.R., Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, New York (1979)
- [8] Millen, J.K.: Covert channel capacity. In: Proceedings of the 1987 IEEE Symposium on Security and Privacy, pp. 60–66. IEEE, Los Alamitos (1987)
- [9] Rényi, A.: On measures of entropy and information. In: *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*, pp. 547–561. University of California Press, Berkeley (1961)
- [10] Sibson, R.: Information radius. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete* **14**(2), 149–160 (1969)

Bibliography

- [11] Smith, G.: On the foundations of quantitative information flow. In: L. de Alfaro (ed.) Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS 2009), *Lecture Notes in Computer Science*, vol. 5504, pp. 288–302. Springer, Berlin (2009)
- [12] Tatikonda, S., Mitter, S.: The capacity of channels with feedback. *IEEE Transactions on Information Theory* **55**(1), 323–349 (2009)
- [13] Yasuoka, H., Terauchi, T.: Quantitative information flow — verification hardness and possibilities. In: Proceedings of the 2010 IEEE 23rd Computer Security Foundations Symposium (CSF 2010), pp. 15–27. IEEE, Los Alamitos (2010)



Chapter 8

Composition of channels

As we saw in Chap. 6, our framework for quantitative information flow can provide a robust theory for deriving security properties from a system’s representation as a channel, as long as we can determine what that channel is — but determining an appropriate channel to model a system is itself often a non-trivial task. Moreover, even when they can be determined, some channels turn out to be so large that their security analyses are infeasible in practice. It’s fortunate therefore that many channels can be understood as compositions of other, simpler channels; and in this chapter we consolidate the compositions we have already encountered in §4.6, and go on to suggest others that describe scenarios that might be encountered in practice. We will see that we sometimes can determine bounds on the vulnerability of a compound channel as functions of the vulnerabilities of its components, a possibility we explore in Exercises 8.11 and 8.12 of this chapter.

In Chap. 9 to follow, we will emphasize that an important issue is whether channel compositions support “stepwise refinement” (§9.1), that is that one can “improve” component channels wrt. their security properties and be confident that the result can only be an improvement of the whole composition (i.e. not a degradation). In particular, we will be interested in whether the compositions preserve leakage properties — that is, given a “specification” channel S in some context \mathcal{C} , which we can write $\mathcal{C}(S)$, whether replacing S by some “implementation” channel I of it will give a $\mathcal{C}(I)$ that has at least as good security properties as $\mathcal{C}(S)$ had — which property is called “monotonicity”. It turns out that for some of the compositions we have already seen (such as cascading §4.6.2) that is not so; but for some others (such as the convex combination of abstract channels,¹ the “external choice” of §4.6.1), it is.

8.1 Compositions of (concrete) channel matrices

We start our discussion of compositions by considering *concrete* channels, i.e. in their matrix representations. Recall (from Chap. 4) that a channel matrix with input set \mathcal{X} and output set \mathcal{Y} can be interpreted as having type $\mathcal{X} \rightarrow \mathbb{D}\mathcal{Y}$, or equivalently $\mathcal{X} \times \mathcal{Y} \rightarrow [0, 1]$ with for the latter the additional constraint that the rows sum to one.

¹ Recall however that convex combination of *concrete* channels is not the same: it corresponds to hidden probabilistic choice, and is the topic of §8.1.5.

In this section we will usually write the type of such matrices as $\mathcal{X} \rightarrow \mathcal{Y}$, thinking of them as “probabilistic functions” from \mathcal{X} to \mathcal{Y} .

We say that two channels are *compatible* if they have the same input set, and we denote by $\mathcal{C}_{\mathcal{X}}$ the set of all channels with the same input set \mathcal{X} ; note however that compatible channels’ output sets can differ. In the special case that compatible channels have the same output set as well, we say they are of the same *type*.

We will formalize channel-matrix compositions as binary operators that take two compatible channel matrices and return a channel matrix compatible with the two components. More precisely, a binary channel-matrix operator is a function of type $\mathcal{C}_{\mathcal{X}} \times \mathcal{C}_{\mathcal{X}} \rightarrow \mathcal{C}_{\mathcal{X}}$.² All binary operators we define can however be generalized to n -ary form in the natural way.

To simplify the presentation, we extend the definition of a channel matrix $C: \mathcal{X} \rightarrow \mathcal{Y}$ in such a way that for all y' not in \mathcal{Y} and all $x: \mathcal{X}$ we have $C_{x,y'} = 0$. Note that this extension does not carry unwanted consequences wrt. the vulnerability and leakage of the channel, because the corresponding reduced channel is equivalent under refinement. (Refinement, which we have not yet addressed directly, is the topic of Chap. 9 to come.) Finally, we will sometimes be using *disjoint union* (\sqcup) when we are composing channels whose output sets overlap only incidentally.

We now consider a number of compositions in turn.

8.1.1 Parallel composition

Consider a database that contains some user’s personal data, and which permits two queries: one that reveals the user’s age, and another that reveals the user’s address. Naturally, an adversary with access to both queries can combine the result from each of them to obtain both age and address, i.e. more comprehensive knowledge than either query would have provided on its own. Intuitively, the end result for this adversary should be the same as the one she would obtain by using a single query that revealed both the user’s age and address at the same time.

To model such scenarios we use *parallel composition*, capturing situations where the adversary observes the results of two compatible channels operating independently on the same secret.

Definition 8.1 (Parallel composition of channel matrices)

Let $C^1: \mathcal{X} \rightarrow \mathcal{Y}^1$ and $C^2: \mathcal{X} \rightarrow \mathcal{Y}^2$ be compatible channels. Their *parallel composition* $C^1 \parallel C^2$ is of type $\mathcal{X} \rightarrow \mathcal{Y}^1 \times \mathcal{Y}^2$ and is defined so that for all $x: \mathcal{X}$ and $y_1: \mathcal{Y}^1$ and $y_2: \mathcal{Y}^2$ we have

$$(C^1 \parallel C^2)_{x,(y_1,y_2)} := C^1_{x,y_1} \times C^2_{x,y_2} .^3$$

² An exception to that however is channel *cascade*: see §8.1.8 below.

³ In Exercise 8.2 and §9.7.1 we will temporarily write concrete parallel composition as $(|||)$.

Note that our mathematical definition of parallel (just above) is appropriate to describe only events that are independent in the sense of elementary probability theory, i.e. when $p(y_1, y_2|x) = p(y_1|x)p(y_2|x)$. As an example, we illustrate the parallel composition of two channels C^1, C^2 with

$$\begin{array}{c|cc} C^1 & y_1 & y_2 \\ \hline x_1 & 0.4 & 0.6 \\ x_2 & 0.8 & 0.2 \end{array} \parallel \begin{array}{c|cc} C^2 & y_1 & y_3 \\ \hline x_1 & 1 & 0 \\ x_2 & 0.3 & 0.7 \end{array} = \begin{array}{c|cccc} C^1 \parallel C^2 & (y_1, y_1) & (y_1, y_3) & (y_2, y_1) & (y_2, y_3) \\ \hline x_1 & 0.4 & 0 & 0.6 & 0 \\ x_2 & 0.24 & 0.56 & 0.06 & 0.14 \end{array} .$$

To compose a channel C many times in parallel with itself, we write $C^{(n)}$; see §14.6.3 and Chap. 19 for examples of that. It is called “repeated independent runs”.

8.1.2 External fixed-probability choice

Consider a protocol that, upon receiving a request, uses a fixed probability to select one of two possible servers for it (based e.g. on the current network traffic), but in such a way that afterwards it is known (e.g. to a possible adversary) which server was selected. From the point of view of a user, the behavior of the protocol is equivalent to the expected behavior of each possible server, weighted by the probability that each server is used.

Protocols such as the one above can be modeled using *external fixed-probability choice* composition. The “external” means that the choice is known (afterwards), and the “fixed” means that the probability used to make the choice does not depend on the input.

Thus with some probability p , the system directs the secret to the first channel and (therefore) with probability $1-p$ directs it to the second channel. In the end, the system reveals the output produced, together with an identification of which channel was used. (Compare the convex combination of channels in §4.6.1.)

Definition 8.2 (External fixed-probability choice between channel matrices)

Let $C^1: \mathcal{X} \rightarrow \mathcal{Y}^1$ and $C^2: \mathcal{X} \rightarrow \mathcal{Y}^2$ be compatible channels. Their *external probabilistic choice with probability p* is of type $\mathcal{X} \rightarrow (\mathcal{Y}^1 \uplus \mathcal{Y}^2)$ and is defined

$$(C^1 \xrightarrow{p} C^2)_{x,y} := \begin{cases} p C^1_{x,y} & \text{if } y \text{ originated from } \mathcal{Y}^1 \\ (1-p) C^2_{x,y} & \text{if } y \text{ originated from } \mathcal{Y}^2 \end{cases} .$$

(Note that our use of disjoint union in the output type $\mathcal{Y}^1 \uplus \mathcal{Y}^2$ has the effect of acting as if the two channels did not share output values, by introducing a distinction among equal outputs whenever it is necessary.)

An example of fixed-probability external choice is

$$\begin{array}{|c|cc|} \hline C^1 & y_1 & y_2 \\ \hline x_1 & 0.4 & 0.6 \\ x_2 & 0.8 & 0.2 \\ \hline \end{array} \quad \begin{matrix} 1/4 \boxplus \\ \text{---} \end{matrix} \quad \begin{array}{|c|cc|} \hline C^2 & y_1 & y_3 \\ \hline x_1 & 1 & 0 \\ x_2 & 0.3 & 0.7 \\ \hline \end{array} = \begin{array}{|c|cccc|} \hline C^1 \underset{1/4}{\boxplus} C^2 & y_1^1 & y_2 & y_1^2 & y_3 \\ \hline x_1 & 0.1 & 0.15 & 0.75 & 0 \\ x_2 & 0.2 & 0.05 & 0.225 & 0.525 \\ \hline \end{array},$$

where the probability is $1/4$ no matter what the input. By the superscripts y_1^1 and y_1^2 we are indicating our use of disjoint union, equivalently that if the adversary observes output y_1 she will know which channel produced it.

8.1.3 External conditional choice

Consider now a protocol that, upon receiving a request, uses some property of the input (instead of a fixed probability) to select externally between two possible servers — again meaning by “external” that afterwards it is known which server was selected, i.e. that knowledge is “externalized”. The scenario is captured by *external conditional choice*, modeling situations in which the adversary observes the output of a composite channel and knows which of its components was used.

Definition 8.3 (External conditional choice between channel matrices)

Let $C^1: \mathcal{X} \rightarrow \mathcal{Y}^1$ and $C^2: \mathcal{X} \rightarrow \mathcal{Y}^2$ be compatible channels. Their *external conditional choice* wrt. a subset \mathcal{A} of \mathcal{X} is (again) of type $\mathcal{X} \rightarrow (\mathcal{Y}^1 \uplus \mathcal{Y}^2)$ and is defined

$$(C^1 \triangleleft \mathcal{A} \triangleright C^2)_{x,y} := \begin{cases} C^1_{x,y} & \text{if } x \in \mathcal{A} \text{ and } y \text{ originated from } \mathcal{Y}^1 \\ C^2_{x,y} & \text{if } x \notin \mathcal{A} \text{ and } y \text{ originated from } \mathcal{Y}^2 \\ 0 & \text{otherwise.} \end{cases}$$

As an example, consider the secret input set $\mathcal{X} = \{x_1, x_2, x_3, x_4\}$, and the condition \mathcal{A} on it defined by the subset $\{x_1, x_2\}$. The external conditional choice $C^1 \triangleleft \mathcal{A} \triangleright C^2$, between the two channels C^1 and C^2 below is given by

$$\begin{array}{|c|cc|} \hline C^1 & y_1 & y_2 \\ \hline x_1 & 0.5 & 0.5 \\ x_2 & 0.3 & 0.7 \\ x_3 & 0 & 1 \\ x_4 & 0.6 & 0.4 \\ \hline \end{array} \quad \triangleleft \{x_1, x_2\} \triangleright \quad \begin{array}{|c|cc|} \hline C^2 & y_1 & y_3 \\ \hline x_1 & 0.1 & 0.9 \\ x_2 & 0.7 & 0.3 \\ x_3 & 0.4 & 0.6 \\ x_4 & 0.8 & 0.2 \\ \hline \end{array} = \begin{array}{|c|cccc|} \hline C^1 \triangleleft \{x_1, x_2\} \triangleright C^2 & y_1^1 & y_2 & y_1^2 & y_3 \\ \hline x_1 & 0.5 & 0.5 & 0 & 0 \\ x_2 & 0.3 & 0.7 & 0 & 0 \\ x_3 & 0 & 0 & 0.4 & 0.6 \\ x_4 & 0 & 0 & 0.8 & 0.2 \\ \hline \end{array},$$

where again we use superscripts (where needed) to recognize the use of disjoint union.

8.1.4 External (general) probabilistic choice

Both external fixed-probability choice and external conditional choice are in fact special cases of a more general form of external probabilistic choice, where the choice between two channels is made probabilistically, and the probability used can depend on the input. Again, since the choice is external, the channel used is revealed afterwards to the adversary.

The fixed probabilistic choice is the special case where the dependence on the input is trivial, i.e. is always some constant p , so that one could equivalently say that the probability *function* is actually a constant function of the input. And the conditional choice is where the probability is indeed a function, but in fact that probability can only be 0 or 1.

More precisely, for each secret value x in \mathcal{X} there is a probability $P(x)$ that the left-hand channel will be used; and with probability $1-P(x)$ the right-hand channel will be used instead.

Definition 8.4 (External probabilistic choice between channel matrices)

Let $C^1: \mathcal{X} \rightarrow \mathcal{Y}^1$ and $C^2: \mathcal{X} \rightarrow \mathcal{Y}^2$ be compatible channels. Their *external probabilistic choice* wrt. function P of type $\mathcal{X} \rightarrow [0, 1]$ is of type $\mathcal{X} \rightarrow (\mathcal{Y}^1 \uplus \mathcal{Y}^2)$ and is defined

$$(C^1 \underset{P}{\boxplus} C^2)_{x,y} := \begin{cases} P(x) C^1_{x,y} & \text{if } y \text{ originated from } \mathcal{Y}^1 \\ (1-P(x)) C^2_{x,y} & \text{if } y \text{ originated from } \mathcal{Y}^2. \end{cases}$$

As an example, consider the secret input set $\mathcal{X} = \{x_1, x_2, x_3\}$, and let P be the probability-valued function

x	$P(x)$	$1-P(x)$
x_1	0.25	0.75
x_2	1	0
x_3	0.5	0.5

Note that P above is not a probability distribution on \mathcal{X} , as indeed is reflected by the fact that $\sum_{x \in \mathcal{X}} P(x)$ is not 1. In fact, for each x , the distribution $(P(x), 1-P(x))$ is over the selection of channel C_1 or C_2 to be activated during the composition.

The external probabilistic choice $C^1 \underset{P}{\boxplus} C^2$, between channels C^1 and C^2 below, is

$$\begin{array}{|c|cc|} \hline C^1 & y_1 & y_2 \\ \hline x_1 & 0.5 & 0.5 \\ x_2 & 0.3 & 0.7 \\ x_3 & 0 & 1 \\ \hline \end{array} \underset{P \boxplus}{=} \begin{array}{|c|cc|} \hline C^2 & y_1 & y_3 \\ \hline x_1 & 0.1 & 0.9 \\ x_2 & 0.7 & 0.3 \\ x_3 & 0.4 & 0.6 \\ \hline \end{array}$$

$$\begin{array}{|c|cccc|} \hline C^1 \underset{P}{\boxplus} C^2 & y_1^1 & y_2 & y_1^2 & y_3 \\ \hline x_1 & 0.125 & 0.125 & 0.075 & 0.675 \\ x_2 & 0.3 & 0.7 & 0 & 0 \\ x_3 & 0 & 0.5 & 0.2 & 0.3 \\ \hline \end{array},$$

where yet again we use superscripts (where needed) to indicate disjoint union.

8.1.5 Internal fixed-probability choice

Now consider a scenario in which an interviewer wants to learn the proportion of a population satisfying some sensitive criterion (related to e.g. consumption of illegal drugs, or to political opinion). The following protocol can be used to protect the privacy of respondents with respect to such a sensitive “yes/no” question:

Warner’s protocol — When presented with the question, each respondent flips a fair coin *in secret*, and behaves in one of two ways depending on the outcome: if the coin comes up heads, he answers the question truthfully; but if it comes up tails, he flips it (privately) again, and answers “yes” for heads and “no” for tails.

That is effectively a probabilistic choice between two channels; the identity channel $\mathbf{0}$ (tell the truth), and the reveal-nothing channel $\mathbf{1}$ (answer randomly).⁴ What makes the choice “internal” is that the adversary does not know directly which of the two channels was used.

It can be shown that in Warner’s protocol the interviewer is able to estimate accurately the results of truthful answers –and, hence, estimate the proportion of the population satisfying the sensitive criterion– by subtracting the expected number of random answers from the total number of answers. (If the coin is fair, the interviewer knows 25% of all answers are expected to be a random “yes” response and 25% a random “no” response, and the relative proportion of “yes”/“no” answers in the remaining 50% is expected to be close to the real proportion of each answer in the population.) However, if the interviewer does not know the result of the coin tosses she cannot know whether any given respondent’s answer is truthful, which grants a degree of *plausible deniability*.

The above scenario is modeled with *internal fixed-probability choice*, used in situations where the system has a choice of feeding its secret input to one component (with probability p), or to another component (with probability $1-p$). In the end, the system reveals the output produced, but, unlike external choice, internal choice does not reveal explicitly which channel was used. Hence when the observations are randomized between the two channels, the adversary cannot in general identify which channel produced the observation.

Definition 8.5 (Internal fixed-probability choice between channel matrices)

Let $C^1: \mathcal{X} \rightarrow \mathcal{Y}^1$ and $C^2: \mathcal{X} \rightarrow \mathcal{Y}^2$ be compatible channels. Their *internal probabilistic choice with fixed probability p* is of type $\mathcal{X} \rightarrow (\mathcal{Y}^1 \cup \mathcal{Y}^2)$ and is defined

$$(C^1 \underset{p}{\oplus} C^2)_{x,y} := \begin{cases} p C^1_{x,y} + (1-p) C^2_{x,y} & \text{if } y \in \mathcal{Y}^1 \cap \mathcal{Y}^2 \\ p C^1_{x,y} & \text{if } y \in \mathcal{Y}^1 - \mathcal{Y}^2 \\ (1-p) C^2_{x,y} & \text{if } y \in \mathcal{Y}^2 - \mathcal{Y}^1 \end{cases} .$$

Note that we do not use disjoint union to give the type of internal choice — instead, we use ordinary union. If however \mathcal{Y}^1 and \mathcal{Y}^2 are disjoint, then internal choice reduces to external choice because whether y is in \mathcal{Y}^1 or \mathcal{Y}^2 reveals which channel was used. (That is why we said “in general” above.)

⁴ Recall from Example 4.12 that the assignment of “0” to the virtue of telling the truth (and complementarily assigning “1” to the vice of evasion) is supported by the intuition –for us here– that a channel that tells the truth to an adversary is a bad thing.

As an example, the internal probabilistic choice, fixed wrt. $p = 1/4$, of the two channels C^1, C^2 below is

$$\begin{array}{|c|cc|} \hline C^1 & y_1 & y_2 \\ \hline x_1 & 0.4 & 0.6 \\ x_2 & 0.8 & 0.2 \\ \hline \end{array} \quad \begin{matrix} 1/4 \oplus \\ \quad \quad \quad \end{matrix} \quad \begin{array}{|c|cc|} \hline C^2 & y_1 & y_3 \\ \hline x_1 & 1 & 0 \\ x_2 & 0.3 & 0.7 \\ \hline \end{array} = \quad \begin{array}{|c|ccc|} \hline C^1_{1/4 \oplus} C^2 & y_1 & y_2 & y_3 \\ \hline x_1 & 0.85 & 0.15 & 0 \\ x_2 & 0.425 & 0.05 & 0.525 \\ \hline \end{array} .$$

Further examples of internal choice are given in §4.6.1 (Boris and Natasha) and at (17.3) in §17.4 (Ms. Vowel and Mrs. Early).

8.1.6 Internal conditional choice

Consider again a protocol that, upon receiving a request, uses some property of the input (instead of a fixed probability) to select between two possible servers to process it. Contrarily to external choice, however, here the protocol's choice is “internal”, in the sense that afterwards it is not explicitly revealed to the user which server was selected. We capture that with *internal conditional choice*, modeling situations in which the adversary observes the output of a composite channel but does not know which of its two components produced the output.

The formalization of internal conditional choice is analogous to that of its external counterpart, as discussed in §8.1.3, and is left to the reader as Exercise 8.6.

8.1.7 Internal (general) probabilistic choice

Analogously to their external counterparts, both internal fixed-probability choice and internal conditional choice are in fact special cases of a more general form of internal probabilistic choice, where the choice between two channels is made probabilistically, and the probability used can depend on the input. Here, since the choice is internal, the channel used is not explicitly revealed to the adversary. Again, for each secret value x in \mathcal{X} there is a probability $P(x)$ that the left-hand channel will be used; and with probability $1-P(x)$ the right-hand channel will be used instead.

The formalization of internal (general) probabilistic choice is analogous to that of its external counterpart, given above in §8.1.4, and is left to the reader as Exercise 8.7.

8.1.8 Cascading

Finally, we mention for completeness the *cascading composition*, which was defined in §4.6.2: it models a scenario in which the output of a (concrete) channel is “captured” by a second concrete channel, and in fact never reaches the adversary directly. Instead it becomes the input to a second (concrete) channel and it is only the second output that is observed.

Here the component channels are not compatible (in the sense above) because the input of the second is the output (not the input) of the first. For *communication* channels that models the operational idea of connecting the “output wires” of the first to the “input wires” of the second one, and then determining the information-theoretic leakage of the two channels taken together, with their interface –the connected wires– hidden from the adversary.⁵

8.2 Compositions of abstract channels

8.2.1 The issue of compositionality

In this section we will present operations on abstract channels, and they will correspond (in a way to be explained below) with the operations on concrete channels that we presented in §8.1.

By “correspond” we mean that it will not matter whether we model our system with concrete channels, and take e.g. concrete parallel composition (as defined above) or model with abstract channels and take abstract parallel composition (to be defined below). What we mean by “will not matter” is made precise by the concept of compositionality.

Definition 8.6 (Compositionality) Suppose we have a concrete model (like concrete channels) and an abstract model (like abstract channels) and an abstraction function $\llbracket - \rrbracket$ that takes concrete elements to abstract elements (as in our taking concrete channel C to abstract channel C by writing $C = \llbracket C \rrbracket$).

Suppose further that there is a concrete operator, say generically (\heartsuit) , that operates between elements of the concrete model. (Here we will concentrate on binary operators.)

Then the abstraction $\llbracket - \rrbracket$ is *compositional* for the (binary) concrete operator (\heartsuit) just when there is an abstract operator (\diamond) corresponding to the concrete (\heartsuit) such that for all concrete elements $C^{1,2}$ we have

$$\llbracket C^1 \heartsuit C^2 \rrbracket = \llbracket C^1 \rrbracket \diamond \llbracket C^2 \rrbracket . \quad (8.1)$$

□

We can make that more compelling with this definition.

Definition 8.7 (Semantic equivalence) Write $C^1 \equiv C^2$ for $\llbracket C^1 \rrbracket = \llbracket C^2 \rrbracket$.

□

Then Def. 8.6 is equivalent to

$$C^1 \equiv D^1 \quad \text{and} \quad C^2 \equiv D^2 \quad \text{implies} \quad C^1 \heartsuit C^2 \equiv D^1 \heartsuit D^2 . \quad (8.2)$$

(See Exercise 8.10.)

An intuitive example of *non*-compositionality in everyday life is the abstraction *eye color* of a person: from the point of view of (8.1) we would say there is no abstract operation that gives the (distribution of) a child’s eye color from the eye colors of the parents (the operands) and the concrete operation of “making a child” that produced her; and that is because the abstraction here has discarded too much. With (8.2) we would (equivalently) say that it’s possible for two sets of parents to have the same eye-color pair (say the two fathers’ eye colors agree, and the two mothers’) while their respective children’s (distribution of) eye colors differ.

⁵ Cascading is thus an example of a binary operator which is defined between non-compatible channel matrices.

The abstraction eye-color *alleles* however (i.e. dominant and recessive) is compositional for the operation of producing children — that is, one can determine (the distribution of) children’s eye-color alleles from the eye-color alleles of their parents (8.1).⁶ And if two sets of parents’ alleles agree, then the (distribution of) their respective children’s alleles will agree as well (8.2).

Finally — in our more extensive discussion of compositions, to come in §9.7, we will refer to possible “contexts”, by which we will mean systems constructed by placing an element of interest –a channel– in an expression built from the operations we have defined. In that case we will speak of an abstraction’s being “compositional” for a given set of contexts, i.e. more generally than simply for certain named operators.

We now turn to the definition of our abstract operators.

8.2.2 Parallel composition

Here is the abstract definition of parallel composition.

Definition 8.8 (Parallel composition between abstract channels) For abstract channels $C^{1,2}: \mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$ the parallel composition (\parallel) is given by

$$(C^1 \parallel C^2)(\pi) := \sum_{\delta: \mathbb{D}\mathcal{X}} C^1(\pi)_\delta \times C^2(\delta) \quad , \quad ^7$$
(8.3)

where by $C^1(\pi)_\delta$ we mean that abstract channel C^1 is first applied to the prior π (using the alternative notation given after Def. 4.7). Then the resulting hyper –a distribution over inner distributions– is itself applied (using the notation of Def. 4.5) to an arbitrary distribution δ in $\mathbb{D}\mathcal{X}$ as introduced by the summation, giving the outer probability that hyper $C^1(\pi)$ assigns to that inner. Of course the summation could be restricted to letting δ range over only the support of $C^1(\pi)$, which is conceptually simpler but gives more notational clutter.⁸

□

Now of course the abstract operator (\parallel) here in Def. 8.8 is different –indeed has a different type– from the operator (\parallel) used in the concrete version of parallel composition defined above in §8.1.1. That is because the abstract version of (\parallel) here corresponds to (\diamond) in (8.1), and the concrete (\parallel) in Def. 8.1 corresponds to (\heartsuit). See Exercise 8.2 concerning the proof that (8.1) is satisfied in this case, i.e. that parallel composition has indeed been defined compositionally for our abstraction $\llbracket - \rrbracket$. (A more general view is given in §9.7.1.)

8.2.3 External fixed-probability choice

Here is the abstract definition of external fixed-probability choice.

Definition 8.9 (External fixed-probability choice between abstract channels) For abstract channels $C^{1,2}: \mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$ the external fixed-probability choice is given by

$$(C^1 \underset{p}{\boxplus} C^2)(\pi) := p \times C^1(\pi) + (1-p) \times C^2(\pi) \quad .$$

It is simply the p -wise linear combination of the two arguments’ output hypers. □

⁶ Mendel showed how to do that: Versuche über Pflanzenhybriden, 1866.

⁷ Although this definition is asymmetrically expressed, it is in fact commutative since it abstracts from concrete-channel parallel composition (Def. 8.1) which is trivially commutative. See Exercises 8.1 and 8.2.

⁸ In §14.3.1 we will recognize the right-hand side of (8.3) as Kleisli composition.

8.2.4 External conditional choice

Here is the abstract definition of external conditional choice.

Definition 8.10 (External conditional choice between abstract channels)

Write $\pi(\mathcal{A})$ for the probability that π assigns to \mathcal{A} , and write $\pi|_{\mathcal{A}}$ for π conditioned on \mathcal{A} , and write $\overline{\mathcal{A}}$ for $\mathcal{X} - \mathcal{A}$. For abstract channels $C^{1,2}: \mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$ the external conditional choice is then given by

$$(C^1 \triangleleft \mathcal{A} \triangleright C^2)(\pi) := \pi(\mathcal{A}) \times C^1(\pi|_{\mathcal{A}}) + \pi(\overline{\mathcal{A}}) \times C^2(\pi|_{\overline{\mathcal{A}}}),$$

with the proviso that if either $\pi(\mathcal{A})$ or $\pi(\overline{\mathcal{A}})$ is 1 then the other summand is ignored. In this case, we simply condition the prior π on \mathcal{A} resp. $\overline{\mathcal{A}}$, and then combine the output hypers resulting from those as priors with the probability that π associates with \mathcal{A} resp. $\overline{\mathcal{A}}$.

□

8.2.5 External (general) probabilistic choice

Here is the abstract definition of external (general) probabilistic choice.

Definition 8.11 (External general probabilistic choice between abstract channels)

Write $\pi(P)$ for $\sum_x \pi_x P(x)$ and write $\pi(\overline{P})$ for $\sum_x \pi_x (1-P(x))$. Let $(\pi/P)_x$ be $\pi_x P(x)/\pi(P)$ and similarly for \overline{P} .

For abstract channels $C^{1,2}$ in $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$ the external (general) probabilistic choice is given by

$$(C^1 \boxplus C^2)(\pi) := \pi(P) C^1(\pi/P) + \pi(\overline{P}) C^2(\pi/\overline{P}),$$

with the proviso that if either $\pi(P)$ or $\pi(\overline{P})$ is 1 then the other summand is ignored.
□

These three versions of choice are compositional (Exercise 8.4) and, as for concrete external choices, choices §8.2.3 and §8.2.4 are both special cases of §8.2.5 (Exercise 8.3).

8.2.6 The internal choices, and cascading

For internal probabilistic choice (of any kind) and for cascading, there are no corresponding abstract operators.⁹

With Def. 8.6 at our disposal, we can say that the reason for that is that our abstraction from channel matrices to abstract channels is *not compositional* for those operators. That does not mean that cascading or the internal choices cannot or should not be used — indeed, they are very important operators. But what it *does* mean is that we cannot deduce the quantitative-information-flow, i.e. the *QIF* properties of say the internal choice between two channels, if all we know is the *QIF* properties of those channels in isolation. We will now give some intuitive suggestions for why that is so.

We have already observed in §4.4 that the leakage properties we study are independent of the column (our output) labels of the channels that produce them. We make that precise as follows.

⁹ In spite of that, multiplication of matrices *can* be expressed at the abstract level $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$, as we show in §14.4.8 to come.

Definition 8.12 (Label equivalence) Say that two concrete channels $C^{1,2}$ of type $\mathcal{X} \rightarrow \mathcal{Y}$ are *equivalent up to labeling*, written $C^1 \sim C^2$, just when one can be converted into the other via permuting their column labels. \square

Lemma 8.13 (Label independence of QIF properties) If two channels are label equivalent, then their leakage properties are the same. That is, for any two concrete channels $C^{1,2}$ of the same type $\mathcal{X} \rightarrow \mathcal{Y}$ we have that

$$C^1 \sim C^2 \quad \text{implies} \quad V_g[\pi \triangleright C^1] = V_g[\pi \triangleright C^2]$$

for all priors π and gain functions g . ¹⁰ \square

Lem. 8.13 is of course foundational to our whole approach (and its proof is trivial, since the definition of $V_g[\pi \triangleright C]$ does not refer to labels). We would like to carry it through, as far as possible, with our operators as well.

Definition 8.14 (Label independence of operators) Say that an operator (\heartsuit) on concrete channels is *label independent* just when for any concrete channels $C^{1,2}$ and $D^{1,2}$ of the same type and any gain function g , we have

$$C^1 \sim D^1 \quad \text{and} \quad C^2 \sim D^2 \quad \text{implies} \quad V_g[\pi \triangleright (C^1 \heartsuit C^2)] = V_g[\pi \triangleright (D^1 \heartsuit D^2)] .$$

If an operator is not label independent, we say that it is *label dependent*. \square

Two simple examples of dependence and independence are these. A channel cascade is label dependent, because the output labels of the first component are used to “connect” those outputs to the inputs of the second component; and external choice is label independent because the two output types are combined by *disjoint* union. A less immediately obvious case is internal choice, which is label dependent because it depends on which labels are shared between the two output types (if any).

It turns out that label independence has a lot to do with whether or not our channel operations can be defined abstractly, i.e. whether $\llbracket - \rrbracket$ is compositional for them (recalling Def. 8.6).

Lemma 8.15 If operator (\heartsuit) is label dependent then the abstraction $\llbracket - \rrbracket$ is not compositional for it, which is to say that it has no abstract counterpart (\diamondsuit) . That is, there exist channels $C^{1,2}$ and $D^{1,2}$ such that

$$C^1 \equiv D^1 \quad \text{and} \quad C^2 \equiv D^2 \quad \text{but} \quad C^1 \heartsuit C^2 \not\equiv D^1 \heartsuit D^2 . \quad (8.4)$$

Proof. If operator (\heartsuit) is label dependent then from Def. 8.14 there are $C^{1,2}$ and $D^{1,2}$ with $C^1 \sim D^1$ and $C^2 \sim D^2$ but $V_g[\pi \triangleright (C^1 \heartsuit C^2)] \neq V_g[\pi \triangleright (D^1 \heartsuit D^2)]$ for some prior π and gain function g .

But in general $C \sim D$ implies $C \equiv D$; and $C' \equiv D'$ implies $V_g[\pi \triangleright C'] = V_g[\pi \triangleright D']$ for all priors π and gain functions g . Thus (8.4) contradicts the formulation (8.2) of compositionality. \square

We can now see that both parallel and the external choices defined above are label independent, but that cascading and the internal choices are label dependent. In other words, the label dependence of the last two gives us an intuitive indication of why they cannot be abstracted.

¹⁰ The converse is not true.

Historically, compositionality was seen as crucial to the semantics of sequential programs — precisely because it is a way to break a large proof of correctness into smaller pieces. Operators that preserve security properties are therefore paramount in verification. The same can be said here, i.e. that operators that preserve leakage properties have the potential to ease the calculation of leakage properties for large systems by breaking the calculation into smaller parts that can be reliably combined. We explore these ideas further with an abstract semantics for the sequential programs that we will see in Part IV.

8.3 Exercises

Exercise 8.1 Recall Def. 8.8 of abstract-channel parallel composition. Use the matrices $C^{1,2}$ from §8.1.1 on concrete-channel parallel composition, and the uniform prior distribution ϑ over $\mathcal{X} = \{x_1, x_2\}$, to illustrate the correspondence between the abstract and the concrete definitions.

In particular, show that $[\vartheta \triangleright C^1]$ is the hyper-distribution

	3/5	2/5
x_1	1/3	3/4
x_2	2/3	1/4

so that the inners δ, δ' , over which the summation on the right-hand side of Def. 8.8 is taken, are the two shown above: that is, $(1/3, 2/3)$ and $(3/4, 1/4)$. The corresponding values of $C^1(\vartheta)_\delta$ and $C^1(\vartheta)_{\delta'}$ are then the two outers $3/5$ for δ and $2/5$ for δ' .

Then calculate the hypers $[\delta \triangleright C^2]$ and $[\delta' \triangleright C^2]$ to complete the summation in Def. 8.8, and verify that the result is indeed $[\vartheta \triangleright (C^1 \parallel C^2)]$ where $C^1 \parallel C^2$ is as calculated in the example following Def. 8.1.

Will $C^2 \parallel C^1$ give the same result? □

Exercise 8.2 Write (\parallel) for the binary operator from §8.1.1, on compatible channel matrices say of type $\mathcal{X} \rightarrow \mathcal{Y}^1$ and $\mathcal{X} \rightarrow \mathcal{Y}^2$, that makes a new channel matrix of type $\mathcal{X} \rightarrow (\mathcal{Y}^1 \times \mathcal{Y}^2)$ representing the two component matrices acting in parallel. Compositionalty of our abstraction $\llbracket - \rrbracket$ wrt. the operation of parallel composition can then be expressed as

$$\llbracket C \parallel D \rrbracket = \llbracket C \rrbracket \parallel \llbracket D \rrbracket , \quad (8.5)$$

where (\parallel) on the left is the concrete version, now (but only temporarily) distinguished notationally from the abstract version (\parallel) on the right as defined in §8.2.2. That is, the (\parallel) on the right is the parallel operator on abstract channels: it is of different type to (\parallel) , but has the same conceptual purpose. The effect of parallel composition, whether abstract or concrete, is as if both constituent channels were applied to the same input X and the outputs $Y^{1,2}$ collected from each. Compositionalty (8.5) then says that the syntactic (\parallel) and the semantic (\parallel) have essentially the same effect, even though their definitions are quite different.

Prove (8.5), i.e. that our abstraction is compositional for parallel composition. □

Exercise 8.3 Show that Def. 8.9 and Def. 8.10 are indeed both special cases of Def. 8.11. □

Exercise 8.4 Recall Definitions 8.4 and 8.11 and the notations introduced there. Prove that our abstraction $\llbracket - \rrbracket$ is compositional for external choice. □

Exercise 8.5 Our motivating example for internal choice was in fact the composition $\mathbb{O}_{1/2} \oplus \mathbb{1}$, where we interpreted $\mathbb{O}, \mathbb{1}$ in their *concrete* form, i.e. both as matrices of type $\{x_1, x_2\} \rightarrow \{y_1, y_2\}$. What is the channel matrix for that composition? \square

Exercise 8.6 Recall §8.1.6. Give a definition of internal conditional choice analogous to Def. 8.3. Work out the behavior of this new composition on the same channels used in §8.1.3 to illustrate the behavior of external conditional choice. \square

Exercise 8.7 Recall §8.1.7. Give a definition of internal (general) probabilistic choice analogous to Def. 8.4. Work out the behavior of this new composition on the same channels used in §8.1.4 to illustrate the behavior of external (general) probabilistic choice. \square

Exercise 8.8 Give an example to show that cascading is not compositional. \square

Exercise 8.9 Show that internal fixed-probability choice can be defined by external fixed-probability choice followed by a cascade.

[Hint: Analyze whether the cascading that you construct depends on the labels of the channel it is post-processing.] \square

Exercise 8.10 Recall §8.2.1. By defining a suitable (\diamond) where necessary, show that the two characterizations (8.1) and (8.2) of compositionality are equivalent. \square

Exercise 8.11 Prove that $\mathcal{ML}_1^{\times}(\mathbb{D}, C_1 \parallel C_2) \leq \mathcal{ML}_1^{\times}(\mathbb{D}, C_1) \times \mathcal{ML}_1^{\times}(\mathbb{D}, C_2)$. \square

Exercise 8.12 Show that posterior g -vulnerability $V_g(-)$ is:

(a) Linear wrt. external fixed-probability choice, i.e. that

$$V_g[\pi \triangleright C^1 p \boxplus C^2] = p \times V_g[\pi \triangleright C^1] + (1-p) \times V_g[\pi \triangleright C^2].$$

(b) Convex wrt. internal fixed-probability choice, i.e. that

$$V_g[\pi \triangleright C^1 p \oplus C^2] \leq p \times V_g[\pi \triangleright C^1] + (1-p) \times V_g[\pi \triangleright C^2].$$

\square

8.4 Chapter notes

The protocol for releasing statistical information about a “yes”/“no” question, while granting plausible deniability for participants, is discussed as an example of internal fixed-probability choice; it is based on Warner’s randomized-response protocol [8]. (An example of its use in differential privacy is given in Exercise 23.1.)

One of the early examples of compositionality for security properties appears in McCullough’s “hook-up theorem” where he shows how to compose a collection of secure-restrictive systems to form a secure-restrictive composite system [3]. Such compositionality results are less common in quantitative information flow generally, primarily because for a long time there was no notion of refinement that expressed how leakage properties could be preserved. An early qualitative example of refinement and operators was Morgan’s Shadow semantics [5, 6, 7], also described in §17.4, in which a distinction is also made between internal and external choice. (The context there however is demonic choice rather than probabilistic choice, as in Chap. 17 below.)

8 Composition of channels

Espinoza and Smith [2] derive a number of bounds on multiplicative Bayes capacity (which they call *min-capacity*) for different channel compositions, including cascading and parallel composition. McIver et al. [4] explore compositionality generally for a number of operators, including the abstract operators described here and later in §14.4. Américo et al. [1] provide a set of operators modeling system-component interactions, and study their algebraic properties with respect to the security-preserving (i.e. compositional) refinement relation.

Bibliography

- [1] Américo, A., Alvim, M.S., McIver, A.: An algebraic approach for reasoning about information flow. In: K. Havelund, J. Peleska, B. Roscoe, E. de Vink (eds.) Proceedings of the 22nd International Symposium on Formal Methods (FM 2018), *Lecture Notes in Computer Science*, vol. 10951, pp. 55–72. Springer, Berlin (2018)
- [2] Espinoza, B., Smith, G.: Min-entropy as a resource. *Information and Computation* **226**, 57–75 (2013)
- [3] McCullough, D.: A hookup theorem for multilevel security. *IEEE Transactions on Software Engineering* **16**(6), 563–568 (1990)
- [4] McIver, A., Meinicke, L., Morgan, C.: Compositional closure for Bayes risk in probabilistic noninterference. In: S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide, P.G. Spirakis (eds.) Proceedings of the 37th International Colloquium on Automata, Languages, and Programming (ICALP 2010), *Lecture Notes in Computer Science*, vol. 6199, pp. 223–235. Springer, Berlin (2010)
- [5] Morgan, C.: *The Shadow Knows*: Refinement of ignorance in sequential programs. In: T. Uustalu (ed.) Proceedings of the International Conference on Mathematics of Program Construction (MPC 2006), *Lecture Notes in Computer Science*, vol. 4014, pp. 359–378. Springer, Berlin (2006)
- [6] Morgan, C.: *The Shadow Knows*: Refinement of ignorance in sequential programs. *Science of Computer Programming* **74**(8), 629–653 (2009)
- [7] Morgan, C.: Compositional noninterference from first principles. *Formal Aspects of Computing* **24**(1), 3–26 (2012)
- [8] Warner, S.L.: Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association* **60**(309), 63–69 (1965)



Chapter 9

Refinement

As a technical term in Computer Science, “refinement” is understood as a relation between systems (programs, channels...) that, in its most austere form, is simply *preservation of properties*. One system S is refined by another system I , which eventually we will write $S \sqsubseteq I$, just when every property satisfied by S is satisfied by I also. Less formally, we could say that if a specification S is “good enough”, and an implementation I refines it, then we can be sure that I is good enough too.¹

The term was coined by Niklaus Wirth, who used it to describe a hierarchical method of program development with which one “refined” larger, possibly less precise descriptions into smaller, i.e. *finer*, more precise ones. And it did so in a way that each larger description was “implemented” by some combination of the smaller ones that replaced it, and as far as possible independently of other refinements going on elsewhere in the same overall development. (We will return to “independently” when we discuss compositionality in §9.7.) That is, the method was hierarchical in the sense that the smaller components themselves could be refined further still and –in principle at least– independently of other components. The whole process would stop when the now perhaps very small components were in fact directly implementable in some programming language or on some hardware — because of course it cannot go on forever.² In essence, one continues refining until the remaining pieces, the “tips of the development tree”, are programming-language code.

9.1 Refinement: for the *customer*; for the *developer*

Since its introduction, the meaning of “refinement” has become more abstract than as recalled above: it now no longer refers only to the process of splitting components into pieces. Instead it refers as well to the relation between the *meanings* of the small pieces and the larger one they replace, that the properties of the larger one should be preserved by the combination of small ones — just as mentioned in the paragraph at the start of this section.

But *which* properties? A crucial part of refinement’s definition, a parameter in fact, is exactly that: an unambiguous description of the properties that it must preserve. They could be called refinement’s “terms of reference”, and could be as simple as

¹ “Refinement is just implication.” *Eric C.R. Hehner*.

² Big fleas have little fleas upon their backs to bite 'em,
And little fleas have lesser fleas, and so, ad infinitum.

Augustus De Morgan

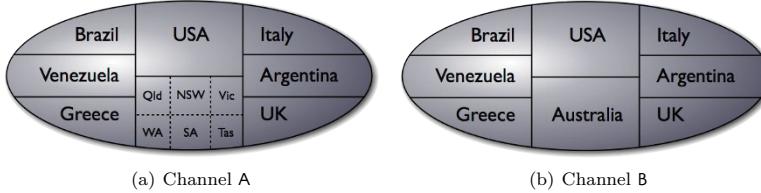


Figure 9.1 Partitions induced by channels A and B

functionality — that if $S \sqsubseteq I$ then any answer that I could give is an answer that S could also have given (from the same input). It could be time performance, that execution takes no more than so-many seconds on a certain hardware configuration; it could be both of those; it could be that some event can (or cannot) occur.

Or it could be “is at least as secure as”.

For our refinement here, we will concentrate on that last, that is on security properties as we have developed them so far. Our systems will be channels, and our properties will be based on “being secure”. Roughly speaking, for channels S, I we will have $S \sqsubseteq I$ just when I is at least as secure as S . More precisely however — as we are about to see — we will consider the *contexts* in which security might be an issue, those contexts defined, for example, by the prior π and by the criteria with which a security violation is measured (e.g. the gain- or loss function). And we will see that there are two important ways of formulating refinement. One, for the developer, we will call “structural refinement” — it is how refinement is *done* by the developer. The other we will call “testing refinement” — it is how refinement is *experienced* by the customer. A major contribution of our approach is to show that, suitably formulated, those two forms of refinement are —in fact— exactly the same.

9.2 Structural refinement: the developer’s point of view

9.2.1 Structural refinement for deterministic channels

We begin by defining structural refinement for *deterministic channels*. Any deterministic channel matrix from \mathcal{X} to \mathcal{Y} , i.e. of type $\mathcal{X} \rightarrow \mathcal{Y}$ (but, being deterministic, equivalently of type $\mathcal{X} \rightarrow \mathcal{Y}$), induces a *partition* on \mathcal{X} , which is a set of mutually disjoint subsets of \mathcal{X} that we call cells: inputs x_1 and x_2 in \mathcal{X} belong to the same cell just when the channel maps them to the same output observation in \mathcal{Y} . That is, each cell of the partition is the pre-image of some output y .³

Consider for example a deterministic channel A taking a secret person X to his location of birth, giving just the *country* of birth normally, but giving the *state* in the case of Australians.⁴ The induced partition is illustrated in Fig. 9.1(a).

Bearing in mind that we intend “is a refinement of” to mean “is at least as secure as”, how can we refine deterministic channel A to another deterministic channel B? One condition that is intuitively sufficient is that B’s partition should be “coarser” than A’s partition, in that each cell of B’s partition is formed by *merging* one or more cells of A’s partition. As an example, let channel B instead reveal only the *country*

³ Deterministic channels’ inducing equivalence relations is discussed also in §17.1, where it is remarked that the relation induced is called the *kernel* of the channel as a function.

⁴ Recall from §1.1.3 that X is the *name* of a secret, in this case e.g. “a person of interest”. Lower-case x is used for the actual *values* that X might take.

of birth in all cases. Its partition is illustrated in Fig. 9.1(b), where we can see that B 's partition is formed by merging the cells corresponding to the six Australian states (Queensland, New South Wales, Victoria, Western Australia, South Australia, and Tasmania) into a single cell (Australia).

That leads to the following definition of structural refinement (\sqsubseteq_o) for deterministic channels.

Definition 9.1 (Structural refinement (\sqsubseteq_o), deterministic case) Two deterministic channels A and B on input X are said to be in the *structural-refinement* relation, written $A \sqsubseteq_o B$, just when the partition induced by B is coarser than that induced by A , in that each of B 's cells is formed by *merging* one or more of A 's cells. \square

The above definition is essentially the same as an early definition of Landauer and Redmond, as explained later in §17.1, although they were not considering probabilistic priors.

It is intuitively clear that a security-conscious “customer” will *always* prefer B to A , whatever the input prior π or leakage measure might be (and an adversary’s preference will be exactly the opposite), since B never gives more information about X than A would have. That intuition is supported by the following theorem, which says that partition coarsening is *sound* with respect to Bayes leakage at least.⁵

Theorem 9.2 If A and B are deterministic channels and $A \sqsubseteq_o B$, then the Bayes leakage of B is never greater than that of A , regardless of the prior π .

Proof. Recall from Thm. 5.15 that posterior Bayes vulnerability is simply the sum of the column maximums of the joint matrix. In case $A \sqsubseteq_o B$, note that the joint matrix for B (under any prior π) is formed by *merging* some of the columns of the joint matrix for A , which can only decrease that sum (by converting a sum of maximums into a maximum of sums). \square

Later we will extend soundness to probabilistic channels (§9.4).

More interestingly, the converse implication holds as well. This means that partition coarsening is *complete* for Bayes leakage.

Theorem 9.3 (Deterministic coriaceous) If A and B are deterministic channels and the Bayes leakage of B is never greater than that of A , regardless of the prior π , then $A \sqsubseteq_o B$.

Proof. We argue the contrapositive. If $A \not\sqsubseteq_o B$, then there must exist x_1 and x_2 that belong to the *same* cell of A , but to *different* cells of B . On a prior that gives nonzero probability only to x_1 and x_2 , that is whose support is just $\{x_1, x_2\}$, channel A leaks nothing about X , while B leaks everything. \square

Just as for soundness, we return later to completeness for probabilistic channels (§9.5).

The Bayes leakage featuring in the two theorems above is of course a *testing-*, that is a customer-oriented (rather than a structural) artifact: the customer “experiences” with what probability his secrets can be guessed. We have discussed it here only to bolster the intuition, and later we return to the two theorems above when we discuss the equivalence of structural and testing refinement in general.

⁵ The “at least” reminds us that Bayes leakage is only one of many criteria a customer might have.

9.2.2 Structural refinement for probabilistic channels

9.2.2.1 Concrete channels

We now consider the question of how we can generalize structural refinement, that is partition-coarsening $A \sqsubseteq_{\circ} B$, from deterministic to probabilistic channels.

In the deterministic case, we saw that A 's partition is refined by B 's just when we can convert A into B by doing a “post-processing” step in which certain of A 's outputs are merged — and that in our example corresponds to federating the states of Australia to get from Fig. 9.1(a) to Fig. 9.1(b).⁶ Taking advantage of our matrix representation for channels, we can express merging as a *cascade*, i.e. that B is the *matrix product* of A and a channel matrix R : one imagines the observables of A “pouring in” to the input of R .⁷

Theorem 9.4 Let $A: \mathcal{X} \rightarrow \mathcal{Y}$ and $B: \mathcal{X} \rightarrow \mathcal{Z}$ be deterministic channel matrices. Then $A \sqsubseteq_{\circ} B$ just when there is a deterministic matrix $R: \mathcal{Y} \rightarrow \mathcal{Z}$ such that $B = AR$.

Proof. If $B = AR$ for some deterministic R , then $A(x_1) = A(x_2)$ implies that $B(x_1) = R(A(x_1)) = R(A(x_2)) = B(x_2)$, where because the matrices are deterministic we can consider them as functions — effectively we read $A: \mathcal{X} \rightarrow \mathcal{Y}$ as $A: \mathcal{X} \rightarrow \mathcal{Y}$. Hence every cell of A 's partition is entirely contained in a cell of B 's partition, which implies that $A \sqsubseteq_{\circ} B$. Note that each column of R determines a cell of B , where the 1's in that column indicate which of A 's cells were merged to make it.

Conversely, if $A \sqsubseteq_{\circ} B$, then for every y we know that B maps all x 's in $A^{-1}(y)$ to the same value, say y'_x . If we define (deterministic) R to map each y to y'_x , then indeed $B = AR$. \square

Now the formulation of structural refinement (\sqsubseteq_{\circ}) given by Thm. 9.4 has the advantage that it suggests a generalization that makes sense for probabilistic channels as well: remarkably, the type of the “refinement matrix” R , though not corresponding to an actual channel, has the same type as the channels for which it is used to define (structural) refinement. (In §17.3 we see that the same happens for demonic channels: there R too is demonic, having only 0's and 1's like a deterministic channel, but more general in that it might have several 1's in each row.) That leads us to make the following definition, in which the difference from Thm. 9.4 is only that A, B and R can be probabilistic.⁸ Again a cascade is used.

Definition 9.5 (Structural refinement (\sqsubseteq_{\circ}), probabilistic case)

For (probabilistic) channel matrices A and B , we say that A is structurally refined by B , again written $A \sqsubseteq_{\circ} B$, just when there exists a –possibly probabilistic– matrix R such that $AR = B$.

The refinement matrix R can be considered to be a *witness* for (structural) refinement, that is a single artifact that shows the refinement to hold. On the other hand, not having a witness R does not show that structural refinement does not hold — your not having one does not in itself mean that there isn't one. Perhaps you were simply not clever enough to find it. In §9.5 we see what a witness for *non-* refinement is.

⁶ It is an example of the *Data-Processing Inequality*.

⁷ The Panama Canal is a cascade of its locks.

⁸ We remark that, in some of the previous literature, refinement for information flow has been defined with the order of the arguments *reversed* wrt. our usage here. As mentioned earlier, we have ordered the arguments so that refinement goes from a less secure- to a more secure system, as is done in program refinement generally.

It is easy to see that (\sqsubseteq_\circ) is *reflexive* (since $A = A\mathbf{I}$, where \mathbf{I} is the identity matrix) and *transitive* (since $B = AR$ and $C = BR'$ implies $C = (AR)R' = A(RR')$ by associativity).⁹ Taken together, those two facts establish that structural refinement is (at least) a pre-order.

9.2.2.2 Abstract channels

We note however that Def. 9.5 is in terms of concrete channels, i.e. channel *matrices*, whereas our main object of study is actually *abstract* channels. It is possible to give an equivalent definition in terms of abstract channels directly, but for now we use a simpler, one could say “hybrid” definition that is sufficient (and equivalent).¹⁰

Definition 9.6 (abstract refinement) —

Let A and B be abstract channels over the same input \mathcal{X} . Then A is said to be structurally refined by B , again written $A \sqsubseteq_\circ B$, just when for *all* channel matrices \mathbf{A} and \mathbf{B} realizing A and B (i.e. for any \mathbf{A}, \mathbf{B} with $\llbracket \mathbf{A} \rrbracket = A$ and $\llbracket \mathbf{B} \rrbracket = B$), we have $\mathbf{A} \sqsubseteq_\circ \mathbf{B}$.

To show that the quantification over *all* concrete matrices makes sense in Def. 9.6, we recall *reduced matrices* from our earlier Cor. 4.11, and prove the following theorem.

Theorem 9.7 For any channel matrix C , we have both $C \sqsubseteq_\circ C^r$ and $C^r \sqsubseteq_\circ C$.

Proof. The reduced matrix C^r of channel matrix C is defined in Def. 4.9 via a series of operations: deleting all-zero columns, summing similar columns together, and ordering the resulting columns lexicographically. Each of those can be effected via post-multiplication by a simple channel matrix; and so their overall effect is achieved via multiplication by the matrix product of all those simple matrices taken together, again a channel matrix. Hence $C \sqsubseteq_\circ C^r$.

For the reverse direction the operations are adding an all-zero column, splitting a column into several similar columns, and reordering columns. Again all of those can be achieved by post-multiplication. Hence $C^r \sqsubseteq_\circ C$. \square

Example 9.8 Let channel matrix C be

C	y_1	y_2	y_3	y_4
x_1	1	0	0	0
x_2	$1/4$	$1/2$	0	$1/4$
x_3	$1/2$	$1/3$	0	$1/6$

The reduced matrix C^r is then

C^r		
x_1	1	0
x_2	$1/4$	$3/4$
x_3	$1/2$	$1/2$

⁹ Here we are using that the product of two stochastic matrices is again stochastic.

¹⁰ The hyper-native definition is discussed briefly in the Chapter Notes.

and we have both

$$\begin{array}{|c|cc|} \hline C^r & & \\ \hline x_1 & 1 & 0 \\ x_2 & 1/4 & 3/4 \\ x_3 & 1/2 & 1/2 \\ \hline \end{array} = \begin{array}{|c|cccc|} \hline C & y_1 & y_2 & y_3 & y_4 \\ \hline x_1 & 1 & 0 & 0 & 0 \\ x_2 & 1/4 & 1/2 & 0 & 1/4 \\ x_3 & 1/2 & 1/3 & 0 & 1/6 \\ \hline \end{array} \begin{array}{|c|cc|} \hline R & & \\ \hline y_1 & 1 & 0 \\ y_2 & 0 & 1 \\ y_3 & 0 & 0 \\ y_4 & 0 & 1 \\ \hline \end{array}$$

and

$$\begin{array}{|c|cccc|} \hline C & y_1 & y_2 & y_3 & y_4 \\ \hline x_1 & 1 & 0 & 0 & 0 \\ x_2 & 1/4 & 1/2 & 0 & 1/4 \\ x_3 & 1/2 & 1/3 & 0 & 1/6 \\ \hline \end{array} = \begin{array}{|c|cc|} \hline C^r & & \\ \hline x_1 & 1 & 0 \\ x_2 & 1/4 & 3/4 \\ x_3 & 1/2 & 1/2 \\ \hline \end{array} \begin{array}{|c|cccc|} \hline R' & y_1 & y_2 & y_3 & y_4 \\ \hline & 1 & 0 & 0 & 0 \\ & 0 & 2/3 & 0 & 1/3 \\ \hline \end{array},$$

so that indeed $C \sqsubseteq_{\circ} C^r \sqsubseteq_{\circ} C$. \square

Corollary 9.9 Let A and B be abstract channels over the same input \mathcal{X} , and let \mathbf{A} and \mathbf{B} be arbitrary realizations of A and B as channel matrices. Then $A \sqsubseteq_{\circ} B$ iff $\mathbf{A} \sqsubseteq_{\circ} \mathbf{B}$.

Proof. The forward implication is immediate from Def. 9.6.

For the backward implication, let \mathbf{A}' and \mathbf{B}' be any (other) realizations of A and B , so that by Cor. 4.11 we have $(\mathbf{A}')^r = \mathbf{A}^r$ and $(\mathbf{B}')^r = \mathbf{B}^r$. Hence, using Thm. 9.7 we have

$$\mathbf{A}' \sqsubseteq_{\circ} (\mathbf{A}')^r = \mathbf{A}^r \sqsubseteq_{\circ} \mathbf{A} \sqsubseteq_{\circ} \mathbf{B} \sqsubseteq_{\circ} \mathbf{B}^r = (\mathbf{B}')^r \sqsubseteq_{\circ} \mathbf{B}'.$$

Hence, since (\sqsubseteq_{\circ}) is reflexive and transitive, we have $\mathbf{A}' \sqsubseteq_{\circ} \mathbf{B}'$. \square

We remark that we can test efficiently whether $\mathbf{A} \sqsubseteq_{\circ} \mathbf{B}$, by formulating it as a *linear-programming* problem.

9.3 Testing refinement: the customer's point of view

We will base our *tests* for refinement on the g -parameterized vulnerability that was the subject of earlier chapters. As we saw there, a gain function g is intended to capture and quantify the value $g(w, x)$ to the adversary of secrets x we are trying to protect from her, and to set out the actions w she has for trying to discover and exploit them. Thus we base testing *refinement* on what we call the “strong g -vulnerability order”.

Definition 9.10 (Testing refinement, probabilistic case)

Given abstract channels A and B , over the same input \mathcal{X} , we say that A is testing-refined by B , written $A \sqsubseteq_{\mathbb{G}} B$, if for any prior π and gain function $g: \mathbb{G} \times \mathcal{X}$ we have $V_g[\pi \triangleright A] \geq V_g[\pi \triangleright B]$.

Equivalently, we could use loss functions (Def. 3.4) for the same definition, i.e. that for any loss function ℓ we have $U_{\ell}[\pi \triangleright A] \leq U_{\ell}[\pi \triangleright B]$.

Note that in Def. 9.10 we could have compared leakages rather than vulnerabilities, suggesting the name “strong g -leakage order”, but the result would have been the same (regardless of whether we measure leakage multiplicatively or additively) because we have

$$\begin{aligned} \frac{V_g[\pi \triangleright A]}{V_g(\pi)} &\geq \frac{V_g[\pi \triangleright B]}{V_g(\pi)} \\ \text{iff} \quad V_g[\pi \triangleright A] &\geq V_g[\pi \triangleright B] \\ \text{iff} \quad V_g[\pi \triangleright A] - V_g(\pi) &\geq V_g[\pi \triangleright B] - V_g(\pi) \end{aligned}$$

using (in the first step) the fact that g 's being in $\mathbb{G}\mathcal{X}$ ensures $V_g(\pi) \geq 0$. (In the multiplicative case we assume that $V_g(\pi) \neq 0$.)

Now that we have defined both structural and testing refinement, we can devote the rest of the chapter to studying how they relate to each other: how the developer and the customer interact.

9.4 Soundness of structural refinement

By “soundness” of structural refinement we mean that if structural refinement is used by a developer, then he will achieve testing refinement for the customer. Put less formally: if he follows sound engineering practices, then he won't get sued by a litigious customer.

The soundness of (\sqsubseteq_{\circ}) is established by the following theorem, which can be understood as a generalized *data-processing inequality*.¹¹

Theorem 9.11 (Soundness of structural refinement)

For abstract channels A, B over the same input space \mathcal{X} , we have that $A \sqsubseteq_{\circ} B$ implies $A \sqsubseteq_{\mathbb{G}} B$.

Proof. Let abstract channels A, B (both of type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$) be realized by matrices $\mathbf{A}: \mathcal{X} \rightarrow \mathcal{Y}$ and $\mathbf{B}: \mathcal{X} \rightarrow \mathcal{Z}$. Recall from Cor. 9.7 that it does not matter which realizations we choose.

Suppose therefore that $\mathbf{A} \sqsubseteq_{\circ} \mathbf{B}$, which means that there exists a stochastic matrix $\mathbf{R}: \mathcal{Y} \rightarrow \mathcal{Z}$ such that $\mathbf{B} = \mathbf{A}\mathbf{R}$. To show $\mathbf{A} \sqsubseteq_{\mathbb{G}} \mathbf{B}$ we must show that for any prior π and gain function g we have $V_g[\pi \triangleright \mathbf{A}] \geq V_g[\pi \triangleright \mathbf{B}]$.

Now recall the *trace-based* formulation¹² of posterior g -vulnerability that was established in Thm. 5.24: it is

$$\begin{aligned} V_g[\pi \triangleright \mathbf{A}] &= \max_{\mathbf{S}_A} \text{tr}(\mathbf{G}^\top \pi \llcorner \mathbf{A} \mathbf{S}_A) \quad \text{and} \\ V_g[\pi \triangleright \mathbf{B}] &= \max_{\mathbf{S}_B} \text{tr}(\mathbf{G}^\top \pi \llcorner \mathbf{B} \mathbf{S}_B) \end{aligned}$$

where \mathbf{S}_A and \mathbf{S}_B are *strategies*, meaning stochastic matrices that select (possibly probabilistically) the action w an (optimal) adversary could take on each channel output. (Hence \mathbf{S}_A is of type $\mathcal{Y} \rightarrow \mathcal{W}$ and \mathbf{S}_B of type $\mathcal{Z} \rightarrow \mathcal{W}$.) Now observe that if \mathbf{S}_B

¹¹ The information-theoretic data-processing inequality is often described with the slogan, “post-processing can only destroy information”. To see that this theorem is indeed the g -leakage version of the data-processing inequality, note that if $\mathbf{B} = \mathbf{A}\mathbf{R}$, where \mathbf{B} goes from \mathcal{X} to \mathcal{Z} , \mathbf{A} goes from \mathcal{X} to \mathcal{Y} , and \mathbf{R} goes from \mathcal{Y} to \mathcal{Z} , then for any prior π we have a Markov chain from X to Y to Z . The data-processing inequality says in this case that $I(X; Z) \leq I(X; Y)$.

¹² The use of the trace-based formulation assumes $g: \mathbb{G}^{\text{fin}}\mathcal{X}$, but the proof can be extended to $\mathbb{G}\mathcal{X}$.

is a strategy for B and R is a stochastic matrix from \mathcal{Y} to \mathcal{Z} , then RS_B is a strategy for A . Hence we can reason as follows:

$$\begin{aligned}
 & V_g[\pi \triangleright B] \\
 = & \max_{S_B} \text{tr}(G \lceil_{\pi} BS_B) && \text{“Thm. 5.24”} \\
 = & \max_{S_B} \text{tr}(G \lceil_{\pi} (AR)S_B) && \text{“$B = AR$”} \\
 = & \max_{S_B} \text{tr}(G \lceil_{\pi} A(RS_B)) && \text{“matrix multiplication is associative”} \\
 \leq & \max_{S_A} \text{tr}(G \lceil_{\pi} AS_A) && \text{“S_A can be RS_B”} \\
 = & V_g[\pi \triangleright A] , && \text{“Thm. 5.24”}
 \end{aligned}$$

which gives the inequality $V_g[\pi \triangleright A] \geq V_g[\pi \triangleright B]$, hence also the $V_g[\pi \triangleright A] \geq V_g[\pi \triangleright B]$ that we seek. \square

Recall that we saw a simpler version of soundness, for deterministic channels and Bayes-vulnerability testing, in Thm. 9.2 above. A concrete example of the utility of soundness is given in §15.8 of Chap. 15, referring to the “Lovers’ protocol” developed in §15.6 just before it.

9.5 Completeness of structural refinement: the Coriaceous theorem

By “completeness” of structural refinement (\sqsubseteq_{\circ}) we mean that if $A \not\sqsubseteq_{\circ} B$ then there is a prior π and a gain function g such that $V_g[\pi \triangleright A] \not\geq V_g[\pi \triangleright B]$.¹³ The idea is that structural refinement is not more stringent than necessary—that it is “justified” as a development method, however expensive—because a failure of structural refinement implies a failure of testing refinement as well.

There are two reasons why that is important.

The first reason is that it gives the customer redress against shoddy or unscrupulous “developers”. If the customer for whatever reason believes that the developer has not followed the “industry best practice” by using (\sqsubseteq_{\circ}) to get from specification S (what the customer asked for) to implementation I (what the developer delivered), then to win a court case he would have to prove that there is *no* R that the developer could have used to get from S to I . And he would not necessarily be able to force the developer to reveal what that R could have been, via legal means, because the R might be a proprietary technique that the developer wants to keep secret. Completeness however shows that the customer, by exhibiting a *single* prior π and gain function g pair, can show that there *cannot* be any R that the developer used to get from S to I —however much he might claim that there was, that he did follow “best practice”. And here we have the witness to non-refinement that we promised earlier, just after Def. 9.5: the combination (π, g) is a single artifact that establishes $S \not\sqsubseteq_{\mathbb{G}} I$ beyond any doubt—and Thm. 9.11 above means that in that case, also $S \not\sqsubseteq_{\circ} I$. The import of Thm. 9.12 just below is that if refinement fails, there is always such a witness (π, g) .

The second reason is that it gives the developer confidence that using structural refinement is really worth the effort (and the expense). For the π, g pair mentioned just above might come not from a litigious customer, but rather from an unexpected context in which his delivered I has been placed. The developer wants to be sure that his I will not fail—and the theorem below is how he knows that the only way he can be sure is to use (\sqsubseteq_{\circ}). So he sees that it’s worth it.

¹³ Of course that is equivalent to $V_g[\pi \triangleright A] < V_g[\pi \triangleright B]$. But we write it that way to emphasize its purpose.

We can now present completeness, the converse to Thm. 9.11; again we have generalized the earlier case Thm. 9.3 which concerned deterministic channels and Bayes-vulnerability testing.

Theorem 9.12 (Completeness of structural refinement: Coriaceous¹⁴)

For abstract channels A, B over the same input space \mathcal{X} , we have that $A \sqsubseteq_{\mathbb{G}} B$ implies $A \sqsubseteq_{\circ} B$.

Proof. We argue the contrapositive, showing that if $A \not\sqsubseteq_{\circ} B$, then we can construct a gain function g and a prior π such that $V_g[\pi \triangleright A] < V_g[\pi \triangleright B]$, implying that $A \not\sqsubseteq_{\mathbb{G}} B$.

Let channel matrices $\mathbf{A}: \mathcal{X} \rightarrow \mathcal{Y}$ and $\mathbf{B}: \mathcal{X} \rightarrow \mathcal{Z}$ realize abstract channels A and B , respectively. If $\mathbf{A} \not\sqsubseteq_{\circ} \mathbf{B}$, then by definition there exists no stochastic matrix $\mathbf{R}: \mathcal{Y} \rightarrow \mathcal{Z}$ such that $\mathbf{B} = \mathbf{A}\mathbf{R}$. If we therefore define

$$\mathbf{A}^\dagger = \{\mathbf{A}\mathbf{R} \mid \mathbf{R} \text{ is a stochastic matrix from } \mathcal{Y} \text{ to } \mathcal{Z}\} ,$$

then our assumption $A \not\sqsubseteq_{\circ} B$ becomes $\mathbf{B} \notin \mathbf{A}^\dagger$.

Because matrix \mathbf{B} and the matrices in \mathbf{A}^\dagger are of type $\mathcal{X} \rightarrow \mathcal{Z}$, they can be embedded into Euclidean space of dimension $N = |\mathcal{X}| \times |\mathcal{Z}|$ by gluing their columns together in order. Then \mathbf{A}^\dagger becomes a set of points in N -space — and by linearity and continuity of matrix multiplication, we see that it is both convex and closed. And \mathbf{B} becomes a point in N -space that does not belong to \mathbf{A}^\dagger .

By the *Separating-Hyperplane Lemma* there is thus a hyperplane in N -space with point \mathbf{B} strictly on one side, and all of the set \mathbf{A}^\dagger strictly on the other side. Figure 9.2 gives a three-dimensional illustration of our use of the Separating-Hyperplane Lemma: the green point \mathbf{B} is separated from the gray convex region \mathbf{A}^\dagger by the blue hyperplane shown; the red line \mathbf{G} (used just below) is the hyperplane's normal.

Because \mathbf{G} is the normal of the hyperplane, also an N -vector thus, then we have that $\mathbf{B} \cdot \mathbf{G} > \mathbf{A}' \cdot \mathbf{G}$ for all \mathbf{A}' in \mathbf{A}^\dagger .¹⁵ Note that we can assume a ($>$)-separation without loss of generality, because we can negate \mathbf{G} if necessary. Moreover we can assume again wlog. that the elements of \mathbf{G} are non-negative, since we can add a constant k to each entry: that has the effect of increasing both sides of the inequalities above by exactly $k|\mathcal{X}|$, because with \mathbf{B} and each \mathbf{A}' derived from “glued” channel matrices of the same input type \mathcal{X} , as vectors they all sum to the same value $|\mathcal{X}|$.

Now by “ungluing” we can view \mathbf{G} , a vector in N -space, as a non-negative matrix (though not necessarily a stochastic matrix) indexed by $\mathcal{X} \times \mathcal{Z}$. Thus we can view \mathbf{G} as a *gain function* g in $\mathbb{G}^+ \mathcal{X}$. And so we have $g: \mathcal{Z} \times \mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$ (noting the transpose) using \mathcal{Z} as the set of actions and defined by $g(z, x) = \mathbf{G}_{x,z}$.

The g just defined turns out to be precisely the gain function we need to show that \mathbf{B} can leak more than \mathbf{A} under the uniform prior ϑ . For by Thm. 5.24 we have

$$V_g[\vartheta \triangleright \mathbf{A}] = \max_{\mathbf{S}_A} \text{tr}(\mathbf{G}^T \vartheta \lrcorner \mathbf{A} \mathbf{S}_A)$$

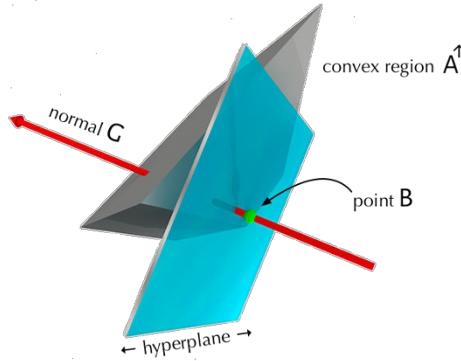
and

$$V_g[\vartheta \triangleright \mathbf{B}] = \max_{\mathbf{S}_B} \text{tr}(\mathbf{G}^T \vartheta \lrcorner \mathbf{B} \mathbf{S}_B) ,$$

where \mathbf{S}_A and \mathbf{S}_B respectively range over *strategies* for \mathbf{A} (i.e. channels from \mathcal{X} to \mathcal{Y}) and for \mathbf{B} (i.e. channels from \mathcal{X} to \mathcal{Z}). Note also that the *identity matrix* \mathbf{I} is a strategy for \mathbf{B} , and that each $\mathbf{A}\mathbf{S}_A \in \mathbf{A}^\dagger$. Hence we can reason as follows:

¹⁴ *Coriaceous* means “resembling or having the texture of leather”; the theorem is so called because of the difficulty of its proof.

¹⁵ We are using the vector forms here, and (\cdot) is used for their dot products.



In 3-space, “hyper-” planes are just ordinary planes.

Figure 9.2 A separating hyperplane

$$\begin{aligned}
 & V_g[\vartheta \triangleright A] \\
 = & \max_{S_A} \text{tr}(G^T \triangleright \vartheta \downarrow AS_A) && \text{“Thm. 5.24”} \\
 = & \text{tr}(G^T \triangleright \vartheta \downarrow AS_A^o) && \text{“choose } S_A^o \text{ to be optimal”} \\
 = & \frac{1}{|\mathcal{X}|} \text{tr}(G^T AS_A^o) && \text{“}\vartheta\text{ is uniform over }\mathcal{X}\text{”} \\
 = & \frac{1}{|\mathcal{X}|} \sum_z (G^T AS_A^o)_{z,z} && \text{“definition of tr”} \\
 = & \frac{1}{|\mathcal{X}|} \sum_z \sum_x G_{z,x}^T (AS_A^o)_{x,z} && \text{“definition of matrix multiplication”} \\
 = & \frac{1}{|\mathcal{X}|} \sum_{x,z} G_{x,z} (AS_A^o)_{x,z} && \text{“reorganize sum, remove transpose”} \\
 = & \frac{1}{|\mathcal{X}|} G \cdot (AS_A^o) && \text{“taking dot product in vector form”} \\
 < & \frac{1}{|\mathcal{X}|} G \cdot B && \text{“separation; } AS_A^o \in A^\uparrow\text{”} \\
 = & \text{tr}(G^T \triangleright \vartheta \downarrow B) && \text{“reverse of steps above; I is identity”} \\
 \leq & \max_{S_B} \text{tr}(G^T \triangleright \vartheta \downarrow BS_B) && \text{“}\vartheta \downarrow \text{ can be I”} \\
 = & V_g[\vartheta \triangleright B] && \text{“Thm. 5.24”}
 \end{aligned}$$

Finally, we remark that the gain function g constructed in this proof is of a restricted form: it is non-negative, with finitely many actions — that is, it is an element of $\mathbb{G}^+ \mathcal{X} \cap \mathbb{G}^{\text{fin}} \mathcal{X}$.¹⁶ □

With Thm. 9.11 and Thm. 9.12 together, we now have the equivalence of our two refinement orders.

Theorem 9.13 (Equivalence of structural and testing refinement orders)

The structural and the testing refinement orders on channels are the same: that is we have $(\sqsubseteq_\circ) = (\sqsubseteq_{\mathbb{G}})$.¹⁷

¹⁶ This fact will turn out to be significant in the proof of Thm. 10.4.

¹⁷ As a matter of notation, we write binary operators within parentheses when they are used *without arguments*, and without parentheses otherwise: thus we write (\sqsubseteq) in the former case and $A \sqsubseteq B$ in the latter.

And with that we can introduce an important convention:

With Thm. 9.13 we are justified in writing just “ \sqsubseteq ” from now on for the refinement order between channels, whether abstract or concrete, and whether structure or tests are used. (9.1)

We continue to use “ \sqsubseteq_{\circ} ” and “ $\sqsubseteq_{\mathbb{G}}$ ” however whenever we want to emphasize the way in which a refinement is established (or refuted).

One important implication of the equivalence Thm. 9.13 is that we can efficiently test whether $A \sqsubseteq B$ by testing whether $A \sqsubseteq_{\circ} B$, and (as already mentioned) that can be done using linear programming. In contrast, testing $A \sqsubseteq_{\mathbb{G}} B$ naïvely would require considering all priors and all gain functions.

9.6 The structure of abstract channels under refinement

On channel matrices, refinement (\sqsubseteq) is reflexive and transitive but not antisymmetric. On abstract channels, however, it *is* antisymmetric, making it a *partial order*.

Theorem 9.14

The refinement relation (\sqsubseteq) is antisymmetric on abstract channels.

Proof. In contrapositive form, antisymmetry says that if abstract channels A and B are distinct, then they cannot refine each other, and that is equivalent to saying that distinct *reduced matrices* A and B cannot refine each other.

We argue in two steps:

1. We first show that if $A \sqsubseteq B$, where A and B are distinct reduced matrices, then there exists a prior π such that the posterior Bayes vulnerability under A is *strictly greater* than that under B : that is $V_1[\pi \triangleright A] > V_1[\pi \triangleright B]$.
2. If also $B \sqsubseteq A$, then by Thm. 9.11 we must have $V_1[\pi \triangleright B] \geq V_1[\pi \triangleright A]$. But given the strict inequality in the first step, that is impossible.

For the first step, suppose that there exists R such that $B = AR$. A fundamental property of matrix multiplication is that each column of B is then a linear combination of the columns of A , with the corresponding column of R (which, being stochastic, has entries only between 0 and 1) as the coefficients. Now observe that there must be a column of R with two or more nonzero entries, since otherwise the columns of B would all be similar to the columns of A , which is impossible if A and B are distinct reduced matrices. That means that at least one column of B is formed by adding some fractions of two or more columns of A .

Now pick any two of the columns of A that are added. Because they are not similar, they must contain two positions where the ratio of the values to each other in the first column differs from the ratio of the values in the second column. That difference in ratios is important for posterior Bayes vulnerability, since (by Thm. 5.15) the posterior Bayes vulnerability of A under prior π is simply the sum of the column maximums of the joint matrix $J^A = \pi \triangleright A$. And, given the difference in ratios, we can choose a prior π whose support includes only those two positions such that the column maximums of the two corresponding columns of J^A are in *different* positions.

Finally we show that such a prior π causes $V_1[\pi \triangleright \mathbf{B}]$ to be strictly less than $V_1[\pi \triangleright \mathbf{A}]$. First note that the columns of \mathbf{J}^B are formed from the columns of \mathbf{J}^A in the same way that the columns of \mathbf{B} are formed from the columns of \mathbf{A} . (We have $\overline{\pi}_\square(\mathbf{A}\mathbf{R}) = (\overline{\pi}_\square \mathbf{A})\mathbf{R}$ by associativity.) Now note that the chosen column of \mathbf{B} gives rise to a column of \mathbf{J}^B formed by adding fractions of columns of \mathbf{J}^A whose maximums are *not all in the same position*. As a result, the maximum of that column of \mathbf{J}^B will be strictly less than the scaled sum of the maximums of those columns of \mathbf{J}^A . Hence $V_1[\pi \triangleright \mathbf{A}] > V_1[\pi \triangleright \mathbf{B}]$, and the proof of Step 1 is complete.

With Step 2 (an appeal to already proved facts) the proof of antisymmetry is complete. \square

A geometric argument for antisymmetry is given §12.5. (See also the Chapter Notes for a discussion of an even more abstract –and shorter– argument for antisymmetry.)

Example 9.15 To see a concrete illustration of the proof of Thm. 9.14, suppose that \mathbf{A} and \mathbf{B} and \mathbf{R} are as follows, where we have labeled the columns for convenience:

\mathbf{B}	z_1	z_2	z_3	z_4	=	\mathbf{A}	y_1	y_2	y_3	\mathbf{R}	z_1	z_2	z_3	z_4
x_1	1	0	0	0		x_1	1	0	0	y_1	1	0	0	0
x_2	0	0.5	0.3	0.2		x_2	0	0.6	0.4	y_2	0	0.5	0.5	0
x_3	0.3	0.35	0.25	0.1		x_3	0.3	0.5	0.2	y_3	0	0.5	0	0.5

Here we observe from column z_2 of \mathbf{R} that column z_2 of \mathbf{B} is formed by adding half of column y_2 and half of column y_3 of \mathbf{A} .

Since columns y_2 and y_3 of \mathbf{A} are not similar, they have two positions where the ratio of the entries in column y_2 differs from the ratio of the corresponding entries in column y_3 . Indeed, positions x_2 and x_3 have that property, since

$$\frac{\mathbf{A}_{x_2,y_2}}{\mathbf{A}_{x_3,y_2}} = \frac{0.6}{0.5} = 1.2 ,$$

while

$$\frac{\mathbf{A}_{x_2,y_3}}{\mathbf{A}_{x_3,y_3}} = \frac{0.4}{0.2} = 2 .$$

As a result, we can choose a prior π such that in the joint matrix \mathbf{J}^A , the maximums of columns y_2 and y_3 do not occur in the same place. Indeed, if $\pi = (0, 0.4, 0.6)$, then the joint matrix that we get from \mathbf{A} is

\mathbf{J}^A	y_1	y_2	y_3	,
x_1	0	0	0	
x_2	0	0.24	0.16	
x_3	0.18	0.3	0.12	

and the maximum for column y_2 (0.3) occurs at position x_3 , while the maximum for column y_3 (0.16) occurs at position x_2 . To understand why we chose the π that we did, note that we just needed to arrange that

$$\frac{\mathbf{A}_{x_2,y_2}}{\mathbf{A}_{x_3,y_2}} = 1.2 < \frac{\pi_{x_3}}{\pi_{x_2}} < 2 = \frac{\mathbf{A}_{x_2,y_3}}{\mathbf{A}_{x_3,y_3}}$$

since then

$$\mathbf{J}_{x_2,y_2}^A = \pi_{x_2} \mathbf{A}_{x_2,y_2} < \pi_{x_3} \mathbf{A}_{x_3,y_2} = \mathbf{J}_{x_3,y_2}^A$$

and so

$$\mathbf{J}_{x_3,y_3}^A = \pi_{x_3} \mathbf{A}_{x_3,y_3} < \pi_{x_2} \mathbf{A}_{x_2,y_3} = \mathbf{J}_{x_2,y_3}^A .$$

Since we can make the ratio of the two nonzero entries in π whatever we like, it is clear that we can always do that.¹⁸

Now consider the joint matrix that we get on π from \mathbf{B} , that is

J^B	z_1	z_2	z_3	z_4
x_1	0	0	0	0
x_2	0	0.2	0.12	0.08
x_3	0.18	0.21	0.15	0.06

We see that column z_2 of J^B is formed by adding half of column y_2 and half of column y_3 of J^A , but those columns of J^A have their maximums in different places. Observe that the contribution that column z_2 makes to $V_1[\pi \triangleright \mathbf{B}]$ is 0.21, while the scaled contributions of the summed columns y_2 and y_3 is $0.5 \cdot 0.3 + 0.5 \cdot 0.16 = 0.23$, reflecting the fact that J^B is (in effect) forced to choose x_3 rather than x_2 for y_3 , resulting in a loss of $0.5 \cdot (0.16 - 0.12) = 0.02$. Note finally that this loss is the only one that occurs here, as

$$V_1[\pi \triangleright \mathbf{A}] = 0.18 + 0.3 + 0.16 = 0.64 ,$$

while

$$V_1[\pi \triangleright \mathbf{B}] = 0.18 + 0.21 + 0.15 + 0.08 = 0.62 .$$

□

Finally, we mention that even though abstract channels under refinement (\sqsubseteq) are a partial order (as we have just shown), they are *not* a lattice: a counterexample is given in §12.4.

9.7 Refinement and monotonicity

9.7.1 Compositionality for contexts

As we saw in Def. 8.6, compositionality is the principle that the meaning of a compound is determined by the meanings of its components. We continue to take the view that the meaning of a channel matrix C is the abstract channel $\llbracket C \rrbracket$ that it denotes, i.e. thinking of the matrices as syntax and the abstract channels as semantics.¹⁹ Then we recall that compositionality of our abstraction $\llbracket - \rrbracket$ wrt. the operation of parallel composition requires that $\llbracket C \parallel D \rrbracket = \llbracket C \rrbracket \parallel \llbracket D \rrbracket$, where (\parallel) is the concrete version of parallel composition and (\parallel) is the abstract, as explained in Exercise 8.2 (p. 142).

Generalizing the above slightly, we could regard “run in parallel with D or D' ” as a *context* into which some C or C' is deployed. We could write the two contexts –syntactic and semantic– as $(-\parallel D)$ and $(-\parallel D')$ respectively, and call them $\mathcal{P}_D(-)$ and $\mathcal{P}_{D'}(-)$ for “run in parallel with D/D' ”. Then compositionality wrt. *contexts* (dropping the subscripts and parentheses) would be in this case that $\llbracket \mathcal{P}C \rrbracket = \mathcal{P}\llbracket C \rrbracket$, from which we can see that if $\llbracket C \rrbracket = \llbracket C' \rrbracket$ then also $\llbracket \mathcal{P}C \rrbracket = \llbracket \mathcal{P}C' \rrbracket$. That is, if you swap one component C for another C' of the same meaning (even if differently written), the meaning of the compound $\mathcal{P}(-)$ in which the components occur is unaffected. Recall from Def. 8.7 that we write $C \equiv C'$ for $\llbracket C \rrbracket = \llbracket C' \rrbracket$, so that the above is equivalently that $C \equiv C'$ implies $\mathcal{P}C \equiv \mathcal{P}C'$.

¹⁸ Note in particular that it is not a problem if some of the ratios would involve division by zero.

¹⁹ That view is reinforced by our seeing reduced channel matrices as (syntactic) normal forms, so that two matrices have the same meaning, as channels, just when they have the same normal form.

Thus if $C \equiv C'$ implies $\mathcal{C}C \equiv \mathcal{C}C'$ more generally (i.e. not just for \mathcal{P}_D), then that's compositionality with respect to contexts, of which our earlier discussion of compositionality wrt. certain operators was a special case.²⁰

9.7.2 Monotonicity with respect to refinement

With the conceptual preparation of the previous section, monotonicity is easy to define, and it applies to the concrete- and the abstract levels separately. Here we will give it for the concrete level.

Definition 9.16 (Concrete monotonicity) A concrete context \mathcal{C} is monotonic for refinement just when for all concrete channels C, D we have

$$C \sqsubseteq D \text{ implies } \mathcal{C}(C) \sqsubseteq \mathcal{C}(D) .$$

□

Without a monotonicity property at the concrete level, rigorous software engineering is very difficult — perhaps impossible. Because it is so important, we restate the message that might have been obscured by the technical details above: why –in security analysis– compositionality really does matter. It matters because security protocols are not used in a vacuum — but often they are *analyzed* in a vacuum, and that might be necessary for practical reasons. The point is that even if we are forced to analyze in a vacuum, we'd better make sure we use an analysis method where the conclusions still hold when we deploy the mechanism back in “the atmosphere” — i.e. so that we know that when the protocol (or whatever) is no longer being used in the nice, safe clean-room context of the analysis, where everyone plays fair and we know the prior and we know the gain function, it will still function as intended.

9.8 Why does refinement (\sqsubseteq) have to be so complicated?

9.8.1 Who gets to define refinement, anyway?

To answer the question posed for this section, we should begin with the title of this subsection — to which the answer is “both the developer and the customer”.

The customer determines what he will accept — since he's paying, it's his choice. We will suggest below (although we are neither developers nor customers) that “no increase in Bayes vulnerability” is a reasonable guess as to what customers will accept. It is, after all, so intuitive: “The probability of an adversary's figuring out the value of the secret in my system must not be more than advertised.” Here “advertised” is the specification, and “my system” is the implementation.

So (further below) we will take that as our starting point for the customer, introducing a relation (\preccurlyeq) just for that purpose, i.e. so that $A \preccurlyeq B$ means that for all priors π the Bayes vulnerability of $[\pi \triangleright B]$ must not be more than the Bayes vulnerability of $[\pi \triangleright A]$. That is what we have been calling the *strong Bayes-vulnerability order*, with “strong” meaning “over all priors”.

²⁰ Looking at it from the other direction, one says equivalently that (\equiv) is a congruence for the algebra induced by contexts \mathcal{C} . It's also known as *Referential Transparency*: see the Chapter Notes.

The developer might therefore decide to use (\preccurlyeq) as well, to keep things simple, but—if he did—he would quickly be out of business. There are two reasons, but they are essentially the same: *context*. Although the developer might ensure that indeed $S \preccurlyeq I$, the customer might deploy I within a context \mathcal{C} (of the customer's choice), and he would be disappointed—and possibly litigious—if he found that $\mathcal{C}S \not\preccurlyeq \mathcal{C}I$. Thus it is in the developer's interest to use some refinement relation ($\sqsubseteq_?$), possibly stronger than (\preccurlyeq), such that

$$S \sqsubseteq_? I \quad \text{implies} \quad \mathcal{C}S \preccurlyeq \mathcal{C}I \quad (9.2)$$

for as many \mathcal{C} 's as possible.²¹ That is, the developer might have to use some strange, stronger ($\sqsubseteq_?$) in house in order to achieve (\preccurlyeq) in the field—actually, “in as many fields as possible”. Later, in Def. 9.19, we will define that as “safety” of ($\sqsubseteq_?$).

The second reason is that for products of any realistic size, the developer will split the construction among independent teams, whose developer-mandated ($\sqsubseteq_?$) that the teams use independently of each other must be such that when the teams' work is combined, still in house, he will have ($\sqsubseteq_?$) for the combination. Thus again this is a question of context, but not from the unpredictable customer: rather in this case it is from within the developer's own workforce. But the effect is the same.

The problem is that the variety of contexts \mathcal{C} for which (\preccurlyeq) is safe for refinement of channels is not very large. In particular, we note that although we have been talking about channels as matrices, developers do not deliver actual matrices to their customers. Rather they deliver systems whose behavior can be *described* by those matrices, or by the corresponding abstract channels.

In Fig. 9.3 we anticipate Fig. 20.1 from Chap. 20 to come, a program that performs integer exponentiation in logarithmic time: it's used in public-key encryption and decryption. Although it looks like a simple sequential program, we have for the example given it a (side) channel *inside* it—as indicated. The channel itself, in isolation, can easily be described by a matrix (its input is what remains to be used of the private key e , and its output is the least significant bit of that)—but its *context* is the program text surrounding it, which is not a matrix at all. And yet we must take such contexts into account, *for they are where our channels ultimately will be*. We see in Part IV how to deal with that: but for now we note that the contexts \mathcal{C} in which our channels sit can be far more complicated, much more diverse, than simply “other channels”.

A second example of the diversity of contexts is discussed in Chap. 10, where we consider the effect of known correlations between a channel's secret prior and other data that seem to have nothing to do with the channel at all—the *Dalenius effect*.

²¹ It's normal for developers to limit the contexts in which a product may be deployed: for example “underwater” might not be a permitted context for a barbecue set. But the more contexts he can permit, the more popular his product will be.

²² Because we are using information-theoretic security, we must assume here that the adversary cannot see p .

```

- Global variables.
VAR B,E,p           - The Base B, the Exponent E and the power p.

{ VAR b,e:= B,E
  p:= 1
  WHILE e≠0 {
    VAR r:= e MOD 2
    IF r≠0 THEN p:= p*b
    b,e:= b2,e÷2
  }
}

- Local variables.
- Side channel.
- At the end of the program we have p = BE.22

```

Here we are assuming that the “branch on high”, the IF $r \neq 0$ THEN, has introduced a timing side channel into this program. By detecting whether or not the branch is taken, that is by observing the succession of values assumed by r , the adversary can learn the bits of exponent E —which is the secret key— one by one. When the loop ends, she will have learned them all.

Figure 9.3 Insecure implementation of public/private key decryption

The solution for the developer is to discover a refinement relation $(\sqsubseteq?)$, stronger than the customer’s (\preccurlyeq) , that satisfies (9.2), i.e. is safe for as many contexts \mathcal{C} as possible. But it’s important that it is not *too* strong, since the stronger it is the more constrained he will be in his development techniques, and the more they will cost. Later, in Def. 9.19, we will define that as “necessity” of $(\sqsubseteq?)$, meaning that a potential refinement should not be disallowed unless necessary (i.e. is disallowed only because a context exists that would make it so).

We now go over the above arguments in more detail.

9.8.2 A subjective argument: keeping the customer satisfied

As noted above (Thm. 9.13), now that we recognize (\sqsubseteq_0) and $(\sqsubseteq_{\mathbb{G}})$ as essentially the same thing, we will write just (\sqsubseteq) unless we have a particular point to make about its two possible formulations. We begin by taking the customer’s point of view, considering a refinement order simpler than (\sqsubseteq) and wondering “Why isn’t that good enough?” Given the results above, it does not matter whether we wonder about abstract or concrete channels, and so we will use C (instead of \mathcal{C}) throughout unless we are referring to specific matrices — and generally we will refer simply to channels, not specifying whether they are abstract or concrete. Similarly we will use \mathcal{C} rather than \mathcal{C} for contexts. Here is our simpler order.

Definition 9.17 (Strong Bayes-vulnerability order) Channels A and B , both with input space \mathcal{X} , are in the *strong Bayes-vulnerability order*, written $A \preccurlyeq B$, just when for any prior π the Bayes leakage of B does not exceed that of A — that is $V_1[\pi \triangleright A] \geq V_1[\pi \triangleright B]$ for all π .²³ By analogy, for distributions π, π' we say that $\pi \preccurlyeq \pi'$ just when $V_1(\pi) \geq V_1(\pi')$. \square

²³ Note again that comparing leakages is the same as comparing vulnerabilities, once the prior is fixed.

So why isn't (\preccurlyeq) good enough as it is? That question was abundantly answered in §3.2, where we saw that “the probability of guessing the secret in one try” is too simple to characterize many everyday situations in which our secret might be attacked by an adversary. And so a more cautious customer might wonder “What if she has two tries rather than just one?” or “What if some secrets are more valuable to her than others?” For a moment (but only a brief one), the customer might hope that (\preccurlyeq) is enough to capture all those other scenarios as well — but consider that $V_1(0.6, 0.2, 0.2) > V_1(0.5, 0.5, 0)$,²⁴ so that $(0.6, 0.2, 0.2) \preccurlyeq (0.5, 0.5, 0)$, but — alas — $V_2(0.6, 0.2, 0.2) < V_2(0.5, 0.5, 0)$ where $V_2(-)$ is the probability of guessing the secret in *two* tries.

Still, it might be thought that for channels (rather than individual priors) we might have that (\preccurlyeq) implies (\sqsubseteq) , since now the $V_1(-)$ -inequality has to apply for all priors — and that makes it much stronger. But the following example shows that not to be true either.

Example 9.18 Consider the two channel matrices

\mathbf{A}	y_1	y_2	y_3	\mathbf{B}	z_1	z_2
x_1	$1/2$	$1/2$	0	x_1	$2/3$	$1/3$
x_2	$1/2$	0	$1/2$	x_2	$2/3$	$1/3$
x_3	0	$1/2$	$1/2$	x_3	$1/4$	$3/4$

and

Although $\mathbf{A} \preccurlyeq \mathbf{B}$, still $\mathbf{A} \not\sqsubseteq \mathbf{B}$. Here is why.

The Bayes leakage bound $\mathbf{A} \preccurlyeq \mathbf{B}$ can be verified using the linear-programming-based Algorithm 5.27.

Yet we can show that $\mathbf{A} \not\sqsubseteq \mathbf{B}$ by exhibiting a prior and a gain function that together cause \mathbf{B} to leak more than \mathbf{A} . Suppose that x_1 and x_2 are *male* and x_3 is *female*, and the adversary cares only about the *sex* of the secret, as specified by the gain function

g_{sex}	x_1	x_2	x_3
<i>male</i>	1	1	0
<i>female</i>	0	0	1

Under a uniform prior ϑ and gain function g_{sex} we find that \mathbf{B} leaks more than \mathbf{A} , because $V_{g_{\text{sex}}}[\vartheta \triangleright \mathbf{A}] = 2/3$ yet $V_{g_{\text{sex}}}[\vartheta \triangleright \mathbf{B}] = 25/36$. (Intuitively, channel \mathbf{B} is useless for distinguishing between x_1 and x_2 , but g_{sex} does not penalize it for that.) Hence by Thm. 9.11 we conclude that $\mathbf{A} \not\sqsubseteq \mathbf{B}$. \square

An important consequence of Example 9.18 is that it shows that structural refinement (\sqsubseteq_\circ) is *not complete* for the strong Bayes-vulnerability order (\preccurlyeq) in the sense of its completeness in Thm. 9.12 for $(\sqsubseteq_\mathbb{G})$ — that is, it is possible to have two channels \mathbf{A}, \mathbf{B} such that

$$\mathbf{A} \preccurlyeq \mathbf{B} \quad \text{but still} \quad \mathbf{A} \not\sqsubseteq_\circ \mathbf{B} . \quad (9.3)$$

(See also Exercise 9.3 and Exercise 9.4.)

At this point however we seem to be opening Pandora's Box — how could we *ever* be sure then that \mathbf{A} is refined by \mathbf{B} in *all* scenarios and for *all* priors? That hoped-for “robustness” of refinement, that it apply in all situations, is surely too demanding a criterion...

That it is *not* too demanding is the remarkable thing about the definition of g -vulnerability and the soundness of structural refinement (\sqsubseteq_\circ). The definition Def. 3.2 of g -functions, and the g -vulnerabilities they induce, are actually surprisingly simple

²⁴ Recall that increasing vulnerability suggests decreasing security.

for the generality achieved. The definition involves only maximum and dot product, and yet the vulnerabilities one can express with that are astonishingly wide-ranging: all functions on distributions that are convex (or concave, for loss functions) and continuous — a form of expressive completeness that we explain in §11.2.1 below. And that leads to the second remarkable fact: in spite of the rampant generality of possible V_g 's, our structural refinement —itself not much more than matrix multiplication— suffices to achieve no loss of security with respect to every single one: that was *soundness* (Thm. 9.11).

Possibly we could convince the developer therefore to use (\sqsubseteq) by those essentially subjective arguments; but, if they don't work, we could take a more technical approach — to which we now turn.

9.8.3 An objective argument: compositional closure

We now give another, more technical argument for the “inevitability” of our definition of (\sqsubseteq) . Imagine that a customer has asked for channel S (the specification) and the developer has delivered channel K (the putative implementation).²⁵ And suppose further that $V_g[\pi \triangleright S] \geq V_g[\pi \triangleright K]$ for all π 's and *almost* all g 's — certainly, the developer insists, “enough g 's to describe any adversary one might meet”. Maybe he considered all n -guess vulnerabilities $V_n(-)$ for any $n \geq 1$. (The developer's point of view is motivated perhaps by the fact that the fewer the g 's he has to worry about, the cheaper his development method can be.)

So —to continue the story— the customer says “That's all very well; but in fact for *this* prior $\hat{\pi}$ and *this* gain function \hat{g} , encountered where I deployed your K , it turns out that $V_{\hat{g}}[\hat{\pi} \triangleright S] < V_{\hat{g}}[\hat{\pi} \triangleright K]$, and so I am not going to pay you for K .?” The developer replies by asserting that such a \hat{g} is ridiculous, a “monster”, and that the customer is being unreasonable.²⁶

Who is right? The customer... always — no matter how monstrous his $\hat{\pi}$ and \hat{g} might be. To see that, we look at the above scenario slightly more abstractly.

Customers generally will *not* know the intricacies of cyber-security and *QIF* — but still they will in *some* cases, simple ones, know without any doubt that security really *has* been compromised.²⁷ A good example of that, we have argued, is when $(\not\leq)$ — for if $S \not\leq K$ then even the developer —and probably a judge, too— would agree that the developer has botched the job: that failed inequality means that, for some priors, the probability of guessing the secret exactly, with K , can be strictly more than with S . It's a blatant failure. The difficulty comes for the customer when $S \not\leq K$ *does* hold, and his problem is revealed only in a more exotic (but still legitimate) setting, described by a gain function \hat{g} that the developer claims “in practice can never happen”. The customer can no longer appeal to the “any fool can see” argument that a glaring $(\not\leq)$ would provide.

Here is how the customer proves that, in spite of that, the “rogue” \hat{g} is worth worrying about. He exhibits a context \mathcal{F} (for “fail”) such that $\mathcal{F}S \not\leq \mathcal{F}K$, and his argument becomes “If I run your K in context \mathcal{F} , then its failure is revealed in just one guess. And any fool can see *that*, at least.” That technique, which we call “compositional closure”, is how we move from the simple refinement definition (\leq) that everyone *accepts* to the (perhaps) more complicated one that everyone *needs*. We start with a definition of (\leq) that is generally agreed to be reasonable. (That of course

²⁵ The mnemonics used in examples like this are S for specification and I for implementation, but K (kludge) for something claimed to be an implementation but that in fact is not.

²⁶ Compare the polyhedral monsters of Lakatos.

²⁷ “I don't know much about security, but I know what I don't like.”
with apologies to Gelett Burgess

remains a subjective issue.) Then the developer consults *QIF* experts (for which he need not involve the customer) and asks “What refinement relation ($\sqsubseteq_?$) do I need to use in house so that if I establish $S \sqsubseteq_? I$ then I can be sure that in the field, i.e. for any context C , the customer will always find that $CS \preccurlyeq CI$?“

The developer is in fact asking the *QIF* expert for the compositional closure of (\preccurlyeq) with respect to allowed contexts C from some agreed-upon set \mathbb{C} of contexts.

Suppose then that we are considering some customer-plausible relation ($\preccurlyeq_?$) of refinement. The *QIF* expert will use the following procedure to recommend a suitable ($\sqsubseteq_?$) for the developer.

Definition 9.19 (Compositional closure) The *compositional closure* of a primitive refinement relation ($\preccurlyeq_?$) with respect to set of contexts \mathbb{C} is the unique relation ($\sqsubseteq_?$) such that

($\sqsubseteq_?$) is **safe** for ($\preccurlyeq_?$) — for all channels A, B and contexts C in \mathbb{C} we have that $A \sqsubseteq_? B$ implies $CA \preccurlyeq_? CB$. (The customer will never be disappointed, no matter how B is deployed.)

($\sqsubseteq_?$) is **necessary** for ($\preccurlyeq_?$) — for all channels A, B we have that $A \not\sqsubseteq_? B$ implies there is some context C in \mathbb{C} such that $CA \not\preccurlyeq_? CB$. (The developer is not restricting his refinement techniques unnecessarily.)²⁸

□

And now we can give the *technical* justification for the refinement order (\sqsubseteq) that we use — it is simply that for contexts C like the program of Fig. 9.3 (which programs are the topic of Part IV) *and also* for the “Dalenius contexts” that we will encounter in Chap. 10, the appropriate approach is compositional closure. We summarize that as follows:

Our refinement order is a compositional closure of Bayes vulnerability

Our refinement order (\sqsubseteq) is the *compositional closure* under a set of contexts \mathbb{C} of the strong Bayes-vulnerability order (\preccurlyeq).

We assume always that \mathbb{C} includes the identity context and that it is closed under composition, i.e. that if C, C' are in \mathbb{C} then so is $C \circ C'$.

Since the failure of (\preccurlyeq) is not tolerated by anyone –by customer or developer, by judge or jury– we could argue that our (\sqsubseteq) is the inevitable –and unique– choice for refinement of channels, accounting for both the programs into which the customer might put them and the Dalenius contexts they might encounter.

We finish with some interesting facts about compositional closure (although we do not need them).²⁹ The compositional closure ($\sqsubseteq_?$) of a refinement-like relation ($\preccurlyeq_?$) for a given set \mathbb{C} of contexts has these properties:

- it is unique (as mentioned above);
- it acts monotonically for all contexts in \mathbb{C} ;
- it is the smallest relation that is necessary (Def. 9.19) for ($\preccurlyeq_?$);

²⁸ Note that our closure here –unusually– seems to be making the relation ($\sqsubseteq_?$) smaller, while the word “closure” might suggest that it’s making something bigger. What is being made bigger is in fact the *rejection* ($\sqsubseteq_?$) relation, i.e. the complement: a customer adds to the “not refined” relation a pair S, K because he knows a context C such that $CS \not\sqsubseteq_? CK$.

²⁹ Recall that when we say “(\sqsubseteq) is monotonic” with respect to a set of contexts \mathbb{C} , we mean more precisely that every context in \mathbb{C} is monotonic, as a function, with respect to (\sqsubseteq).

- it is the largest relation that is safe (Def. 9.19) for $(\preceq?)$; and
- it is the weakest strengthening of $(\preceq?)$ that is monotonic (and so is $(\preceq?)$ itself if $(\preceq?)$ is itself monotonic).

A similar argument to the above is given in §17.8 for inevitability of the definition of refinement for demonic channels.

9.9 Capacity is unsuitable as a criterion for refinement

An alternative to Def. 9.10 for refinement might be to use capacities, as developed in Chap. 7, since they can be made independent of the prior and the gain function, depending on which kind of capacity is used. For instance, we might require for refinement that the multiplicative Bayes capacity of B be no more than that of A . Since the Miracle theorem 7.5 says that the multiplicative Bayes capacity is an upper bound on multiplicative g -leakage, regardless of the prior or the gain function g in $\mathbb{G}^+\mathcal{X}$, we might then conclude that B should be regarded as “more secure” than A if its worst-case multiplicative leakage is smaller. But an ordering on maximums does not imply an ordering on individuals: there could be particular π ’s and g ’s where the leakage of that B is, in fact, greater than that of the A . If such a pair characterizes the context in which our system is deployed, then we might well regret replacing A with B — as the following example shows.

Example 9.20 Consider the two channel matrices

\mathbf{A}	z_1	z_2	z_3		\mathbf{B}	y_1	y_2	y_3	
x_1	$3/4$	$1/8$	$1/8$		x_1	1	0	0	
x_2	$1/8$	$3/4$	$1/8$	and	x_2	0	$1/2$	$1/2$	
x_3	$1/8$	$1/8$	$3/4$		x_3	0	$1/2$	$1/2$	

Suppose we compare \mathbf{A} and \mathbf{B} with respect to Bayes capacity. Because for any π

$$\mathcal{L}_1^\times(\pi, \mathbf{A}) \geq \mathcal{L}_1^\times(\pi, \mathbf{B}) \quad \text{iff} \quad \mathcal{L}_1^+(\pi, \mathbf{A}) \geq \mathcal{L}_1^+(\pi, \mathbf{B}) ,$$

we might expect that it makes no difference whether we compare multiplicative or additive Bayes capacity. But this is not so. For, using Thm. 7.2, we have

$$\mathcal{ML}_1^\times(\mathbb{D}, \mathbf{A}) = 3/4 + 3/4 + 3/4 = 9/4 > 2 = 1 + 1/2 + 1/2 = \mathcal{ML}_1^\times(\mathbb{D}, \mathbf{B}) .$$

On the other hand, by trying all sub-uniform distributions (see §7.2), we find that $\mathcal{ML}_1^+(\mathbb{D}, \mathbf{A})$ is realized on the uniform prior $\vartheta = (1/3, 1/3, 1/3)$ and

$$\mathcal{ML}_1^+(\mathbb{D}, \mathbf{A}) = V_1[\vartheta \triangleright \mathbf{A}] - V(\vartheta) = 3/4 - 1/3 = 5/12 ,$$

while $\mathcal{ML}_1^+(\mathbb{D}, \mathbf{B})$ is realized on a non-uniform prior $\pi = (1/2, 1/2, 0)$ and

$$\mathcal{ML}_1^+(\mathbb{D}, \mathbf{B}) = V_1[\pi \triangleright \mathbf{B}] - V(\pi) = 1 - 1/2 = 1/2 .$$

Hence we have $\mathcal{ML}_1^\times(\mathbb{D}, \mathbf{A}) > \mathcal{ML}_1^\times(\mathbb{D}, \mathbf{B})$ and $\mathcal{ML}_1^+(\mathbb{D}, \mathbf{A}) < \mathcal{ML}_1^+(\mathbb{D}, \mathbf{B})$. Notice that these two inequalities reflect the behavior of \mathbf{A} and \mathbf{B} on *different priors*: the first reflects that on prior $\vartheta = (1/3, 1/3, 1/3)$, channel \mathbf{A} ’s multiplicative and additive Bayes leakage exceeds \mathbf{B} ’s; the second reflects that on prior $\pi = (1/2, 1/2, 0)$, we see that \mathbf{B} ’s multiplicative and additive Bayes leakage exceeds \mathbf{A} ’s. \square

9.10 Exercises

Exercise 9.1 Explain intuitively the effect of allowing only *deterministic* refinement-matrices R when refining *probabilistic* channels. Clearly there might be fewer refinements: but how might you describe the ones that could be missing? \square

Exercise 9.2 Give an example to show that cascading (of channel matrices) is not refinement-monotonic in its left-hand argument. Is it monotonic in its right-hand argument? \square

Exercise 9.3 Explain how (9.3) follows from Example 9.18. \square

Exercise 9.4 Recall (9.3) and Exercise 9.3. Use a matrix-based argument to show *directly* that $A \not\leq_{\circ} B$ — that is, show that there can be no matrix R such that $AR = B$. \square

Exercise 9.5 (Refer §9.8.3.) Show that the compositional closure of a primitive refinement relation ($\preceq_?$) for a set of contexts is unique. \square

Exercise 9.6 (Refer §9.8.3.) Show that the compositional closure ($\sqsubseteq_?$) of a primitive refinement relation ($\preceq_?$) for a set C of contexts is monotonic for C . \square

Exercise 9.7 (Refer §9.8.3.) Show that the compositional closure ($\sqsubseteq_?$) of a primitive refinement relation ($\preceq_?$) for a set C of contexts is ($\preceq_?$) itself if ($\preceq_?$) is (already) monotonic for C . \square

Exercise 9.8 (Refer §9.8.3.) Show that the compositional closure of a primitive refinement relation ($\preceq_?$) for a set C of contexts is the smallest relation necessary for ($\preceq_?$) wrt. C . \square

Exercise 9.9 (Refer §9.8.3.) Show that the compositional closure of a primitive refinement relation ($\preceq_?$) for a set C of contexts is the largest relation safe for ($\preceq_?$) wrt. C . \square

Exercise 9.10 (Refer §9.8.3.) Show that the compositional closure of a primitive refinement relation ($\preceq_?$) for a set C of contexts is the weakest strengthening of ($\preceq_?$) that is monotonic for C . \square

9.11 Chapter notes

Although refinement as presented here, with all its theoretical properties, might seem a rather esoteric approach, the Crowds protocol in Part V gives a very practical and direct example of how useful it can be. (See §18.6).

The theory of refinement for Information Flow began however with Landauer and Redmond [8], who formalized a lattice of information in order to understand how to compare qualitative information flows. The importance of that lattice, for computing quantitative differences between qualitative leaks as expressed e.g. by Shannon entropy, was explored by Clark et al. [4], and used by Malacaria to prove some results establishing order relation correspondences between different quantitative approaches (i.e. Shannon entropy, guessing entropy, probability of guessing) on deterministic channels [9]. Hardness properties for computing information-flow measurements were explored by Yasuoka and Terauchi [19].

The structural refinement relation $\mathsf{A} \sqsubseteq_{\circ} \mathsf{B}$ of Def. 9.5 can be generalized mathematically to the case where A and B are arbitrary real-valued matrices having the same number of rows; the more general relation is known as *majorization*, and its properties have been studied by Dahl [5].

The realization that a more general partial order was necessary for comparing partial information flows came about through the extension of functional refinement on programs to include quantitative security properties; the definition used here appeared originally in McIver, Meinicke and Morgan's work on program semantics [10] and (independently) for channels by Alvim et al. [1]. The Coriaceous conjecture was identified and articulated by the latter [1], but in fact had been proved already by the former [10] as a step on the way to determining compositional closure (§9.8.3). That proof, derived by McIver from a construction in the probabilistic/demonic semantics for *pGCL* [11], is the basis for the presentation in this chapter. Later it was pointed out to us that it is an instance of Blackwell's much earlier results on comparison of experiments [2].

The relation between compositional closure wrt. refinement and Bayes leakage wrt. refinement was described by McIver, Meinicke and Morgan [10], and compositionality properties are discussed more generally by Morgan [14].

A good explanation of referential transparency is given by Stoy [16].

The “hyper-native”, i.e. abstract definition of refinement was given by McIver et al. [12], using essentially the insight that the refinement matrix R that witnesses the structural refinement is implicitly of type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}\mathcal{X}$, because the “anonymous” column labels of a hyper can unambiguously be taken to be the inners themselves. The witness in the abstract setting is thus a distribution of hypers, equivalently a hyper $\underline{\Delta}$ on $\mathbb{D}\mathcal{X}$, and then hyper Δ_S is refined by Δ_I just when there is a $\underline{\Delta}$ in $\mathbb{D}^3\mathcal{X}$ with $\Delta_S = \mu\underline{\Delta}$ and $\Delta_I = (\mathbb{D}\mu)\underline{\Delta}$.

A more direct proof of refinement's antisymmetry in terms of abstract channels (i.e. in terms of hypers) can be based on entropy [13, Thm 6], or alternatively on a more general technique of “color mixing” [15].

The results from linear programming are all standard, and can be found e.g. in the work of Trustrum [17], and we note that refinement definitions for functional properties of programs have been used to formalize the Stepwise Refinement method of Wirth [18].

Finally, the quote from Rick Hehner is in his book *A Practical Theory of Programming* [6]. The phrase “I don't know much about art, but I know what I like.” comes from the book *Are you a Bromide?* by Gelett Burgess [3]. The “monsters” of Lakatos come from his book *Proofs and Refutations* [7], though we are using them in a different way. Rather than bar them, adjust them or treat them as exceptions, we accept them as unavoidable –weird adversarial contexts (our monsters) are inevitable in a hostile setting– and so we make sure, using compositional closure to adjust our *definition* of refinement, that they are properly treated in a completely general way.

Bibliography

- [1] Alvim, M.S., Chatzikokolakis, K., Palamidessi, C., Smith, G.: Measuring information leakage using generalized gain functions. In: Proceedings of the 2012 IEEE 25th Computer Security Foundations Symposium (CSF 2012), pp. 265–279. IEEE, Los Alamitos (2012)
- [2] Blackwell, D.: Comparison of experiments. In: Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability, pp. 93–102. University of California Press, Berkley (1951)
- [3] Burgess, G.: Are You a Bromide? Or, The Sulphitic Theory Expounded and Exemplified According to the Most Recent Researches Into the Psychology of Boredom, Including Many Well-known Bromidioms Now in Use. B. W. Huebsch, New York (1906)
- [4] Clark, D., Hunt, S., Malacaria, P.: Quantitative analysis of the leakage of confidential data. In: Proceedings of the 1st Workshop on Quantitative Aspects of Programming Languages (QAPL 2001), *Electronic Notes in Theoretical Computer Science*, vol. 59, pp. 238–251 (2002)
- [5] Dahl, G.: Matrix majorization. *Linear Algebra and its Applications* **288**, 53–73 (1999)
- [6] Hehner, E.C.R.: A Practical Theory of Programming. Texts and Monographs in Computer Science. Springer, Berlin (1993)
- [7] Lakatos, I.: Proofs and Refutations. Cambridge Philosophy Classics. Cambridge University Press, New York (1976)
- [8] Landauer, J., Redmond, T.: A lattice of information. In: Proceedings of the 1993 IEEE Computer Security Foundations Workshop (CSFW 1993), pp. 65–70. IEEE, Los Alamitos (1993)
- [9] Malacaria, P.: Algebraic foundations for quantitative information flow. *Mathematical Structures in Computer Science* **25**(2), 404–428 (2015)
- [10] McIver, A., Meinicke, L., Morgan, C.: Compositional closure for Bayes risk in probabilistic noninterference. In: S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide, P.G. Spirakis (eds.) Proceedings of the 37th International Colloquium on Automata, Languages, and Programming (ICALP 2010), *Lecture Notes in Computer Science*, vol. 6199, pp. 223–235. Springer, Berlin (2010)
- [11] McIver, A., Morgan, C.: Abstraction, Refinement and Proof for Probabilistic Systems. Monographs in Computer Science. Springer, Berlin (2005)

Bibliography

- [12] McIver, A., Morgan, C., Rabehaja, T.: Abstract hidden Markov models: a monadic account of quantitative information flow. In: Proceedings of the 2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2015), pp. 597–608 (2015)
- [13] McIver, A., Morgan, C., Smith, G., Espinoza, B., Meinicke, L.: Abstract channels and their robust information-leakage ordering. In: M. Abadi, S. Kremer (eds.) Proceedings of the 3rd Conference on Principles of Security and Trust (POST 2014), *Lecture Notes in Computer Science*, vol. 8414, pp. 83–102. Springer, Berlin (2014)
- [14] Morgan, C.: Compositional noninterference from first principles. *Formal Aspects of Computing* **24**(1), 3–26 (2012)
- [15] Sonin, I.M.: The decomposition-separation theorem for finite nonhomogeneous Markov chains and related problems. In: Markov Processes and Related Topics: A Festschrift for Thomas G. Kurtz, vol. 4, pp. 1–15. Institute of Mathematical Statistics, Beachwood (2008)
- [16] Stoy, J.E.: Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory. MIT Press, Cambridge (1977)
- [17] Trustrum, K.: Linear Programming. Library of Mathematics. Routledge and Kegan Paul, London (1971)
- [18] Wirth, N.: Program development by stepwise refinement. *Communications of the ACM* **14**(4), 221–227 (1971)
- [19] Yasuoka, H., Terauchi, T.: Quantitative information flow — verification hardness and possibilities. In: Proceedings of the 2010 IEEE 23rd Computer Security Foundations Symposium (CSF 2010), pp. 15–27. IEEE, Los Alamitos (2010)



Chapter 10

The Dalenius perspective

We now consider the possibility that the adversary knows interesting *correlations* among different secrets, and we study the implications of that for information leakage.

Given a secret X with prior distribution π , and a channel C , our concern so far has been for the leakage of X caused by C . But what if there is another secret Z , apparently having nothing to do with C , that is *correlated* with X via some joint distribution $J: \mathbb{D}(\mathcal{Z} \times \mathcal{X})$ that is known to the adversary? We will see, in that case, that channel C can also leak information about Z . And the particular danger there is that we might not even know of the existence of Z , let alone of its correlation via J with X . So how can we defend against Z 's being leaked in this “accidental” way?

Here is an example of such a leak. Suppose that a number of *beneficiaries* reside in a region comprising three *counties* A, B, and C, and that they have a variety of ages. If a beneficiary is selected at random, then we might regard the beneficiary's county and *age* as secrets Z and X respectively. Now suppose further that a census has been taken and the following table of “macrostatistics” (from Dalenius) has been published:

Number of beneficiaries by county and age

County	Age class			
	Under 65	65–69	70–74	75 & over
A	3	15	11	8
B	7	60	34	20
C	0	4	0	0

We can normalize the table by dividing each entry by 162 (since there are 162 beneficiaries in total), giving a *joint distribution* which we here express as a matrix J indexed by \mathcal{Z} (the county, as rows) and \mathcal{X} (the age, as columns):¹

J	Under 65	65–69	70–74	75 & over
A	3/162	15/162	11/162	8/162
B	7/162	60/162	34/162	20/162
C	0	4/162	0	0

(10.1)

Moreover, by summing the rows and the columns of J we can compute marginal

¹ As with channels, we write J for the correlation when using a matrix presentation. Recall also that we use X, Z for the *names* of the secrets, and \mathcal{X}, \mathcal{Z} for the sets from which their values are drawn.

distributions ρ and π for Z and X respectively: they are

$$\begin{aligned} \rho &= (37/162, 121/162, 4/162) \\ \text{and} \quad \pi &= (10/162, 79/162, 45/162, 28/162) \end{aligned} .$$

Those marginals themselves are of course already useful to an adversary interested in secrets Z or X separately. More worrying, however, is that the existence of the correlation means that a channel C leaking information about age will *also* leak information about county. For instance, if C reveals that the age is not in the range 65–69, then the adversary can deduce that the county is not C. Note especially that knowledge of the channel C is not necessarily related in any way to knowing there is a correlation J : it is the *adversary* who knows J , and the data collected and partly revealed by C could be via some other study entirely.

We call such leakage the *Dalenius leakage* of Z caused by channel C under correlation J , and our goal in this chapter is to develop its theory.²

For another real-world example of Dalenius leakage, we refer the reader forward to §22.2.2 on voting.

10.1 Dalenius scenarios

In the scenario of interest here, there is a channel C that (potentially) leaks information about a secret X , and an adversary who actually cares about a *different* secret Z , and who knows not only the prior π of X but also the *correlation* between Z and X . Given what she might find out about X , she could then go further: first, her observation of C acting on prior π induces a posterior distribution (an inner) $p_{X|y}$ on X , which is what she learns about X ; and then, with the correlation between Z and X , she can use that $p_{X|y}$ to induce a posterior distribution on Z . We describe the Z -to- X correlation symmetrically, as a joint distribution $J: \mathbb{D}(\mathcal{Z} \times \mathcal{X})$, and we note that its X -marginal is the prior π for C .

From J , we get marginal distribution $\pi: \mathbb{D}\mathcal{X}$ (as just mentioned) but also marginal $\rho: \mathbb{D}\mathcal{Z}$ for Z , which together represent the adversary's prior knowledge about X and about Z , respectively. Moreover, when we express J as a matrix J as at (10.1) above, we can *factor* it via $J = \lceil \rho \rceil \mathbf{B}$, for some (stochastic) matrix $\mathbf{B}: \mathcal{Z} \rightarrow \mathcal{X}$ — meaning that $\rho_z \mathbf{B}_{z,x} = J_{z,x}$ for all z and x or, more succinctly, that $J = \rho \triangleright \mathbf{B}$ if we treat \mathbf{B} as a (concrete) channel. Each row $\mathbf{B}_{z,-}$ of \mathbf{B} is found simply by normalizing row $J_{z,-}$.³

Now suppose we take a concrete realization C of our abstract channel C . By the theory of *cascading* developed in §4.6.2, we know that $p(y|z) = (\mathbf{BC})_{z,y}$.⁴ The remarkable conclusion is that the adversary's "Dalenius knowledge" about Z that results from running C and revealing Y is simply the hyper-distribution $[\rho \triangleright \mathbf{BC}]$.

That justifies the following definition.

² It could also be called "collateral" leakage, in the sense that the adversary's actual target (via C) is age, but (the secrecy) of county is damaged because it is "nearby" (via J).

³ ... unless of course $\rho_z = 0$, in which case $\mathbf{B}_{z,-}$ is arbitrary since in that case $J_{z,-}$ is itself all zeros no matter what is chosen.

⁴ We are using concrete channels here because, as we remarked in §8.2.6, cascade of channels cannot be expressed at the abstract level. Multiplication of matrices, however, can be — as we show in §14.4.8 to come. And so these results can be expressed entirely at the abstract level if we wish.

Definition 10.1 (Dalenius g -vulnerability)

Suppose we are given a channel C on X and a joint distribution $J: \mathbb{D}(\mathcal{Z} \times \mathcal{X})$ that represents an adversary's knowledge of a correlation between \mathcal{Z} and \mathcal{X} . Let \mathbf{J} be a matrix realization of J , and factor it into marginal distribution $\rho: \mathbb{D}\mathcal{Z}$ and stochastic matrix $\mathbf{B}: \mathcal{Z} \rightarrow \mathcal{X}$, i.e. so that $\mathbf{J} = \rho \triangleright \mathbf{B}$. Let \mathbf{C} be a concrete realization of C .

Then, for any gain function $g: \mathbb{G}\mathcal{Z}$, the *Dalenius g -vulnerability* of J and C is defined

$$V_g^D(J, C) := V_g[\rho \triangleright \mathbf{B}\mathbf{C}] .^5$$

The special case when g is g_{id} is called *Dalenius Bayes vulnerability*, written V_1^D .

An example of the above is given in §22.2.3, applied to privacy in elections.

The definition however goes much further — it also implies the following important result, which shows that refinement (\sqsubseteq) is also equivalent to the *strong Dalenius vulnerability ordering*.

Theorem 10.2

For any (abstract) channels C and D on X ,⁶ we have $C \sqsubseteq D$ iff the Dalenius g -vulnerability of D never exceeds that of C , for any correlation J between X and some Z and any gain function g .

Proof. Write (\sqsubseteq^D) for the strong Dalenius g -vulnerability order (i.e. that for all g and J we have $V_g^D(J, C) \geq V_g^D(J, D)$); and recall that $(\sqsubseteq_\circ) = (\sqsubseteq) = (\sqsubseteq_G)$.

Then (\sqsubseteq^D) implies (\sqsubseteq_G) because (\sqsubseteq_G) is the special case of (\sqsubseteq^D) where $Z=X$ and J as a matrix is $\pi_{\mathcal{Z}}$. (This is where the Coriaceous theorem 9.12 is used.)

For the opposite direction, that (\sqsubseteq_\circ) implies (\sqsubseteq^D) , let concrete channels \mathbf{C} and \mathbf{D} realize C and D , respectively. Note that although \mathbf{C} and \mathbf{D} must have the same input type \mathcal{X} , they can have different output types say \mathcal{Y}^C and \mathcal{Y}^D .

Now from Def. 9.5 we have that $\mathbf{C} \sqsubseteq_\circ \mathbf{D}$ gives $\mathbf{C}\mathbf{R}=\mathbf{D}$ for some \mathbf{R} of type $\mathcal{Y}^C \rightarrow \mathcal{Y}^D$, whence immediately $(\mathbf{B}\mathbf{C})\mathbf{R}=\mathbf{B}\mathbf{D}$ and thus $\mathbf{B}\mathbf{C} \sqsubseteq_\circ \mathbf{B}\mathbf{D}$, giving our desired inequality $V_g[\rho \triangleright \mathbf{B}\mathbf{C}] \geq V_g[\rho \triangleright \mathbf{B}\mathbf{D}]$ for all ρ, g — which is (\sqsubseteq^D) by Def. 10.1. \square

We conclude this section with an example calculation of Dalenius vulnerability.

Example 10.3 Suppose that we have a 2-bit secret X and a channel C , expressed concretely as \mathbf{C} of type $\mathcal{X} \rightarrow \{0, 1\}$, that leaks the least-significant bit of X :

\mathbf{C}	0	1
00	1	0
01	0	1
10	1	0
11	0	1

⁵ That the definition does not depend on the precise realizations of J and C is a consequence of Def. 9.6.

⁶ As usual, we state our results in terms of abstract channels if possible; the proof (as here) might however involve taking their concrete realizations.

10 The Dalenius perspective

And suppose further that there is a 1-bit secret Z that is correlated with X according to the joint-distribution matrix J^{ZX} , where the superscript ZX is included for clarity:

J^{ZX}	00	01	10	11
0	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{4}$	$\frac{1}{16}$
1	$\frac{1}{16}$	$\frac{1}{4}$	$\frac{1}{16}$	$\frac{1}{8}$

As defenders of X we might be completely unaware of that data Z and —as such— have no interest in its security, and so take no measures to ensure it. What leakage do J^{ZX} and C imply?

From J^{ZX} we obtain marginal distributions ρ, π on \mathcal{Z}, \mathcal{X} respectively: they are

	0	1		and		00	01	10	11		(10.2)
ρ	$\frac{1}{2}$	$\frac{1}{2}$	π	$\frac{3}{16}$	$\frac{5}{16}$	$\frac{5}{16}$	$\frac{3}{16}$				

And now to calculate the posterior Bayes vulnerability of X due to C 's acting on prior π , we must first compute the joint matrix J^{XY} from π and C , that is

J^{XY}	0	1	
00	$\frac{3}{16}$	0	
01	0	$\frac{5}{16}$	
10	$\frac{5}{16}$	0	
11	0	$\frac{3}{16}$	

whence by Thm. 5.15 we can compute the posterior Bayes vulnerability as the sum of the column maximums of J^{XY} . That gives

$$V_1[\pi \triangleright C] = 5/16 + 5/16 = 5/8 .$$

Since X 's prior Bayes vulnerability $V_1(\pi)$ is $5/16$, we observe either multiplicative or additive leakage as we please: that X is leaked by C is what we expect.

What about Z though, which indeed we might not even have heard of — how is it affected by C ? Following Def. 10.1, we first get B by normalizing the rows of J^{ZX} and then we compute the cascade BC : it is

B	00	01	10	11	C	0	1	BC	0	1	
0	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{2}$	$\frac{1}{8}$	00	1	0	0	$\frac{3}{4}$	$\frac{1}{4}$	
1	$\frac{1}{8}$	$\frac{1}{2}$	$\frac{1}{8}$	$\frac{1}{4}$	01	0	1	1	$\frac{1}{4}$	$\frac{3}{4}$	
					10	1	0				
					11	0	1				

from which the Dalenius Bayes vulnerability of Z is

$$V_1^D(J, C) = V_1[\rho \triangleright BC] = 3/4 ,$$

since ρ in (10.2) is $(1/2, 1/2)$. Since Z 's prior Bayes vulnerability is $1/2$, we see that C together with $J: \mathbb{D}(\mathcal{Z} \times \mathcal{X})$ causes leakage of Z — which is what one might *not* expect. But defenders of X don't know that (because they are ignorant of Z); and defenders of Z don't know that (because they are ignorant of C).

The whole Dalenius issue is this — how can we defend secrets like Z from attacks on X ? We will return to that question in §10.3 below. Our very next topic however is the fact that the existence of Dalenius threats can be used to justify our definition of refinement: for that we return to the idea of compositional closure, from §9.8.3. \square

10.2 Compositional closure for Dalenius contexts

10.2.1 Safety and necessity with respect to Dalenius contexts

In §9.8.3 we saw an argument that our definition of refinement (\sqsubseteq) could in a sense be made “inevitable” once the set of allowable contexts had been determined. One set of contexts, it was suggested there, was sequential programs in which (side-) channels might occur. Another set of contexts, it turns out, is provided by Dalenius leakage, for which –given the material above– we can now be precise.

In fact we have proved safety already, that if $C \sqsubseteq D$ then $V_1^D(J, C) \geq V_1^D(J, D)$ for any J , because Thm. 10.2 proved the stronger result (in its forward direction) for *any* vulnerability V_g , not just Bayes vulnerability. That leaves necessity, for which we have the following theorem.

Theorem 10.4 (Dalenius necessity)

Let C, D be channels such that $C \not\sqsubseteq D$. Then there is a Dalenius correlation J such that $V_1^D(J, C) \not\geq V_1^D(J, D)$.

That is, distinctions made with our refinement order (\sqsubseteq) can be reduced to distinctions made with the strong Bayes-vulnerability order (\preccurlyeq) via a suitable Dalenius context J .

Proof. Let π and g be the prior and gain function that establish $C \not\sqsubseteq D$, that is

$$V_g[\pi \triangleright C] \not\geq V_g[\pi \triangleright D] , \quad (10.3)$$

and let $C: \mathcal{X} \rightarrow \mathcal{Y}^C$ and $D: \mathcal{X} \rightarrow \mathcal{Y}^D$ be concrete channels corresponding to C, D .⁷

With the trace-based formulation of posterior g -vulnerability given in Thm. 5.24, from (10.3) we have

$$V_g[\pi \triangleright C] = \max_S \text{tr}(G \lceil \pi \rfloor CS) ,$$

where S ranges over strategies from \mathcal{Y}^C to \mathcal{W} , and G is the matrix representation of g , and as usual $\lceil \pi \rfloor$ denotes the $\mathcal{X} \times \mathcal{X}$ matrix with π on its diagonal and zeroes elsewhere. We choose 1-summing joint-distribution matrix J and positive scalar c so that $cJ = G \lceil \pi \rfloor$.⁸ Continuing, we reason

$$\begin{aligned} & V_g[\pi \triangleright C] \\ = & \max_S \text{tr}(G \lceil \pi \rfloor CS) && \text{“from above”} \\ = & \max_S \text{tr}(c \times JCS) && \text{“choice of } c \text{ and } J\text{”} \\ = & c \times \max_S \text{tr}(I \lceil \rho \rfloor BCS) && \text{“factorize } J = \rho \triangleright B; \\ & & & \text{take } c \text{ outside max; identity matrix } I\text{”} \\ = & c \times V_1[\rho \triangleright BC] && \text{“Thm. 5.24”} \\ = & c \times V_1^D(J, C) ; && \text{“Def. 10.1”} \end{aligned}$$

and similarly for D we can show $V_g[\pi \triangleright D] = c \times V_1^D(J, D)$.⁹

Our result now follows from (10.3) above. \square

⁷ Recall from Def. 9.6 that whether or not refinement holds is unaffected by which specific corresponding concrete realizations we take.

⁸ We are assured that we can do this because the proof of Thm. 9.12 always constructs a gain function g that is non-negative, with finitely many actions.

⁹ In that argument our strategies S would be from \mathcal{Y}^D to \mathcal{W} .

Together with safety, Thm. 10.4 gives us that our refinement (\sqsubseteq) is the (unique) compositional closure of (\preccurlyeq), that is the strong Bayes-vulnerability order with respect to Dalenius contexts.

10.2.2 Justifying refinement: an example

We saw earlier in §9.8 that our definition of refinement (\sqsubseteq) could be justified by showing, in a sense, that it could be reduced to the more intuitive strong Bayes-vulnerability order (\preccurlyeq). With Dalenius contexts, we have shown precisely how that happens; and we can now look at an example.

Recall the concrete channels considered in Example 9.18: they were

C	y_1	y_2	y_3	D	z_1	z_2
x_1	$1/2$	$1/2$	0	x_1	$2/3$	$1/3$
x_2	$1/2$	0	$1/2$	x_2	$2/3$	$1/3$
x_3	0	$1/2$	$1/2$	x_3	$1/4$	$3/4$

and

As explained there, in spite of the fact that $C \not\sqsubseteq D$, these two channels in particular satisfy $C \preccurlyeq D$, i.e. that under *Bayes* vulnerability D never introduces more vulnerability than C does, *no matter what the prior π might be*. It is a strong condition –for all priors– and is why we call (\preccurlyeq) the *strong* Bayes-vulnerability order. That being so, are we still justified in insisting that $C \not\sqsubseteq D$? Is it really true that D is sometimes less secure than C ?

Yes, it is: and in fact there is a convincing Dalenius scenario J that demonstrates that. Let the joint-distribution matrix J^{ZX} for J be

J^{ZX}	x_1	x_2	x_3	
z_1	$1/3$	$1/3$	0	,
z_2	0	0	$1/3$	

and note that its X -marginal is uniform, whereas its Z -marginal is $(2/3, 1/3)$. Given the Dalenius correlation J , what are the *Dalenius* Bayes vulnerabilities of C and D ?

To find them, we use C to carry out the matrix multiplication

J^{ZX}	x_1	x_2	x_3	C	y_1	y_2	y_3	J^{ZY}	y_1	y_2	y_3
z_1	$1/3$	$1/3$	0	x_1	$1/2$	$1/2$	0	z_1	$1/3$	$1/6$	$1/6$
z_2	0	0	$1/3$	x_2	$1/2$	0	$1/2$	z_2	0	$1/6$	$1/6$
				x_3	0	$1/2$	$1/2$				

and hence $V_1^D(J, C) = 1/3 + 1/6 + 1/6 = 2/3$. But for D we have

J^{ZX}	x_1	x_2	x_3	D	z_1	z_2	J^{ZY}	z_1	z_2	
z_1	$1/3$	$1/3$	0	x_1	$2/3$	$1/3$	z_1	$4/9$	$2/9$,
z_2	0	0	$1/3$	x_2	$2/3$	$1/3$	z_2	$1/12$	$1/4$	
				x_3	$1/4$	$3/4$				

and that gives $V_1^D(J, D) = 4/9 + 1/4 = 25/36 > 2/3$. Thus the Dalenius Bayes vulnerability with respect to the correlation J is *more* for D than for C in spite of the fact that $C \preccurlyeq D$, i.e. that $V_1[\pi \triangleright C] \geq V_1[\pi \triangleright D]$ for all priors π .

To make the example more concrete, let $x_{1,2,3}$ be Jack, John and Jill, and let $y_{1,2,3}$ be Yes, No and Maybe. Then C gives the probabilities that each of those people will give each of the possible answers: for example Jack says “Yes” and “No” with equal probability, but never says “Maybe”. And no matter what the prior distribution on people might be, the strong Bayes-vulnerability ordering $C \preceq D$ means that an adversary, having heard what is said, is no more likely with D than with C to guess in one try who it was that spoke.

But now let $z_{1,2}$ be genders, i.e. Male and Female, so that J expresses the correlation between name and gender as well as the two marginals: that the person is uniformly likely to be any of Jack, John or Jill, and that the gender Male is ¹⁰ twice as likely as Female. Our calculation shows that an adversary using D is *more* likely than with C to be able to guess the speaker’s gender in one try. And that is precisely what $C \not\leq D$ warned us might happen.

It is especially important to realize that the above construction is not limited to this particular example (nor is it dependent on the use of *concrete* channels). For Thm. 9.12 (Coriaceous) shows that if $C \not\leq D$ there is *guaranteed* to exist a Dalenius context J in which $V_1^D(J, C) \geq V_1^D(J, D)$, however “weird” that context might be.

And indeed it is the case that the Dalenius correlation Thm. 9.12 delivered might be weird in the sense that it is difficult to concoct a plausible story (e.g. involving genders) of how it might arise in reality. But still the conclusion is very striking; and we must remember that weird contexts are exactly what adversaries devote their time to finding. That is why it is important to develop concepts and methods that apply to all contexts — not merely to the ones we might think are “reasonable” or “likely”. ¹¹

10.3 Bounding Dalenius leakage

As we have just seen, a channel C acting on a secret input X might induce vulnerability of an apparently unrelated secret Z that happens to have an interesting correlation J with X . Because Dalenius effects are worrisome –it seems so difficult to foresee the correlations that might be discovered to hold between secrets– upper bounds on Dalenius insecurity would be very desirable. It turns out that Dalenius *leakage* is a good way to investigate that.

Thus we now turn from considering only vulnerability to considering leakage which, as we know, compares vulnerability before and after a channel’s output is observed.

Definition 10.5 (Dalenius g -leakage)

Suppose we are given a channel C with concrete realization $C: \mathcal{X} \rightarrow \mathcal{Y}$ and joint distribution $J: \mathbb{D}(\mathcal{Z} \times \mathcal{X})$ whose realization J factors into marginal distribution $\rho: \mathbb{D}\mathcal{Z}$ and stochastic matrix $B: \mathcal{Z} \rightarrow \mathcal{X}$, i.e. so that $J = \rho \triangleright B$. Then, for any gain function $g: \mathbb{G}\mathcal{Z}$, we define the *Dalenius g -leakage* of J and C by

$$\mathcal{DL}_g^X(J, C) := \frac{V_g[\rho \triangleright BC]}{V_g(\rho)} = \mathcal{L}_g^X(\rho, BC) \quad (\text{multiplicative})$$

and

$$\mathcal{DL}_g^+(J, C) := V_g[\rho \triangleright BC] - V_g(\rho) = \mathcal{L}_g^+(\rho, BC). \quad (\text{additive})$$

¹⁰ There is no “therefore” here: it is the correlation J that expresses the conventional correspondence of gender with name — and so it is necessary to make the argument.

¹¹ See also §17.8 for a similar perspective for non-quantitative, i.e. demonic channels.

10 The Dalenius perspective

Our first, remarkable result is that for any channel C and any π and g it is possible to find a correlation J that reduces the multiplicative g -leakage $\mathcal{L}_g^\times(\pi, C)$ to the multiplicative Dalenius Bayes leakage $\mathcal{DL}_1^\times(J, C)$ with respect to that J . To see that, we note first that for any ρ we can express $V_g(\rho)$ as $V_g[\rho \triangleright \mathbf{1}]$, where (recall) the channel $\mathbf{1}$ leaks nothing.¹²

Then we can use the approach in the proof of Thm. 10.4 to reason

$$\begin{aligned}
& \mathcal{L}_g^\times(\pi, C) \\
= & V_g[\pi \triangleright C] / V_g(\pi) && \text{"definition } \mathcal{L}_g^\times\text{"} \\
= & V_g[\pi \triangleright C] / V_g[\pi \triangleright \mathbf{1}] && \text{"explained above"} \\
= & c \times V_1^D(J, C) / c \times V_1^D(J, \mathbf{1}) && \text{"reasoning as in Thm. 10.4,"} \\
& & & \text{for } J \text{ depending on } \pi, g \\
= & V_1^D(J, C) / V_1^D(J, \mathbf{1}) && \text{"canceling } c\text{'s"} \\
= & V_1[\rho \triangleright BC] / V_1[\rho \triangleright B\mathbf{1}] && \text{"use concrete realizations; } J = \rho \triangleright B\text{"} \\
= & V_1[\rho \triangleright BC] / V_1(\rho) && \text{"B}\mathbf{1}=\mathbf{1}, \text{ then again as explained above"} \\
= & \mathcal{DL}_1^\times(J, C) , && \text{"Def. 10.5"}
\end{aligned}$$

thus establishing the fundamental connection mentioned above between *multiplicative Dalenius Bayes leakage under arbitrary correlations* and *multiplicative g -leakage under arbitrary non-negative gain functions*. We have proved the following.

Theorem 10.6

Let C be a channel on X . For any $\pi: \mathbb{D}\mathcal{X}$ and non-negative gain function $g: \mathbb{G}^+\mathcal{X}$, there exists a correlation J in $\mathbb{D}(\mathcal{Z} \times \mathcal{X})$ such that the multiplicative g -leakage of X is equal to the multiplicative Dalenius Bayes leakage of Z : that is

$$\mathcal{L}_g^\times(\pi, C) = \mathcal{DL}_1^\times(J, C) .$$

With Thm. 10.6 one very general bound can be proved from the fact that, under Def. 10.5, multiplicative Dalenius g -leakage of C (expressed concretely) is also (ordinary) multiplicative g -leakage of the *cascade* BC , where as we saw above B is derived from the joint distribution J .

Expressing multiplicative Dalenius g -leakage as (ordinary) multiplicative g -leakage of a cascade is useful because we can prove bounds on the multiplicative Bayes capacity of a cascade, and those bounds carry over (by the Miracle theorem 7.5) to multiplicative g -leakage for any non-negative gain function g .

Theorem 10.7 Because we are using cascade, we express this theorem at the concrete level. For any concrete channels $B: \mathcal{Z} \rightarrow \mathcal{X}$ and $C: \mathcal{X} \rightarrow \mathcal{Y}$, we have

$$\mathcal{ML}_1^\times(\mathbb{D}, BC) \leq \min\{\mathcal{ML}_1^\times(\mathbb{D}, B), \mathcal{ML}_1^\times(\mathbb{D}, C)\} .$$

Proof. By the data-processing inequality (Thm. 9.11) we know that for any ρ we have

$$\mathcal{L}_1^\times(\rho, BC) \leq \mathcal{L}_1^\times(\rho, B) \leq \mathcal{ML}_1^\times(\mathbb{D}, B) .$$

Hence

$$\mathcal{ML}_1^\times(\mathbb{D}, BC) = \sup_{\rho} \mathcal{L}_1^\times(\rho, BC) \leq \mathcal{ML}_1^\times(\mathbb{D}, B) .$$

¹² As a matrix, it is a single column of 1's.

To obtain the upper bound with respect to C , we continue by observing that

$$\begin{aligned}
 & \mathcal{ML}_1^{\times}(\mathbb{D}, BC) \\
 = & \sum_{y: \mathcal{Y}} \max_{z: \mathcal{Z}} (BC)_{z,y} && \text{"Thm. 7.2"} \\
 = & \sum_{y: \mathcal{Y}} \max_{z: \mathcal{Z}} \sum_{x: \mathcal{X}} B_{z,x} C_{x,y} && \text{"definition of matrix multiplication"} \\
 \leq & \sum_{y: \mathcal{Y}} \max_{z: \mathcal{Z}} \sum_{x: \mathcal{X}} B_{z,x} \max_{x': \mathcal{X}} C_{x',y} \\
 = & \sum_{y: \mathcal{Y}} \max_{z: \mathcal{Z}} \max_{x': \mathcal{X}} C_{x',y} && \text{"row } z \text{ of } B \text{ sums to 1"} \\
 = & \sum_{y: \mathcal{Y}} \max_{x': \mathcal{X}} C_{x',y} \\
 = & \mathcal{ML}_1^{\times}(\mathbb{D}, C) . && \text{"Thm. 7.2"}
 \end{aligned}$$

Hence we have $\mathcal{ML}_1^{\times}(\mathbb{D}, BC) \leq \min\{\mathcal{ML}_1^{\times}(\mathbb{D}, B), \mathcal{ML}_1^{\times}(\mathbb{D}, C)\}$. \square

Now we can apply Thm. 10.7 to prove a very general bound on multiplicative Dalenius g -leakage.

Theorem 10.8

For any channel C , non-negative gain function g , and correlation J , we have $\mathcal{DL}_g^{\times}(J, C) \leq \mathcal{ML}_1^{\times}(\mathbb{D}, C)$.

Proof. Using concrete realizations as usual, we have

$$\begin{aligned}
 & \mathcal{DL}_g^{\times}(J, C) \\
 = & \mathcal{L}_g^{\times}(\rho, BC) && \text{"Def. 10.5"} \\
 \leq & \mathcal{ML}_1^{\times}(\mathbb{D}, BC) && \text{"Miracle theorem 7.5"} \\
 \leq & \mathcal{ML}_1^{\times}(\mathbb{D}, C) . && \text{"Thm. 10.7"}
 \end{aligned}$$

\square

This is an important and robust result, since it does not require us to make *any* assumptions about the correlation or the gain function (except that the gain function has to be non-negative).

10.4 Chapter notes

This chapter is largely based on results of Alvim et al. [1] and of Bordenabe and Smith [2].

We have adopted the name “Dalenius leakage” in honor of Tore Dalenius, whose 1977 paper [3] anticipated many of the concerns of modern quantitative information flow. (The table of macrostatistics at the beginning of this chapter is taken from that paper.) Working in the context of census data, Dalenius considered the sorts of information leakage (which he called “disclosure”) that could result from releasing census statistics, and gave the following definition:

If the release of statistics S makes it possible to determine the value D_K more accurately than is possible without access to S , a disclosure has taken place. [3, p. 433]

We note that this definition appears to be the source of what Dwork calls *Dalenius’ Desideratum*:

In 1977 the statistician Tore Dalenius articulated an “*ad omnia*” (as opposed to *ad hoc*) privacy goal for statistical databases: Anything that can be learned about a respondent from the statistical database should be learnable without access to the database. [4, p. 90]

Dwork argues for the impossibility of that goal by imagining the auxiliary information “Turing is two inches taller than the average Lithuanian woman.” With such information, revealing statistical information about Lithuanian women reveals information about Alan Turing.

But in fact the name “Dalenius’ Desideratum” appears to us to be quite unfair to Dalenius, as he does *not* make elimination of disclosure a *desideratum*. Indeed, contrary to the impression given by Dwork’s account,¹³ Dalenius explicitly recognizes the need to accept some disclosure. He writes [3, pp. 439–440]

A reasonable starting point is to discard the notion of *elimination* of disclosure. Two arguments for doing so are:

- i. it would be unrealistic to aim at elimination: such a goal is not operationally feasible;
- ii. it would place unreasonable restrictions on the kind of statistics that can be released; it may be argued that elimination of disclosure is possible only by elimination of statistics.

What has just been said is in fact the reason for our use of the term “statistical disclosure *control*” rather than “prevention” or “avoidance”.

Moreover, Dalenius was well aware of the possibility of information leakage due to auxiliary information (as in Dwork’s parable about Lithuanian woman and Alan Turing). Denoting by S the statistics released from the survey and by E the “extra-objective data”, he writes [3, pp. 441–442]

$S \times E$ -based disclosure could easily prove to be a much more serious problem than S -based disclosure: the statistician might not know about E , or –if he does– he might not have the authority to control it.

Finally, Dalenius even suggests the need for two measures: an M “giving the amount of disclosure associated with the release of some statistics and the extra-objective data”, and a B giving “the benefit associated with the statistics”; and he suggests using a criterion for statistical disclosure control that maximizes B for some accepted level M_0 of disclosure [3, p. 440].

An important inspiration for our §10.2.2 was given by McIver, Meinicke, and Morgan [6]. That work identifies the crucial *refinement* relation (\sqsubseteq) which is defined there on sequential imperative *QIF*-programs, rather than on channels alone, and justifies its strength by arguing that if program P is not refined by program Q , that is if $P \not\sqsubseteq Q$, then there is guaranteed to be a context C (involving probabilistic updates to secret variables) such that the posterior Bayes vulnerability of $C(Q)$ is greater than that of $C(P)$. Analogously, our Thm. 10.6 allows us to conclude that if $C \not\sqsubseteq D$, then there is a “Dalenius context” in which the posterior Bayes vulnerability (and Bayes leakage) of D is greater than that of C . In both cases, we find that if we accept the importance of Bayes vulnerability over a sufficiently rich set of contexts, then we are compelled also to accept the importance of refinement.

Turning to §10.3, we note that Thm. 10.7 was formulated and proved by Espinoza and Smith [5]. Also a rich variety of bounds on Dalenius leakage were developed by Alvim et al. [1, Section IX], including quite intricate bounds on additive Dalenius leakage.

¹³ Consider, for example, Dwork’s sentence “The last hopes for Dalenius’s goal evaporate in light of the following parable, which again involves auxiliary information.” [4, p. 90]

Bibliography

- [1] Alvim, M.S., Chatzikokolakis, K., McIver, A., Morgan, C., Palamidessi, C., Smith, G.: Additive and multiplicative notions of leakage, and their capacities. In: Proceedings of the 2014 IEEE 27th Computer Security Foundations Symposium (CSF 2014), pp. 308–322. IEEE, Los Alamitos (2014)
- [2] Bordenabe, N.E., Smith, G.: Correlated secrets in quantitative information flow. In: Proceedings of the 2016 IEEE 29th Computer Security Foundations Symposium (CSF 2016), pp. 93–104. IEEE, Los Alamitos (2016)
- [3] Dalenius, T.: Towards a methodology for statistical disclosure control. *Statistik Tidskrift* **15**, 429–444 (1977)
- [4] Dwork, C.: A firm foundation for private data analysis. *Communications of the ACM* **54**(1), 86–95 (2011)
- [5] Espinoza, B., Smith, G.: Min-entropy as a resource. *Information and Computation* **226**, 57–75 (2013)
- [6] McIver, A., Meinicke, L., Morgan, C.: Compositional closure for Bayes risk in probabilistic noninterference. In: S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide, P.G. Spirakis (eds.) *Proceedings of the 37th International Colloquium on Automata, Languages, and Programming (ICALP 2010)*, *Lecture Notes in Computer Science*, vol. 6199, pp. 223–235. Springer, Berlin (2010)



Chapter 11

Axiomatics

In previous chapters we discussed how secrecy can be quantified: we described information measures that map a prior to a real number reflecting the amount of threat to which the secret is subjected; and we introduced g -vulnerabilities as a rich family of such information measures that can capture a variety of significant operational scenarios. In particular, we determined that g -vulnerabilities can express Bayes vulnerability as a special case, and furthermore that by using properly constructed loss functions (the dual of gain functions), we can express Shannon entropy and guessing entropy as well. All of that suggests that g -vulnerabilities constitute a quite general family of information measures — but we have not yet developed independently a clear sense of what a *general definition* of “vulnerability measure” ought to be.¹

In this chapter therefore we investigate two significant questions regarding how general g -vulnerabilities really are, as information measures. First, are all instances of g -vulnerabilities “reasonable”? And second, are there “reasonable” vulnerability measures that cannot be captured as g -vulnerabilities?

We address those questions by considering a set of axioms that, we hope, will characterize intuitively reasonable properties that vulnerability measures might satisfy. We separately consider axioms for prior vulnerability, axioms for posterior vulnerability, and axioms for the *relationship* between prior and posterior vulnerability. As a result, we are also able to derive properties of leakage, which is defined in terms of comparison between the posterior- and prior vulnerabilities. As a particularly interesting outcome of that approach, we uncover a series of *dependencies* among the various axioms, which helps characterize the significance of g -vulnerability as a general information measure.

11.1 An axiomatic view of vulnerability

Earlier, we derived vulnerability measures by quantifying the adversary’s success in specific operational scenarios; and we extended vulnerability measures on priors to vulnerability measures on posteriors, by averaging vulnerabilities over the hypers.

Now we will follow a different approach: we axiomatize the study of the vulnerabilities themselves. We begin by considering generic vulnerability functions of type

$$\text{prior vulnerability} = V: \mathbb{D}\mathcal{X} \rightarrow \mathbb{R}_{\geq 0} \quad \text{and} \quad (11.1)$$

$$\text{posterior vulnerability} = \hat{V}: \mathbb{D}^2\mathcal{X} \rightarrow \mathbb{R}_{\geq 0} , \quad (11.2)$$

¹ The presentation in this chapter will be mainly in terms of gain functions and vulnerability; but they have straightforward duals for loss functions and uncertainty.

Axioms for prior vulnerabilities \mathbb{V}	
CNTY	$\forall \pi: \mathbb{V} \text{ is a continuous function of } \pi$
CVX	$\forall \sum_i a_i \pi^i: \mathbb{V}(\sum_i a_i \pi^i) \leq \sum_i a_i \mathbb{V}(\pi^i)$
Q-CVX	$\forall \sum_i a_i \pi^i: \mathbb{V}(\sum_i a_i \pi^i) \leq \max_i \mathbb{V}(\pi^i)$

Axioms for posterior vulnerabilities $\hat{\mathbb{V}}$	
NI	$\forall \pi: \hat{\mathbb{V}}[\pi] = \mathbb{V}(\pi)$
DPI	$\forall \pi, C, R: \hat{\mathbb{V}}[\pi \triangleright C] \geq \hat{\mathbb{V}}[\pi \triangleright CR]$ ²
MONO	$\forall \pi, C: \hat{\mathbb{V}}[\pi \triangleright C] \geq \mathbb{V}(\pi)$

Possible relationships between \mathbb{V} and $\hat{\mathbb{V}}$	
AVG	$\forall \Delta: \hat{\mathbb{V}}\Delta = \mathcal{E}_\Delta \mathbb{V}$
MAX	$\forall \Delta: \hat{\mathbb{V}}\Delta = \max_{[\Delta]} \mathbb{V}$

The prior vulnerability is $\mathbb{V}: \mathbb{D}\mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$; the posterior is $\hat{\mathbb{V}}: \mathbb{D}^2\mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$.

Table 11.1 Summary of axioms and their mnemonics

and we examine a variety of properties that “reasonable” instantiations of those generic functions might be expected to have. We then formalize those properties as a set of *axioms* for vulnerability functions, and investigate their consequences.

We would like to be clear however that we are not treating axiomatics here as “self-evident truths”. In fact a variety of (sets of) axioms might appear intuitively reasonable, and while it is sensible to consider justifications for each of them, such justifications should not be considered absolute. Rather, the axiomatics help us better to understand the logical dependencies among different properties, so that we might identify a minimal set of axioms sufficient to imply all the properties we care about.

We begin in §11.2 by focusing on the prior case, that is by giving axioms for prior vulnerabilities \mathbb{V} alone. We then propose convexity and continuity (made precise below) as generic properties of vulnerability, and show that they lead to g -vulnerability exactly. After that, we turn our attention to axioms considering both \mathbb{V} and $\hat{\mathbb{V}}$, and finally to axioms for posterior vulnerabilities $\hat{\mathbb{V}}$ alone.

Moreover, we study two ways of constructing $\hat{\mathbb{V}}$ from \mathbb{V} : by taking the average over the hyper, as we have been doing so far; and by considering instead the maximum vulnerability over the hyper. It turns out, in each case, that several of the axioms become equivalent. An important observation is that the axioms affect only the *relationship* between prior and posterior vulnerabilities, and as such are orthogonal to the way \mathbb{V} and $\hat{\mathbb{V}}$ are compared when used to measure leakage (e.g. multiplicatively or additively). Hence the results we obtain about the relationship among axioms are valid under both definitions of leakage.

Table 11.1 summarizes the axioms we shall consider.

² As before, concrete channels are used when explicit matrix operations are needed.

11.2 Axiomatization of prior vulnerabilities

We begin by introducing axioms that deal solely with generic prior vulnerabilities \mathbb{V} .

The first property we consider is that “small” changes on the prior π have a “small” effect on \mathbb{V} applied to that prior. That intuition is formalized in the following axiom.

Definition 11.1 (Axiom of continuity (CNTY))

A vulnerability \mathbb{V} is a continuous function of π with respect to the standard topology on $\mathbb{D}\mathcal{X}$, where by *standard* topology on $\mathbb{D}\mathcal{X}$ we mean that it is generated by the *Manhattan metric*

$$d(\pi, \pi') := \sum_{x: \mathcal{X}} |\pi_x - \pi'_x| .^3$$

Intuitively, the CNTY axiom describes adversaries who are not infinitely risk-averse. For instance, the non-continuous vulnerability function

$$\mathbb{V}^\lambda(\pi) := \begin{cases} 1, & \text{if } \max_x \pi_x \geq \lambda \\ 0, & \text{otherwise} \end{cases}$$

would correspond to an adversary who requires the probability of guessing correctly to be above a certain threshold λ in order to consider an attack effective at all. But this is arguably an unnatural behavior if we assume that the risk to the adversary of changing the probability to $\lambda - \epsilon$, for negligible ϵ , should not be arbitrarily large.⁴

The second property we consider is that \mathbb{V} is a convex function of the prior.⁵ More precisely, recall from §4.6.1 that a *convex combination* of distributions π^1, \dots, π^N is a sum $\sum_n a_n \pi^n$ where the multipliers a_n are non-negative reals adding up to 1. Since $\mathbb{D}\mathcal{X}$ is a convex set, a convex combination of distributions is itself a distribution. The convexity property is then formalized as the following axiom.

Definition 11.2 (Axiom of convexity (CVX))

A vulnerability \mathbb{V} is a convex function of π — that is, for all convex combinations $\sum_i a_i \pi^i$ of distributions we have

$$\mathbb{V}\left(\sum_n a_n \pi^n\right) \leq \sum_n a_n \mathbb{V}(\pi^n) .$$

The CVX axiom can be motivated as follows. Consider a game in which a secret x in \mathcal{X} (say a password) is drawn from one of two possible prior distributions π^1 and π^2 in $\mathbb{D}\mathcal{X}$. More precisely, the choice of prior distribution is itself random: we first select n in $\{1, 2\}$ with $n=1$ having probability say a_1 and $n=2$ probability $a_2 = 1 - a_1$, and —after that— we use π^n to draw the actual secret x .

³ The same topology (and thus definition of continuity) is obtained by using the Euclidean distance between the distributions considered as points in Euclidean $|\mathcal{X}|$ -space; and that in turn is equivalent to the Kantorovich metric over the discrete metric on \mathcal{X} itself.

⁴ There are however situations in probabilistic systems where surprising discontinuities occur: for example `loop "exit with probability ε" end` terminates with probability one if $\varepsilon > 0$, no matter how small ε might be; but when $\varepsilon = 0$ the loop is guaranteed *never* to terminate.

⁵ Note that, given the duality between vulnerability and uncertainty measures, for uncertainty measures a reasonable property would be *concavity* rather than *convexity*.

Now consider the following two scenarios for this game: in the first scenario, the value of n (i.e. the selection of prior) is revealed to the adversary.⁶ Using information in that π^n (whichever one it was) the adversary performs an attack on X whose expected success is therefore quantified as $\mathbb{V}(\pi^n)$. The expected measure of success overall will therefore be $a_1\mathbb{V}(\pi^1) + a_2\mathbb{V}(\pi^2)$.

In the second scenario, the choice n is not disclosed to the adversary: instead she knows only that, on average, the secret is drawn from the “overall” prior $a_1\pi^1 + a_2\pi^2$. With only that knowledge, her expected success is now quantified as $\mathbb{V}(a_1\pi^1 + a_2\pi^2)$.

With respect to the above game, the **CVX** axiom corresponds to the intuition that, since in the first scenario the adversary has more information, the effectiveness of an attack can never be smaller. A second way of understanding this axiom is to realize that an adversary should get no less information from a_1 , π^1 and π^2 than from $a_1\pi^1 + a_2\pi^2$, since the last value can be calculated if the first three are known.

Note that, in the definition of **CVX**, it is sufficient to use convex combinations of just *two* priors, i.e. of the form $a\pi^1 + (1-a)\pi^2$ as in our example above (where a_2 was a_1-1); and indeed we often use such combinations in proofs. Note also that **CVX** actually implies continuity everywhere except on the *boundary* of the domain, i.e. except on priors having some element with probability exactly 0. The **CNTY** axiom, however, explicitly enforces continuity everywhere.

An alternative to **CVX** however is to note that since the vulnerabilities $\mathbb{V}(\pi^n)$ in its definition are weighted by the probabilities a_n , we could have cases when the expected vulnerability $\sum_n a_n\mathbb{V}(\pi^n)$ is small even though some individual $\mathbb{V}(\pi^n)$ is large. Then one might argue that the bound imposed by **CVX** is too strict and could be loosened by requiring that $\mathbb{V}(\sum_i a_i\pi^i)$ is bounded only by the maximum of the individual vulnerabilities. That weaker requirement is formalized as the following axiom.

Definition 11.3 (Axiom of quasiconvexity (Q-CVX))

A vulnerability \mathbb{V} is a quasiconvex function of π — i.e. for all convex combinations $\sum_n a_n\pi^n$ we have

$$\mathbb{V}(\sum_n a_n\pi^n) \leq \max_n \mathbb{V}(\pi^n).$$

(Analogously, a quasiconcave function reverses the inequality and uses min.)

11.2.1 Soundness and completeness of V_g with respect to continuous, convex functions

Before discussing further the justification of **CVX** and **Q-CVX** as two possible generic properties for studying information flow, we end this section with some results concerning the characterization of g -vulnerabilities in terms of convexity and continuity. In particular we show that any real-valued function over priors that is both convex and continuous can be expressed as a g -vulnerability for some g , provided we allow infinitely many possible choices \mathcal{W} — and that implies that g -vulnerabilities are fundamental to an understanding of information flow when **CVX** is taken as axiomatic.

The characterization of convex and continuous functions as g -vulnerabilities can already be seen from Figs. 3.1 and 3.2, which illustrates each action available to an adversary as a line, a piecewise linear part of V_g as a real-valued function of π . More generally, we observe that a vulnerability V_g over an arbitrary $\mathbb{D}\mathcal{X}$ satisfies **CVX**, but it is also continuous provided that it is finite over its whole domain.

⁶ She knows n by assumption, because we revealed it; she knows π^n because the association between n and π^n is public, another example of Kerckhoffs’ Principle.

As usual, a real-valued function f over distributions is continuous at π if $f(\pi')$ converges to $f(\pi)$ when π' converges to π , in this case wrt. Def. 11.1. And f is continuous everywhere if it is continuous at all distributions. Our first result is that any g -vulnerability V_g for g in $\mathbb{G}\mathcal{X}$ (§3.3.1) is convex and continuous.

Theorem 11.4

If g is a gain function in $\mathbb{G}\mathcal{X}$, then the vulnerability V_g is convex and continuous.

Proof. Convexity of V_g follows easily from its definition, since it is the supremum over actions w of linear functions f_w in $\mathbb{D}\mathcal{X} \rightarrow \mathbb{R}$ of the form

$$f_w(\pi) := \sum_x g(w, x) \pi_x .$$

Continuity is due to the Gale-Klee-Rockafellar theorem, which implies that any bounded convex function over $\mathbb{D}\mathcal{X}$ is necessarily continuous.⁷ \square

Our second result is the converse, that CVX and CNTY together characterize g -vulnerabilities. To see that we use a well-known representation theorem for convex and continuous real-valued functions, which states that any such function is the supremum of its “subgradients”. Here a subgradient of f is a linear function that bounds f from below. For example, in Fig. 3.2, the straight lines are all subgradients of the V_g graphed in Fig. 3.1.⁸

Now we show that any convex and continuous function over $\mathbb{D}\mathcal{X}$ is expressible as a V_g for some g .

Theorem 11.5

Let $f: \mathbb{D}\mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$ be convex and continuous. Then f is realized as a g -vulnerability V_g for some gain function g in $\mathbb{G}\mathcal{X}$.

Proof. Write N for the size $|\mathcal{X}|$ of \mathcal{X} , and let \mathcal{X} itself be $\{x_1, \dots, x_N\}$. Then each π in $\mathbb{D}\mathcal{X}$ is a point $(\pi_{x_1}, \dots, \pi_{x_N})$ in \mathbb{R}^N — and $\mathbb{D}\mathcal{X}$ as a whole is polyhedral, because it is the convex polyhedron defined by the intersection of the regions

$$\pi_{x_1} + \dots + \pi_{x_N} = 1 , \quad \text{and} \quad \pi_{x_n} \geq 0 , \quad \text{for } 1 \leq n \leq N .$$

With that observation, we can now express any convex and continuous f as the supremum of its subgradients, that is⁹

$$f(\pi) = \sup_{f^* \leq f} f^*(\pi) ,$$

where f^* ranges over bounded linear functions $\mathbb{R}^N \rightarrow \mathbb{R}$ that lie everywhere below f (on $\mathbb{D}\mathcal{X}$). But now we note that each f^* contributes an action (which we call f^* itself) to a gain function defined overall as

$$g(f^*, x_n) := f^*([x_n]) ,$$

⁷ See the Chapter Notes.

⁸ In Chap. 12 we discuss this type of geometrical representation more generally for arbitrary \mathcal{X} , and illustrate how to use geometrical insights to obtain some important results about channel refinement.

⁹ See Calculus of Variations II by Mariano Giaquinta, Stefan Hildebrandt, Theorem 5, page 88.

because f^* is linear and $\pi = \sum_n \pi_{x_n}[x_n]$. The result now follows, since V_g takes the supremum over all actions, i.e. over all f^* 's in this case.¹⁰ \square

The justifications we have so far provided for the axioms of CVX and Q-CVX might not strike us as very intuitive at first, but it turns out that they can in fact be justified as natural consequences of fundamental axioms relating prior and posterior vulnerabilities, and specific choices for constructing \widehat{V} . We now address those relationships in detail, in §11.3 just below.

11.3 Axiomatization of posterior vulnerabilities

We will now consider axioms for posterior vulnerabilities alone, and axioms that relate posterior and prior vulnerabilities to each other. We consider three of them, and investigate how different definitions of posterior vulnerabilities shape their interactions.

The first property that we consider states that a prior π and the point hyper $[\pi]$ are equally good for the adversary.

Definition 11.6 (Axiom of noninterference (NI)) —

The vulnerability of a point hyper equals the vulnerability of the unique inner of that hyper: for all distributions π in $\mathbb{D}\mathcal{X}$ we have

$$\widehat{V}[\pi] = V(\pi) .$$

We can see why the above is called the *axiom of noninterference* if we recall Def. 4.14 of noninterference for a channel: the axiom states that a noninterfering channel should leak nothing when its leakage is measured by comparing \widehat{V} and V .

The second axiom we consider is an analog of the famous *Data-Processing Inequality* for mutual information,¹¹ and is formalized as follows. (Since the DPI axiom uses cascading, it cannot be expressed in terms of abstract channels: they must be concrete.¹²)

Definition 11.7 (Axiom of data-processing inequality (DPI)) —

Post-processing does not increase vulnerability: for all distributions π in $\mathbb{D}\mathcal{X}$ and (conformal) channel matrices C, R we have

$$\widehat{V}[\pi \triangleright C] \geq \widehat{V}[\pi \triangleright CR] .$$

The DPI axiom can be interpreted as follows. Consider a secret that is fed into a (concrete) channel C , after which the produced output is post-processed by being fed into another (concrete) channel R (whose input set must therefore be the same as the output set of C). Now consider two adversaries A and A' such that A can observe only the output of channel C , and A' can observe only the output of the cascade $C' = CR$. For any given prior π on secret values, adversary A' 's posterior knowledge about the

¹⁰ Recall that $[x_n]$ is the point distribution on x_n . Observe also that this construction actually makes the index set \mathcal{W} of f^* 's uncountable, although we can reduce to countability as follows. Since \mathcal{X} is finite and the rationals are dense in \mathbb{R} , and any linear function can be expressed in the form $a_1x_1 + \dots + a_Nx_N + c$ for constants a_1, \dots, a_N, c we can approximate all linear functions with linear functions that use rational coefficients.

¹¹ The data-processing inequality was discussed in §4.6.2 and §9.4. Its version for mutual information states that if $X \rightarrow Y \rightarrow Z$ forms a Markov chain, then $I(X; Y) \geq I(X; Z)$.

¹² See §8.2.6 for an explanation of why cascading cannot be defined for abstract channels. But see also §14.4.8.

secret is given by the hyper $[\pi \triangleright C]$, whereas that of A' is given by $[\pi \triangleright C']$. Now from A 's knowledge it is always possible to reconstruct what A' knows, but the converse is not necessarily true. To see that, note that A can use π and C to compute $[\pi \triangleright CR']$ for any R' , including the particular R used by A' . On the other hand, A' knows only π and C' and, in general, the decomposition of C' into a cascade of two channels is not unique (i.e. there may be many pairs C'' and R'' of matrices satisfying $C' = C''R''$), and so it is not always possible for A' to uniquely recover C from C' and then compute $[\pi \triangleright C]$. Given that asymmetry, the DPI formalizes that a vulnerability \hat{V} should not evaluate A 's information as any less of a threat than A' 's.

The third property we consider is that by observing the output of a channel an adversary cannot lose information about the secret; in the worst case, the output can simply be ignored if it is not useful.¹³ We formalize that property as follows.

Definition 11.8 (Axiom of monotonicity (MONO))

Pushing a prior through a channel does not decrease vulnerability: for all distributions π in $\mathbb{D}\mathcal{X}$ and channels C we have

$$\hat{V}[\pi \triangleright C] \geq V(\pi) .$$

The MONO axiom has two direct consequences on the additive and multiplicative notions of leakage. Since posterior vulnerabilities are never smaller than the corresponding prior vulnerabilities, additive leakage is always non-negative, and multiplicative leakage is never smaller than 1.

11.3.1 Possible definitions of posterior vulnerabilities

Having introduced the axioms of NI, DPI and MONO, we now turn our attention to how posterior vulnerabilities can be defined so as to respect them. In contrast with the case of prior vulnerabilities, in which the axioms considered (CVX and CNTY) uniquely determined the set of prior g -vulnerabilities, in the case of posterior vulnerability the axioms considered so far do not uniquely determine the set of posterior g -vulnerabilities. For that reason, in the following we shall consider alternative definitions of posterior vulnerabilities, and discuss the interrelations of axioms each definition induces.

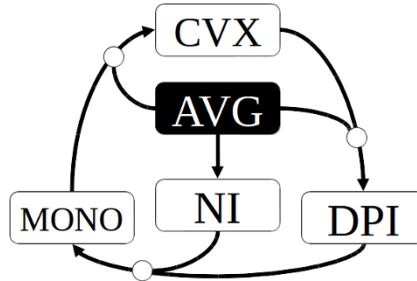
11.3.1.1 Posterior vulnerability as expectation

As we have seen, the posterior versions of Bayes vulnerability and g -vulnerability, as well as the uncertainty measures of Shannon entropy and guessing entropy, are all defined as the expectation of the corresponding prior measures over the (hyper-)distribution

¹³ In spite of MONO, it is still possible for an adversary to realize after running a channel that she is in a worse position than she *believed* she was before running it. This “dynamic” view was illustrated in Example 5.1, where before running the channel (knowing only the prior, and not the actual value of input x), she believed her chance of guessing the secret was $9/10 = 90\%$. But if she runs the channel and happens to observe y_2 , she then realizes that her luck was bad and that her chance right now –on this very run– is only 25%, much less than the 90% she’d hoped for at the beginning. The vulnerability seems to have decreased.

Next time she runs the channel, however, she could of course get lucky and observe y_1 , in which case her chance of guessing the secret is 100%, i.e. *more* than she formerly believed. Here the vulnerability appears to have increased.

Taking both of those possibilities into account (in general “all” possibilities) is called the “static” view, and what MONO tells us is that, if that is done, then *overall* a channel cannot decrease the vulnerability. That is, even if vulnerability decreases on some observations specifically (like y_2), that decrease will be compensated for by vulnerability’s increase on other potential outputs (like y_1) and –in the long run– both y_1 ’s and y_2 ’s will occur. (See also §14.1.1.)



The merging arrows indicate joint implication: for example, given **AVG** we have that **CVX** implies **DPI**.

Figure 11.1 Equivalence of axioms given **AVG**

of posterior distributions, i.e. weighted by the probability of each posterior's being realized. That definition of posterior vulnerability as expectation is formalized as the following axiom.

Definition 11.9 (Axiom of averaging (AVG))

The vulnerability of a hyper is the expected value wrt. the outer distribution of the vulnerabilities of its inner: for all hyper-distributions Δ in $\mathbb{D}^2\mathcal{X}$ we have

$$\hat{\mathbb{V}}\Delta = \mathcal{E}_\Delta \mathbb{V} \quad ,^{14}$$

where the hyper $\Delta : \mathbb{D}\mathcal{X}$ typically results from $\Delta = [\pi \triangleright C]$ for some π, C .

We will now consider the consequences of taking **AVG** as an axiom. As it turns out, by imposing **AVG** on a prior/posterior pair $(\mathbb{V}, \hat{\mathbb{V}})$ of vulnerabilities, we can uncover a series of interesting relations among other axioms: if the axiom of **AVG** holds, then so does the axiom of **NI**; and the axioms of **CVX**, **DPI** and **MONO** become equivalent to each other. Figure 11.1 summarizes those relations, which we shall now demonstrate.

We begin by showing that **AVG** implies **NI**.

Theorem 11.10 (AVG** \Rightarrow **NI**)** If a pair of prior/posterior vulnerabilities $(\mathbb{V}, \hat{\mathbb{V}})$ satisfies **AVG**, then it also satisfies **NI**.

Proof. If **AVG** is assumed then for any prior π we have $\hat{\mathbb{V}}[\pi] = \mathcal{E}_{[\pi]} \mathbb{V} = \mathbb{V}(\pi)$, since $[\pi]$ is a point hyper. \square

¹⁴ Recall that $\mathcal{E}_\alpha F$ is the expected value of real-valued function F over a distribution α . Here F is \mathbb{V} , a function from inner to real, and α is Δ , a distribution on inner.

Second, we show that the axioms of NI and DPI, taken together, imply MONO.

Theorem 11.11 (NI \wedge DPI \Rightarrow MONO) If a pair of prior/posterior vulnerabilities $(\mathbb{V}, \hat{\mathbb{V}})$ satisfies NI and DPI, then it also satisfies MONO.

Proof. Take any π, C , and recall that $\mathbf{1}$ denotes the noninterfering channel with only one column and as many rows as the columns of C . Then we reason

$$\begin{aligned} & \hat{\mathbb{V}}[\pi \triangleright C] \\ \geq & \hat{\mathbb{V}}[\pi \triangleright C\mathbf{1}] \quad ^{15} && \text{"by DPI"} \\ = & \hat{\mathbb{V}}[\pi \triangleright \mathbf{1}] && \text{"C}\mathbf{1} = \mathbf{1}" \\ = & \hat{\mathbb{V}}[\pi] && \text{"definition } [-\triangleright-]" \\ = & \mathbb{V}(\pi) . && \text{"by NI"} \end{aligned}$$

□

Third, we show that the axioms AVG and MONO together imply CVX.

Theorem 11.12 (AVG \wedge MONO \Rightarrow CVX) If a pair of prior/posterior vulnerabilities $(\mathbb{V}, \hat{\mathbb{V}})$ satisfies AVG and MONO, then it also satisfies CVX.

Proof. Let $\mathcal{X} = \{x_1, \dots, x_N\}$ be finite, and let π^1 and π^2 be distributions over \mathcal{X} . Let $0 < a < 1$ so that also $\pi^3 = a\pi^1 + (1-a)\pi^2$ is a distribution on \mathcal{X} . (In case $a=0$ or $a=1$ we would have $\pi^3 = \pi^1$ or $\pi^3 = \pi^2$, respectively, and convexity would follow trivially.) Define C^* to be the two-column channel matrix

$$C^* = \begin{bmatrix} a\pi_1^1/\pi_1^3 & (1-a)\pi_1^2/\pi_1^3 \\ \vdots & \vdots \\ a\pi_n^1/\pi_n^3 & (1-a)\pi_n^2/\pi_n^3 \\ \vdots & \vdots \\ a\pi_N^1/\pi_N^3 & (1-a)\pi_N^2/\pi_N^3 \end{bmatrix} \quad (11.3)$$

in which there is a row for each element x_n of \mathcal{X} with $\pi_n^3 \neq 0$. (Note that if $\pi_n^3 = 0$ for some n , then x_n is in the support of neither π^1 nor π^2 (since $0 < a < 1$), and wlog. we can remove element x_n from both priors and from the channel matrix C^* above.)

By pushing π^3 through C^* we obtain the hyper $[\pi^3 \triangleright C^*]$ with outer distribution $(a, 1-a)$, and associated inners π^1 and π^2 . Since AVG is assumed, we have

$$\hat{\mathbb{V}}[\pi^3 \triangleright C^*] = a\mathbb{V}(\pi^1) + (1-a)\mathbb{V}(\pi^2) . \quad (11.4)$$

But note that by MONO, we also have

$$\hat{\mathbb{V}}[\pi^3 \triangleright C^*] \geq \mathbb{V}(\pi^3) = \mathbb{V}(a\pi^1 + (1-a)\pi^2) . \quad (11.5)$$

Taking (11.4) and (11.5) together, we obtain CVX. □

(See §14.7 for a proof of Thm. 11.12 at a more abstract level.)

Finally, we show that the axioms AVG and CVX together imply DPI. For that, we will need the following lemma.

Lemma 11.13 Let $X \rightarrow Y \rightarrow Z$ form a Markov chain whose triply joint distribution is given by $p(x, y, z) = p(x)p(y|x)p(z|y)$ for all (x, y, z) in $\mathcal{X} \times \mathcal{Y} \times \mathcal{Z}$. Then we have that $\sum_y p(y|z)p(x|y) = p(x|z)$ for all x, y, z .

¹⁵ Again we use concrete channels here because of the cascade.

Proof. First we note that the probability of z depends only on the probability of y , and not x , and so $p(z|x,y) = p(z|y)$ for all x,y,z . Then we can use the fact that

$$p(y,z) p(x,y) = p(x,y,z) p(y) \quad (11.6)$$

to reason

$$\begin{aligned} & \sum_y p(y|z)p(x|y) \\ = & \sum_y p(y,z)/p(z)) \times p(x,y)/p(y) && \text{"by definition of conditional"} \\ = & \sum_y p(x,y,z)p(y) / p(z)p(y) && \text{"by (11.6)"} \\ = & \sum_y p(x,y | z) && \text{"by definition of conditional"} \\ = & p(x|z) . && \text{"by marginalization"} \end{aligned}$$

□

Theorem 11.14 (AVG \wedge CVX \Rightarrow DPI) If a pair of prior/posterior vulnerabilities $(\mathbb{V}, \widehat{\mathbb{V}})$ satisfies AVG and CVX, then it also satisfies DPI.

Proof. Let \mathcal{X} , \mathcal{Y} and \mathcal{Z} be sets of values. Let π be a prior on \mathcal{X} , and let C be a (concrete) channel from \mathcal{X} to \mathcal{Y} , and R be a (concrete) channel from \mathcal{Y} to \mathcal{Z} . Note that the cascade CR of channels C and R is a channel from \mathcal{X} to \mathcal{Z} .

Let $p(x,y,z)$ be the triply joint distribution defined $p(x,y,z) = \pi_x C_{x,y} R_{y,z}$ for all (x,y,z) in $\mathcal{X} \times \mathcal{Y} \times \mathcal{Z}$. By construction, this distribution has the property that the probability of z depends only on the probability of y , and not x , that is that $p(z|x,y) = p(z|y)$.

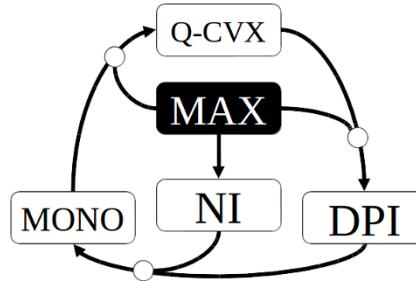
Note that, by pushing prior π through channel C , we obtain hyper $[\pi \triangleright C]$, in which the outer distribution on y is $p(y)$, and the inners are $p_{X|y}$. Thus we can reason

$$\begin{aligned} & \widehat{\mathbb{V}}[\pi \triangleright C] \\ = & \sum_y p(y) \mathbb{V}(p_{X|y}) && \text{"by AVG"} \\ = & \sum_y (\sum_z p(z)p(y|z)) \mathbb{V}(p_{X|y}) && \text{"by marginalization"} \\ = & \sum_z p(z) \sum_y p(y|z) \mathbb{V}(p_{X|y}) && \text{"moving constants wrt. the sum"} \\ \geq & \sum_z p(z) \mathbb{V}\left(\sum_y p(y|z)p_{X|y}\right) && \text{"by CVX"} \\ = & \sum_z p(z) \mathbb{V}(p_{X|z}) && \text{"by Lem. 11.13"} \\ = & \widehat{\mathbb{V}}[\pi \triangleright CR] . && \text{"by AVG"} \end{aligned}$$

□

11.3.1.2 Posterior vulnerability as maximum

An important consequence of AVG is that an observable's happening with very small probability will have a negligible effect on $\widehat{\mathbb{V}}$, even if that observable reveals the secret completely. If that is not acceptable, an alternative approach is to consider the *maximum* (rather than average) information that can be obtained from any single output of the channel –produced with nonzero probability– no matter how small that probability might be. This conservative approach represents a defender who worries about the worst possible amount of threat to the secret. The definition of posterior vulnerability in those terms is formalized as the following axiom.



The merging arrows indicate joint implication: for example, given **MAX** we have that **Q-CVX** implies **DPI**. (Compare Fig. 11.1.)

Figure 11.2 Equivalence of axioms under **MAX**

Definition 11.15 (Axiom of maximum (MAX))

The vulnerability of a hyper is the maximum value among the vulnerabilities of the inners in the support of its outer: thus for all hypers Δ we have

$$\hat{\mathbb{V}}\Delta = \max_{[\Delta]} \mathbb{V} \quad ,^{16}$$

where the hyper $\Delta: \mathbb{D}\mathcal{X}$ typically results from $\Delta = [\pi \triangleright C]$ for some π, C .

Note that the definition above takes the support of the outer distribution because we ignore the vulnerability of inners that cannot happen (i.e. that have probability zero).

We shall now consider the consequences of taking **MAX** as an axiom. As it turns out, by imposing **MAX** on a prior/posterior pair $(\mathbb{V}, \hat{\mathbb{V}})$ of vulnerabilities, we can derive relations among other axioms, just as we did for **AVG**. But they are different relations.

More precisely, if the axiom of **MAX** is satisfied, then again the axiom of **NI** is implied; but this time it's the axioms of **Q-CVX**, **DPI** and **MONO** that become equivalent to each other. Figure 11.2 summarizes those relations, which we shall now demonstrate.

We begin by showing that **MAX** implies **NI**.

Theorem 11.16 ($\text{MAX} \Rightarrow \text{NI}$) If a pair of prior/posterior vulnerabilities $(\mathbb{V}, \hat{\mathbb{V}})$ satisfies **MAX**, then it also satisfies **NI**.

Proof. If the **MAX** axiom is assumed, for any prior π we will have

$$\hat{\mathbb{V}}[\pi] = \max_{[\pi]} \mathbb{V} = \mathbb{V}(\pi) \quad ,$$

because $[[v]] = \{v\}$ for any value v , and thus for $v = \pi$ in particular. \square

However, in contrast to the case of **AVG**, the symmetry among **CVX**, **MONO** and **DPI** is broken under **MAX**: although the axioms of **MONO** and **DPI** are still equivalent (a result that we shall soon demonstrate), they are weaker than the axiom of **CVX**. Indeed, the

¹⁶ We are writing $\max_S f$ for the maximum value of function f over arguments taken from set S . Here the function is \mathbb{V} on distributions, and the arguments are the inners of Δ , equivalently the elements of the set $[\Delta]$.

following example shows a pair of prior/posterior vulnerabilities $(\mathbb{V}, \widehat{\mathbb{V}})$ satisfying the axioms of MAX, MONO and DPI, but not the axiom of CVX.

Example 11.17 (MAX \wedge MONO \wedge DPI $\not\Rightarrow$ CVX) Consider the pair $(\mathbb{V}_1, \widehat{\mathbb{V}}_1)$ such that for every prior π and channel C :

$$\begin{aligned}\mathbb{V}_1(\pi) &= 1 - \left(\min_x \pi_x \right)^2, \\ \widehat{\mathbb{V}}_1[\pi \triangleright C] &= \max_{\llbracket \pi \triangleright C \rrbracket} \mathbb{V}_1.\end{aligned}$$

We can see that \mathbb{V}_1 does not satisfy CVX by considering distributions $\pi^1 = (0, 1)$ and $\pi^2 = (1/2, 1/2)$, and their convex combination $\pi^3 = 1/2\pi^1 + 1/2\pi^2 = (1/4, 3/4)$. We then calculate $\mathbb{V}_1(\pi^1) = 1 - 0^2 = 1$ and $\mathbb{V}_1(\pi^2) = 1 - (1/2)^2 = 3/4$ and finally $\mathbb{V}_1(\pi^3) = 1 - (1/4)^2 = 15/16$, which gives $1/2\mathbb{V}_1(\pi^1) + 1/2\mathbb{V}_1(\pi^2) = 7/8$ — so that we can conclude that $\mathbb{V}_1(\pi^3) > 1/2\mathbb{V}_1(\pi^1) + 1/2\mathbb{V}_1(\pi^2)$, and indeed CVX is not satisfied.

The pair $(\mathbb{V}_1, \widehat{\mathbb{V}}_1)$ satisfies MAX by construction. To show that $(\mathbb{V}_1, \widehat{\mathbb{V}}_1)$ satisfies MONO and DPI, we first notice that \mathbb{V}_1 is quasiconvex. Using results from Figure 11.2 (more precisely, that $\text{MAX} + \text{Q-CVX} \Rightarrow \text{MONO} + \text{DPI}$, proved later in this section), we conclude that MONO and DPI are also satisfied. \square

The vulnerability function used in the counterexample above is quasiconvex. That turns out not to be a coincidence: by replacing CVX with Q-CVX (a weaker property), the symmetry between the axioms can be restored. The remainder of this section establishes the equivalence of Q-CVX, MONO and DPI under MAX.

We first show that the axioms MAX and MONO together imply Q-CVX.

Theorem 11.18 (MAX \wedge MONO \Rightarrow Q-CVX) If a pair of prior/posterior vulnerabilities $(\mathbb{V}, \widehat{\mathbb{V}})$ satisfies MAX and MONO, then it also satisfies Q-CVX.

Proof. Assume for a contradiction that $(\mathbb{V}, \widehat{\mathbb{V}})$ satisfy MAX and MONO, but do not satisfy Q-CVX.

Since Q-CVX is not satisfied, there must exist a value $0 \leq a \leq 1$ and three distributions π^1, π^2, π^3 such that $\pi^3 = a\pi^1 + (1-a)\pi^2$ and

$$\mathbb{V}(\pi^3) > \max(\mathbb{V}(\pi^1), \mathbb{V}(\pi^2)). \quad (11.7)$$

Now consider the (concrete) channel C^* defined as in (11.3). Then the hyper-distribution $[\pi^3 \triangleright C^*]$ has outer distribution $(a, 1-a)$, and corresponding inner distributions π^1 and π^2 . Since MAX is assumed, we have that

$$\widehat{\mathbb{V}}[\pi^3 \triangleright C^*] = \max(\mathbb{V}(\pi^1), \mathbb{V}(\pi^2)), \quad (11.8)$$

and because we assumed MONO, we also have that

$$\widehat{\mathbb{V}}[\pi^3 \triangleright C^*] \geq \mathbb{V}(\pi^3). \quad (11.9)$$

By replacing (11.8) in (11.9), we derive that $\mathbb{V}(\pi^3) \leq \max(\mathbb{V}(\pi^1), \mathbb{V}(\pi^2))$, which contradicts our assumption in (11.7). \square

We now show that the axioms MAX and Q-CVX together imply DPI.

Theorem 11.19 (MAX \wedge Q-CVX \Rightarrow DPI) If a pair of prior/posterior vulnerabilities $(\mathbb{V}, \widehat{\mathbb{V}})$ satisfies MAX and Q-CVX, then it also satisfies DPI.

Proof. Let π be a prior on \mathcal{X} , and C, R be channels from \mathcal{X} to \mathcal{Y} and from \mathcal{Y} to \mathcal{Z} , respectively, with joint distribution $p(x, y, z)$ defined in the same way as in the proof of Thm. 11.14.

Note that, by pushing prior π through channel CR, we obtain hyper $[\pi \triangleright CR]$ in which the outer distribution on z is $p(z)$, and the inners are $p_{X|z}$. Thus we can reason

$$\begin{aligned}
 & \hat{\mathbb{V}}[\pi \triangleright CR] \\
 = & \max_z \mathbb{V}(p_{X|z}) && \text{"by MAX"} \\
 = & \max_z \mathbb{V}\left(\sum_y p(y|z)p_{X|y}\right) && \text{"by Lem. 11.13"} \\
 \leq & \max_z \left(\max_y \mathbb{V}(p_{X|y})\right) && \text{"by Q-CVX"} \\
 = & \max_y \hat{\mathbb{V}}(p_{X|y}) && \text{"z not free"} \\
 = & \hat{\mathbb{V}}[\pi \triangleright C] && \text{"by MAX"}
 \end{aligned}$$

□

Finally, note that although Q-CVX is needed to recover the full equivalence of the axioms, CVX is strictly stronger than Q-CVX; hence, using a convex vulnerability measure (such as any V_g), MONO and DPI are still guaranteed under MAX.

Corollary 11.20 (MAX \wedge CVX \Rightarrow MONO \wedge DPI) If a pair $(\mathbb{V}, \hat{\mathbb{V}})$ satisfies MAX and CVX, then it also satisfies MONO and DPI.

Proof. Follows from the results of Figure 11.2 and the fact that CVX \Rightarrow Q-CVX. □

11.3.1.3 Other definitions of posterior vulnerabilities

Defining posterior vulnerabilities using the axioms of AVG or MAX is certainly reasonable in many scenarios, but we may wonder whether other definitions might also be meaningful in some context. In this section we discuss the consequences of defining posterior vulnerabilities with something more relaxed than AVG or MAX. We shall, however, require that $\hat{\mathbb{V}}$ should be related to \mathbb{V} by the following condition: the vulnerability of a hyper-distribution should be bounded by the vulnerabilities of the inner distributions in its support. The next axiom formalizes that restriction.

Definition 11.21 (Axiom of bounds (BNDS))

The vulnerability of a hyper lies non-strictly between the vulnerabilities of the inners in its support: for all hypers Δ we have

$$\min_{[\Delta]} \mathbb{V} \leq \hat{\mathbb{V}}[\Delta] \leq \max_{[\Delta]} \mathbb{V},$$

where the hyper Δ in $\mathbb{D}^2 \mathcal{X}$ typically results from $\Delta = [\pi \triangleright C]$ for some π, C .

Intuitively, the BNDS axiom states that the vulnerability of a hyper resulting from pushing a prior through a channel can only be as high as the vulnerability induced by the “most leaky” observable, and it can only be as low as the vulnerability induced by the “least leaky” observable.

We shall now consider the consequences of taking BNDS as an axiom. Whereas BNDS turns out to be strong enough to ensure NI, by replacing MAX with BNDS, the equivalence among Q-CVX, DPI and MONO no longer holds.

We begin by showing that the axiom of BNDS is sufficient to guarantee NI.

Theorem 11.22 (BNDS \Rightarrow NI) If a pair of prior/posterior vulnerabilities $(\mathbb{V}, \widehat{\mathbb{V}})$ satisfies BNDS, then it also satisfies NI.

Proof. If $(\mathbb{V}, \widehat{\mathbb{V}})$ satisfies BNDS, then

$$\min_{[\Delta]} \mathbb{V} \leq \widehat{\mathbb{V}}\Delta \leq \max_{[\Delta]} \mathbb{V}$$

for every hyper Δ . Consider, then, the particular case when Δ is $[\pi]$. Since $[\pi]$ is a point hyper with inner π , as in Thm. 11.16 above we have that $\min_{[\pi]} \mathbb{V} = \max_{[\pi]} \mathbb{V} = \mathbb{V}(\pi)$. That in turn implies that $\mathbb{V}(\pi) \leq \widehat{\mathbb{V}}[\pi] \leq \mathbb{V}(\pi)$, which is NI. \square

The next example shows that under BNDS, not even CVX –which is stronger than Q-CVX– is sufficient to ensure MONO or DPI.

Example 11.23 (BNDS \wedge CVX $\not\Rightarrow$ MONO and BNDS \wedge CVX $\not\Rightarrow$ DPI)

Consider the pair $(\mathbb{V}_2, \widehat{\mathbb{V}}_2)$ such that for every prior π and hyper Δ , we have

$$\begin{aligned} \mathbb{V}_2(\pi) &= \max_x \pi_x \quad \text{and} \\ \widehat{\mathbb{V}}_2\Delta &= (\max_{[\Delta]} \mathbb{V}_2 + \min_{[\Delta]} \mathbb{V}_2) / 2 \end{aligned}$$

The pair $(\mathbb{V}_2, \widehat{\mathbb{V}}_2)$ satisfies the axiom of BNDS, since $\widehat{\mathbb{V}}_2$ is the simple arithmetic average of the maximum and minimum vulnerabilities of the inners. The pair $(\mathbb{V}_2, \widehat{\mathbb{V}}_2)$ also satisfies the axiom of CVX, since $\mathbb{V}_2(\pi)$ is just the Bayes vulnerability of π .

To see that the pair $(\mathbb{V}_2, \widehat{\mathbb{V}}_2)$ does not satisfy the axiom of MONO, consider the prior $\pi^2 = (9/10, 1/10)$ and the (concrete) channel

$$C_2 = \begin{array}{|c|c|} \hline 8/9 & 1/9 \\ \hline 0 & 1 \\ \hline \end{array} .$$

We can calculate that $\mathbb{V}_2(\pi^2) = 9/10$, and that $[\pi^2 \triangleright C_2]$ has outer distribution $(4/5, 1/5)$, and inner distributions $(1, 0)$ and $(1/2, 1/2)$. Hence

$$\widehat{\mathbb{V}}_2[\pi^2 \triangleright C_2] = (1 + 1/2) / 2 = 3/4 ,$$

which violates MONO because $\widehat{\mathbb{V}}_2[\pi^2 \triangleright C_2] < \mathbb{V}_2(\pi^2)$.

Now to see that the pair $(\mathbb{V}_2, \widehat{\mathbb{V}}_2)$ does not satisfy DPI, consider the prior $\pi^3 = (3/7, 4/7)$ and the (concrete) channels

$$C_3 = \begin{array}{|c|c|} \hline 1/3 & 2/3 \\ \hline 1/4 & 3/4 \\ \hline \end{array} \quad \text{and} \quad R_3 = \begin{array}{|c|c|} \hline 1/4 & 3/4 \\ \hline 3/4 & 1/4 \\ \hline \end{array} .$$

We can calculate that $[\pi^3 \triangleright C_3]$ has outer distribution $(2/7, 5/7)$, and inners $(1/2, 1/2)$ and $(2/5, 3/5)$. Hence

$$\widehat{\mathbb{V}}_2[\pi^3 \triangleright C_3] = (1/2 + 3/5) / 2 = 11/20 = 0.55 .$$

On the other hand, the cascade $C_3 R_3$ yields the channel

$$C_3 R_3 = \begin{array}{|c|c|} \hline 7/12 & 5/12 \\ \hline 5/8 & 3/8 \\ \hline \end{array} ,$$

and we can calculate $[\pi^3 \triangleright C_3 R_3]$ to have outer distribution $(17/28, 11/28)$, and inners $(7/17, 10/17)$ and $(5/11, 6/11)$. Hence

$$\widehat{\mathbb{V}}[\pi^3 \triangleright C_3 R_3] = (10/17 + 6/11) / 2 = 106/187 \approx 0.567 ,$$

which makes $\widehat{\mathbb{V}}[\pi^3 \triangleright C_3 R_3] > \widehat{\mathbb{V}}[\pi^3 \triangleright C_3]$ and violates the axiom of DPI. \square

11.4 Applications of axiomatization to understanding leakage measures

The relationships we have uncovered among axioms helps us better to understand the multitude of possible leakage measures one can adopt (e.g. what V_g to use, and what version of leakage –additive or multiplicative– to employ).

A first instance of that concerns the robustness of the *refinement relation* (\sqsubseteq) that was discussed in Chap. 9. Recall that refinement is *sound* (Thm. 9.11) with respect to the strong g -leakage ordering ($\geq_{\mathbb{G}}$). Still, we might worry that refinement implies a leakage ordering *only* with respect to g -leakage, leaving open the possibility that the leakage ordering might conceivably fail for some yet-to-be-defined leakage measure. But Theorems 11.12 (AVG \wedge MONO \Rightarrow CVX) and 11.14 (AVG \wedge CVX \Rightarrow DPI) show that if the hypothetical new leakage measure is defined using AVG, and never gives negative leakage, then it also satisfies the data-processing inequality DPI. And hence refinement would be sound for the new leakage measure as well.

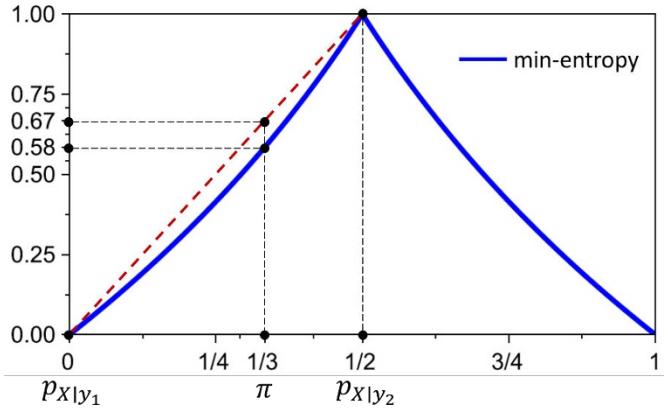
Another application concerns the possibility of negative leakage for some information measures. As an example, consider Rényi entropy which, we recall from §2.6, is a set of entropy measures that has been used in the context of quantitative information flow. The set is defined by

$$H_\alpha(\pi) = \frac{1}{1-\alpha} \log_2 (\sum_{x \in \mathcal{X}} \pi_x^\alpha)$$

for $0 \leq \alpha \leq \infty$ (taking limits in the cases of $\alpha=1$, which gives Shannon entropy, and $\alpha=\infty$, which gives min-entropy). It would be natural to use Rényi entropy to introduce a set of leakage measures by defining posterior Rényi entropy \widehat{H}_α (notated by analogy with $\widehat{\mathbb{V}}$) using AVG and defining Rényi leakage by

$$\mathcal{L}_\alpha(\pi, C) = H_\alpha(\pi) - \widehat{H}_\alpha[\pi \triangleright C] .$$

However, it turns out that H_α is not concave for $\alpha > 2$. Therefore, by the dual version of Thm. 11.12 (AVG+MONO \Rightarrow CVX), we find that Rényi leakage \mathcal{L}_α for $\alpha > 2$ would sometimes be *negative*. As an illustration, Figure 11.3 shows how the nonconcavity of min-entropy H_∞ can cause posterior min-entropy to be greater than prior min-entropy, giving negative min-entropy leakage. This problem is avoided, however, if we do not define posterior min-entropy via AVG, but instead by using $\widehat{H}_\infty[\pi \triangleright C] := -\log_2 V_1[\pi \triangleright C]$.



A picture showing how posterior min-entropy can be greater than prior min-entropy. Pushing prior $\pi := (1/3, 2/3)$ through concrete channel C defined

$$C := \begin{pmatrix} 0 & 1 \\ 1/2 & 1/2 \end{pmatrix}$$

gives hyper $[\pi \triangleright C]$ with outer $(1/3, 2/3)$ and the two inners $p_{X|y_1} = (0, 1)$ and $p_{X|y_2} = (1/2, 1/2)$.

Thus $H_\infty(\pi) = -\log_2 2/3 \approx 0.58$ and $\widehat{H}_\infty[\pi \triangleright C] = 1/3 \cdot 0 + 2/3 \cdot 1 \approx 0.67$.

Figure 11.3 Posterior min-entropy exceeding prior min-entropy

11.5 Chapter notes

The Gale-Klee-Rockafellar theorem is proved by Gale et al. [9]. The theorem was then used by Howe to show continuity “automatically” [11]. Observe also that for finite \mathcal{X} , we have that $\mathbb{D}\mathcal{X}$ is so-called “polyhedral” since it can be described by finitely many (hyper-) planes in $\mathbb{R}^{|\mathcal{X}|}$.

Our axioms for posterior vulnerabilities can be seen as generalizations of key properties of Shannon entropy [5]. As we mentioned, our axiom of data-processing inequality (DPI) is a straightforward generalization of the data-processing inequality for mutual information, which states that if $X \rightarrow Y \rightarrow Z$ forms a Markov chain, then $I(X;Y) \geq I(X;Z)$. Our axiom of monotonicity (MONO) is a generalization of Shannon entropy’s “*information can’t hurt*” property,¹⁷ which states that for every pair X, Y of random variables we have $H(X|Y) \leq H(X)$. Finally, our axiom of monotonicity (NI) can be seen as a generalization of the property that, for every pair of independent random variables X and Y , we have $H(X|Y) = H(X)$.

The conservative approach represented by the use of the axiom of maximum (MAX) is employed, for instance, in the original definition of *differential privacy* [8].

Csiszár has surveyed [6] the most commonly used postulates for a function f of the uncertainty contained in a finite probability distribution (p_1, \dots, p_N) for $N > 0$. They are (P1) *positivity*: $f(p_1, \dots, p_N) \geq 0$; (P2) *expansibility*: $f(p_1, \dots, p_N, 0) = f(p_1, \dots, p_N)$; (P3) *symmetry*: $f(p_1, \dots, p_N)$ is invariant under permutations of (p_1, p_2, \dots, p_N) ; (P4) *continuity*: $f(p_1, \dots, p_N)$ is a continuous function of (p_1, \dots, p_N) , for fixed N ; (P5) *additivity*: $f(P \times Q) = f(P) + f(Q)$, where $P \times Q$ is the product distribution of P and Q (i.e. the distribution in which events have probability $p_i q_j$ for each $p_i \in P$ and $q_j \in Q$); (P6) *subadditivity*: $f(A, B) \leq f(A) + f(B)$, where A and B are discrete random variables; (P7) *strong additivity*: $f(A, B) = f(A) + f(B|A)$; (P8) *recursivity*: $f(p_1, p_2, \dots, p_N) = f(p_1+p_2, p_3, \dots, p_N) + (p_1+p_2)f(p_1/(p_1+p_2), p_2/(p_1+p_2))$; and (P9) *sum-property*: $f(p_1, \dots, p_N) = \sum_{n=1}^N g(p_n)$ for some function g .

Shannon entropy is the only uncertainty measure to satisfy all axioms (P1–9) listed by Csiszár; but in fact various proper subsets of those axioms are sufficient to characterize Shannon entropy fully. In particular, Shannon himself showed that continuity, strong additivity, and the property that the uncertainty of a uniform distribution should not decrease as the number of elements in the distribution increases, are sufficient to determine entropy up to a constant factor [22]. Khinchin proved a similar result using strong additivity, expansibility, and the property that the maximum uncertainty should be realized in a uniform distribution [12].

Rényi explored ways to relax the axiomatization of Shannon entropy to derive more general uncertainty measures [21]. He showed that Shannon entropy could be characterized by five postulates: (R1) symmetry; (R2) continuity; (R3) $f(1/2, 1/2) = 1$; (R4) additivity; and (R5) the entropy of the union of two subdistributions is the arithmetic weighted average of each individual distribution. By replacing the weighted average in postulate (R5) with the (more relaxed) exponential mean, Rényi uniquely determined the set of Rényi entropies for full probability distributions $H_\alpha(p_1, p_2, \dots, p_n) = 1/(1-\alpha) \log_2(\sum_{k=1}^n p_k^\alpha)$, where $0 < \alpha < \infty$, with $\alpha \neq 1$, is a parameter. In the limit of α tending to 1, Rényi entropy H_α coincides with Shannon entropy, and in the limit of α tending to infinity, it is min-entropy (i.e. the negated logarithm of Bayes vulnerability).

Following Denning’s seminal work [7], Shannon entropy has been widely used in the field of quantitative information flow for the leakage of confidential information [4, 17, 3, 18, 20, 19, 1]. But as the field of quantitative information flow continued

¹⁷ From our adversarial perspective, however, this becomes “information always hurts”.

to evolve, new measures of uncertainty and of information were proposed. Contrary to Rényi's motivation, however, most measures were not derived from mathematical principles, but instead were motivated by specific operational scenarios. That was the case for guessing entropy, Bayes vulnerability, and g -vulnerability, for instance. Although many *healthiness conditions* have been proved after the fact for those measures (e.g. non-negativity, non-decrease of uncertainty by post-processing etc.) there has not always been a derivation of such measures from basic principles, nor attempts to verify whether they can be unified in a more general framework.

Naturally, since measures other than Shannon entropy cannot satisfy all postulates (P1–9), the axioms for vulnerability considered in this chapter differ from those listed by Csiszár. Some differences are unimportant: they are just adaptations of axioms of uncertainty to axioms of vulnerability (e.g. conditioning of random variables reduces uncertainty, but increases vulnerability, so some inequalities must be reversed).

Other differences are more fundamental, however, as they reflect our departure from Shannon's indifference to the *meaning* of different secret values. The axiom of symmetry (P3), for instance, assumes that all secret values are equally informative — and in many scenarios that is false. For instance, some bank accounts are more attractive to an attacker than others, and so evidently in that case a permutation on the probabilities of every particular account being broken into does not amount to the same vulnerability. The axioms of additivity (P5), subadditivity (P6) and strong additivity (P7) assume that the uncertainty of a pair of joint random variables is a function only of the correlation of the random variables, which is also not a valid assumption in many security scenarios: the information of the combination of two secrets may exceed the information content of each secret separately: for instance, the benefit of knowing someone's PIN *and* bank-account number at the same time greatly surpasses the sum of the benefits of knowing each one on its own. Recursivity (P8) and the sum-property (P9) assume that the probability of each secret value contributes on equal terms to the overall uncertainty of the probability distribution, which also is a false assumption for many relevant measures. Bayes vulnerability, for instance, satisfies neither recursivity nor the sum-property, as the information of a probability distribution is a function of the maximum probability only.

In the field of statistical disclosure control, query mechanisms must be evaluated according to both the level of privacy they provide for individuals in a database and the level of utility of the provided answers to the queries. In a series of papers [13, 15, 14, 16], Lin and Kifer propose an axiomatic characterization of “good” properties that sanitization mechanisms should provide, focusing in particular on *privacy-* and *utility measures*. They consider utility as *information preservation*, which captures how *faithful* the output of the mechanism is to its input,¹⁸ and as such is closely related to the notion of vulnerability we study in this book.

Boreale and Pampaloni have conducted one of the first studies of adaptive adversaries in the context of quantitative information flow [2]. Although they do not consider an axiomatic framework explicitly, for the sake of generality they adopt a generic notion of entropy, specified by a few properties which turn out to be our axioms of concavity, continuity, and averaging. Furthermore, they point out a known theorem in decision theory, which states that a function $H: \mathbb{D}\mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$ satisfies concavity and continuity iff it is of the form $H(\pi) = \sum_x \pi_x S(x, \pi)$, where $S: (\mathcal{X} \times \mathbb{D}\mathcal{X}) \rightarrow \mathbb{R}_{\geq 0}$ is any function satisfying the condition that $\sum_x \pi_x S(x, \pi')$ is minimal when $\pi' = \pi$. Such a function S , called the *Proper Scoring Rule* in decision theory, is similar to *loss functions*, the

¹⁸ This is in contrast with utility as *usability*, which expresses how easily the output can be used. An example of the difference is provided by an encryption mechanism, which perfectly preserves information, but whose output is not usable except by users who know the decryption key.

dual of the gain functions we have so far used to define g -vulnerability,¹⁹ and therefore the above definition is related to that of prior g -entropy. Thus this result is similar to that of the completeness of g -vulnerability with respect to our axiomatization of prior vulnerability.

Finally, many of the axioms here have very succinct formulations in terms of Giry's "probabilistic monad" [10], as discussed in §14.7.

¹⁹ In fact we use loss functions in later sections of this book, when our focus is on programming-language semantics.

Bibliography

- [1] Alvim, M.S., Andrés, M.E., Palamidessi, C.: Quantitative information flow in interactive systems. *Journal of Computer Security* **20**(1), 3–50 (2012)
- [2] Boreale, M., Pampaloni, F.: Quantitative information flow under generic leakage functions and adaptive adversaries. *Logical Methods in Computer Science* **11**(4) (2015)
- [3] Chatzikokolakis, K., Palamidessi, C., Panangaden, P.: Anonymity protocols as noisy channels. *Information and Computation* **206**(2-4), 378–401 (2008)
- [4] Clark, D., Hunt, S., Malacaria, P.: Quantitative information flow, relations and polymorphic types. *Journal of Logic and Computation* **15**(2), 181–199 (2005)
- [5] Cover, T.M., Thomas, J.A.: Elements of Information Theory. Wiley Series in Telecommunications and Signal Processing. Wiley-Interscience (2006)
- [6] Csiszár, I.: Axiomatic characterizations of information measures. *Entropy* **10**(3), 261–273 (2008)
- [7] Denning, D.E.: Cryptography and Data Security. Addison-Wesley (1983)
- [8] Dwork, C.: Differential privacy. In: M. Bugliesi, B. Preneel, V. Sassone, I. Wegener (eds.) *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP 2006)*, *Lecture Notes in Computer Science*, vol. 4052, pp. 1–12. Springer, Berlin (2006)
- [9] Gale, D., Klee, V., Rockafellar, R.T.: Convex functions on convex polytopes. *Proceedings of the American Mathematical Society* **19**(4), 867–873 (1968)
- [10] Giry, M.: A categorical approach to probability theory. In: B. Banaschewski (ed.) *Categorical Aspects of Topology and Analysis*, *Lecture Notes in Mathematics*, vol. 915, pp. 68–85. Springer, Berlin (1981)
- [11] Howe, R.: Automatic continuity of concave functions. *Proceedings of the American Mathematical Society* **103**(4), 1196–1200 (1988)
- [12] Khinchin, A.I.: Mathematical Foundations of Information Theory. Dover Books on Mathematics. Dover Publications (1957)
- [13] Lin, B.R., Kifer, D.: Towards an axiomatization of statistical privacy and utility. In: *Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS 2010, pp. 147–158. ACM, New York (2010)

Bibliography

- [14] Lin, B.R., Kifer, D.: An axiomatic view of statistical privacy and utility. *Journal of Privacy and Confidentiality* **4**(1), 5–49 (2012)
- [15] Lin, B.R., Kifer, D.: Reasoning about privacy using axioms. In: 2012 Conference Record of the Forty Sixth Asilomar Conference on Signals, Systems and Computers (ASILOMAR), pp. 975–979. IEEE, Los Alamitos (2012)
- [16] Lin, B.R., Kifer, D.: Information measures in statistical privacy and data processing applications. *ACM Transactions on Knowledge Discovery from Data* **9**(4), 28 (2015)
- [17] Malacaria, P.: Assessing security threats of looping constructs. In: Proceedings of the 34th ACM Symposium on Principles of Programming Languages (POPL 2007), pp. 225–235. ACM, New York (2007)
- [18] Malacaria, P., Chen, H.: Lagrange multipliers and maximum information leakage in different observational models. In: Proceedings of the 3rd Workshop on Programming Languages and Analysis for Security (PLAS 2008), pp. 135–146. ACM, New York (2008)
- [19] Moskowitz, I.S., Newman, R.E., Crepeau, D.P., Miller, A.R.: Covert channels and anonymizing networks. In: Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society (WPES 2003), pp. 79–88. ACM, New York (2003)
- [20] Moskowitz, I.S., Newman, R.E., Syverson, P.F.: Quasi-anonymous channels. In: Proceedings of the IASTED International Conference on Communication, Network, and Information Security (CNIS 2003), pp. 126–131. IASTED, Calgary (2003)
- [21] Rényi, A.: On measures of entropy and information. In: Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics, pp. 547–561. University of California Press, Berkeley (1961)
- [22] Shannon, C.E.: A mathematical theory of communication. *Bell System Technical Journal* **27**(3), 379–423, 623–656 (1948)



Chapter 12

The geometry of hypers, gains and losses

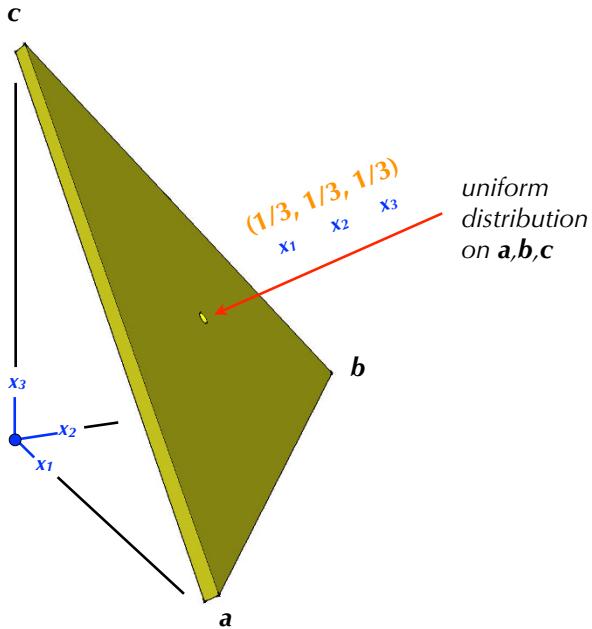
Many of the constructions we have seen so far have a compelling geometric interpretation. Information is not only numbers and formulae — it is also curves and tangents, shapes and planes. For example, over a state space \mathcal{X} of size N , the discrete distributions $\mathbb{D}\mathcal{X}$ can be represented as points in Euclidean N -space and, since they sum to one, in fact they lie on the (hyper-) plane $x_1 + \dots + x_N = 1$ of one fewer dimensions, a regular polytope.

We have seen that when $N=2$ and \mathcal{X} is say $\{\mathbf{a}, \mathbf{b}\}$ the distributions over \mathcal{X} are on a line of unit length from \mathbf{a} to \mathbf{b} along the (horizontal) x_1 -axis, with each point x_1 there representing a distribution $\mathbf{a}_p \oplus \mathbf{b}$ for p varying between 0 and 1. Any vulnerability V_g can then be graphed as a curve in the 2-dimensional x_1, y plane, that is $y = V_g(\mathbf{a}_p \oplus \mathbf{b})$ where $\mathbf{a}_p \oplus \mathbf{b}$ is the distribution represented by x_1 . We saw an example of that in Fig. 3.1 on p. 28.

When $N=3$, i.e. one dimension more, the distributions are found on an equilateral triangle with vertices labeled $\mathbf{a}, \mathbf{b}, \mathbf{c}$ say, the part of the plane $x_1 + x_2 + x_3 = 1$ lying in the positive octant of 3-space — which is itself the base of the right tetrahedron obtained by adding the origin $(x_1, x_2, x_3) = (0, 0, 0)$ as apex. Each point on the base has three coordinates, which give the probabilities assigned to each of $\mathbf{a}, \mathbf{b}, \mathbf{c}$, so that e.g. the point distribution $[\mathbf{b}]$ has coordinates $(x_1, x_2, x_3) = (0, 1, 0)$. And all distribution-representing points lie on the plane $x_1 + x_2 + x_3 = 1$ mentioned above. That is illustrated in Fig. 12.1.

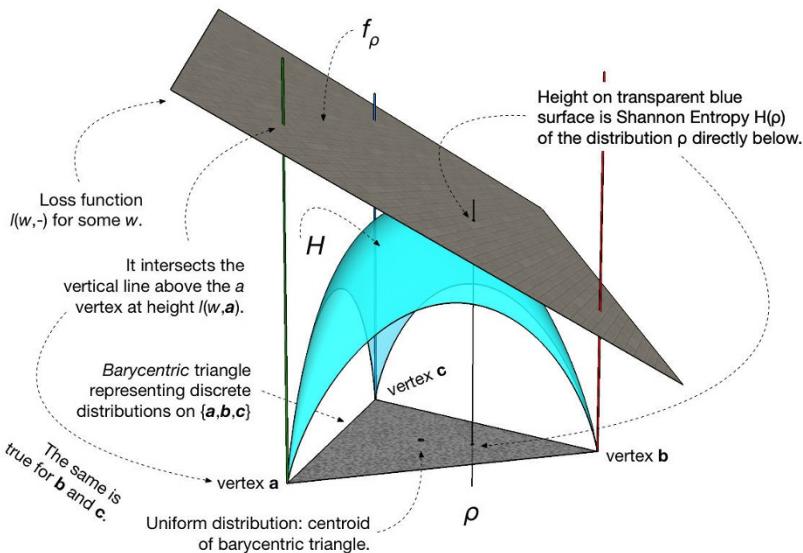
If we remain with $N=3$ but concentrate on the equilateral triangle alone, moving thus back into 2-space, then a point within the triangle (or on an edge, or indeed even at a vertex) represents a probability distribution given by the unique linear combination of the vertices that determines the point. The center of mass of the triangle, for example, is obtained by weighting each vertex by $1/3$, and so represents the uniform distribution; each vertex is a point distribution; and the side-midpoints are sub-uniform distributions on the two vertices at either end of the side. Vulnerabilities then become surfaces, rather than curves, lying above a triangle instead of a line; and that is illustrated in Fig. 12.2 for Shannon entropy, where the barycentric equilateral triangle of Fig. 12.1 is detached from its right-tetrahedron and becomes the base above which the entropy indeed appears as a surface.

In general, such a representation is called “barycentric”, and we will see later that we can represent not only distributions (Fig. 12.1) and entropies (Fig. 12.2), but also hyper-distributions (Fig. 12.3 in §12.2).



The barycentric representation of probabilities over a 3-element state space is the equilateral triangle at the base of a right tetrahedron with its apex at the origin.

Figure 12.1 The barycentric representation in 3-space



The barycentric representation of $\mathbb{D}\{a, b, c\}$ is the equilateral triangle at bottom, with the Shannon entropy $H(\rho)$ of each point, i.e. distribution ρ being the height of the blue surface above it. The tangential plane represents a loss-function component $\ell(w, -)$ for a particular $w: \mathcal{W}$: it is a linear function on the triangle, and its coefficients are given by its intersections with the vertical lines rising from the triangle's vertices.

Figure 12.2 Barycentric representation of uncertainties and loss functions

12.1 Barycentric representation of gain/loss functions

We now make the above ideas more precise. A discrete distribution on finite \mathcal{X} is a function from \mathcal{X} into $[0, 1]$, with the property of course that its range sums to 1. If \mathcal{X} is of size N , then that naturally suggests a representation as a point in the N -dimensional unit cube, where the axes are labeled with elements of \mathcal{X} and a point's coordinates for each axis give the probability associated with the label. Because of the 1-summing property, however, all those distributions will in fact lie on a subset of that cube, on a (hyper-)plane of dimension $N-1$.

We consider 3-space by letting \mathcal{X} be $\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$, and associating $\mathbf{a}, \mathbf{b}, \mathbf{c}$ with the x_1, x_2, x_3 -axes resp. in its Cartesian representation. Distributions over \mathcal{X} will all lie on the plane $x_1+x_2+x_3=1$ and within the octant $x_1, x_2, x_3 \geq 0$. That shape is the base of a right tetrahedron, with its apex at the origin $(0, 0, 0)$ and its base an equilateral triangle with vertices at $(1, 0, 0)$, $(0, 1, 0)$ and $(0, 0, 1)$; and so all distributions lie within that triangular base (or on its boundary).

The barycentric representation of $\mathbb{D}\mathcal{X}$ is obtained by taking just that triangle on its own, with the advantage that we move back into 2-space: the triangle's three vertices are labeled $\mathbf{a}, \mathbf{b}, \mathbf{c}$ and any point within the triangle is a summing-to-one linear interpolation of those three vertices. The point distributions are the vertices themselves, since that is an interpolation of $(1\times)$ that vertex and $(0\times)$ each of the other two; and the uniform distribution is the center of mass of the triangle, an interpolation of $(1/3\times)$ for each of them. The closer a distribution's point is to a vertex, the higher the probability contributed by that vertex. The sub-uniform distribution on just $\{\mathbf{a}, \mathbf{b}\}$, for example, that is $(1/2\times)$ for each, is represented by the midpoint of the triangle's edge that connects vertex \mathbf{a} to vertex \mathbf{b} .

With the now freed-up third dimension we can represent a *function* on $\mathbb{D}\mathcal{X}$, specifically a gain or loss function, as a surface above the barycentric triangle.

As we will see in §12.2 below, there is a barycentric representation of hyper-distributions as well. For the moment however we concentrate on the connection between vulnerability/uncertainty functions in general (continuous convex/concave functions on $\mathbb{D}\mathcal{X}$) and their representation as maximums/minimums of gain/loss functions g/ℓ on $\mathcal{W} \times \mathcal{X}$.

We return briefly to the simpler $N=2$ case. As we saw earlier, the barycentric representation there is a unit line; a distribution is a point on that line, which we will think of as the x -axis for graphing functions. A vulnerability/uncertainty is a convex/concave continuous function whose y -value at any point is the vulnerability/uncertainty of the x -represented distribution directly beneath it on the x -axis. We saw examples of that earlier, in §3.1.1 and, in particular, in Fig. 3.1. Here in our more general context however we must be careful about the representation: if the origin, that is $x=0$, corresponds to value \mathbf{a} and $x=1$ to value \mathbf{b} then the point $x=p$ on the x -axis is the distribution $\mathbf{b} \oplus \mathbf{a}$. (In Fig. 3.1 it was the other way around, which we admit does at first feel more natural; but it does not generalize easily.)

The gain/loss function components that *generate* the vulnerability/uncertainty, i.e. the $g(w, -)$ that determines the V_g or the $\ell(w, -)$ that determines the U_ℓ , are w -indexed tangents to the curve V_g or U_ℓ , lying either (non-strictly) below it or above it depending on whether we are doing gain or loss. Ignoring some details,¹ the completeness of gain/loss functions wrt. vulnerability/uncertainty can be visualized informally as the fact that one can go from one representation to the other, i.e. from V_g to g 's or from g 's to V_g , by taking all tangents (to give the gain/loss functions) or—in the opposite direction—taking the boundary of the intersections of all half-planes

¹ Recall §11.2.1 for a discussion of those details, however.

(to give the vulnerability/uncertainty). As we noted earlier (Fig. 3.5(b) in §3.2.7) the lines represent actions, and the curve is their envelope. We see immediately that if the set \mathcal{W} of actions is finite, then the associated gain/loss function is piecewise linear.

We now move back to the more interesting case of $N=3$ and $\mathcal{X}=\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ and our example for Shannon entropy specifically in Fig. 12.2: since it's an entropy, aka. uncertainty, we will use loss functions rather than gain functions.

The barycentric representation of our state space is the horizontal gray equilateral triangle at bottom, with its three vertices $\mathbf{a}, \mathbf{b}, \mathbf{c}$. Above it, the light-blue surface gives the Shannon entropy for all the distributions below: for example, at the vertices which represent the point distributions, the entropy is zero. The midpoints of the sides give entropy 1, correct for a uniform distribution over its two endpoints; and the maximum entropy $\log_2 3 \approx 1.58$ occurs above the triangle's centroid, since it represents the uniform distribution on all of \mathcal{X} .

The dark, angled plane is one of the infinitely many tangential planes that are the envelope of the Shannon-entropy surface: each one corresponds to a particular action w , and defines three coefficients, say a_w, b_w, c_w , that can be recovered geometrically as the height of the plane's intersections with the vertex-verticals at $\mathbf{a}, \mathbf{b}, \mathbf{c}$. Each particular w gives $a_w\pi_a + b_w\pi_b + c_w\pi_c$ as the loss associated with choosing action w for prior $\pi = (\pi_1, \pi_2, \pi_3)$ — that is we have $\ell(w, a) = a_w$ etc.

The fact that the plane labeled f_ρ in Fig. 12.2 is tangential above ρ to the Shannon surface labeled H represents geometrically that indeed that loss function gives the *least* loss for an attack on the prior ρ below; and from Gibbs' Inequality² we know that the plane's coefficients a_w, b_w, c_w will in fact be $-\log_2 \pi_a, -\log_2 \pi_b, -\log_2 \pi_c$ respectively.

In more detail, let the Shannon surface H be a function given by the Shannon-entropy function $H(\delta)$ of any distribution δ in the barycentric triangle, fix some particular distribution ρ among those δ 's, and then consider the linear function $f_\rho(\delta) := -\delta_a \log_2 \rho_a - \delta_b \log_2 \rho_b - \delta_c \log_2 \rho_c$ of arbitrary δ expressed wrt. δ 's barycentric coordinates $(\delta_a, \delta_b, \delta_c)$ — being linear, it is of course a plane lying above the barycentric triangle. At vertex \mathbf{a} of the triangle, say, corresponding to distribution $\delta = [\mathbf{a}] = (1, 0, 0)$, we have $f_\rho(\delta) = f_\rho[\mathbf{a}] = -\log_2 \rho_a$, and similarly for \mathbf{b}, \mathbf{c} . Thus indeed the coefficients of those f_ρ -planes are given by their intersections with the verticals at the vertices.

Now by the definition of Shannon entropy we have that $f_\rho(\rho) = H(\rho)$, so that the plane f_ρ and the surface H intersect above distribution ρ . And it is a tangent there: for if the intersection were not tangential, then by continuity there would be some nearby distribution ρ' such that $f(\rho') < H(\rho')$, which contradicts Gibbs' inequality. Again by Gibbs' inequality, we know that in fact $H(\delta)$ lies strictly below $f_\rho(\delta)$ for all distributions $\delta \neq \rho$, which confirms that H is concave. Thus the tangential plane for H at δ is indeed f_δ , and H itself is the (below-)envelope of all the $f_\rho(-)$ planes that we get by allowing ρ to range over all distributions δ . Thus in the Shannon-entropy case the set \mathcal{W} of actions is $\mathbb{D}\mathcal{X}$ itself.

It is that construction that shows Shannon entropy to be representable as a loss function, at the cost however of needing infinite \mathcal{W} and taking inf rather than min. As we know, any continuous, concave loss function can be so represented (§11.2.1).

² See §3.2.8 for Gibbs' inequality.

12.2 Barycentric representation of hypers and their refinement

In Fig. 4.1 (p. 53) we saw a barycentric representation not only of a prior, but of the inners produced by a channel applied to it. It showed not only their positions within the barycentric triangle, but also the probability assigned to each one by the outer distribution: the more probability assigned by the outer, the bigger the circle centered on the inner. A similar representation is given in Fig. 12.3, and we now define refinement in those terms.

Definition 12.1 (Structural refinement of hypers, geometrically) Let N be the size of our state-space \mathcal{X} , so that the barycentric representation of $\mathbb{D}\mathcal{X}$ is a regular polytope of dimension $N-1$.³ For convenience, focusing on $N=3$, here we'll call it a triangle.

A hyper Δ in $\mathbb{D}^2\mathcal{X}$ is then a set of (hyper-)disks of total area 1 with their centers non-strictly within the barycentric triangle. Each disk is centered on one of Δ 's inners, and the area of that disk gives the outer (probability) that Δ assigns to that inner. Thus for example the set of disk-centers is the support of hyper Δ .

Another hyper Δ' , similarly represented, is then a (structural) *refinement* of Δ , that is $\Delta \sqsubseteq_\circ \Delta'$, just when it can be constructed from Δ in the following way:

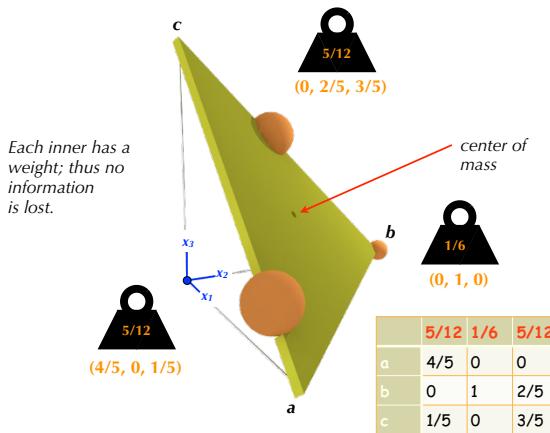
1. Divide each disk of Δ into any number of smaller disks, preserving however the total area of the original disk; and then stack them all at the same place as the original disk. (Think of a single plate on a dining table being replaced, at the same spot, by a stack of smaller plates whose total area equals the area of the original plate.)

As a special case however, the original disk can just be left there, undivided.

2. Select groups of the stacked disks, taking them from different stacks (a dessert plate from here, a side plate from there...) but do not move them yet. Those groups will become the inners of the refined hyper.
3. Now go through the groups, one by one, replacing each group by a new single disk, of area equal to the sum of the group members' areas, and centered on the group's center of mass. (Thus the separate plates of a group become a single plate of the same total area, but in a new place — where the original plates in the group would have “balanced” if you considered them temporarily on a table of their own: literally, where you would have put your hand to hold them all at once in the air above your head.)
4. For every case where there are (now) several disks with their centers on the same point, replace them by a single disk of the same total area and on the same point. (It does not matter whether the disks involved are new or original.)
5. Take the new set of disks to be Δ' .

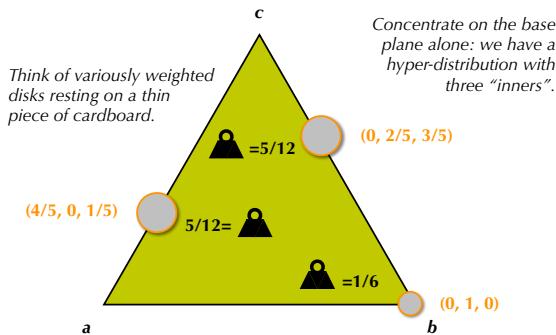
Note that the overall center of mass of the original hyper Δ will be the center of mass of its refinement Δ' as well. It is in fact, in both cases, the prior from which the hyper is made via some channel. \square

³ Polytope is the generalization in N dimensions of line ($N=1$), polygon ($N=2$) and polyhedron ($N=3$). The (hyper-) sides of an N -polytope are themselves $N-1$ -polytopes: thus the “sides” of a line are its two endpoints, the sides of a polygon are lines and the “sides” (faces) of a polyhedron are polygons. Our regular polytopes for a state space $\{x_{1..N}\}$ are: for $N=2$ a line with labels a,b at its endpoints; for $N=3$ an equilateral triangle with labels a,b,c at its vertices; and for $N=4$ we would have a regular tetrahedron (all of it, including its interior) with labels a,b,c,d at its vertices.



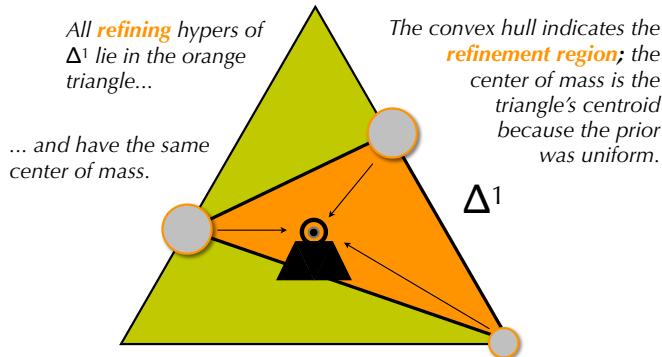
Hypers can be represented as collections of marbles on the base of a right tetrahedron: their positions indicate the inners, and their weights indicate the corresponding outer.

Figure 12.3 Hypers as marbles on a tetrahedron



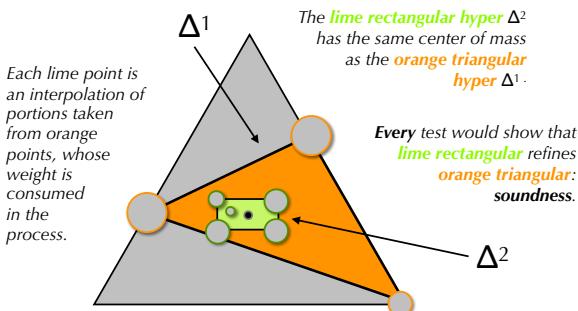
By concentrating on the base of the tetrahedron, we can move back into two dimensions. The marbles become disks.

Figure 12.4 Hypers as disks in a triangle



Refinement of a hyper must result in a new hyper within the convex hull of the original.

Figure 12.5 Refinement defined by convex hull



The **lime** rectangular hyper Δ^2 refines the **orange** triangular hyper Δ^1 ; Δ^2 's own refinements would lie (non strictly) within the rectangle.

Figure 12.6 Rectangle refining triangle

We now present some examples over our three-element state space.

- (1) Consider the concrete channel over state space \mathcal{X} of size 3 given by the matrix here at right

$$\begin{pmatrix} 1/3 \\ 1/3 \\ 1/3 \end{pmatrix} \triangleright \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1/2 & 1/2 \\ 1/4 & 0 & 3/4 \end{pmatrix},$$

and apply to it via (\rightarrow) as shown the uniform distribution at left. The resulting hyper is shown in Fig. 12.3. Each of the three inners can be located as distributions represented by points on the base of the tetrahedron in the figure, and a sphere of size proportional to the outer can be centered on that point. No information is lost in that representation.

- (2) Then by concentrating on just the base of the tetrahedron, we can move back into 2-dimensional space, as shown in Fig. 12.4: we get the barycentric space of distributions on state space $\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ of size 3, with the hyper's inners represented by a collection of disks each of whose size is proportional to that inner's outer probability.
- (3) Since refinement (\sqsubseteq_\circ) of hypers is the weighted merging of (some of) the inners (Def. 12.1), if hyper $\Delta^1 \sqsubseteq \Delta^2$ then all the disks (their centers) of Δ^2 must lie in the convex hull of the disks of Δ^1 , as shown in Fig. 12.5. That is a necessary condition, but it is not sufficient, of course: one obvious extra condition is that the center of mass of the two hypers (each one a collection of disks) must be the same, because that center is the prior from which the two hypers were generated — and for refinement it must be the same prior for each.

Fig. 12.6 shows an example of a refinement: the small lime rectangle refines the larger orange triangle.

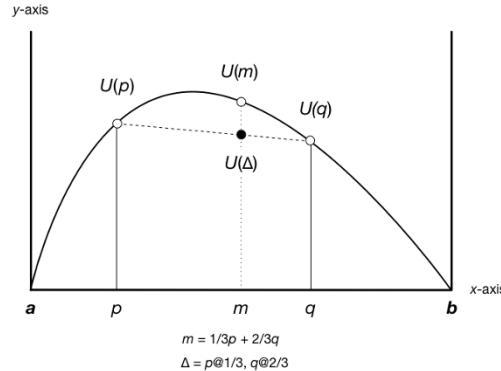
12.3 Primitive hyper-distributions and their refinements

We call a hyper-distribution *primitive* if it has no more inners than the size of the state space, and those inners are independent: none can be obtained by a linear combination of others. When $N=3$, for example, such a hyper would be a proper triangle — that is, three points that are not collinear. (Note that a hyper's being primitive does not depend on its outer.) We now show that primitive hypers have two special geometric properties.

The first geometric property of a primitive hyper is that *all* hypers that lie within its convex hull, whether themselves primitive or not, are refinements of it provided they have the same center of mass. In Fig. 12.6 for example, we can be sure that Δ^2 refines Δ^1 simply because it is “inside” Δ^1 and has the same center of mass. We prove that as follows.

Lemma 12.2 (Refinement of primitive hypers) Over a state space of size N , a hyper with N or fewer inners, all of them independent, can be refined to any hyper lying within its convex closure that has the same center of mass, where by “independent” is meant that no inner can be expressed as an interpolation of others.

Proof. Note first that any point within the convex closure of a set of points can by definition be expressed as a linear interpolation of those points; and when the points themselves are linearly independent, that interpolation is unique.



The curved line is an uncertainty U , and the points labeled p, q on the x -axis represent the distributions $b_p \oplus a$ and $b_q \oplus a$ respectively. (If those seem backward, remember that it is a barycentric representation.) They are inners of a hyper Δ that assigns weight $1/3$ to the inner p and $2/3$ to the inner q , and the hyper's center of mass is labeled m (for "middle"), representing $b_m \oplus a$. (Note that m is closer to q than to p because inner q has the greater outer probability in hyper Δ .)

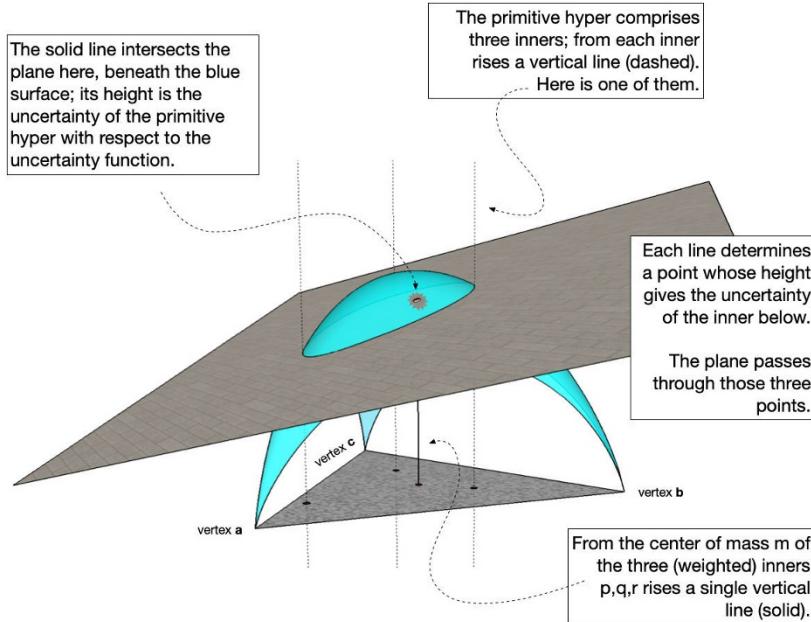
The three white circles give by their height above the x -axis the (prior) uncertainty of the distributions beneath them. The black circle gives the *conditional* uncertainty of the hyper Δ whose two inners are $b_p \oplus a$ and $b_q \oplus a$ with outer probabilities $1/3$ and $2/3$ respectively.

That the point $U(\Delta)$ is below $U(m)$ demonstrates CVX from Chap. 11.

Figure 12.7 Graphical construction of conditional uncertainty when $N=2$

Now take any (unweighted) "target" inner of the (putative) implementation (i.e. of a more-refined hyper, for example Δ^2 in Fig. 12.6), and note that since it is in the convex closure of the specification (i.e. less-refined hyper Δ^1), it can be expressed as an interpolation of the specification's support, i.e. of its unweighted inners. Scale that interpolation by the weight of the target inner to give an interpolation of *weighted* specification inners. (The coefficients of the interpolation will no longer sum to one: instead they will sum to the outer probability associated with that target.) If we do that for each implementation inner, we have expressed each weighted implementation inner as a (sub-)linear combination of specification inners. What we have not shown however is that the sum over all implementation inners of the contribution of each separate specification inner is equal to that specification inner's own outer probability: and that is necessary for refinement. We do it as follows.

Adding all the implementation inners' interpolations together must by associativity and commutativity of addition give the center of mass of the implementation. But that, by assumption, is the center of mass of the specification as well. Thus the weights assigned by the summation to the specification'souters must be such that as a linear combination they give the specification's center of mass. But we know that that linear combination is unique, and so it must agree with the specification inners' actual weights. \square



Again the barycentric representation of $\mathbb{D}\{a, b, c\}$ is the equilateral triangle at bottom; and the uncertainty function is again Shannon entropy. The inners of some primitive hyper are small circles within $\mathbb{D}\{a, b, c\}$, forming a triangle themselves; although the inners' weights are not shown, they are determined uniquely by their own center of mass, shown as a single circle within their triangle.

The three dotted lines rising from the inners give, by the height at which they intersect the curved blue surface, the Shannon entropy of each inner on its own; and the plane passing through those three points interpolates those separate entropies in the same proportion as the interpolation of the inners to give distributions within the triangle they define. Thus in particular the height of the solid line through the three points' center of mass that ends *at that plane* (not at the Shannon surface) is the conditional Shannon entropy of the primitive hyper we started with.

Figure 12.8 Graphical construction of conditional uncertainty when $N=3$

The second geometric property of primitive hypers is that they admit a particularly nice graphical determination of conditional, i.e. posterior uncertainty. We illustrate the property first when $N=2$. In that case, a primitive hyper Δ is determined by the two probabilities p, q representing its two inners: one inner assigns probability p to element a of \mathcal{X} and $1-p$ to b , and similarly for q . A third probability m (for “middle”) gives the distribution of the center of mass, which is of course $\mu\Delta$. When placed on the unit interval, the barycentric representation of $\mathbb{D}\mathcal{X}$ when $N=2$, the points p, q, m must wlog. come in the order p, m, q , because m is an interpolation of the other two: it must lie between them. Furthermore, the (outer) probabilities assigned to the inners p, q are determined uniquely by m and so, knowing only p, q, m , it is meaningful for some uncertainty U to ask “What is $U(\Delta)$?”

The answer is obtained graphically. Take the two verticals from p, q and note their intersections with U . Draw a line between those two intersections, and note its intersection with the vertical from m . The height of that point (above m) is in fact $U(\Delta)$, as illustrated in Fig. 12.7.

Note that the concavity of U means that the point just determined must in fact lie *below* U , a nice graphical illustration of axiom CVX from Def. 11.2: it is a geometric illustration of Thm. 5.8.

When $N=3$, a similar procedure applies. The three points p, q, r in the barycentric triangle representing $\mathbb{D}\mathcal{X}$ are the inners of some primitive hyper Δ , and their being independent means they do not lie on a single line. Let m be their center of mass, and imagine some concave uncertainty function U draped like a tent above the barycentric triangle, as in Fig. 12.8. As before, the center of mass m determines uniquely the outers of the three inners p, q, r of Δ , and so it is meaningful to ask for $U(\Delta)$.

Again the answer is obtained graphically. Take now *three* verticals from p, q, r and note their intersections with U . Construct the plane containing all three intersections, and note its intersection with the vertical from the center of mass m . The height of that point (above m) is in fact $U(\Delta)$.

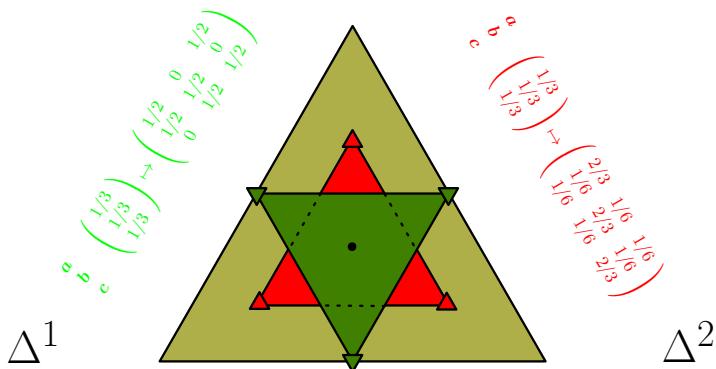
12.4 Hyper-distributions are not a lattice under refinement

In §9.2.1 it was remarked that the partial order (\sqsubseteq) of refinement when restricted to deterministic channels is a lattice, for which we recall the definition just below. And in §17.1 it will be shown that also “demonic channels” (which we have not yet discussed) are a lattice under refinement (Lem. 17.8). Here however we will see that refinement (\sqsubseteq) over hyper-distributions is *not* a lattice. We give a counterexample.

Definition 12.3 (Lattice) A partial order (\sqsubseteq) is called a *lattice* just when every two elements in it have a least upper bound, called their *join*, and a greatest lower bound, called their *meet*. \square

Fig. 12.9 shows two hypers $\Delta^{1,2}$ in barycentric representation over a three-element state space a, b, c : one hyper Δ^1 is on top (pointing down), and the other hyper Δ^2 is underneath (pointing up); each has three equally weighted inners, at the vertices of the triangles. As shown in the figure, they are generated from the same (uniform) prior, but use different channels. Call them the “triangle” hypers: we will show they have no refinement-join.

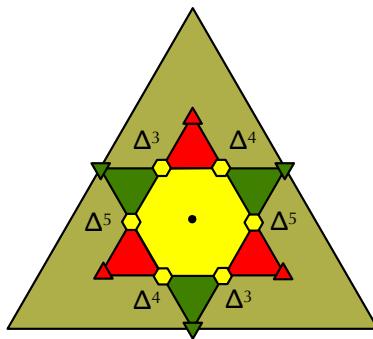
As we stated above, and was illustrated in Fig. 12.6, all refinements of Δ^1 must lie in the upwards-facing triangle, and all refinements of Δ^2 must lie in the downwards-facing triangle, and so any hyper that refines them both must lie in the triangles’ intersection, which is the hexagon shown in Fig. 12.9 with alternating dotted and solid lines. (In



The two hypers $\Delta^{1,2}$ shown are generated by the (differing) channel matrices and (the same) uniform prior shown at the sides; each has three inners, with weight $1/3$ each. We will show that Δ^1 and Δ^2 have no refinement-join.

The dot in the center is the (uniform) prior, and thus is the center of mass of both triangles.

Figure 12.9 Hyper Δ^1 pointing down; hyper Δ^2 pointing up



Three further hypers $\Delta^{3,4,5}$ are made from the vertices of the hexagon that is the intersection of hypers $\Delta^{1,2}$ — hyper Δ^3 for example is made by inners of weight $1/2$ at 11- and 5 o'clock. We will show that $\Delta^{3,4,5}$ have no refinement-meet.

Figure 12.10 Hypers $\Delta^{3,4,5}$, with inners at opposing vertices of the hexagon

fact Lem. 12.2 showed that *any* hyper in the hexagon must be a refinement of both Δ_1 and Δ_2 , provided it has the same center of mass — but we do not need that here.) Say during this argument that all hypers with the same center of mass as $\Delta^{1,2}$ are *centered*.

We now define three further centered hypers $\Delta^{3,4,5}$ formed by pairs of inners of weight $1/2$ on diametrically opposed vertices of the hexagon, as shown in Fig. 12.10. Call them the “diameter” hypers. It should be clear that each diameter is a refinement of each triangle, which six refinement-instances can be written compactly as $\Delta^{1,2} \sqsubseteq \Delta^{3,4,5}$.

We will show that there is no hyper Δ such that

$$\Delta^{1,2} \sqsubseteq \Delta \sqsubseteq \Delta^{3,4,5}. \quad (12.1)$$

But first we need a technical lemma concerning general refinements of centered hypers within the hexagon.

Lemma 12.4 If (centered) hypers Δ', Δ'' lie (non-strictly) within the hexagon, and $\Delta' \sqsubseteq \Delta''$, then the weight of any hexagon-vertex inner of Δ'' cannot exceed the weight of the same inner in Δ' .

Proof. Because refinement interpolates inners, and the hexagon vertices are not interpolations of any other points within the hexagon, a refinement cannot increase the weight of any vertex-inner there (although it can decrease it). \square

Corollary 12.5 There is no Δ satisfying (12.1).

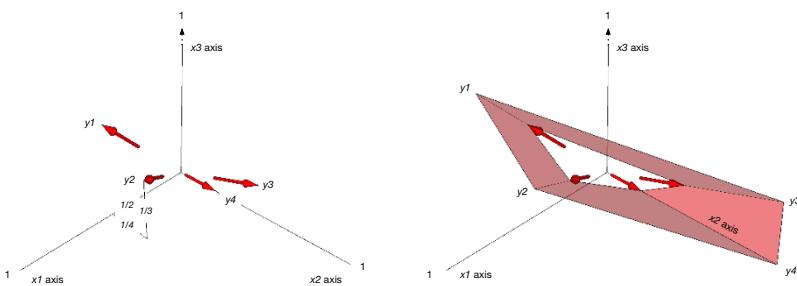
Proof. Any Δ that refines both $\Delta^{1,2}$ must lie within the hexagon; and by Lem. 12.4 if it is refined by all three of $\Delta^{3,4,5}$ then its weight at all six of the hexagon’s vertices must be at least $1/2$ for each — which is impossible (because its overall weight would then be 3). Thus there can be no such Δ . \square

And that gives us our counterexamples showing that hypers with (\sqsubseteq) are neither an upper nor a lower semi-lattice, thus perforce also not a lattice. For the join $\Delta^1 \sqcup \Delta^2$, if it existed, would satisfy (12.1) as Δ ; and the meet $\Delta^3 \sqcap \Delta^4 \sqcap \Delta^5$, if it existed, would satisfy (12.1) as well. But by Cor. 12.5, neither is possible.

It could be however that a quantitative information-flow domain *with* demonic choice, i.e. with probability, security and demonic choice all together, might give at least a lower semi-lattice for QIF with demonic choice. See the Notes to Chap. 17.

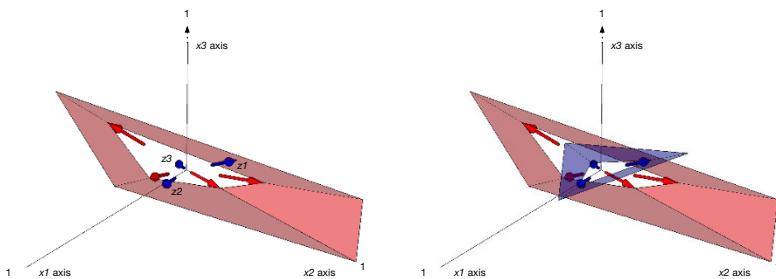
12.5 A geometric proof of antisymmetry of refinement

We now turn to another style of geometric argument: a proof of Thm. 9.14, that two different reduced matrices cannot refine each other. Suppose that $A: \mathcal{X} \rightarrow \mathcal{Y}$ and $B: \mathcal{X} \rightarrow \mathcal{Z}$ are two reduced matrices with $A \sqsubseteq B$, so that there is a stochastic matrix R such that $B = AR$. That means that each column z of B is a linear combination of the columns y of A , with column z of R (whose entries are hence between 0 and 1) giving the coefficients. Viewed geometrically, we see that the columns of B , as vectors, must lie within the conic hull of the columns of A , where the *conic hull* of some M is the set of vectors generated by $M \cdot v$ with v any vector having non-negative entries. (Note that v is not necessarily one-summing.)



(a) Each red arrow's tip is at the coordinates given by matrix A 's column (vector) of that label. Coordinates for y_2 are indicated in the diagram.

(b) A portion of the conic hull of A 's vectors is shown “wrapped around” the red arrows (extended), which as it happens are all extreme vectors of it.



(c) Because $A \sqsubseteq_{\circ} B$, the blue arrows generated by B 's vectors all lie within the conic hull generated by A 's.

(d) But the blue conic hull of B (only a portion shown) clearly does not contain the vectors generated by A .

In the first figure the four column vectors of matrix A are shown as red arrows. In the second, part of their conic hull appears, “wrapped around” those four arrows. In fact the hull extends from the origin to infinity; but here we show only the parts of it lying above the arrows and extending for a fixed distance.

In the third figure we add the three columns of B in blue, and they all lie within the conic hull of A , because $A \sqsubseteq_{\circ} B$. Thus for example z_1 is a combination of y_1 and y_3 , as column z_1 of R dictates, and in fact it lies on A 's conic hull — but it cannot lie outside it. Column z_3 of B lies however strictly within A 's conic hull, and is a combination of y_1 and y_4 .

In the fourth figure we see that indeed the four red arrows (columns of A) do not lie within the conic hull of B , as they would have to if $B \sqsubseteq_{\circ} A$. (In fact none of them do; but that is an accident of this example.)

The overall point is that if $A \sqsubseteq_{\circ} B \sqsubseteq_{\circ} A$ then the red and the blue hulls would have to lie inside each other, i.e. be equal — and in particular their extreme vectors would then be the same. Once those vectors are removed from both A and B , the process can be repeated until no vectors are left. Thus antisymmetry is established.

Figure 12.11 Illustration of conic hulls and antisymmetry

Example 12.6 Given

$$\begin{array}{|c|ccc|} \hline \mathbf{B} & z_1 & z_2 & z_3 \\ \hline x_1 & 1/4 & 1/2 & 1/4 \\ x_2 & 1/2 & 1/3 & 1/6 \\ x_3 & 5/12 & 1/3 & 1/4 \\ \hline \end{array} = \begin{array}{|c|cccc|} \hline \mathbf{A} & y_1 & y_2 & y_3 & y_4 \\ \hline x_1 & 1/2 & 1/2 & 0 & 0 \\ x_2 & 0 & 1/4 & 1/2 & 1/4 \\ x_3 & 1/2 & 1/3 & 1/6 & 0 \\ \hline \end{array} \cdot \begin{array}{|c|ccc|} \hline \mathbf{R} & z_1 & z_2 & z_3 \\ \hline y_1 & 1/2 & 0 & 1/2 \\ y_2 & 0 & 1 & 0 \\ y_3 & 1 & 0 & 0 \\ y_4 & 0 & 1/3 & 2/3 \\ \hline \end{array},$$

Figure 12.11 shows the columns of \mathbf{A} (in red), of \mathbf{B} (in blue), and their conic hulls. \square

Now if we *also* have $\mathbf{B} \sqsubseteq_{\circ} \mathbf{A}$, then the columns of \mathbf{A} must in turn lie within the conic hull of the columns of \mathbf{B} , so that the conic hulls must actually be equal. And therefore in particular the *extreme vectors* of those two (equal) conic hulls are equal — those vectors that (minimally) generate the hulls. But extreme vectors have the special property that they cannot be generated by linear combinations of others and so, since the matrices are reduced, the columns of \mathbf{A} and \mathbf{B} that led to each of those extreme vectors must be the same (i.e. not merely similar to each other). We can then peel away those extreme vectors, like the layers of an onion, and continue in the same way, showing eventually that \mathbf{A} and \mathbf{B} are equal as a whole.

12.6 Exercises

Exercise 12.1 Explain why the first action of a channel on a prior seems to reveal more (non-negative leakage), but subsequent multiplications (by refinement/post-processing matrices) loses information (the data-processing inequality [DPI](#) of §4.6.2). \square

12.7 Chapter notes

The relationship between geometric ideas and algebra is well known in mathematics generally. In probabilistic systems it has been particularly useful to provide an intuition between probabilistic logics and probabilistic relational models [1]. The proof that hyper-distributions do not form a lattice is just one example of how geometrical ideas can supply important results to the theory.

A non-geometric argument for antisymmetry was given in Thm. 9.14, in terms of matrices; an even more abstract and very direct proof, almost a “one liner” in terms of abstract channels, is given in [2, Thm 6].

Bibliography

- [1] McIver, A., Morgan, C.: Abstraction, Refinement and Proof for Probabilistic Systems. Monographs in Computer Science. Springer, Berlin (2005)
- [2] McIver, A., Morgan, C., Smith, G., Espinoza, B., Meinicke, L.: Abstract channels and their robust information-leakage ordering. In: M. Abadi, S. Kremer (eds.) Proceedings of the 3rd Conference on Principles of Security and Trust (POST 2014), *Lecture Notes in Computer Science*, vol. 8414, pp. 83–102. Springer, Berlin (2014)

Part IV

Information Leakage in Sequential Programs



Chapter 13

Quantitative information flow in sequential computer programs

Until this point, we have modeled systems as *channels*: either concretely as matrices of type $\mathcal{X} \rightarrow \mathcal{Y}$ for some input X and observation Y , or abstractly as functions $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$ on \mathcal{X} alone.¹ In this part of the book, we extend that, and other previously developed ideas, to systems represented as *programs* written in a probabilistic imperative language. As well as being more concise than channels as matrices, and closer to real implementations, programs can assign to the secret, leading us naturally to consider the possibility that the value of secrets can change over time — as for example when a user switches to a new password.

Just as earlier in §4.2 we assumed that an adversary knows how the channel works, here we will assume that the adversary knows the program source code.²

To make this generalization, we begin by focusing on two imperative-programming-language primitives: *markovs*, written as assignment statements that update the secret, and *channels*, written as “leak statements” using the keyword PRINT, that leak information about the secret.³

Now that we have markovs (as well as channels) we should point out that the state space (usually \mathcal{X}) is entirely hidden: both the initial state, although we know its distribution, and the final state, whose distribution we can nevertheless calculate from the initial state and the program code. The channels, as sequential-program fragments, do not update the state; instead, their effect is to leak information to the adversary, *about* the state, during the program’s execution. The markovs, on the other hand, do update the program state.

A more conventional setting, say with high- and low-security inputs and low-security outputs, would for us be realized by inserting a leak statement (PRINT) at the very beginning of the program that revealed the low-security projection of the initial state, and a second leak statement –at the end– that revealed the low-security projection of the final state. (We return to this in more detail at §13.4.4.) Using the syntax that will be introduced below, that would be something like this program:

¹ Recall that X is the name of the secret, whereas \mathcal{X} is the mathematical set from which its values (typically) x are drawn.

² This is again Kerckhoffs’ Principle.

³ We write the noun “markov” to distinguish our use here from the more general adjective “Markov”.

VAR	high	- High security variable (input).
	low	- Low security variable (input and output).
PRINT	low	- Reveal low initial state.
	"Existing program that sets...	
	... low's final value based on...	
	... the initial values of high and low."	
PRINT	low .	- Reveal low final state.

13.1 Markovs don't leak; and channels don't update

Our first primitive imperative construct is the “markov”, by which we mean a simple, but possibly probabilistic assignment statement; our second is the “channel”, which is just what we have studied intensively already but now reclothe as a program fragment. Later, and more generally, by “program” we will mean constructions built from those markovs and channels, using sequential composition, conditionals and loops; and our aim is to be able to reason about them all in a common framework. We will now introduce those primitives.

However our aim is not somehow to “glue” the primitives, the markovs and channels together — we go in precisely the other direction. Instead we take “Hidden Markov Models”, or *HMM*'s, whose details are the subject of Chap. 14, as our primitive building blocks, and we show that they specialize to channels and to markovs. That is, both channels and markovs will be special cases of something more general that exists already: a channel is an *HMM* that declines to update but can leak; and complementarily a markov is an *HMM* that declines to leak, but can update. The *extant* constructors for combining *HMM*'s generally then work together, automatically, to build programs that can do both, from the markovs and channels that are their special cases.

But we defer general *HMM*'s, for now, and in this section consider only the primitives: channels and markovs.

The first issue we encounter when treating channels and markovs together is that they appear to be of different types: whereas a channel takes its inputs in the state-space \mathcal{X} to its outputs in (some) observables \mathcal{Y} , it would seem that a markov should take its initial states (again in \mathcal{X}) to its final states (in the same \mathcal{X}). Thus although we can identify channels' inputs and markovs' initial states —they are both \mathcal{X} — we cannot identify channels' outputs and markovs' final states. Not only are \mathcal{X} and \mathcal{Y} possibly different, we can within the same system find channels that, while over the same input \mathcal{X} , have varying output types $\mathcal{Y}^1, \mathcal{Y}^2$ etc. Luckily however that varying-output problem has already been solved: we have recognized that we are not interested in the channels' outputs *per se*; rather, by abstracting from those outputs, we concentrate only on what they tell us about the channels' inputs — that is, we use the *abstract channels* introduced in Chap. 4.4, whose type is $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$ no matter what their individual \mathcal{Y} 's might have been when they were matrices. All the potential $\mathcal{Y}^1, \mathcal{Y}^2, \dots$ are swept under the carpet of that abstraction.

Having eliminated the \mathcal{Y} 's from channels, we now turn to how we “add a second \mathbb{D} ” to markovs, so that we can see them also as of type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$.

The simplest kind of markov is one that in fact is not (even) probabilistic: it is just an assignment statement over a state space \mathcal{X} , thus a function of type $\mathcal{X} \rightarrow \mathcal{X}$ from initial to final state. And it is readily generalized: one popular instance is the nondeterministic or “demonic” assignment statements, easily modeled as functions $\mathcal{X} \rightarrow \mathbb{P}\mathcal{X}$ that take initial states to *sets* of possible output states, with the degenerate, deterministic programs being the special case where every output-state set is singleton. We don't consider demonic programs any further here; but they suggest that, by analogy, we could define markovs to be functions $\mathcal{X} \rightarrow \mathbb{D}\mathcal{X}$ that take initial states to probability distributions over final states, with the degenerate case of “non-probabilistic markovs” now being those that produce only point distributions. For example, a (proper) markov that either increments or decrements x uniformly would be modeled as the function that takes x of type \mathcal{X} to the distribution $x+1_{1/2} \oplus x-1$ of type $\mathbb{D}\mathcal{X}$. The degenerate markov that only increments takes x to the point distribution on $x+1$; even though it is in fact a probability-free assignment statement, here it has type $\mathcal{X} \rightarrow \mathbb{D}\mathcal{X}$ so that it can be combined with other markovs that *are* properly probabilistic.

The classical mathematical structure corresponding to a markov is of course a Markov matrix, whose rows are labeled with input states and columns with output states: the real number in row x and column x' of the matrix gives the probability that from input state x the markov will move to output state x' ; and each row sums to 1, so that the whole matrix is called *stochastic*. And the special case of standard (non-probabilistic, “ordinary”) markovs is the matrices that have exactly one 1 in each row, and 0's elsewhere. The connection of Markov matrices with the function model above $\mathcal{X} \rightarrow \mathbb{D}\mathcal{X}$ is then that for a given x the function produces the distribution enumerated by row x , in which for each potential x' you will find a probability: that is, each row of a Markov matrix is a (possibly different) probability distribution of type $\mathbb{D}\mathcal{X}$; and the function that the Markov matrix describes takes each row-label to the (distribution described by) the row it labels. The special degenerate case of non-probabilistic programs is those whose output distributions are *point* distributions, each assigning probability 1 to a single x' and probability 0 to all the others.

However another, and perhaps more common view of Markov matrices is as functions of type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}\mathcal{X}$, where the argument is a row vector whose columns (i.e. elements) are x -labeled, giving not an input state alone but rather a distribution over input states. Matrix multiplication, with the row vector on the left, then gives (another) row vector whose columns are x' -labeled, and it is the distribution on final states that results from applying that Markov matrix to the initial-state distribution. In this way we can see a markov (i.e. the computation that a Markov matrix describes) as having type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}\mathcal{X}$ — it takes an input-state distribution π to some output-state distribution π' . We are almost there...

But not quite: we want a markov to have type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$, i.e. to be a function from distributions on the input state to *hypers* on the output state. Since a markov leaks no information, for inspiration we could ask “What does an abstract channel look like that leaks nothing?” That “trivial channel” $\mathbf{1}$ takes its prior π to the hyper $[\pi]$, and with $[-]$ makes a point distribution on its argument, that is the *point* hyper whose inner is the very prior that the channel started with: after applying the trivial channel to a prior π you know for sure, i.e. with probability 1 (its outer is the point distribution) that the posterior is... just the prior again, the π you started with.⁴ The channel leaks nothing; and an adversary learns nothing from it. Thus a markov, which is trivial from the *leaking* point of view, could be seen as of type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$ by

⁴ When written as a channel *matrix*, the identity channel has all columns similar to each other: a single column of all 1's is the canonical example. In particular, the identity channel is not the identity matrix (which matrix in fact represents the “leak everything” zero channel \mathbf{O}).

taking a prior distribution π on its input state not to final distribution π' but rather to the *point* hyper $[\pi']$ on that π' . Its “markovian” aspect is that the resulting point hyper is not necessarily $[\pi]$, the prior again: the markov might have updated the state, giving instead $[\pi']$ for some $\pi' \neq \pi$. But it is still a *point* hyper.⁵

Thus now we have the agreement we need: both channels and markovs have type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$, but as special cases:

a markov takes prior π to hyper $[\pi']$, i.e. the point hyper on the distribution π' , where π' itself is the effect on π of the Markov matrix describing the markov, i.e. the update it carries out. Our slogan is

markovs don't leak; but if π' is different from π , then the markov has carried out an update;

and **a channel** takes prior π to hyper Δ such that the weighted average of Δ 's inners is π again. This time it's

channels don't update; but if Δ is non-singleton, then the channel has leaked some information.

Note that a joint specialization is the statement **SKIP**, the identity of sequential composition, and both a markov and a channel: it's equal as a program fragment to the trivial update (which we can write $x := x$) and the trivial leak say of a constant value (which we can write **PRINT K** for some constant K).

We have now shown that markovs and channels are, semantically, each special cases of more general structures. Before exploiting that (§13.4 below), we review specifications, implementations and refinement: they too must work for markovs and channels together.

13.2 Specifications and implementations: a review

13.2.1 When is one program better than another, and why?

Comparing sequential programs, i.e. saying that one is better than another, has a long history, focusing originally on programs' *functional* behavior: that is, what final outputs they produce from what initial inputs. Specification of functional behavior is usually written as a triple

$$\{\text{precondition}\} \text{ program } \{\text{postcondition}\} , \quad (13.2)$$

meaning that provided the program's initial state satisfies the precondition it is guaranteed to terminate in a state satisfying the postcondition. This insight led later to the idea of one program's being *refined* by another, meaning in the classical sense precisely that any precondition/postcondition pair (13.2) satisfied by the less-refined program would be satisfied by the more-refined one as well. Calling the less-refined program a specification and the more-refined program an implementation, one wrote *specification* \sqsubseteq *implementation* for “specification is refined by implementation”. That binary relation (\sqsubseteq) is made precise by the definition⁶

⁵ In the trivial, non-leaking channel case both the prior and (unique) posterior are the same. In the markov case the prior and its unique posterior are different, but related by the underlying Markov matrix. Thus in the channel case the “implicit” matrix is the identity.

⁶ If defined with sufficient generality, it induces a lattice structure on programs.

Refinement of sequential programs

$\text{specification} \sqsubseteq \text{implementation}$

$$\begin{array}{lllll} \text{means} & \text{if} & \{ \text{precondition} \} & \text{specification} & \{ \text{postcondition} \} \\ & \text{then} & \{ \text{precondition} \} & \text{implementation} & \{ \text{postcondition} \} \end{array} \quad (13.3)$$

for all *precondition* and *postcondition*.

This (13.3) is the genesis of “comparing programs” — and it is in agreement with all of the subsequent treatments of stepwise development of sequential programs, continuing through the many related specification/development methods that followed and, indeed, some that came before. A customer expresses his satisfaction with a specification in the language of triples (or two-place before/after predicates) and then the refinement relation guarantees that in the same terms he will be happy with any implementation of that specification.

This last point is crucial: if the customer can express his requirements in terms of pre- and postconditions, and he is satisfied that the specification meets those requirements, then he does not have to check the implementation at all: he does not even have to see it. If the implementation refines the specification, then his satisfaction is guaranteed. That is exactly what the refinement relation is for.⁷

13.2.2 When is one *channel* better than another, and why?

We have already studied extensively the ways in which channel-induced information flow can be measured. And if one settles on a particular style of measurement, say Shannon entropy for example, one can compare two channels’ leakage with respect to that. What is more interesting however is that, as we have shown, it is possible to relate channels in a way that ensures that one leaks less⁸ than another with respect to *all* (reasonable) styles of measurement. It is surprising (at least at first) that such generality is possible: but that generality is precisely the significance of our refinement order between channels:

Refinement of channels

$\text{specification} \sqsubseteq \text{implementation}$

$$\begin{array}{llll} \text{means} & \text{if} & (g\text{-leakage wrt. prior } \pi \text{ of specification}) & \leq t \\ & \text{then} & (g\text{-leakage wrt. prior } \pi \text{ of implementation}) & \leq t \end{array} \quad (13.4)$$

for all gain functions g , priors π and reals t (for “tolerance”).

And here we have the analog of the program case: if a customer is satisfied that the specification channel does not leak too much, where the definition of “too much” is his to make — which gain function g , which only-just-tolerable threshold t — then he does not have to check the implementation channel. Any complicated calculations of how many bits are actually leaked (e.g. for Shannon-based leakage measures) can be done on the specification alone. (See §15.8.) He does not have to do calculations

⁷ If he cannot express his requirements as pre- and postconditions, then he needs a more expressive language of specifications — for which he pays in increased reasoning complexity. It’s worth choosing the least-expressive formalism that suffices — if one has a choice.

⁸ By “less” is meant more exactly “no more than”.

on the implementation channel, and in particular he does not have to know how much it leaks according to the gain function he chose to express his requirements. Again, his satisfaction is guaranteed automatically by the refinement relation: the implementation leaks no more than the specification, and since he was happy with the former (because it was no more than t) he doesn't care about the actual value of the latter (because its performance is not more than t either).

13.2.3 Programs and channels together: what is “better” for both?

Given the similarity of (13.3) and (13.4), it's no coincidence that we use the same symbol \sqsubseteq above for the notion “ $program_1$ is refined by $program_2$ ” as we used in earlier sections where it meant “ $channel_1$ is refined by $channel_2$ ”. Refinement for programs means “is more reliable and predictable”⁹ and refinement for channels means “is more secure”.¹⁰ If a program contains both markovs and channels, what then is the refinement relation for programs that results in refinement of its markovs *and* its channels? That is how we will get “satisfaction guaranteed” for markovs and channels, i.e. for functional and information-flow behavior at the same time.

That is, when we take programs and channels together, here meaning for markovs and channels, the joint refinement order (\sqsubseteq) must make sense simultaneously in both respects, i.e. indicating better behavior with respect to both programmed updates *and* information leaks. That is, “is refined by” will mean both “requires only a weaker precondition and/or ensures a stronger postcondition” (functional refinement) and “leaks no more than” (information-flow refinement). To achieve this, we make two final adjustments as described in the next section. One is to generalize Hoare triples; and the other is to invert our gain functions.

And our reward is based on the importance of monotonicity: if we program-refine a markov within a big system, the system as a whole should be program-refined; and if we *QIF*-refine a channel within a big system, the system as a whole should be *QIF*-refined.

13.3 Aligning functional refinement with information-flow refinement

13.3.1 Generalizing Hoare logic for probability

The first adjustment we make concerns Hoare triples.

Probabilistic programs generalize *standard* programs, i.e. conventional sequential programs like those treated by Hoare logic, by introducing an extra operator $p \boxplus$ that chooses between its operands with probability p .¹¹ But it is, or at least was not obvious how Hoare triples should correspondingly be generalized.

⁹ The accepted technical meaning for those terms in sequential program refinement is that *reliable* means “is guaranteed to terminate” and *predictable* means “can have its possible outputs determined in advance”. Thus $r := \pm \sqrt{s}$ is refined by $r := +\sqrt{|s|}$ in both senses: the latter terminates even when s is negative, and for any s the possible assignments to r for the latter are a subset of those for the former.

¹⁰ Again by “more” we intend also “is equal”.

¹¹ An interesting point, but not essential here, is that $p \boxplus$ generalizes the standard conditional IF in a very direct way. Since p need not be a constant, but can be a $[0, 1]$ -valued expression P in the state variables, the standard IF b THEN `thenBranch` ELSE `elseBranch` is simply `thenbranch` $p \boxplus$ `elseBranch` provided the expression P has value 1 when b holds and value 0 when it does not. See §14.4.6.

Although an appealing proposal for a probabilistically extended interpretation of the Hoare triple

$$\{ \text{true} \} \quad \text{coin} := \text{Heads } \frac{1}{2} \boxplus \text{coin} := \text{Tails} \quad \{\text{coin} = \text{Heads}\} \quad (13.5)$$

might then be that “The triple is valid with probability $1/2$,” in fact such “probabilistic Hoare triples” turn out *not* to be the best choice for us, and it’s not simply a matter of taste or opinion: if abstraction (or equivalently demonic choice) is introduced into the programming language as well, say where it is possible to write “Assign to x some value between 0 and 9.” then it’s a mathematical fact that probabilistic Hoare triples in the style of (13.5) are not compositional. Although we do not treat such abstractions here, we want to avoid taking a conceptual step that would preclude them. (See the Chapter Notes, and Chap. 17 for a treatment of demonic instead of probabilistic information flow.)

What does suit us is a generalization made by Kozen. Hoare’s logical formulae, equivalently Boolean-valued expressions over the state variables, become non-negative real-valued expressions instead. Here we’ll call them pre- and post-*expectations*. Since the program itself is now a function from initial state to a *distribution* of final states, i.e. of type $\mathcal{X} \rightarrow \mathbb{D}\mathcal{X}$ (whereas standard deterministic programs are of type $\mathcal{X} \rightarrow \mathcal{X}$), it is type-correct to say that the judgment

$$\{ \text{pre-exp} \} \quad \text{prog} \quad \{ \text{post-exp} \}$$

means that the average, i.e. expected value of *post-exp* over the distribution of final states is at least the actual value of *pre-exp* in the initial state from which *prog* started execution. This beautifully generalizes standard Hoare logic, for if we silently read Booleans as reals –taking *false* to 0 and *true* to 1, and interpreting implies as (\leq) – then we have the “Kozen triple”

$$\{ \frac{1}{2} \} \quad \text{coin} := \text{Heads } \frac{1}{2} \boxplus \text{coin} := \text{Tails} \quad \{\text{coin} = \text{Heads}\}$$

for the coin-flipping program just above; it can be read as saying that, regardless of the prior distribution on *coin*, the program results in *coin*’s being *Heads* with probability at least $1/2$ (and of course the same holds for *Tails*). This approach in the long run works better than the “obvious” proposal mentioned earlier;¹² in fact it is strictly more general.

13.3.2 Using loss functions

The second adjustment concerns gain functions.

The gain functions whose use for channels we have discussed so thoroughly in earlier sections measure “gain” for the *adversary* — thus their increase indicates a channel’s becoming more attractive to the adversary, *less* secure from our point of view. But in the Kozen-inspired generalization of Hoare logic, as above, pre-expectations measure a program’s utility for *us*, the defenders — that is, their increase indicates a program’s becoming *more* useful from our point of view. The increases have exactly opposing significance: better for one, worse for the other — and so we must flip one of them. That is, we would like to include *both* predicates about functional properties and about leakage in our pre- and post-expectations. But if we use vulnerability we have a mismatch, since greater expectations for functional properties makes the program

¹² In a nutshell, rather than convert the Hoare triples as a whole to numbers, we converted the formulae inside them.

better, while greater vulnerability makes it *worse*. Therefore we need to use uncertainty, i.e. loss functions instead.

Thus we flip gain functions: from here on, at least when dealing with programs and channels together, we will use *loss* functions rather than gain functions. In any case, gain- and loss functions are duals. They are equally expressive, and the main technical change is only that instead of using max as in Def. 3.2 we use min as in Def. 3.4.¹³

13.3.3 Refinement in general

From now on, programs S (for specification) and I (for implementation) will not just be probabilistic updates (markovs), or information flows (channels), but potentially both together. We will use the common and general convention –from program semantics– that for two programs S, I we induce a refinement relation $S \sqsubseteq I$ between programs just when for all priors π we have $S(\pi) \sqsubseteq I(\pi)$, where $S(\pi)$, resp. $I(\pi)$ are the hypers produced by running S, I resp. from that prior π , and (\sqsubseteq) is between hypers.

Definition 13.1 (Refinement order between QIF programs generally) The type of *QIF* programs' semantics is

functions from distributions on a state-space \mathcal{X} to hyper-distributions on the same state-space, that is $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$.

The refinement order on those programs is induced, “lifted” from the refinement order (existing) on the codomain, that is the refinement order on hypers $\mathbb{D}^2\mathcal{X}$ alone. Thus for two programs S (for specification) and I (for implementation) we define that

S is refined by I , written $S \sqsubseteq I$, just when for all priors π in $\mathbb{D}\mathcal{X}$ we have that $S(\pi) \sqsubseteq I(\pi)$,

where the occurrence of (\sqsubseteq) on the right is refinement between hypers. □

Refinement of hypers Def. 13.1 just above depends on the definition of refinement between hypers; and by now we have several (equivalent) formulations of that. They include

defined via concrete channels Any two hypers Δ, Δ' can be constructed from prior/channel combinations $[\pi \triangleright C]$ and $[\pi' \triangleright C']$ respectively (proof of Thm. 4.16). We have $\Delta \sqsubseteq \Delta'$ just when $C \sqsubseteq_0 C'$ which, incidentally, implies $\pi = \pi'$ (Def. 9.5).¹⁴

defined via abstract channels Abstract channels are just the case of *QIF* programs restricted to pure channels, i.e. without markovs.

defined via geometry In Chap. 12, hypers are given a geometric representation as sets of weighted disks resting in a barycentric representation of $\mathbb{D}\mathcal{X}$, and refinement is a procedure of splitting and then merging those disks (Def. 12.1).

¹³ See §3.2.2 for an example where loss functions are the intuitive choice even when you're not considering programs.

¹⁴ Returning briefly to the world of concrete channels, we are now saying that our earlier judgment that *channels* S and I satisfy $S \sqsubseteq I$, i.e. relating two stochastic matrices, is instead to be thought of as that for all π we have $(\pi \triangleright S) \sqsubseteq (\pi \triangleright I)$, in which we are relating two joint distributions (the concrete counterpart of hypers). From the definition of *[prior > channel matrix]*, we see however that those two definitions are equivalent: it makes no difference to the refinement-mediated merging of matrix columns whether prior π is “pushed in” beforehand or afterwards.

defined via gain- or loss functions Here one hyper Δ is defined to be refined by another Δ' iff

- for all gain functions g we have $V_g(\Delta) \geq V_g(\Delta')$, or equivalently
- for all loss functions ℓ we have $U_\ell(\Delta) \leq U_\ell(\Delta')$.

(Recall Def. 9.10.)

defined via the \mathbb{D} monad There is a more abstract “hyper native” definition of refinement between hypers (§9.11 and §14.7(10)), but we won’t need it here. It is however the one most useful for investigating the structure of the relation itself, e.g. whether limits of chains exist and how hypers and their refinement order can be generalized to proper measures.

13.3.4 Initial-final correlations, and Dalenius

Although we have so far (and just above) described the effect of *QIF* programs to be the construction of final hyper-distributions from initial distributions (and we will continue to do so), there are important situations in which that is not enough, because in an *HMM* each markov step actually creates a *correlation between* its initial and final states, not only a distribution on final states alone. Indeed, by concentrating our analysis on the uncertainty wrt. the final state only, we have obtained a relatively simple model — but its utility is based on our assumption that the program variables we have declared are the only secrets we care about. For example, if the adversary is trying to obtain our password, it’s our *current* password that we protect: yesterday’s password, we might argue, is no risk to us — because it won’t work today.

One situation in which the above view is too simple is of course where we change our password in some systematic way. We shouldn’t, but still sometimes we do.¹⁵ And in that case, an adversary with knowledge of our last two passwords might be able to gain some knowledge about what our current one could be. To capture that in our *QIF* programming semantics, we would have to model somehow in our programming code the procedure a user follows to generate a new password: but that can’t be deterministic, because our attacker model mandates that the adversary can see our program code; and similarly it can’t be an external probabilistic choice, because she would see that too. One approach however might be to model our “irresponsible update” as an *internal* probabilistic choice that is not uniform, but rather is biased in a way that depends on the password’s current value. But we don’t pursue that issue further here. (See however Exercise 13.2.)

The other situation in which our view is too simple is that there might be *other* variables that are neither read nor written by our program — and so are not declared at all in our program source code— but which nevertheless have a correlation with our own program’s variables, at least initially. That however does not affect our own security analysis, because whatever correlation there might be with other variables is sufficiently captured —for us, at least— by our assumption that we make about our own variables’ prior.

But correlations like the above do cause a security risk for those *other* variables, i.e. for other people — and that is in fact precisely the Dalenius issue which we discussed in Chap. 10. With the example (13.6) below we show that in its starkest form.

¹⁵ That is one reason that forced changes of passwords are a bad idea, and can be argued to decrease security rather than increase it. Once-responsible users are forced into “algorithmic” updates because they are sick of thinking of new passwords every three months.

Suppose that x is our sole program variable, for our own secret, and note that we have a valid program refinement, actually an equality between two programs, with

$$\text{PRINT } x; \quad x \in X \quad = \quad x \in X \quad (13.6)$$

where we have written “ $x \in X$ ” for “Choose x uniformly from its type X .” and `PRINT x` was introduced in Prog. (13.1). In each case the distribution of the final value of x is exactly the same: uniform. Such an analysis is independent of the information flow wrt. the initial value of x , which is why it leads to a simpler model.

However there might be an adversary whose target is “somebody else’s” variable z and who knows, furthermore, that $x = z$ just at the moment before either of these two programs is run. Then clearly the left-hand program reveals z while the right-hand program does not. Recall however that z does not appear in our program at all, and that our security is unaffected; but whoever is in charge of the security of z will probably be disappointed.

And indeed the above is a very unusual and surprising situation from the point of view of any reasonable program algebra: it means that two programs that have the same effect can be made to differ simply by adding a declaration `VAR z` of a fresh variable that neither program refers to, that probably neither of the programs’ developers has even heard of.

We do not feel however that this is a situation we have created for ourselves, i.e. that it is a side-effect of our formalization, because it is exactly the *same* surprise that Dwork suggested Alan Turing might feel if he learned that information about his height had been leaked by an adversary’s queries concerning Lithuanian women (§10.4). What has happened here is that our close and fundamental look at *QIF* semantics has revealed how *Dalenius leakage* applies to programming generally, i.e. not only to statistical databases.

One approach to the above is to incorporate what we have learned about the Dalenius perspective described in Chap. 10, and to treat the markov part of an *HMM* in terms of a correlation between initial and final states, rather than concentrating on final states alone as we do now. That leads to a slightly more elaborate program semantics of type $\mathbb{D}\mathcal{X}^2 \rightarrow \mathbb{D}^2\mathcal{X}^2$, where programs take initial, prior *correlations* to final posterior correlations. (See the Chapter Notes.)

Finally, we point out that the Dalenius-aware semantics, its slight “complification”, is not necessary in situations where the initial variables are only read, never overwritten — which is very often the case in the standard *QIF* examples, where we are concerned about leaks of the initial state only, and do not update it. (The program might well introduce and update other local program variables, but —being local— their initial states are irrelevant. There are many examples of that in this and the following chapter, and in for example the fast-exponentiation program of our case study Fig. 20.2 to come in which the initial E , the secret, is copied and itself never updated.) In cases where the initial variables *are* overwritten, it is sufficient to declare (but not refer to) an unused extra variable of the same type. Our existing and simpler $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$ semantics then is robust against Dalenius scenarios as well.

13.4 Larger information-flow-aware programs

We have now decided to use the type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$ for both channels and markovs, and hence also for the general programs we shall make from them: they will be built using *sequential composition*, *conditional* and *iteration*.

13.4.1 Sequential composition

One startling advantage of our common type is that sequential composition is already defined: there is well established theory in Computer Science that tells us how to compose two “computations” of type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$ to yield a larger computation of the same type (in the style of Moggi). It is “Kleisli” composition, to which we return in the next chapter (§14.3.1). It will tell us, for example, what two channels in succession do and what two markovs in succession do: but in fact for those “homogeneous” sequential compositions we already know the answer from mathematics. And so for those cases we will be checking that our generalization gives us that same answer — it’s an objective criterion, either the same answer or not. And it’s a “sanity check”.

But the generalization *also* will determine what a channel and then a markov should do, and what a markov then a channel should do, and here our criteria will be subjective: “Is this reasonable?” is the best we can do, since there is no extant generally agreed-upon definition of how those components interact.

And finally we will have to design from scratch the effect of conditionals and iterations: for that we will perform some “*gedanken* experiments”.

We begin with some general remarks. Programs have type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$, but now the interpretation is that the input represents the adversary’s prior knowledge about the *initial* value of \mathbf{x} , while the output represents her posterior knowledge about the *final* value of \mathbf{x} . Note that this is consistent with our previous understanding of abstract channels, since there the initial and final values of \mathbf{x} are equal. In the case of a markov, which naturally has type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}\mathcal{X}$, we need only convert the final distribution π' to the point hyper $[\pi']$.

Now given two programs P and P' of that type, how then should we sequentially compose them? There is an obvious answer. If $P(\pi) = \sum_i a_i[\delta^i]$, that is can be written as a linear combination of point hypers $[\delta^i]$,¹⁶ then we should “run” the subsequent P' on *each* of the inners δ^i that P produced, and then combine the resulting hypers using the respective outer probabilities a_i to obtain

$$(P; P')(\pi) = \sum_i a_i P'(\delta^i) .$$

That is in fact an example of the standard construction known as Kleisli composition.

With that preparation, we continue with the homogeneous compositions: markov-then-markov and channel-then-channel. In the examples below we will have M (possibly decorated) as a markov acting as a program fragment, and C will be a channel acting as a program fragment, and they will be *described* in conventional terms as matrices called C and M . That is, program M will “act like” M , and C like C similarly. Our state-space is $\mathcal{X}=\{\text{heads}, \text{tails}\}$, our (single) observation space is $\mathcal{Y}=\{\text{true}, \text{false}\}$; and our matrices are both of size 2×2 , as follows.

¹⁶ Here each δ^i is an inner, and a_i is its corresponding output probability.

$$\begin{array}{ccc}
 & \mathcal{X} & \mathcal{Y} \\
 & \downarrow \text{heads} & \downarrow \text{true} \\
 & \downarrow \text{tails} & \downarrow \text{false} \\
 \mathbf{M}: \quad \mathcal{X} \left\{ \begin{array}{c} \text{heads} \\ \text{tails} \end{array} \right. & \left(\begin{array}{cc} 1 & 0 \\ 1/2 & 1/2 \end{array} \right) & \subset: \quad \mathcal{X} \left\{ \begin{array}{c} \text{heads} \\ \text{tails} \end{array} \right. & \left(\begin{array}{cc} 1 & 0 \\ 1/2 & 1/2 \end{array} \right) & (13.7)
 \end{array}$$

Note that (here) \mathbf{M} and \mathbf{C} are simply names standing for matrices: writing the actual matrices in place would not alter the meaning of what we are about to say. (It merely complicates the typesetting.) Programming-language syntax for those matrices will be introduced in the next section (§13.5).

And note especially that the contents of the two example matrices are *the same*, numerically: that is done deliberately, so that we will see how they behave differently when used for two different purposes. But the column types differ: the (final state) \mathcal{X} of \mathbf{M} is $\{\text{heads}, \text{tails}\}$, and \mathbf{M} will be used to construct a program-fragment markov M ; the \mathcal{Y} of \mathbf{C} is $\{\text{true}, \text{false}\}$, and the matrix \mathbf{C} will be used to construct a program-fragment channel C .

Now we consider the four possible cases of sequential composition.

(1) The first (homogeneous) case is **two markovs** in succession.

In general, if we have two markovs $M^{1,2}$ (thus of type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$) representing matrices $\mathbf{M}^{1,2}$, their sequential composition $M^1; M^2$ must be the same as the single markov corresponding to the multiplication $M^1 \cdot M^2$ of the corresponding Markov matrices from which they came: either way, the state is updated (probabilistically) according to the Markov transition that M^1 represents; and then it is updated, again, according to M^2 . This is the objective criterion referred to above.

As explained in §14.2.2 to come, the (Kleisli-) sequential composition for $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$ does make that happen.

As an example, we let our $M^{1,2}$ in the composition as above be markovs representing the same Markov matrix \mathbf{M} from (13.7): as a temporary measure we'll write “update according to \mathbf{M} ” for them. (More realistic syntax is given in §13.5 below; we do not use it here because we do not want to give the impression that the embeddings we discuss are somehow dependent on the way we write the channels or the markovs concretely.)

The markov that is “update according to \mathbf{M} ” has of course type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$ but, recall, the matrix \mathbf{M} it represents is itself just four real numbers and so it is “update according to \dots ” that converts that matrix into a markov, namely the mapping $\pi \mapsto [\pi\mathbf{M}]$, where $\pi\mathbf{M}$ is the vector-matrix product. Our criterion here is that, as an example of reasoning about programs, we must have

“update according to \mathbf{M} ”; “update according to \mathbf{M} ” = “update according to \mathbf{N} ”

where \mathbf{N} is the matrix \mathbf{MM} , that is the matrix \mathbf{M} *multiplied by itself* in the usual composition-of-Markov-transitions style from mathematics:

$$\mathbf{N} = \mathbf{MM}: \quad \mathcal{X} \left\{ \begin{array}{c} \text{heads} \\ \text{tails} \end{array} \right. \left(\begin{array}{cc} 1 & 0 \\ 3/4 & 1/4 \end{array} \right) \quad \begin{array}{c} \mathcal{X} \\ \downarrow \text{heads} \\ \downarrow \text{tails} \end{array} \quad (13.8)$$

That is, it must not matter whether we first convert M to a markov and then compose it sequentially with itself, or first compose M with itself using matrix multiplication and then convert that product matrix to a markov: we must get the same either way.

For M ; M , on a uniform prior $\pi := (1/2, 1/2)$ the distribution on x after the first run of M is given by a vector-matrix product, converted to a point hyper: $[(1/2, 1/2) \cdot M] = [(3/4, 1/4)]$.¹⁷ Then the distribution on x after the second run of M is given by $[(3/4, 1/4) \cdot M] = [(7/8, 1/8)]$.

We can confirm that this is the same result that we get for MM.

- (2) The second homogeneous case is **two channels**.

If we have two channels $C^{1,2}$ their sequential composition $C^1; C^2$ must be the single channel $C^1 \parallel C^2$ corresponding to C^1 and C^2 's being independently run in parallel with each other.¹⁸ This surprising commutativity (of sequential composition of channels) is because, firstly, channels do not change the state (and so it does not matter “which one does not change the state first”) and, secondly, if they are running independently it does not matter in which order the adversary receives the information: at the end she has both channels’ leaks, and the state has not changed.

Again §14.2.2 shows that the sequential composition for $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$ makes that happen, as we now illustrate.

Let both $C^{1,2}$ in $C^1; C^2$ be the program-fragment channel representing channel-matrix C from above: we write “leak according to C ” for that, namely the mapping $\pi \mapsto [\pi \triangleright C]$, where we recall from Def. 4.5 that $[-]$ here is converting the joint distribution $\pi \triangleright C$ of type $\mathbb{D}(\mathcal{X} \times \mathcal{Y})$ into a hyper-distribution (of type $\mathbb{D}^2 \mathcal{X}$).

This time our example is that

“leak according to C”; “leak according to C” = “leak according to D”

where D is the matrix representing the *parallel composition* of (the channel described by) C with itself: we have

$$\mathsf{D} = \mathsf{C} \parallel \mathsf{C} : \quad \mathcal{X} \quad \left\{ \begin{array}{ll} \text{heads} & \left(\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 1/4 & 1/4 & 1/4 & 1/4 \end{array} \right) \\ \text{tails} & \end{array} \right.$$

But since the last three columns of D are the same, we can simplify it (i.e. *reduce* it via Def. 4.9) by adding those columns together; we get

$$\mathsf{E} : \quad \mathcal{X} \left\{ \begin{array}{c} \textit{heads} \\ \textit{tails} \end{array} \right. \quad \left(\begin{array}{cc} 1 & 0 \\ 1/4 & 3/4 \end{array} \right) \quad , \quad (13.9)$$

¹⁷ In a matrix or vector product where one or both operands are constants, we write an explicit multiplication (\cdot).

¹⁸ Parallel composition for channel matrices was defined at Def. 8.1 in §8.1.1 (p. 132), and for abstract channels at Def. 8.8 in §8.2.2 (p. 139).

where as well as now having only two columns we have dropped the column labels: for channels-as-leakage, the labels are not necessary anyway; that is, as abstract channels D and E are the same. Thus in fact we have

$$\begin{aligned} & \text{"leak according to } C\text{"}; \text{"leak according to } C\text{"} \\ = & \quad \text{"leak according to } D\text{"} \\ = & \quad \text{"leak according to } E\text{"} \end{aligned} \quad (13.10)$$

Done with hypers, the case of $C; C$ on a uniform prior gives first

$$[(1/2, 1/2) \triangleright C] = 3/4 \cdot [(2/3, 1/3)] + 1/4 \cdot [(0, 1)] .$$

Then running C on each of the inners after the first run gives

$$\begin{aligned} [(2/3, 1/3) \triangleright C] &= 5/6 \cdot [(4/5, 1/5)] + 1/6 \cdot [(0, 1)] \\ [(0, 1) \triangleright C] &= [(0, 1)] . \end{aligned}$$

And finally we can combine those two hypers, weighted by the outer probabilities from the first run of C , which gives overall $5/8 \cdot [(4/5, 1/5)] + 3/8 \cdot [(0, 1)]$.

We can confirm that this is the same result that we get for the channel E , whose (reduced) matrix was given at (13.9).

Summarizing the homogeneous cases (1,2), we can compare (13.8) and (13.9) to see that even though M and C are the same as matrices, i.e. contain the same numbers, the result of their self-sequential-composition differs: it depends on whether they are interpreted as channels or as markovs. But the *same* definition of sequential composition is used in each case. The difference therefore must come from the way “update according to –” and “leak according to –” convert their arguments into program components of type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$.

An additional benefit of the example is that the right-hand equality in (13.10) emphasizes that the interpretation of a matrix into the programming language depends only on its abstract-channel semantics. That is, if D and E denote the same abstract channel –equivalently, have the same reduced matrix– then the programs “leak according to D ” and “leak according to E ” are the same as well.

Having channels and programs together in the same framework now allows us to consider more heterogeneous cases: channels and markovs together.

(3) The first heterogeneous case is **channel then markov**, that is

$$\text{"leak according to } C\text{"}; \text{"update according to } M\text{"} ,$$

which is a leak followed by an update as might happen if someone’s password is (partially) leaked via C , and then (e.g. as a countermeasure) he updates it according to a probabilistic algorithm M .¹⁹ To understand that, we begin by reviewing the effect of “leak according to C ” on its own; then we discuss the effect of that leak on a subsequent “update according to M ”.

Leaking according to C The adversary is assumed to know the initial distribution of the states; that is as we know called the *prior* distribution, or the *prior*. When “leak according to C ” is executed she

¹⁹ Services that insist you change your password regularly can have this effect: “Oh, I’ll just change the last four digits to the date I was forced to change it (replacing the four digits already there from last time).” From the adversary’s point of view, that is a probabilistic update.

will make an observation, and as a result will (probably) revise her knowledge of the prior on the initial state: this new distribution on the initial state is the *posterior* distribution.

Of course for a channel the initial and final states are equal: that is, the state itself has not changed. (We mentioned that in the two-channel case (2) above.) What has changed is the adversary's knowledge of the state's distribution, and the “before” and “after” inherent in the words “prior” and “posterior” refer to before and after the observation respectively, i.e. what she knows before vs. what she knows after.

Later, on the other hand, we will use the words “initial” and “final” to refer to the states before and after the run respectively — and that is not the same thing. For example the adversary can get *prior* knowledge about the eventual final state before running the program, based on her prior knowledge of the input state and of the program code: for small programs, she can do that calculation by hand without even going near the computer. After the program is run, however, she revises her knowledge because of the observations she made during the run: that gives her *posterior* knowledge of the final state. (See also §13.4.2 below.)

Updating according to M after a leak The effect of running “update according to M ” after the “leak according to C ”, as opposed to just running “update according to M ” on its own, can be described as follows. Call the two programs M and C . If M is run on its own, it has a single initial distribution —which the adversary knows— and from knowing as well what M is (in effect knowing the M it represents, i.e. knowing the source code) she can deduce what its final distribution will be. In fact (in principle) she doesn't even have to run M : it leaks nothing, and there is nothing for her to observe.

But if she runs C *before* M , even though she knows the initial/prior distribution π of C still she does not know what M 's initial distribution is until *after* she has run C . And she can't calculate it in advance (as she could with running M alone), because she must first see what C leaks. After running C and observing whatever it leaks, the adversary replaces that prior π with one of possibly many posteriors. But which one? That depends on what she observed when C ran — each observation y say has an associated posterior ρ_y (an inner) and an associated probability p_y (the outer), which depend on π and C . No matter which posterior ρ_y it is, she then takes that as the initial distribution of M and deduces (as above) what the final distribution ρ'_y of M will be for that initial distribution ρ_y .

The overall hyper resulting from running $C; M$ on π is then all those ρ'_y 's as inners, each with outer probability p_y .

In the case of $C; M$ on a uniform prior, we would find that

$$[(1/2, 1/2) \triangleright C] = 3/4 \cdot [(2/3, 1/3)] + 1/4 \cdot [(0, 1)] ,$$

and then we would continue by running M on each of those inners separately, giving

$$[(2/3, 1/3) \cdot M] = [(5/6, 1/6)] \quad \text{and} \quad [(0, 1) \cdot M] = [(1/2, 1/2)] .$$

Finally we combine those two hypers, weighted by the outer probabilities from the run of C , to give $3/4 \cdot [(5/6, 1/6)] + 1/4 \cdot [(1/2, 1/2)]$.

- (4) For the second heterogeneous case, we consider **markov then channel**, that is

“update according to M ”; “leak according to C ” ,

which is an update followed by a leak: it is the previous case (3) but in the opposite order. Here the user has changed his password (perhaps he was forced to), but was observed after he did that. From an initial/prior π the adversary can calculate a final/prior ρ for M , and it becomes the initial/prior distribution for C . From that, channel C generates a hyper over final/posteriors.

In the case of $M; C$ on a uniform prior, we found above that first running M yields the “intermediate” hyper $[(3/4, 1/4)]$. Then we continue with C , to find

$$[(3/4, 1/4) \triangleright C] = 7/8 \cdot [(6/7, 1/7)] + 1/8 \cdot [(0, 1)] .$$

Finally, note that in all four cases the hyper-distribution refers to what the adversary knows about the *final* state of the program: here, again, we are matching standard program semantics where also, traditionally, one’s interest is in the final state rather than the initial state. (Note that in Case (2) the initial and final states are in fact the same; but in the other three cases the **update** has updated the state.) Interestingly, however, in Case (4)

“update according to M ”; “leak according to C ” ,

the Dalenius perspective lets us speak also about the adversary’s posterior knowledge about the *initial* state. For, since M determines the *correlation* between the initial state and the final state, we see from the discussion in §10.1 that her posterior knowledge about the initial state is simply $[\pi \triangleright MC] = 7/8 \cdot [(4/7, 3/7)] + 1/8 \cdot [(0, 1)]$.

13.4.2 On the terms *prior*, *posterior*, *initial* and *final*

In the enriched environment of channels and markovs together, i.e. in the heterogeneous cases of §13.4.1 just above, we have encountered the following interesting distinctions: before vs. after *execution* is initial vs. final (as in a markov); but before vs. after *observation* is prior vs. posterior (as in a channel).

All four combinations are possible, and meaningful, and we summarize them here:

initial & prior is the adversary’s knowledge (represented as a distribution) *before* she runs the program about what the initial value of the secret *is*.

initial & posterior is the adversary’s knowledge *after* she runs the program about what the initial value *was*.

final & prior is the adversary’s knowledge *before* she runs the program about what the final value of the secret *will be*.

final & posterior is the adversary’s knowledge *after* she runs the program about what the final value *is*.

Only when you mix channels and programs do you get those interesting artifacts. With channels alone, the initial and final states are the same (no updates), so that only prior and posterior are relevant. With markovs alone, the prior- and posterior beliefs are the same (no leaks), so that only initial and final are relevant.

13.4.3 Conditionals

Before designing our semantics for conditionals, we must pause to consider the extent to which a conditional's test –the new²⁰ feature– is vulnerable to observation. This is related to what is called “implicit flow” as in for example

```
IF high THEN low:= 0 ELSE low:= 1
PRINT low ,
```

where the leak is of variable `low` “only” — but the previous conditional gives it what is called a *strong dependency* on `high`, so that `high` is “implicitly” leaked as well. However it will turn out that our situation is more subtle than that: we will consider the deceptively innocuous

```
IF high THEN SKIP ELSE SKIP ,
```

(13.11)

where there is no difference between the two branches and there is no `PRINT` statement.

We recall that we have taken the view that the precise values output from a program are not important: what matters is the posterior reasoning that those values enable. In terms of channels we have agreed, already, that one that maps all inputs to the same output (what we called a “trivial” channel in §13.1 above) leaks nothing, no matter what that one output might be: the Cretan who *always* says “No.” is just as “flowless”²¹ as the Cretan who always says “Yes.” — and both leak less than the Cretan who always lies.²² In terms of programs we therefore are taking the position that `PRINT "Yes"` is the same as `PRINT "No"` is the same as `SKIP` — as programs, they are all three equal.²³ And so by compositionality the above program is equal to

```
IF high THEN PRINT "Yes" ELSE PRINT "No" ,
```

(13.12)

because we can replace the first `SKIP` by `PRINT "Yes"` and the second `SKIP` by `PRINT "No"`.

Yet in Prog. (13.12), the adversary can deduce the value of `high` from whether she hears “Yes” or “No”. Nevertheless that program is semantically equal to Prog. (13.11) —we have after all only replaced equals by equals— and so that one must leak `high` as well. We conclude that since `SKIP` leaks nothing,

*It must be the conditional itself that leaks.*²⁴

We call this “super implicit” flow, and return to it in §13.5 below.

Our conclusion could be summarized as “Don't branch on high.” (Examples of that in practice –and the consequences if ignored– are given in §20.1 and §21.2.1.)

²⁰ It's “new” because neither markovs (assignment statements) nor channels (`PRINT` statements) have it explicitly. But it turns out that we *have* seen it before, as external conditional choice between channels at Secs. 8.1.3 and 8.2.4.

²¹ and clueless

²² Another analogy: the stopped clock that is nevertheless right twice a day.

²³ That is, in the semantics all three give rise to the same abstract channel.

²⁴ “How often have I said to you that when you have eliminated the impossible, whatever remains, however improbable, must be the truth?”

Sherlock Holmes

13.4.4 The power of the adversary: *gedanken* experiments

Our conclusion just above –summarized as “Don’t branch on high.”– is one that we have *derived* from a “*gedanken* experiment” in theory rather than simply recommending it from prudence in practice. That is, rather than trying directly to model what we believe typical *QIF*-behavior to be, instead we conduct experiments in reasoning about program behavior, synthesizing –and often adopting as we do so– principles we believe to be essential for that reasoning to be reliable. Above, those principles were that one can “replace equals by equals”, and that one should abstract from actual output values, retaining only the information they convey.

We now look at what kind of adversary our *gedanken* experiments have let out of the bottle. (We have assumed already that the adversary can read the source code.)

Suppose that in the style of noninterference we have informally classified variables statically into low and high-security: we can model that classification in our current framework by inserting after every assignment to a low variable –say `low`– an explicit `PRINT low` of its (current) value. (That approach is discussed in more detail in Chap. 21.) Since `PRINT`'s cannot be undone, that amounts to what is called “perfect recall”, implying in its starker form that `low:= high; low:=0` leaks `high`, even though `low` is immediately overwritten — that is, we consider it really to be the program `low:= high; PRINT low; low:= 0`.²⁵

Consistent with all that, i.e. with “super implicit” flow and perfect recall, is an adversary who is allowed to run our program under the control of a single-stepping debugger. As she presses `next` `next`... the debugger shows her:

- All the program code;
- Which instruction has just been executed; and
- The names of all variables in scope (which will expand and contract as the steps enter and leave local-variable declaration blocks).

At any time she can click `PRINT` on a visible variable, and its value will be shown. Clicking on a hidden variable however shows nothing (except perhaps a reminder of its type).

(Alternatively, we suppose that all variables are hidden, as in the main part of this chapter, but that the code contains explicit `PRINT` statements. In that case, whenever the adversary single-steps onto a `PRINT` statement, the current value of its argument is shown.)

With the above, the adversary has *perfect recall*, because she can write down the value of any low variable whenever its value is updated, even if it is subsequently overwritten: the value revealed by clicking (or expression revealed by being the argument of a `PRINT` statement), together with where she was in the program code at that moment, and her “piece of paper” will not be affected by the program’s subsequent updates. She profits from *super-implicit flow* as well, because the movement of the debugger’s focus into a conditional’s branches will tell her whether the condition was true or false: and she writes that down too.

Our model also tells us that whether a variable is visible or hidden is not directly related to whether the variable is local or global: all four possibilities are allowed and are meaningful.²⁶ For example, we remarked above that we can mimic the static

²⁵ A compiler could do that in order to use our semantics to interpret the traditional high/low variable declarations of noninterference.

²⁶ That is, each of global-high, global-low, local-high and local-low is possible, and they all make sense. For example a local-visible variable’s value is shown whenever it is within scope of the current program counter.

assignment of attributes “high” and “low” by adding a PRINT statement after any assignment to a low variable, even if it is local.

In the other direction, we can ask “How do you write a PRINT statement when your information flow is determined by static classifications of variables as being high or low?” The answer is to declare a *local* low variable (the scope delimited by $\{\dots\}$) and assign the to-be-leaked expression to that. That is, we define

```
PRINT Exp := { VAR low; low:= Exp } (13.13)
```

so that `Exp` is leaked due to its being assigned to a low variable (while the adversary is watching); but *functionally* both sides of the above are equivalent to SKIP, because (`low`) variable `low` is local.²⁷ The adversary can learn `Exp` by clicking on `low` while the program is still in the local block. (After the program has left the block, she can no longer click on `low` — it is not listed anymore among the program’s variables. But she does not need to, because she still has her piece of paper.)

13.4.5 Iteration

Iteration includes elements we have already discussed –such as conditionals– since a traditional iteration requires a conditional guard to determine whether control will pass to a (further) iteration, or whether the iteration terminates. A complicating factor however is the question of how to treat loops’ potential nontermination in the context of information flow, and it requires some adjustments to hyper-distributions. We return to that question for a thorough treatment in Chap. 16. In the meantime, for simple loops it’s sufficient to unroll and apply the reasoning above for the conditionals that unrolling generates.

13.5 Syntax for probabilistic choice

In the sections above, we have explained informally and with examples how we expect our programs, with their probabilistic updates and leaks, to behave. Precise details of programs’ meanings, i.e. the semantics, will be given in Chap. 14. Here we set out the syntax for probabilistic choice.

The main syntactic innovation is the probabilistic-choice operator $(_P \oplus)$.²⁸ The “ P ” is an expression in the program variables which, as a special (but common) case, can be just a constant p so that e.g. the operator $_{1/2} \oplus$ makes a fair choice between its two operands. The P occurs on the left because it gives the probability to be associated with the left-hand operand; and therefore $1-P$ is associated with the right-hand operand. It can be used both between expressions and between programs.

Probabilistic choice between expressions (internal)

The expression $Exp1 \ _P \oplus Exp2$ denotes a function from the state to a distribution over the type of the two expressions, which two types must therefore be the same. (As a syntactic shortcut we write $Exp1 \oplus Exp2 \oplus Exp3$ etc. for uniform internal choices.) Even if P is a constant p , it does not necessarily follow that the resulting $Exp1 \ _p \oplus Exp2$ is a constant function (distribution-valued) — for that, the two expressions must be constants as well.

²⁷ The two statements are not equivalent to SKIP wrt. information flow, of course. But on the other hand `{ VAR high'; high':= high }` is equivalent to SKIP.

²⁸ This is not our innovation of course: it has been used for decades. It represents an innovation when moving from standard- to probabilistic programs.

More complicated expressions can be written by using multiple instances of $(_P \oplus)$, with different P 's that in some cases might be constants, and in others might depend on the state, and in particular using the special case IF B THEN $Exp1$ ELSE $Exp2$, where B is a Boolean-valued expression. This means exactly what it appears to mean, but can also be understood as $Exp1 [B] \oplus Exp2$ where the “Iverson brackets” $[-]$ convert a Boolean to a number: *true* becomes 1 and *false* becomes 0. With the above syntax, and using x for the state variable, i.e. the coin, the state-to- distribution-on-state function M from (13.7 left) above is written

```
IF x=Heads THEN Heads ELSE (Heads  $_{1/2} \oplus$  Tails) ,
```

and the state-to- distribution-on-Boolean function C from (13.7 right) is

```
IF x=Heads THEN True ELSE (True  $_{1/2} \oplus$  False) ,
```

where in both cases the scalar in the THEN-branch has been implicitly converted to a point distribution.

Statement “update according to M ” is then written using traditional assignment-statement syntax, but with that distribution-valued expression on its right-hand side, becoming

```
x:= (IF x=Heads THEN Heads ELSE Heads  $_{1/2} \oplus$  Tails) , (13.14)
```

and statement `leak C` is written with its new keyword, as

```
PRINT (IF x=Heads THEN True else True  $_{1/2} \oplus$  False) .29
```

Note that all the choices in this subsection were *internal*.

Probabilistic choice between programs (external)

The program $Prog1 \underset{P}{\boxplus} Prog2$ denotes another program that with probability P (which, again, can itself be an expression) acts as $Prog1$ and with probability $1-P$ acts as $Prog2$. (Again, as a syntactic shortcut we write $Prog1 \boxplus Prog2 \boxplus Prog3$ etc. for uniform external choices.) An obvious question is then whether there is a difference between for example the two programs

```
(x:= Exp1)  $_{1/2} \boxplus$  (x:= Exp2) and x:= (Exp1  $_{1/2} \oplus$  Exp2) ,
```

where we met the right-hand statement at Prog. (13.14) just above. (The parentheses are not necessary: here they have been included just to be absolutely clear.)

The above question is not a syntactic accident that we are hoping to legitimize by inventing, after the fact, some highly nuanced semantics that justifies it. In fact it's a bit of good luck that the two syntaxes are so readily available: for the difference in semantics was there already. Recall §13.4.3 in which we concluded that conditionals must leak their condition. A more general argument concludes that probabilistic choices between programs must reveal which branch was taken; and indeed the argument for IF's revealing its condition is then a special case, because –as we remarked above– the IF conditional is a special case of $(_P \oplus)$. With that introduction, we can see the difference between those two programs. The program $(x:= Exp1) \underset{1/2}{\boxplus} (x:= Exp2)$ sets x to $Exp1$ or to $Exp2$ with probability $1/2$ each way, and afterwards the adversary

²⁹ We add the parentheses (\dots) around the IF-expression for clarity — they are not necessary.

knows which assignment was made even though she could not predict it beforehand. This does not, by the way, mean that she knows the value of \mathbf{x} since, for that, she would additionally have had to know the value of Exp1 (or Exp2) before the assignment. On the other hand, the program $\mathbf{x} \in (\text{Exp1} \xrightarrow{1/2} \text{Exp2})$ sets \mathbf{x} to Exp1 or to Exp2 with probability $1/2$ each way, and afterwards the adversary *does not know* which assignment was made. We have already referred in Chap. 8 to those two phenomena as *external*- and *internal* probabilistic choice, in this case between updates.

A similar –but even more interesting– case is external and internal probabilistic choice between leaks, as in

$$(\text{PRINT } \mathbf{b}) \xrightarrow{1/3} (\text{PRINT } \neg\mathbf{b}) \quad \text{and} \quad \text{PRINT } (\mathbf{b} \xrightarrow{1/3} \neg\mathbf{b}), \quad (13.15)$$

where \mathbf{b} is a Boolean-valued state variable. On the left is an external probabilistic choice between two channels, which reveals \mathbf{b} whichever branch was taken because, seeing the Boolean output and knowing which branch produced it, the adversary will know whether to negate the Boolean she saw or not (in order to deduce the value of the Boolean \mathbf{b} she cannot see). On uniform input, it produces the output hyper-distribution

with outer probability $1/2$, the inner distribution “certainly *true*”, and
with outer probability $1/2$, the inner distribution “certainly *false*”.

(Where has the $1/3$ gone? See Exercise 13.4.)

Analogous to the connection between sequential composition of channel programs, and parallel composition of channel matrices (§13.4.1(2)) is the connection between the external probabilistic choice between channel programs and the weighted concatenation of the matrices.

On the right in Prog. (13.15) however, the channel reveals *something* about \mathbf{b} , but not its exact value, because the adversary *does not* know in that case whether \mathbf{b} or $\neg\mathbf{b}$ was revealed: she knows only that $\neg\mathbf{b}$ was more likely to have been revealed (because the biased $1/3$) was chosen for this example), and so her best guess will be that \mathbf{b} is the negation of what she observed — but she might be wrong. Thus again taking \mathbf{b} for example to be uniformly distributed initially, we see that the right-hand program produces the hyper-distribution

with outer probability $1/2$, the inner distribution “ $1/3$ *true* and $2/3$ *false*”, and
with outer probability $1/2$, the inner distribution “ $2/3$ *true* and $1/3$ *false*”.

The outer probability reflects the initial uncertainty (i.e. the prior) of the Boolean’s value. Afterwards, the uncertainty is less — but whether the posterior is skewed towards *true* or *false* depends on what value the Boolean actually has.³⁰

A mnemonic for the syntax we have chosen to use for those distinctions is the idea that expressions are evaluated atomically, but the combination of whole statements is not. (A similar convention is often used in formalizations of concurrency.)

³⁰ An interesting exercise, to which we will return later, is to run the right-hand program in a loop. The further the loop runs, the more is revealed about \mathbf{b} ; it is an example of repeated independent runs, but with the repetition controlled programmatically by a `while`-loop. The exact value of \mathbf{b} is revealed only in the limit. See §14.6.3.

13.6 Summary

The remarkable thing about the constructions of this chapter –the questions raised, and their answers– is that they are treated *automatically* by the simple device of embedding Markov matrices, as markovs, into the same type that abstract channels have already, and then using “off the shelf” the so-called “Kleisli composition” of functions to implement sequential composition of programs — and finally appealing to compositionality to determine the behavior of conditionals.

Just those three steps create all of the material this chapter has presented.³¹

13.7 Exercises

Exercise 13.1 It’s interesting that `PRINT (bp⊕¬b)` is the same as `SKIP` if *and only if* $p = 1/2$. Show that to be the case by using channel matrices directly. \square

Exercise 13.2 (Password histories) Recall §13.3.4, where the issue of repeated password changes was discussed, and consider this program:

```
VAR
    password: {0,1}                                - Either 0 or 1.
    p,q: [0,1]                                     - Between 0 and 1.

REPEAT N TIMES
    PRINT password                                ← Leak current password.

                                         ↓ Internal choice of new password.
    password:= ( IF password=0 THEN 1p⊕0 ELSE 0q⊕1 )
END .
```

It models repeated changes of `password`, which is either 0 or 1: on each occasion the new password is chosen probabilistically, based on its current value — which itself is leaked just before the update. But the probabilities used for the update are themselves also unknown to the adversary. (That is why they are variables rather than constants — not because the program changes them (it doesn’t), but so they can be hidden from the adversary who, remember, can read the source code.)

The theme we explore is that frequent changes of password can endanger the *current* password if its previous values become known: from them, the adversary might learn the password-changing habits of the user.

Describe the adversary’s asymptotic posterior knowledge of the state of this system as N becomes large. Comment on the fact that `password`’s sequence of values plays a role in the adversary’s attack, even though it is continually overwritten.

What role does the adversary’s knowledge of the program code play in her judgment?

What difference would there be if `password` were not leaked? \square

³¹ Kleisli composition is used very extensively in theoretical computer science: we do not have to make excuses for using it. Rather we would have to excuse ourselves if we did *not* use it.

Exercise 13.3 Consider the two programs

$$x := (x+1) \underset{1/2}{\oplus} x \quad \text{and} \quad (x := x+1) \underset{1/2}{\boxplus} (x := x) ,$$

assuming that the prior distribution on x is uniform on the set $\{1, 2, 3\}$. Compute the hyper-distributions produced by the two programs. Explain why the first program is a markov but the second is not, even though it has no PRINT statement. \square

Exercise 13.4 In §13.5 it was stated that the left-hand Prog. (13.15), repeated here

$$(\text{PRINT } b) \underset{1/3}{\boxplus} (\text{PRINT } \neg b) , \quad (13.16)$$

if run on the uniform input distribution in b , would produce the output hyper

with outer probability $1/2$, the inner distribution “certainly true”, and
with outer probability $1/2$, the inner distribution “certainly false”.

It’s true: and in fact it would do that no matter what external-choice probability were used — even the extremes 1 and 0, which would reduce the program to the simpler PRINT b and PRINT $\neg b$ respectively.

Where has the $1/3$ gone? And where has the $1/2$ come from?

And would an observer notice any difference between Prog. (13.16) and the “complementary” program

$$(\text{PRINT } b) \underset{2/3}{\boxplus} (\text{PRINT } \neg b) ?$$

\square

13.8 Chapter notes

Triples for sequential-program specification were introduced by Hoare’s adaptation of Floyd’s work [11, 16]; they are usually called “Hoare triples”. Originally they expressed so-called partial correctness, that the postcondition was achieved if the program terminated; later they were more commonly taken to express as well that the program was guaranteed to terminate (while continuing to establishing the postcondition), which was called total correctness. That was especially true in the approach of weakest preconditions, a further adaptation introduced by Dijkstra [9]. The idea of a mathematical definition of refinement of programs was promoted in many early works including *Z* and *VDM* [1, 15, 17] and for example made usable by industry in the *B*-method of Abrial [2], although it probably takes its name from Wirth’s notion of “stepwise refinement” [35].³² It was explicitly connected to Dijkstra’s weakest preconditions by Ralph-Johan Back in his Ph.D. thesis [3] which led to what was later called *The Refinement Calculus* [4, 27]. It was further developed by Morgan, and by Morris [27, 28, 31], and –finally– systematically explored and presented as a whole by Back and von Wright [5]. Refinement also figured prominently in the *failures model* for Communicating Sequential Processes of Hoare, Brookes and Roscoe [6].

“Noninterference” as a security policy for programs was studied by Goguen and Meseguer [13]. A similar phenomenon was identified even earlier by Cohen [7], who called it “strong dependency”. The basic idea is to model a program as a mechanism

³² *VDM* stands for “Vienna Development Method”, referring to its early development at the IBM Laboratory in Vienna. It is thought that *Z* stands for “Zermelo”, because of its emphasis on using “ordinary” set theory to specify systems. *B* is believed to stand for “Bourbaki”, possibly because of its being based on a very small number of fundamental concepts.

that operates on inputs and outputs, some of which are “high” security and the others “low” security. Information is considered to have leaked if there is some (non-trivial) correlation between the high-security inputs (and, here, outputs) and low-security outputs. Research in the verification of noninterference motivated an interest in adapting refinement-based methods to enable the comparison of programs with respect to their security characteristics, using various abstractions of program behavior to prove complex properties. The underlying philosophy is that a benchmark for a verification task is an abstract (and therefore succinct) specification with which to compare a (possibly highly complex) implementation; most importantly, it should stand as the independent arbiter of the standard required for correctness. Refinement is a natural manifestation of this principle where an abstract program’s behavior can be compared rigorously to that of a more detailed implementation [4, 27, 31]; such mathematical refinement-based techniques have formed the foundation for many successful software verification tools [2, 17].

Early attempts to find a refinement relation that preserves security *as well as* functional properties proved to be troublesome [14], and some suffered from the so-called “refinement paradox”, where apparently “leak-free” specifications were refined by “leak-full” implementations.³³ In response to this problem, various restrictions on refinements were introduced to enable the distinction between safe and unsafe refinement (for security). Mantel [20], for example, proposed a number of refinement operators that can be used to give sound rules for preserving noninterference properties, and Sabelfeld and Sands [33] explored partial-order relations for information flow.

A refinement order for preservation of qualitative security properties (including demonic choice) was proposed by Morgan in his “Shadow” model [29, 30] and is based on a Kripke structure that is able to distinguish between “internal” and “external” choice in much the same way that hyper-distributions distinguish between internal and external probabilities. It used the *gedanken*-experiment approach to motivate its structure. (Sherlock Holmes’ famous remark –essentially the motivation (p. 241) for “We must accept what the experiments tell us.”– comes from Sir Arthur Conan Doyle’s *The Sign of Four* [10].)

An early probabilistic refinement-style model [21] for reasoning about quantitative noninterference properties via refinement used the refinement relation of Chap. 9. The probabilistic programming model presented here can be seen as a further extension of original work by Kozen [19] on probabilistic programs.

While there are several semantics-based models for the verification of information-flow properties, they differ in various ways: how they treat “implicit flow” [8], whether the information flow concerns initial or final values (of program variables) [18], and how to treat perfect recall. The primary principle adopted in this book is that of compositional reasoning in a context where the secrets might change — and that necessarily implies the formalization of the concepts given in Chap. 14.

A model treating probability, information flow *and* (external) demonic choice has been explored: by analogy with the simultaneous treatment of probability and demonic choice [23] that replaced Kozen’s $\mathbb{D}\mathcal{X}$ by $\mathbb{P}\mathbb{D}\mathcal{X}$, that model replaces hyper-distributions $\mathbb{D}^2\mathcal{X}$ by sets of them $\mathbb{P}\mathbb{D}^2\mathcal{X}$ [22]. It also treats hyper-distributions and their refinement over proper measures.

The enhanced “Dalenius aware” semantics mentioned in §13.3.4 is explored by McIver et al. [25, 32].

Our model’s use of monads, and in particular the Kleisli composition of successive program steps (§13.4.1), is an example of the general technique pioneered by Moggi where he wrote that monads abstract and unify common features of “notions of

³³ The name “refinement paradox” seems to be due to Jeremy Jacob.

computation” [26]. Monadic features can be, and have been built into (functional) programming languages like *Haskell* [34], and are what allow the easy implementation (a deep embedding) of the *QIF*-aware programming language *Kuifje* into Haskell [12].

The non-compositionality of the “intuitive” extension of Hoare logic to probabilistic programs, mentioned in §13.3.1, is treated in detail by McIver and Morgan [24, App. A.1]. The example there is as follows.

Let $x := A \sqcap (x := B \text{ } 1/2 \boxplus x := C)$ and $(x := A \sqcap x := B) \text{ } 1/2 \boxplus (x := A \sqcap x := C)$ be two programs combining (external) probabilistic choice and demonic choice. In the intuitive extension, because of the nondeterminism, we can ask only “What is the *least guaranteed* probability of establishing the postcondition?”³⁴ And for the two programs above, the answer is the same for all 8 possible postconditions (corresponding to the 8 subsets of $\{A, B, C\}$). Thus in the intuitively extended semantics of “probabilistic Hoare triples”, the two programs are the same.

But if we follow (i.e. sequentially compose) each with the same further program **IF** $x = A$ **THEN** $x := B \text{ } 1/2 \boxplus x := C$ **ELSE** **Skip**, then the least guaranteed probabilities of establishing postcondition $\{x = B\}$ differ: it is $1/2$ in the first case, but in the second case it is $1/4$. Thus the compositions are different even though the components (semantically) are the same. Thus compositionality has failed here.

³⁴ That is instead of “What is the (exact) probability of establishing the postcondition?”

Bibliography

- [1] Abrial, J.-R.: Data semantics. In: J.W. Klimbie, K.L. Koffeman (eds.) Proceedings of the IFIP Working Conference on Data Base Management, pp. 1–59. North-Holland, Amsterdam (1974)
- [2] Abrial, J.-R.: The B Book: Assigning Programs to Meanings. Cambridge University Press, New York (1996)
- [3] Back, R.-J.: On the correctness of refinement steps in program development. Report A-1978-4, Department of Computer Science, University of Helsinki (1978)
- [4] Back, R.-J.: A calculus of refinements for program derivations. Acta Informatica **25**(6), 593–624 (1988)
- [5] Back, R.-J., von Wright, J.: Refinement Calculus: A Systematic Introduction. Springer, Berlin (1998)
- [6] Brookes, S.D., Hoare, C.A.R., Roscoe, A.W.: A theory of communicating sequential processes. Journal of the ACM **31**(3), 560–599 (1984)
- [7] Cohen, E.: Information transmission in computational systems. In: Proceedings of the 6th ACM Symposium on Operating Systems Principles (SOSP 1977), pp. 133–139. ACM, New York (1977)
- [8] Denning, D.E., Denning, P.J.: Certification of programs for secure information flow. Communications of the ACM **20**(7), 504–513 (1977)
- [9] Dijkstra, E.W.: A Discipline of Programming. Prentice Hall (1976)
- [10] Doyle, A.C.: The Sign of Four. In: Lippincott's Monthly Magazine. Joseph Marshall Stoddart (1890)
- [11] Floyd, R.W.: Assigning meanings to programs. In: J. Schwartz (ed.) Mathematical Aspects of Computer Science, *Proceedings of the Symposium on Applied Mathematics*, vol. 19, pp. 19–32. American Mathematical Society (1967)
- [12] Gibbons, J., McIver, A., Morgan, C., Schrijvers, T.: Quantitative information flow with monads in Haskell. In: G. Barthe, J.P. Katoen, A. Silva (eds.) Foundations of Probabilistic Programming. Cambridge University Press, New York (2020)
- [13] Goguen, J.A., Meseguer, J.: Security policies and security models. In: Proceedings of the 1982 IEEE Symposium on Security and Privacy, pp. 11–20. IEEE, Los Alamitos (1982)

Bibliography

- [14] Graham-Cumming, J., Sanders, J.W.: On the refinement of noninterference. In: Proceedings of the IEEE Computer Security Foundations Workshop (CSFW 2011), pp. 35–42. IEEE, Los Alamitos (1991)
- [15] Hayes, I.: Specification Case Studies. Prentice Hall (1987)
- [16] Hoare, C.A.R.: An axiomatic basis for computer programming. Communications of the ACM **12**(10), 576–580 (1969)
- [17] Jones, C.B.: Systematic Software Development using VDM. Prentice Hall (1986)
- [18] Joshi, R., Leino, K.R.M.: A semantic approach to secure information flow. Science of Computer Programming **37**(1–3), 113–138 (2000)
- [19] Kozen, D.: A probabilistic PDL. In: Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC 1983), pp. 291–297. ACM, New York (1983)
- [20] Mantel, H.: Preserving information flow properties under refinement. In: Proceedings of the 2001 IEEE Symposium on Security and Privacy, pp. 78–91. IEEE, Los Alamitos (2001)
- [21] McIver, A., Meinicke, L., Morgan, C.: Compositional closure for Bayes risk in probabilistic noninterference. In: S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide, P.G. Spirakis (eds.) Proceedings of the 37th International Colloquium on Automata, Languages, and Programming (ICALP 2010), *Lecture Notes in Computer Science*, vol. 6199, pp. 223–235. Springer, Berlin (2010)
- [22] McIver, A., Meinicke, L., Morgan, C.: A Kantorovich-monadic powerdomain for information hiding, with probability and nondeterminism. In: Proceedings of the 2012 27th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2012), pp. 461–470. IEEE, Los Alamitos (2012)
- [23] McIver, A., Morgan, C.: Abstraction, Refinement and Proof for Probabilistic Systems. Monographs in Computer Science. Springer, Berlin (2005)
- [24] McIver, A., Morgan, C.: A novel stochastic game via the quantitative *mu*-calculus. In: A. Cerone, A. de Pierro (eds.) Proceedings of the 3rd Workshop on Quantitative Aspects of Programming Languages (QAPL 2005), *Electronic Notes in Theoretical Computer Science*, vol. 153. Elsevier (2006)
- [25] McIver, A., Morgan, C., Rabehaja, T.: Program algebra for quantitative information flow. Journal of Logical and Algebraic Methods in Programming **106**, 55–77 (2019)
- [26] Moggi, E.: Notions of computation and monads. Information and Computation **93**(1), 55–92 (1991)
- [27] Morgan, C.: The specification statement. ACM Transactions on Programming Languages and Systems **10**(3), 403–419 (1988)
- [28] Morgan, C.: Programming from Specifications, 2nd edn. Prentice Hall (1994)
- [29] Morgan, C.: *The Shadow Knows*: Refinement of ignorance in sequential programs. In: T. Uustalu (ed.) Proceedings of the International Conference on Mathematics of Program Construction (MPC 2006), *Lecture Notes in Computer Science*, vol. 4014, pp. 359–378. Springer, Berlin (2006)

- [30] Morgan, C.: *The Shadow Knows*: Refinement of ignorance in sequential programs. *Science of Computer Programming* **74**(8), 629–653 (2009)
- [31] Morris, J.M.: A theoretical basis for stepwise refinement and the programming calculus. *Science of Computer Programming* **9**(3), 287–306 (1987)
- [32] Rabehaja, T., McIver, A., Morgan, C., Struth, G.: Categorical information flow. In: M.S. Alvim, K. Chatzikokolakis, C. Olarte, F. Valencia (eds.) *The Art of Modelling Computational Systems: A Journey from Logic and Concurrency to Security and Privacy. Essays Dedicated to Catuscia Palamidessi on the Occasion of Her 60th Birthday, Lecture Notes in Computer Science*, vol. 11760, pp. 329–343. Springer, Berlin (2019)
- [33] Sabelfeld, A., Sands, D.: A PER model of secure information flow in sequential programs. *Higher-Order and Symbolic Computation* **14**(1), 59–91 (2001)
- [34] Wadler, P.: Monads for functional programming. In: *Proceedings of the First International Spring School on Advanced Functional Programming Techniques-Tutorial Text*, pp. 24–52. Springer, Berlin (1995)
- [35] Wirth, N.: Program development by stepwise refinement. *Communications of the ACM* **14**(4), 221–227 (1971)



Chapter 14

Hidden-Markov modeling of *QIF* in sequential programs

In the last chapter we made the case for embedding our treatment of *QIF*, developed in Parts I–III as “channels that leak”, in a programming-language setting. From this we should obtain a semantics, and reasoning tools, that allow us to treat leaky *programs* (i.e. not just leaks on their own) in a fundamental way, one that draws on decades of experience on how to treat programming-language semantics generally.

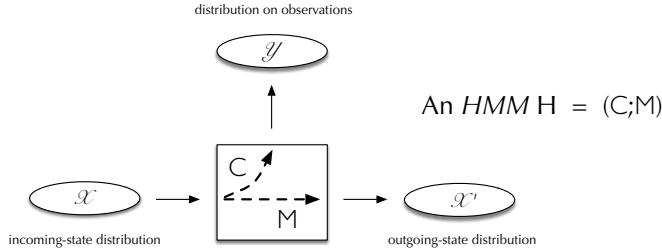
Our approach to that is the “Hidden Markov Model”, which we introduce in the next section in its conventional mathematical form. We then link that presentation back to the previous chapter (as promised there) and –once that is done—move on to “abstract *HMM*’s” in a way inspired by the analogous move we made earlier, from concrete- to abstract channels.

Abstract *HMM*’s give us our semantics for sequential programs with probabilistic choice and information flow and –crucially— it supports a definition of program refinement that agrees both with refinement of channels (Chap. 9) and with long-extant definitions of refinement for “ordinary” sequential programs.

14.1 Concrete Hidden Markov Models

Traditional, that is, “concrete” Hidden Markov Models are, like concrete channels, mathematical structures presented as stochastic matrices: they model both probabilistic state updates (i.e. Markov transitions) and information flow (i.e. channels). As promised, in them we will find both Markov transitions and (concrete) channels (as discussed in Parts I–III) as special cases.

We illustrate a Hidden Markov Model in Fig. 14.1: as for channels, it comprises a set \mathcal{X} of states, and a set \mathcal{Y} of observations; but it has *two* stochastic matrices C and M rather than just one as a channel does (or, indeed, as a Markov transition does). The Markov matrix M gives for any two states $x, x' : \mathcal{X}$ the probability $M_{x,x'}$ that a transition from initial state x will end in final state x' : this is of course the Markovian aspect of the *HMM*. The channel matrix C gives for any state x and observation $y : \mathcal{Y}$ the probability $C_{x,y}$ that y will be emitted, and thus observed, from that same initial state x : this is the channel aspect. We call the two matrices the “channel” and the “markov” (lower case) of the *HMM*. Thus an *HMM* is effectively a channel and a markov running in parallel on the same input: that is, the probabilistic choices in M, C



The *HMM* named H comprises a channel matrix C and a Markov matrix M : we write $H = (C; M)$. It takes an input state $x: \mathcal{X}$ to an output state $x': \mathcal{X}'$, probabilistically; and it emits an observable $y: \mathcal{Y}$, also probabilistically.

Figure 14.1 A Hidden Markov Model

are resolved independently, and as we see in Fig. 14.1 there is no correlation between x' and y . We write that *HMM* as $(C; M)$, using the usual font for “concrete” (as we did for concrete channels).

In the mathematical literature, an *HMM* is typically analyzed over a number of steps $i = 0, 1, \dots$ from some initial distribution π over \mathcal{X} , so that a succession of states x^1, x^2, \dots and observations y^1, y^2, \dots results in which each x^i is related probabilistically to x^{i+1} by M and to y^i by C . The x^i 's are “passed along” from one *HMM*-step to the next; but the y^i 's are just left “lying there”, accumulating, to be observed by whichever adversary cares to look.

Let $\pi: \mathbb{D}\mathcal{X}$ be the distribution over \mathcal{X} of the incoming state, in our terms the prior. The distribution $\rho': \mathbb{D}\mathcal{X}'$ over \mathcal{X}' of the final state x' is the multiplication of π as a row vector on the left by matrix M on the right, so that $\rho'_{x'} = \sum_x \pi_x M_{x, x'}$ — this is just as we saw in Chap. 13, and we continue to write it $\pi M = \rho'$, using juxtaposition for matrix multiplication and assuming π, ρ' are presented as row vectors. Recall that we sometimes write \mathcal{X}' for the final state, even when in fact $\mathcal{X} = \mathcal{X}'$, in order to make their role in discussions easier to disentangle.

Similarly the distribution of observations y is the multiplication πC — that is, $\sum_x \pi_x C_{x, y}$, again as we have seen abundantly already.

The “hidden” aspect of an *HMM*, the “*H*” in its name, is that an adversary cannot see the exact values x and x' of x and x' . (She can however see y .) Her traditional “channel only” *a posteriori* reasoning would allow her to take her knowledge of prior π and observed y and deduce a corresponding hyper Δ of posterior distributions over the initial state; and indeed our earlier examples of channel leakage compare various vulnerabilities (or uncertainties) taken over those two, the prior π and the hyper Δ . Our “*HMM* aware” adversary however goes one step further: she takes that posterior-initial distribution Δ from just above (as explained in §13.4.2 earlier) and uses M to calculate a revised, posterior-final distribution. That is, the adversary is not interested in, indeed forgets, the initial distribution: she focuses on the final distribution alone.¹

¹ We will see later, however, that the Dalenius effect (recall Chap. 10) will sometimes require the adversary to be interested in both the final *and* the initial distributions.

14.1.1 *A priori* versus *a posteriori* reasoning – in more detail

We recall from §13.4.2 that *a priori* reasoning about an *HMM*, say H , is done by the adversary before H is “executed”, when all she knows is the concrete matrices \mathbf{C}, \mathbf{M} and the prior π . With that information alone – i.e. *without* executing H – she can already predict what the distribution $\rho' = \pi \mathbf{M}$ of the final state x' will be. Remember that we call that ρ' *a prior* distribution even though it concerns final values of x' , i.e. after the H has executed: it is “prior” because the adversary’s *knowledge* is nevertheless still “before” any leak.

A posteriori reasoning about H is done by the adversary after H is executed: that can be a “dynamic” approach, where her reasoning is based on a specific y that she has seen; or it can be a “static” approach, where she reasons over (the distribution of) all y ’s that she might see.

Taking momentarily the dynamic view, we imagine that she uses Bayesian reasoning based on the y she sees to revise knowledge about the prior π — we call it π^y just below. Then, from that revision, by applying \mathbf{M} she revises her knowledge about the posterior ρ' as well — below called $(\rho')^y$.² She would calculate as follows.

Recall that we write $\mathsf{J} = \pi \triangleright \mathbf{C}$ to say that J is the joint distribution obtained by “pushing π through \mathbf{C} ”, defined

$$(\pi \triangleright \mathbf{C})_{x,y} := \pi_x \mathbf{C}_{x,y} .$$

Then the resulting *a posteriori* distribution of the input, for which we introduce the temporary notation π^y (with an implicit dependence on \mathbf{C}), is defined

$$\pi^y := \lfloor \mathsf{J}_{-,y} \rfloor ,$$

where we recall that $\mathsf{J}_{-,y}$ is column y of J , which might not sum to one — call that a subdistribution on \mathcal{X} . We are writing $\lfloor \delta \rfloor$ for the normalization of subdistribution δ , obtained by dividing δ ’s probabilities by the weight $\langle \delta \rangle$ of δ , that is by the sum of the probabilities in δ .

The *a posteriori* distribution ρ' of the output is then obtained by applying markov-transition matrix \mathbf{M} to that revised input distribution instead of to the original: thus we have

$$(\rho')^y := \pi^y \mathbf{M} . \quad (14.1)$$

However an alternative calculation of $(\rho')^y$ is as follows. First push the prior π through H as a whole, defining

$$K_{x,y,x'} := (\pi \triangleright \mathsf{H})_{x,y,x'} = \pi_x \mathsf{H}_{x,y,x'} = \pi_x \mathbf{C}_{x,y} \mathbf{M}_{x,x'} .$$

Then project away from x , i.e. take the (y, x') -marginal, giving say $L_{y,x'} := \sum_x K_{x,y,x'}$; and finally condition L on the observed y to give

$$(\rho')^y = \lfloor L_{y,-} \rfloor . \quad (14.2)$$

That turns out to be equivalent to the definition at (14.1) just above. The conceptual advantage of the second view (14.2) however is that it encourages us to consider the *HMM* as a whole rather than as a combination of a channel and a markov, although the latter is still useful to understand the motivation and operation of an *HMM*.

Thus, to summarize, for us a concrete *HMM* is a double-output stochastic matrix, i.e. of type $\mathcal{X} \rightarrow (\mathcal{Y}, \mathcal{X})$ where the left-hand \mathcal{X} is the input and the right-hand \mathcal{X} is the

² By “applying” \mathbf{M} we mean that she reasons about what the effect of executing \mathbf{M} would be. She does not need actually to execute \mathbf{M} in order to carry out that reasoning.

output. Sometimes we will write $\mathcal{X} \rightarrow (\mathcal{Y}, \mathcal{X}')$ to emphasize that distinction, noting “that in fact $\mathcal{X} = \mathcal{X}'$ ”; but it does for example offer the possibility of “heterogeneous” HMM’s where \mathcal{X} and \mathcal{X}' are not the same: they would move probabilistically from one hidden state space to another. An example of that is given in §14.4.8, where it is used to introduce local variables in a sequential QIF-program.

With the general view now established, we define the construction $(C; M)$, the concrete HMM introduced in Fig. 14.1, as a whole: it is given by $(C; M)_{x,y,x'} := C_{x,y} M_{x,x'}$.

14.2 Operations on and specializations of concrete HMM’s

14.2.1 Pure-channel and pure-markov HMM’s

We can now begin to fulfill our promises made in Chap. 13.

A markov, in the sense of Chap. 13, is an HMM whose channel component leaks nothing: we say that it is a (concrete) *pure markov* if it has the form $(C; M)$ with C the trivial channel $\mathbb{1}$ observing only the unit type $\{\bullet\}$, i.e. releasing no information at all; and in that case we can abbreviate it $(; M)$. We thus have $(; M)_{x,\bullet,x'} = M_{x,x'}$, and confirm that a pure markov can change the state, but cannot reveal anything about it. Pure-markov HMM’s represent probabilistic assignment statements.

Similarly a (concrete) HMM is said to be a *pure channel* if it has the form $(C; M)$ with M the identity (matrix) on $\mathcal{X} \rightarrow \mathcal{X}$, in which case we can abbreviate it $(C;)$ and have $(C;)_{x,y,x'} = (C_{x,y} \text{ if } x=x' \text{ else } 0)$. A pure channel can reveal information about the state, but cannot change it. Pure-channel HMM’s represent PRINT statements.

Thus markovs are indeed HMM’s that don’t leak, and channels are HMM’s that don’t update — as we recall from §13.1.

14.2.2 Sequential composition of concrete HMM’s

Further promises were made in §13.4.1 about the behavior of sequential composition: we address them now. Given two concrete HMM’s, we define their sequential composition as follows.

Definition 14.1 (Sequential composition of HMM’s) Given two HMM’s $H^{1,2}$ of types $\mathcal{X} \rightarrow (\mathcal{Y}^{1,2}, \mathcal{X})$ respectively, we have

$$(H^1; H^2)_{x,(y^1,y^2),x'} := \left(\sum_{x''} H^1_{x,y^1,x''} \times H^2_{x'',y^2,x'} \right) .$$

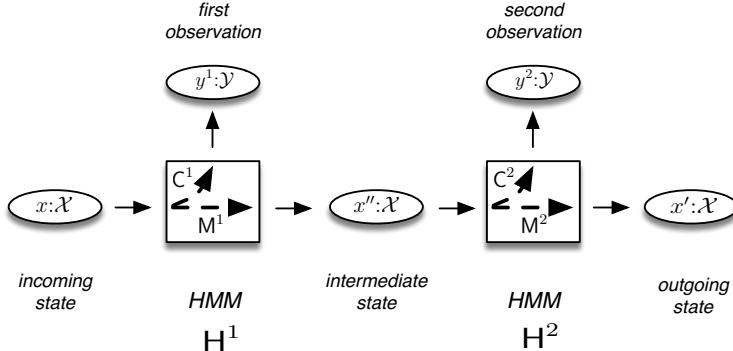
□

The composition is of type $\mathcal{X} \rightarrow (\mathcal{Y}, \mathcal{X})$, where $\mathcal{Y} := \mathcal{Y}^1 \times \mathcal{Y}^2$ and $y = (y^1, y^2)$. It is in fact the matrix multiplication along the x' coordinate of H^1 and the x coordinate of H^2 , with the two observation spaces $\mathcal{Y}^{1,2}$ then merged into one via Cartesian product. We illustrate that in Fig. 14.2.

We now (re-)consider the special cases of sequential composition from §13.4.1. Sequentially composing two pure markovs is effectively markov composition: we have

$$(; M^1); (; M^2) = (; M^1 M^2) ,$$

where as usual the juxtaposition $M^1 M^2$ denotes matrix multiplication. To verify that, we calculate



Each step $H^{1,2}$ takes an input- to an output state in \mathcal{X} ; the observations $y^{1,2}:\mathcal{Y}$ are accumulated. In each step $H^{1,2}$ the output state is determined by a markov $M^{1,2}$ on the input to that step, and the observation is determined independently by a channel $C^{1,2}$ on the same input, i.e. *before* application of the markov.

Figure 14.2 Sequential composition of HMM's

$$\begin{aligned}
 & ((; M^1) ; (; M^2))_{x,\bullet,x'} \\
 = & ((; M^1) ; (; M^2))_{x,(\bullet,\bullet),x'} && “\bullet = (\bullet,\bullet)” \\
 = & \sum_{x''} (; M^1)_{x,\bullet,x''} (; M^2)_{x'';\bullet,x'} && “\text{sequential composition Def. 14.1 of HMM's}” \\
 = & \sum_{x''} M^1_{x,x''} M^2_{x'';\bullet,x'} && “\text{definition } (; M)” \\
 = & (M^1 M^2)_{x,\bullet,x'} && “\text{matrix multiplication}” \\
 = & (; M^1 M^2)_{x,\bullet,x'} && “\text{definition } (; M)”
 \end{aligned}$$

On the other hand, sequentially composing two pure channels is effectively their parallel composition. (Recall Def. 8.1 of (\parallel) and §13.4.1(2).) We have

$$(C^1; ; (C^2;)) = (C^1 \parallel C^2;),$$

which we can now verify with the calculation

$$\begin{aligned}
 & ((C^1; ; (C^2;)))_{x,y,x'} \\
 = & ((C^1; ; (C^2;)))_{x,(y^1,y^2),x'} && “y = (y^1, y^2)” \\
 = & \sum_{x''} (C^1;)_{x,y^1,x''} (C^2;)_{x'',y^2,x'} && “\text{sequential composition Def. 14.1 of HMM's}” \\
 = & \sum_{x''} (C^1_{x,y^1} \text{ if } x=x'' \text{ else } 0) (C^2_{x'',y^2} \text{ if } x''=x' \text{ else } 0) && “\text{definition } (C;)” \\
 = & \sum_{x''} (C^1_{x,y^1} C^2_{x'',y^2} \text{ if } x=x''=x' \text{ else } 0) && “\text{composition of conditionals}” \\
 = & (C^1_{x,y^1} C^2_{x',y^2} \text{ if } x=x' \text{ else } 0) && “\text{one-point rule eliminates } x''” \\
 = & ((C^1 \parallel C^2)_{x,(y^1,y^2)} \text{ if } x=x' \text{ else } 0) && “\text{definition } C^1 \parallel C^2” \\
 = & ((C^1 \parallel C^2);)_{x,y,x'} && “\text{definition } (C;); y = (y^1, y^2)”
 \end{aligned}$$

Then sequentially composing a pure channel and a pure markov is the same as taking them both together: we have

$$(C; ; (; M)) = (C; M),$$

which is indeed the justification of the notation $(C; M)$ in the first place. We calculate

$$\begin{aligned}
 & ((C;);(M))_{x,y,x'} \\
 = & ((C;);(M))_{x,(y,\bullet),x'} && "y = (y, \bullet)" \\
 = & \sum_{x''} (C_x)_{x,y,x''}(M)_{x'',\bullet,x'} && \text{"sequential composition Def. 14.1 of HMM's"} \\
 = & \sum_{x''} (C_{x,y} \text{ if } x=x'' \text{ else } 0) M_{x'',x'} && \text{"definition (C;); definition (;\text{M})"} \\
 = & C_{x,y} M_{x,x'} && \text{"one-point rule eliminates } x''\text{"} \\
 = & (C;M)_{x,y,x'} && \text{"definition (C;M)"}
 \end{aligned}$$

The reason for that simple result is that even though C “goes first” in the sequential composition, it does not change the state — and so the same effect is achieved even if it “goes at the same time” as it does in $(C;M)$.

14.2.3 General (concrete) HMM's

Our fourth case is a pure markov followed by a pure channel, that is $(;\text{M});(C;)$, and is in fact our principal motivation for the generalization we have adopted, that a concrete *HMM* is a stochastic matrix of type $\mathcal{X} \rightarrow (\mathcal{Y}, \mathcal{X}')$ — because $(;\text{M});(C;)$ is *not* in general equal to $(C';M')$ for some C', M' .

The C in $(;\text{M});(C;)$ is acting on the final state, not on the initial state as it is in $(C;M)$. More precisely, in $(;\text{M});(C;)$ the C 's initial state is the M 's final state, as is normal for sequential composition — but since C does not change the state, its initial state is the final state of the whole composition as well.

The reason it is not possible in general to reduce $(;\text{M});(C;)$ to some $(C';M')$ is that, in the latter, the y and the x' must have independent distributions for a given x , no matter what C' and M' might be — the action of M' cannot be affected by the action of C' . But in $(;\text{M});(C;)$ the action of C *is* affected by the action of M . Take for example $\mathcal{X} := \{x_1, x_2\}$ and C the identity matrix (which reveals everything) and M the everywhere $1/2$ matrix (which ignores its input). In $(;\text{M});(C;)$ we then have $y = x'$ for any x — they are not independent.

14.3 Abstract Hidden Markov Models

Just as we moved from channel matrices to abstract channels (§4.4) by abstracting from the observations' values, we now move from *HMM*'s-as-matrices, i.e. concrete *HMM*'s, to abstract *HMM*'s.

Definition 14.2 (Matrix HMM denotes abstract HMM) Let $H: \mathcal{X} \rightarrow \mathcal{Y} \times \mathcal{X}$ be an *HMM* presented as a matrix. Its denotation $\llbracket H \rrbracket$, of type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$, is called an *abstract HMM* and is defined $\llbracket H \rrbracket(\pi) := [J]$, where the prior π is of type $\mathbb{D}\mathcal{X}$, and the joint-distribution matrix $J: \mathcal{X}' \rightarrow \mathcal{Y}$ is over the observations and the *final* value of x , that is the x', y -marginal of J given by $J_{x',y} := \sum_x (\pi \triangleright H)_{x,y,x'} = \sum_x \pi_x H_{x,y,x'}$.

As usual, the brackets $[-]$ take a joint distribution to its corresponding hyper-distribution by abstracting from \mathcal{Y} (Def. 4.6).³

□

When emphasis is needed, we will write the type of an abstract *HMM* as $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}'$ to remind us that the result is a hyper over the *final* state.

³ Note that if we consider a distribution $\delta: \mathbb{D}\mathcal{X}$ to be isomorphically a trivial joint distribution in $\mathbb{D}(\bullet \times \mathcal{Y})$, where \bullet is the unit type introduced in §14.2.1 above, then $[\delta]$ gives the same hyper either way.

14.3.1 Sequential (Kleisli) composition of abstract HMM's

In §14.2.2 we defined sequential composition of concrete HMM's; in §14.3 we abstracted from concrete HMM's to abstract HMM's; and here we show that abstraction to respect sequential composition.

Since abstract HMM's have type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$, they do not readily compose as homogeneous functions do: the result type of the first component will be $\mathbb{D}^2\mathcal{X}$, but the argument type of the second is $\mathbb{D}\mathcal{X}$. This is solved in a general way by noting that \mathbb{D} has associated with it two constructions: one is “lifting”, which allows any function of f of type say $\mathcal{A} \rightarrow \mathcal{B}$ to be converted, i.e. lifted, to a function $\mathbb{D}f$ of type $\mathbb{D}\mathcal{A} \rightarrow \mathbb{D}\mathcal{B}$. The second construction is “multiply” which allows any value of type $\mathbb{D}^2\mathcal{A}$ to be “squashed” or “averaged” back to a value of $\mathbb{D}\mathcal{A}$.⁴

We discuss them in turn, and then show how the “Kleisli composition” mentioned in §13.4.1 is defined in terms of them. It will be our sequential composition operator for abstract HMM's.

lifting – Given $f: \mathcal{A} \rightarrow \mathcal{B}$, its *lifting* $\mathbb{D}f: \mathbb{D}\mathcal{A} \rightarrow \mathbb{D}\mathcal{B}$ is defined so that for $\alpha: \mathbb{D}\mathcal{A}$ and $\beta: \mathbb{D}\mathcal{B}$ we have

$$\beta = (\mathbb{D}f)(\alpha) \quad \text{just when for all } b: \mathcal{B} \text{ we have } \beta_b = \sum_{\substack{a: \mathcal{A} \\ f(a)=b}} \alpha_a \quad .$$

The \mathbb{D} -lifting of f is also called its “push forward” in standard probability texts.

For example if π is the uniform distribution over $\{0, 1, 2\}$ and $f: \mathbb{N} \rightarrow \mathbb{N}$ takes n to $n \div 2$, where \mathbb{N} is the natural numbers, then $(\mathbb{D}f)(\pi) = 0_{2/3} \oplus 1$ because both 0's and 1's contribution of $1/3$ are taken to 0 by f , and they “add up” there. On the other hand, the only contribution to the probability of the final 1 comes from the initial 2.

multiply – We define the *multiply* function, written μ and of type $\mathbb{D}^2\mathcal{A} \rightarrow \mathbb{D}\mathcal{A}$, so that for $\Delta: \mathbb{D}^2\mathcal{A}$ we have

$$(\mu\Delta)_a := \sum_{\delta: [\Delta]} \Delta_\delta \times \delta_a \quad ,$$

which states that the probability assigned by distribution $\mu\Delta$ to any element a is the weighted-by-outer average of the probabilities assigned to a by each inner δ of Δ . We sometimes call this “average” because, applied to a hyper, it averages all the inners together according to their outer probability; and it is sometimes called “squash” because it squashes all the inners of Δ into one.

As an example of these definitions, we note that “lift f then average” is the same as taking the expected value of f as a random variable, i.e. that $\mathcal{E}_\alpha f = (\mu \circ \mathbb{D}f)(\alpha)$.⁵ (Recall that (\circ) is functional composition.)

⁴ These are characteristic properties associated with *monadic functors*, of which \mathbb{D} is an example.

⁵ Earlier, just after Cor. 4.8 in §4.4, we remarked that $\mu\Delta$ is the expected value of a hyper Δ . That's consistent with our remark here: on the one hand, we take the expected value of a distribution; on the other we take the expected value of a random variable over a distribution. They are the same if that random variable is the identity function, and so in the case above we find that

$$\mathcal{E}_\alpha \mathbf{Id} = (\mu \circ \mathbb{D}\mathbf{Id})(\alpha) = (\mu \circ \mathbf{Id})(\alpha) = \mu\alpha \quad .$$

The definitions above are instantiations of more general concepts from monadic functors.⁶ Remaining with our specific case of interest, however, namely the case $\mathcal{A} = \mathbb{D}\mathcal{X}$, we recall that we have met μ earlier. In Cor. 4.8 (p. 58 in Chap. 4) we noted that if we have a prior π and a channel C , then the weighted average of the inners of $[\pi \triangleright C]$ was the prior again — thus $\mu[\pi \triangleright C] = \pi$. The hyper shown at (4.2) on p. 56 comes from Example 4.2 where its generating prior π and channel C are given; and the weighted sum of the inners in (4.2) is indeed the prior given in Example 4.2 — that is, we have

$$\overbrace{1/4 \times \begin{pmatrix} 1/2 \\ 0 \\ 1/2 \end{pmatrix} + 1/3 \times \begin{pmatrix} 3/8 \\ 3/8 \\ 1/4 \end{pmatrix} + 7/24 \times \begin{pmatrix} 0 \\ 6/7 \\ 1/7 \end{pmatrix} + 1/8 \times \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}}^{\mu[\pi \triangleright C]} = \overbrace{\begin{pmatrix} 1/4 \\ 1/2 \\ 1/4 \end{pmatrix}}^{\pi}.$$

In our use of μ here to define Kleisli composition of abstract *HMM*'s, we will however go “one level further up” — instead of averaging a distribution of $\mathbb{D}\mathcal{X}$'s to get a single $\mathbb{D}\mathcal{X}$ again, we will average a distribution of $\mathbb{D}^2\mathcal{X}$'s to get a single $\mathbb{D}^2\mathcal{X}$ again. That is, our \mathcal{A} in the general description of multiply above will be instantiated to $\mathbb{D}\mathcal{X}$ rather than \mathcal{X} , so that μ will be of type $\mathbb{D}^3\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$.⁷

Definition 14.3

Given two abstract *HMM*'s A, B of type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$, their *Kleisli composition* is defined

$$(A; B)(\pi) := \mu(\mathbb{D}B(A(\pi))) .$$

More succinctly, we could write just $A; B := \mu \circ \mathbb{D}B \circ A$.

Note that lifting here is taking B in $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$ to $\mathbb{D}B$ in $\mathbb{D}^2\mathcal{X} \rightarrow \mathbb{D}^3\mathcal{X}$, and multiplying μ is therefore of type $\mathbb{D}^3\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$, as we mentioned above. Thus everything is “one level up” — the \mathcal{A} of our general presentation is in fact $\mathbb{D}\mathcal{X}$ here.

Note the reversed application order in the functional composition within Def. 14.3, i.e. with $\mathbb{D}B$ on the left: that has nothing to do with “Kleisli”. It is just that in sequential composition of programs the left-hand argument “happens first” whereas in composition of functions it is the right-hand argument that happens first.

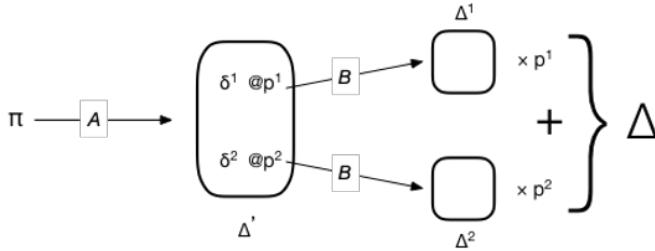
Putting the above informally into words, we would say that in applying $(A; B)$ to some prior π we first apply A “as is” to π , and this gives some intermediate hyper Δ' . In that Δ' are a number of inners δ^n each with its associated outer probability Δ'_{δ^n} — call that probability p^n . Then B is applied separately to those inners δ^n , giving for each a separate hyper Δ^n . That collection of hypers is then averaged together, or squashed, to produce a final hyper $\Delta := \sum_n p^n \times \Delta^n$ — that is, the outers p^n produced by A acting on π are used to average together the Δ^n 's produced by B acting on the inners δ^n that were produced by A .

An illustration of Kleisli composition is given in Fig. 14.3.

We now state the congruence of the concrete- and abstract definitions of sequential composition.

⁶ The lift-then-average construction, which we are using to define Kleisli composition, might seem ad hoc — but in fact it is not: see §14.7.

⁷ An advantage of the general monadic view is that, as here, we can use μ at any level.



The prior π produces hyper $\Delta' = A(\pi)$, which contains two inners $\delta^{1,2}$ with outers $p^{1,2}$ respectively. Then B takes each of $\delta^{1,2}$ to hypers $\Delta^{1,2}$, which are combined together with weights $p^{1,2}$ to produce a single final hyper $\Delta = (A; B)(\pi)$.

Figure 14.3 Kleisli composition of $A; B$ taking π to Δ

Theorem 14.4 (Sequential composition of abstract and concrete HMM's)

Let $A: \mathcal{X} \rightarrow (\mathcal{Y}^1 \times \mathcal{X})$ and $B: (\mathcal{Y}^2 \times \mathcal{X})$ be concrete HMM's, i.e. given as matrices. Then we have $\llbracket A; B \rrbracket = \llbracket A \rrbracket; \llbracket B \rrbracket$, where (\cdot) on the left is as in Def. 14.1 above and on the right is forward Kleisli composition as explained above.

Proof. We must show that for all π in $\mathbb{D}\mathcal{X}$ we have $\llbracket A; B \rrbracket(\pi) = (\llbracket A \rrbracket; \llbracket B \rrbracket)(\pi)$, where (\cdot) on the left is as in Def. 14.1 and on the right as in Def. 14.3. We do that by comparing the inners produced by each pair of observations $y^{1,2}$ from $\mathcal{Y}^{1,2}$, and then comparing the inners themselves x' -wise.

Thus we fix y^1, y^2, x' and argue on the left that

$$= \llbracket A; B \rrbracket(\pi)_{x'} = \sum_x \sum_{x''} \pi_x A_{x,y^1,x''} B_{x'',y^2,x'} \quad \text{“Def. 14.1”}$$

On the right we proceed in two stages. In the first, the “A” stage, only y^1 is used: we have

$$\begin{aligned} &= \llbracket A \rrbracket(\pi)_{x''} \\ &= \sum_x \pi_x A_{x,y^1,x''} \\ &= \pi''_{x''}, \end{aligned} \quad \begin{array}{l} \text{“Def. 14.1”} \\ \text{“say — note the dependence on } y^1\text{”} \end{array}$$

that is, a y^1 -indexed set of inners over the final state x'' of A. Each of those inners is taken separately as the prior for the second, the “B” stage, giving for a particular y^2 and final state x' the value

$$\begin{aligned} &= \llbracket B \rrbracket(\pi'')_{x'} \\ &= \sum_{x''} \pi''_{x''} B_{x'',y^2,x'} \quad \begin{array}{l} \text{“where } \pi''_{x''} \text{ is } \sum_x \pi_x A_{x,y^1,x''}” \\ \text{“Def. 14.1”} \end{array} \\ &= \sum_{x''} (\sum_x \pi_x A_{x,y^1,x''}) B_{x'',y^2,x'} \quad \text{“substitute for } \pi''_{x''}” \end{aligned}$$

which by rearrangement of summations equals the left-hand side. \square

14.4 Syntax and abstract-HMM semantics of *QIF*-programs⁸

In setting out the syntax here, we will use mainly (but not exactly) the notation of *pGCL*, which acronym stands for “*p*robabilistic *GC*ommand *L*anguage”.⁹ We add only two features:

1. A “PRINT” statement that describes information flowing out of a program, but not in the traditional way: it is not extracted from the final state, which remains hidden; rather it is (potentially) observed by an adversary “along the way”, as the program executes. It is our programming-language abstraction of “leaks” and “side channels”.
2. A distinction between two kinds of probabilistic choice (where *pGCL* has only one): “external” probabilistic choice is observable by the adversary, but “internal” probabilistic choice is not.¹⁰ External choice is indicated by “ $_p\boxplus$ ” between program fragments, whereas for internal choice we use “ $_p\oplus$ ” between expressions. In both cases the p is allowed itself to be an expression P , but very often is just a constant probability.¹¹

The point is not to attempt to “legislate” what the syntax of *QIF*-aware languages should be, and in fact we intend exactly the opposite. We want only a “vehicle for the description of (potentially highly sophisticated) abstract mechanisms” in the sense of Dijkstra’s language of guarded commands.

14.4.1 Probabilistic assignment

We write $x \in \text{Exp}$ for an assignment statement in which *Exp* is of type *distribution* over states — that is, is in $\mathbb{D}\mathcal{X}$ rather than simply \mathcal{X} of type state. No matter how we decide to write that distribution concretely (about which more below), we suppose here that it has somehow been converted to a (concrete) markov matrix M . Then we have

$$\llbracket x \in \text{Exp} \rrbracket(\pi) := [\pi M] ,$$

where πM is the distribution on \mathcal{X} resulting from taking π as a row vector, matrix multiplying by M , and taking the resulting row vector as a distribution δ' say in $\mathbb{D}\mathcal{X}$, and the $[-]$ converts that distribution to a point hyper on that distribution. That is, the result is $[\delta']$, which is a (point-)hyper on δ' , rather than just δ' on its own (which would be of the wrong type).

⁸ “There are two situations in which people behave like animals: one is when they are behind the wheel of a car; and the other is when they are discussing concrete syntax.”

Attributed to Robin Milner

⁹ To the authors’ knowledge, Dijkstra did not name his language, although others have called it “the guarded-command language”, sometimes abbreviated *GCL*. Once when asked for a name, he replied (paraphrased)

Let’s call it DOVSPL — “Dijkstra’s Own Very Simple Programming Language”.

¹⁰ In earlier non-quantitative information-flow work a similar distinction was made between external and internal *demonic* choice; some of that is explained in Chap. 17.

¹¹ In earlier work “ $_p\oplus$ ” is used for both kinds of choice, since the operands’ types –whether programs or expressions– make the intention unambiguous. Here, at least initially, we will use two different symbols to be especially clear.

As special cases of assignment we have

Conventional assignment — Here Exp is an expression denoting a single state, i.e. not a distribution, and what is meant implicitly is the point distribution on that state. In that case we use the conventional $\mathbf{x} := \text{Exp}$ for the assignment.

The corresponding markov matrix has exactly one 1 in each row.

Simple binary probabilistic assignment — Here Exp is of the form $\text{Exp1 } p \oplus \text{Exp2}$ with Exp1 and Exp2 expressions denoting single states and p a constant probability in $[0, 1]$.

The markov matrix has exactly one p and one $1-p$ in each row x , found in the columns corresponding to the two values Exp1 and Exp2 for that value x of \mathbf{x} . All other values in the matrix are zero.

Simple general probabilistic assignment — Here Exp is a more complicated expression involving multiple $p \oplus$'s and $q \oplus$'s and (as just above) expressions denoting single final states. For example $(0 \ 1/3 \oplus 1) \ 2/3 \oplus 2$ is the distribution where 1 has probability $(1-1/3) \times 2/3 = 4/9$, etc. A convenient abbreviation for a uniform distribution over a (small) number of values is for example $0 \oplus 1 \oplus 2$ for the uniform distribution on $\{0, 1, 2\}$.

The markov matrix here is the obvious generalization of the binary case, where each row has the same collection of probabilities, but in possibly different positions. In the example just above, the three probabilities would be $2/9$ and $4/9$ and $1/3$.

General probabilistic assignment — This generalizes all the above by allowing the probabilities themselves to depend on the state. Thus for example the assignment $\mathbf{x} := 0 \ (1+x)/3 \oplus 1$ on state space $\mathcal{X} = \{0, 1\}$ gives the markov matrix \mathbf{M} as

$$\mathbf{M}: \quad \mathcal{X} \left\{ \begin{array}{c} 0 \\ 1 \end{array} \right. \quad \underbrace{\left(\begin{array}{cc} 1/3 & 2/3 \\ 2/3 & 1/3 \end{array} \right)}_{\downarrow 0 \quad \downarrow 1} \quad .$$

14.4.2 Information flow via channels: leaking with PRINT

We write `PRINT Exp` for the statement that leaks information about the state \mathbf{x} to an adversary. The type of Exp is arbitrary and, in particular, need not be \mathcal{X} again — one might leak a *property* of \mathbf{x} rather than \mathbf{x} itself.

Just as for assignment, we interpret the right-hand side Exp as a matrix — but this time it is a (concrete) *channel* matrix with rows indexed by \mathcal{X} and columns indexed by \mathcal{Y} , where \mathcal{Y} is the type of Exp .

Then we have

$$[\![\text{PRINT Exp}]\](\pi) := [\pi \triangleright \mathbf{C}] ,$$

where \mathbf{C} is the channel matrix obtained by interpreting Exp in the way described for assignment statements. As usual $[-]$ makes a hyper from its argument, in this case the joint-distribution matrix $\pi \triangleright \mathbf{C}$.

Note that although Exp is of some type \mathcal{Y} , in fact *there need be no variable y , say, of that type in the program*. Instead the calculated value is “exported” from the program into the knowledge of the adversary.

As special cases of PRINT we have

The unit channel — Here Exp is constant, and the corresponding channel matrix has a single column of all 1's. As a program fragment it takes prior π to hyper $[\pi]$, and so is equivalent to the assignment $x := x$ or, even more succinctly, to the no-op `skip`. It is the unit abstract channel $\mathbb{1}$, which preserves secrecy.

The zero channel — Here Exp is just x , the state itself, or more generally an injective function of x , and the corresponding channel is O , with its matrix having a 1 in every column and leaking everything.

A general deterministic channel — Here Exp is an expression in x and, as for conventional assignment above, it is interpreted as a point distribution. The resulting channel matrix C comprises only 0's and 1's, and so is a deterministic channel (§4.1). Note that both the unit channel $\mathbb{1}$ and the zero channel O are deterministic.

General probabilistic channels — The Exp here, as for general probabilistic assignments, allows the probabilities themselves to depend on the state. Thus for example the channel PRINT ("Da" $\xrightarrow{(1+x)/3} \oplus$ "Nyet") on state space $\mathcal{X} = \{0, 1\}$ and observation space $\mathcal{Y} = \{"Da", "Nyet"\}$ gives the channel matrix C as

$$C: \quad \mathcal{X} \left\{ \begin{array}{c} 0 \\ 1 \end{array} \right. \quad \left(\begin{array}{cc} 1/3 & 2/3 \\ 2/3 & 1/3 \end{array} \right) \quad \mathcal{Y} \left\{ \begin{array}{c} \downarrow "Da" \\ \downarrow "Nyet" \end{array} \right.$$

14.4.3 External probabilistic choice

External probabilistic choice between two statements is written $\text{Stmt1} \xrightarrow{p} \boxplus \text{Stmt2}$, and represents a probabilistic choice between its two components Stmt1 and Stmt2 resolved in favor of Stmt1 with probability p , which itself can be an expression in the program variables. However in many cases the probability is just some constant p .

When p is some constant p — Here we have simply

$$\llbracket \text{Stmt1} \xrightarrow{p} \boxplus \text{Stmt2} \rrbracket(\pi) := \llbracket \text{Stmt1} \rrbracket(\pi) \xrightarrow{p} \llbracket \text{Stmt1} \rrbracket(\pi) ,$$

where (we recall) the weighted sum (\xrightarrow{p}) acts between distributions to make a new distribution. (Here those distributions are in fact hyper-distributions.) ¹²

¹² In contrast the $(p\oplus)$ we have been using until now for probabilistic choice is a convenient notation –somewhat abused– for making distributions from scalars. For example, if we had written $(p\oplus)$ on the right, above, we would have made a binary distribution on hyper-distributions — which is not even type correct.

When p is not constant — Given prior π , our first step is to calculate p^1 , the probability that $Stmt1$ is executed: it is the expected value of the expression p (considered as a function on \mathcal{X}) over π : thus if we write P for the function of \mathcal{X} that p denotes, we can define

$$\text{and similarly } \begin{aligned} p^1 &:= \mathcal{E}_\pi P \\ p^2 &:= \mathcal{E}_\pi(1-P) = 1 - \mathcal{E}_\pi P \end{aligned},$$

so that $p^1 + p^2 = 1$ of course. Note that $p^{1,2}$ are constants in $[0, 1]$ — they are not functions of the initial state x , as P is. But if $P(x)$ equals some p for all x , i.e. is a constant function, then indeed $p^1 = p$ as we would expect.

Our second step is to calculate π^1 , the effective prior for $Stmt1$ on its own — it is π conditioned by the case in which $Stmt1$ is chosen to be executed, and thus defined $\pi_x^1 := \pi_x P(x)/p^1$. Similarly we define $\pi_x^2 := \pi_x(1-P(x))/p^2$, and observe that $\pi = p^1\pi^1 + p^2\pi^2$, again as one would expect.

With that preparation, we have

$$[\![Stmt1 \ p \boxplus Stmt2]\!](\pi) := [\![Stmt1]\!](\pi^1) \ p^1 + [\![Stmt2]\!](\pi^2) .$$

14.4.4 (Internal probabilistic choice)

The syntactic difference between internal and external probabilistic choice is that the former occurs between *expressions* whereas the latter occurs between programs. And we have argued that the former conceals its outcome (as far as possible) whereas the latter reveals it.

Our semantics manifests this difference in the fact that internal probabilistic choice in fact does not really belong here in this section §14.4 at all. The same applies to conditional expressions, written e.g. IF n THEN $Exp1$ ELSE $Exp2$. In both cases, and in particular in the description of assignment and PRINT above, the right-hand side was first reduced to a matrix (M or C resp.) — and only then was the HMM-semantics given in terms of that matrix.

Were we to treat internal choice formally, here, we would have a subsidiary semantics that took expressions not to functions say of type $\mathcal{X} \rightarrow \mathcal{Z}$, where \mathcal{Z} is the type of the expression, but rather to functions of type $\mathcal{X} \rightarrow \mathbb{D}\mathcal{Z}$ where the (usual) probability-free semantics is recovered by specializing that to $\mathbb{D}\mathcal{Z}$'s having only point distributions on \mathcal{Z} . With that in hand, we could prove program-algebraic inequalities such as

$$\begin{aligned} (x := Exp1) \ p \boxplus (x := Exp2) &\sqsubseteq x \in (Exp1 \ p \oplus Exp2) \\ \text{and } (\text{PRINT } Exp1) \ p \boxplus (\text{PRINT } Exp2) &\sqsubseteq \text{PRINT } (Exp1 \ p \oplus Exp2) \end{aligned},$$

in both cases expressing that “external choice is refined by internal choice”.

For the reasons above, however, there is no internal-choice operator between statements: we cannot define a semantics of say $[\![Stmt1 \ p \oplus Stmt2]\!]$ in terms of $[\![Stmt1]\!]$ and $[\![Stmt2]\!]$. It is for similar reasons that we noted in Chap. 8 that we cannot define internal probabilistic choice between abstract channels: here that would be equivalent to giving a semantics for $[\![\text{PRINT } (Exp1 \ p \oplus Exp2)]\!]$ in terms of $[\![\text{PRINT } Exp1]\!]$ and $[\![\text{PRINT } Exp2]\!]$ — and as just remarked, that is not possible in general.

14.4.5 Sequential composition

Sequential composition of abstract *HMM*'s is simply their Kleisli composition:

$$\llbracket Stmt1; Stmt2 \rrbracket := \llbracket Stmt1 \rrbracket ; \llbracket Stmt2 \rrbracket .$$

On the left the “;” is the syntax being defined; on the right it is (forward) Kleisli composition. The role of Thm. 14.4 was to show that this is a definition that accords with operational intuition as supported by the concrete *HMM*'s from which *Stmt1* and *Stmt2* arose.

14.4.6 Conditional

The conditional

IF b THEN *Stmt1* ELSE *Stmt2*

is a special case of external probabilistic choice: define p to be the conditional expression IF b THEN 1 ELSE 0, and then take $\text{Stmt1} \parallel_p \text{Stmt2}$.¹³ Note that here b will usually vary with the initial state x , since otherwise the conditional is pointless (but still well defined); and so p will also vary over x , requiring that we use the more general form of external probabilistic choice. If b however did not vary, then it is either constant *true* or constant *false*, in which case the corresponding external choice would be either $0\parallel$ or $1\parallel$ — and then, from both points of view, one of the two branches could be discarded.

14.4.7 Iteration

So far we have assumed that all channels and program fragments terminate. However iteration requires a careful look at the issue of nontermination, i.e. how to model programs that have some nonzero probability of “looping forever”. As mentioned in §13.4.5, we'll provide a thorough treatment of nontermination in Chap. 16, and show there how it can impact information flow.

14.4.8 Local variables

We remarked earlier (§14.1.1) that it is possible to have different initial and final state spaces for an *HMM*; a typical instance of that is the following treatment of local variables, where we assume a global state space \mathcal{X} as usual. In conventional sequential programs, a block that introduces local variables, for example

```
{ VAR 1:L  
    Body  
}
```

- *Initialized uniformly.*
- *Can refer to both x and 1.*

can be seen as beginning with a “block entry” command of type $\mathcal{X} \rightarrow (\mathcal{X} \times \mathcal{L})$, which leaves x unchanged and initializes 1 to some value in its type. (The type “L” in the programming language denotes the set \mathcal{L} in the semantics.) In our quantitative case, conceptually the block-entry command would have type $\mathcal{X} \rightarrow (\mathcal{X} \times \mathcal{L})$ as a markov matrix, with which 1 is –let us say– initialized uniformly from \mathcal{L} .¹⁴ Made into an abstract pure markov *HMM*, it would have type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2(\mathcal{X} \times \mathcal{L})$.

¹³ This is not a circular definition: the conditional expression is conventional mathematics, containing no probabilities.

¹⁴ We will never use un-initialized local variables, so it does not matter what the initialization is.

Then the command Body would execute over the extended state space, having (abstract) type $\mathbb{D}(\mathcal{X} \times \mathcal{L}) \rightarrow \mathbb{D}^2(\mathcal{X} \times \mathcal{L})$.

And finally the corresponding “exit block” command would be of type $(\mathcal{X} \times \mathcal{L}) \rightarrow \mathcal{X}$ as a markov, and $\mathbb{D}(\mathcal{X} \times \mathcal{L}) \rightarrow \mathbb{D}^2 \mathcal{X}$ as an abstract HMM. It “projects away” the local variable, leaving only the original x .

Using matrices, i.e. staying concrete, we can illustrate this easily over small state spaces. Let $\mathcal{X} = \{x_1, x_2\}$ and $\mathcal{L} = \{\ell_1, \ell_2\}$. Then we have two markov matrices

$$\text{block entry: } \mathcal{X} \left\{ \begin{array}{c} x_1 \\ x_2 \end{array} \right. \quad \overbrace{\left(\begin{array}{cccc} 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 1/2 & 1/2 \end{array} \right)}^{\mathcal{X} \times \mathcal{L}}$$

$\downarrow (x_1, \ell_1)$
 $\downarrow (x_1, \ell_2)$
 $\downarrow (x_2, \ell_1)$
 $\downarrow (x_2, \ell_2)$

that introduces the local 1 and initializes it uniformly, and

$$\text{block exit: } \mathcal{X} \times \mathcal{L} \left\{ \begin{array}{c} (x_1, \ell_1) \\ (x_1, \ell_2) \\ (x_2, \ell_1) \\ (x_2, \ell_2) \end{array} \right. \quad \overbrace{\left(\begin{array}{cc} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{array} \right)}^{\mathcal{X}}$$

$\downarrow x_1$
 $\downarrow x_2$

which are turned into pure-markov HMM’s by introducing the unit observation type $\mathcal{Y} = \{\bullet\}$ as in §14.2.1, revealing nothing.

As a further example, we return to the issue of describing “channel” cascade at the abstract level, using the Dalenius scenario from Chap. 10 as motivation.

Recall that the Dalenius scenario involves a channel C on a secret X and a correlation J of type $\mathbb{D}(\mathcal{Z} \times \mathcal{X})$ between X and some other secret Z whose values are drawn from a set \mathcal{Z} . In our earlier treatment (from §10.1), correlation J was converted to a joint-distribution matrix J , which was then factored into a prior ρ on Z and a concrete channel $B: \mathcal{Z} \rightarrow \mathcal{X}$ from Z to X , so that $J = \rho \triangleright B$. The B was then matrix-multiplied with a concrete realization $C: \mathcal{X} \rightarrow \mathcal{Y}$ of (abstract) channel C to make an overall channel BC between Z and Y . And that multiplication is what cannot be done at the abstract level, i.e. with abstract versions B, C of types $\mathbb{D}\mathcal{Z} \rightarrow \mathbb{D}^2\mathcal{Z}$ and $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$ respectively.

The reason for that abstraction failure –given earlier– is that in the abstract type $\mathbb{D}\mathcal{Z} \rightarrow \mathbb{D}^2\mathcal{Z}$ of B “the labels on B ’s output have been lost”, and so there is no way to connect them to the inputs of C . But a related view, one not using concrete artifacts such as labels, is that the operation of cascading *hides* the outputs of B . Thinking of cascading as “joining two wires together”, we could say that before the cascade the values “coming out of the B wire” were visible, but afterwards they are hidden “because B has been plugged into C ”.

That kind of hiding is something we *can* do with local variables, but only if J is interpreted as a markov rather than a channel: that is B takes secret Z and makes secret X from it, according to some (internal choice) probabilistic assignment, and then C –as before– leaks information about that X . The result is that an adversary can determine a posterior in Z by reasoning about the posterior of the X that Z was used to construct. Here is the program, the Dalenius scenario abstractly:

```

VAR z:Z           - Global variable.
    “initialize z according to prior  $\rho$  in  $\mathbb{D}\mathcal{Z}$ ”

{ VAR x:X
    “assign to x based on matrix B in  $\mathcal{Z} \rightarrow \mathcal{X}$ ”      - A markov.
    “leak x based on matrix C in  $\mathcal{X} \rightarrow \mathcal{Y}$ ”          - A channel.
}

```

Thus we see that the key to this representation is first that the B is interpreted as a *markov*, not as a channel; and second, that B 's output is never observable directly by the adversary — instead it is stored in local (hidden) variable x in the program, which is then leaked according to C . Continuing the metaphor above, we could call variable x the connection between the two wires.

The types involved are these:

initialize z	$\mathbb{D}\mathcal{Z}$	ρ is prior distribution on Z
enter local block	$\mathbb{D}\mathcal{Z} \rightarrow \mathbb{D}^2(\mathcal{Z} \times \mathcal{X})$	
assign to x based on B	$\mathbb{D}(\mathcal{Z} \times \mathcal{X}) \rightarrow \mathbb{D}^2(\mathcal{Z} \times \mathcal{X})$	z read but not changed, x changed
leak x based on C	$\mathbb{D}(\mathcal{Z} \times \mathcal{X}) \rightarrow \mathbb{D}^2(\mathcal{Z} \times \mathcal{X})$	x read, z not read, nothing changed
leave scope	$\mathbb{D}(\mathcal{Z} \times \mathcal{X}) \rightarrow \mathbb{D}^2\mathcal{Z}$	x discarded, z not changed

The types above are all consistent with sequential (Kleisli) composition, each with the next; and from that we can see that the overall type is $\mathbb{D}\mathcal{Z} \rightarrow \mathbb{D}^2\mathcal{Z}$, i.e. a (Dalenius) leak of z via a channel C acting on local variable x — and all of that is expressed at the abstract level.

14.5 Leaks caused by conditionals and by external choice

We return now to the theme of §13.4.3, where we argued informally that conditionals must leak their conditions; and indeed the semantics we set out here does have that property. But why?

We begin, still informally, with the observation that external probabilistic choice must leak “which branch was taken”. This too was discussed earlier, in §13.5, where a distinction was drawn between internal and external probabilistic choice. We now look at that more rigorously, on the basis of the semantics we defined above.

The programs we discussed earlier were

$$\text{PRINT } b_{1/3} \boxplus \text{PRINT } \neg b \quad \text{and} \quad \text{PRINT } (b_{1/3} \oplus \neg b) ,$$

and it was argued that the left-hand program's external choice leaks b exactly, whereas the right-hand program's internal choice does not.

The left-hand program leaks b exactly, we argue, because the adversary –knowing which branch was executed– knows also therefore whether to negate the observed value or to take it as it is for the value of b . Suppose that \mathcal{X} is $\{\text{true}, \text{false}\}$ and that the prior π is $\text{true}_p \oplus \text{false}$. The channel matrices $C_{L,R}$ induced by $\text{PRINT } b$ and $\text{PRINT } \neg b$ are in that case

$$\begin{array}{ccc} \overbrace{\mathcal{Y}_L}^{\text{Boolean}} & = & \overbrace{\mathcal{Y}_R}^{\text{Boolean}} \\ \downarrow \text{true} & & \downarrow \text{true} \\ \downarrow \text{false} & & \downarrow \text{false} \\ C_L: \quad \mathcal{X} \left\{ \begin{array}{ll} \text{true} & \left(\begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array} \right) \\ \text{false} & \end{array} \right. & C_R: \quad \mathcal{X} \left\{ \begin{array}{ll} \text{true} & \left(\begin{array}{cc} 0 & 1 \\ 1 & 0 \end{array} \right) \\ \text{false} & \end{array} \right. , \end{array}$$

where the difference is only that on the left the 1's-diagonal goes one way and on the right it goes the other. And from the left we see that $\llbracket \text{PRINT } b \rrbracket(\pi)$ is the hyper $[\text{true}]_p \oplus [\text{false}]$, expressing that with probability p the adversary will know for certain that b is *true*, and with $1-p$ that it is certainly *false*. (Recall that the p comes from the prior π .)

Now, on the right, a resolutely formal¹⁵ calculation for $\llbracket \text{PRINT } \neg b \rrbracket(\pi)$ will give “instead” the hyper $[\text{false}]_{1-p} \oplus [\text{true}]$ — which however is of course the same hyper as for the other case. When they are combined with $(\text{1/3}+)$, as the semantics (§14.4.3) of external choice requires, the overall result is

$$([\text{true}]_p \oplus [\text{false}]) \text{ 1/3}+ ([\text{false}]_{1-p} \oplus [\text{true}]) ,$$

which is just $[\text{true}]_p \oplus [\text{false}]$ of course, since both sides of the $(\text{1/3}+)$ are equal. (And the result would be the same for any other external-choice probability.) Thus we have shown in fact that

$$(\text{PRINT } b) \text{ 1/3} \oplus (\text{PRINT } \neg b) = \text{PRINT } b = \text{PRINT } \neg b ,$$

which is an equality between *programs* — and a simple example of program algebra. The equality shows that an external choice between leaking b and leaking $\neg b$ is just as bad as, indeed is equal to just leaking “just b ” or “just $\neg b$ ” — as long as the observer knows which kind of leak it is.

But now we turn to the right-hand program: its channel matrix is

$$\begin{array}{ccc} \overbrace{\mathcal{Y}}^{\text{Boolean}} & = & \text{Boolean} \\ \downarrow \text{true} & & \\ \downarrow \text{false} & & \\ C: \quad \mathcal{X} \left\{ \begin{array}{ll} \text{true} & \left(\begin{array}{cc} 1/3 & 2/3 \\ 2/3 & 1/3 \end{array} \right) \\ \text{false} & \end{array} \right. , & & \end{array}$$

and so —remembering that π is $\text{true}_p \oplus \text{false}$ — we have that $\llbracket \text{PRINT } (\text{b 1/3} \oplus \neg b) \rrbracket(\pi)$ is the hyper

$$(\text{true}_{p/2-p} \oplus \text{false}) \text{ 2-p/3} \oplus (\text{true}_{2p/1+p} \oplus \text{false}) , \quad (14.3)$$

which actually is something quite complicated arithmetically but —in any case— does not reveal the exact value of b as the external choice did.

¹⁵ By “resolutely formal” is meant strictly applying formal reasoning, as an automated prover might: a human might for example swap the arguments of $1-p \oplus$ for neatness; but a machine would not.

But the detail of (14.3) just above makes another point, actually a “meta point” — that sometimes that kind of detail is not of much use. What we have seen here in more abstract terms is that

$$(\text{PRINT } b)_{1/3} \boxplus (\text{PRINT } \neg b) \sqsubseteq \text{PRINT } (b_{1/3} \oplus \neg b),$$

and that might in fact be all that we care about. If our specification uses external choice, we can implement it as internal choice *without* having to calculate exactly what leakage that internal choice exhibits: it can’t be worse than the specification’s leakage anyway and, after all, the customer has already agreed to that specification. That further example of program algebra is an instance of the *general* algebraic property that external choice is refined by internal choice, something that we can use in stepwise refinement of QIF-programs from specifications to implementations.

We noted earlier that conditionals leak their conditions, again informally. Since we here *define* conditionals in terms of external probabilistic choice, that is to be expected. A way of going in the other direction however, i.e. from leakage of conditionals to leakage of external choices, would be (again informally) to note that we would expect the following algebraic equality to hold, where we use local variables (as introduced in §14.4.8):

$$\begin{aligned} & Stmt1 _p \boxplus Stmt2 \\ = & \{ \text{VAR } b : \in \text{ TRUE } _p \oplus \text{ FALSE} & - \text{ Internal choice of } b \dots \\ & \quad \text{IF } b \text{ THEN } Stmt1 \text{ ELSE } Stmt2 & - \dots \text{ leaked here.} \\ & \} \quad . \end{aligned}$$

If that is indeed so, then the conditional’s “IF b ” leaking b agrees with the leaking of the branch-taken in the external probabilistic choice. (See also Exercise 14.2.)

14.6 Examples of small QIF programs

In this section we provide some small examples of using our programming language.

14.6.1 First example: Bertrand's Boxes

Our first example is taken from the well-known problem of drawing a colored ball randomly from one of three labeled boxes.

Let secret x represent one of three boxes containing balls which are colored white or black according to the following pattern: Box 0 contains two black balls; Box 1 contains one black and one white ball; and Box 2 contains two white balls. Consider now the experiment where we choose x uniformly, and then give the adversary a ball uniformly chosen from that box: she observes the color of the selected ball, but not which box it came from.

The question is “What does she learn about x itself?”

With our programming language we can represent the experiment succinctly as follows, where we are using (program) variable x to represent the secret; thus \mathcal{X} is $\{0, 1, 2\}$. We have

$$\begin{aligned} & x : \in 0 \oplus 1 \oplus 2 & - \text{ Choose } x \text{ uniformly from } \{0, 1, 2\}. \\ & \text{PRINT "white"}_{z/2} \oplus \text{"black"} & (14.4) \end{aligned}$$

The first statement is the original assignment to x of the secret Box — a uniform choice over $\{0, 1, 2\}$. The second statement is a leak statement that “outputs” an observation based on which of the boxes was selected by the assignment. Observe that the probabilistic choice $x/2$ is inside an expression, and so is internal in the sense that x is not observed explicitly; but still the observer will learn something about x from seeing “white” or “black”. If $x=0$ then there is no chance that a white ball will be selected; if $x=1$ then there is an equal probability of selecting white or black; and finally, if $x=2$ then it is certain that a white ball will be selected.

A traditional channel analysis would construct the (abstract) channel explicitly, and answer the question by determining the Bayes vulnerability (or equivalently Bayes risk) of x , the state. With the flexibility of a programming language, an alternative would be to prove equality between the program above at (14.4) and this program:

$$\begin{array}{l} \text{PRINT "black"; } x \in 0_{2/3} \oplus 1 \\ \frac{1}{2} \boxplus \text{PRINT "white"; } x \in 1_{1/3} \oplus 2 \end{array} \quad .^{16} \quad (14.5)$$

Program (14.5) is a nesting of two program operators: the outer probabilistic choice $(\frac{1}{2} \boxplus)$ is observable, i.e. *external* because it is not contained in an expression, but although the result cannot be predicted its resolution can be observed. In fact we have included PRINT statements to make that clear. The second probabilistic choice however, on the right-hand side of the two assignments $x \in \dots$, is *internal*, and in the semantics corresponds to a markov assignment.

What is interesting here is that these two programs Prog. (14.4) and Prog. (14.5) achieve exactly the same thing both in terms of the adversary’s knowledge of the final value of x and in terms of the assignment to x ’s value; but the process by which they do so is quite different. The first program (14.4) proceeds in a traditional information-flow sequence and matches the informal description of the experiment: first the prior is established and then some information about it is leaked. The second program (14.5)’s execution sequence however first randomly selects a color and only then assigns a value to the secret variable x . In fact (14.5) summarizes precisely the information available to the adversary observing (14.4) and describes very directly the two posteriors and the probabilities with which they occur in the output hyper-distribution of (14.4).

Often the scenario is posed as a puzzle:

Bertrand’s Boxes

There are three boxes: one contains two white balls; one contains one white and one black; and one contains two black balls. A box is chosen uniformly at random, and a ball drawn randomly from it. If the ball is white, what is the probability the other ball in that box is white as well?

Program (14.4) describes just the scenario of the puzzle, operationally: in that sense, it is “the question”. And Prog. (14.5) is the answer (given that it is equal to Prog. (14.4)) — because from Prog. (14.5) you can simply read off that if a white ball is drawn, the *rhs* of the probabilistic choice was taken and, in that case, the probability that $x=2$ (i.e. that the other ball is white) is explicitly given as $2/3$.

¹⁶ The $(\frac{1}{2} \boxplus)$ is the probability *before* a box is chosen that a black or a white ball will be seen once the procedure has been followed.

In fact we can go one step further, to a rather startling conclusion: since `PRINT` of a constant is equivalent to `SKIP`, Prog. (14.5) is actually equivalent to the simpler

$$x \in 0_{2/3} \oplus 1 \quad \frac{1}{2} \boxplus \quad x \in 1_{1/3} \oplus 2 \quad (14.6)$$

in which the *only* leak is that the observer knows which branch of the $(\frac{1}{2} \boxplus)$ -choice was taken: we have abstracted completely from *how* she knows. (For example, the boxes could contain frogs and toads instead of colored balls: the information flow would be the same.) But she knows nonetheless.

From a security point of view (rather than as a puzzle), the program equality (14.5) = (14.6) tells us that the adversary will be able to guess the color of the other ball with probability $2/3$ — that is, the posterior Bayes vulnerability of x is $2/3$. And indeed, as certifiers, that might be what we want to know first of all: it's the “executive summary” of this system's security.

14.6.2 Second example: Goldfish or piraña?

An opaque goldfish bowl contains a fish which could with equal probability be either a goldfish or a piraña. A (new) piraña is added to the bowl, which now therefore contains two fish. Then a fish is scooped out at random (before one eats the other), and an observer notes which type of fish it is. What is the probability that the original fish was a piraña if the fish that is removed is a piraña?

The answer to this question can be found in the same style as in the previous example. First we describe the situation with the following program:

$$\begin{aligned} & \text{fish1} \in \text{"goldfish"} \quad \frac{1}{2} \oplus \quad \text{"piraña"} \\ & \text{fish2} := \text{"piraña"} \\ & \text{PRINT fish1 } \frac{1}{2} \oplus \text{fish2} \quad . \end{aligned} \quad (14.7)$$

Just as above, we can rewrite it equivalently as a program that describes the final hyper-distribution, where the (internal) assignments to the `fish` variables are made only *after* an observable probabilistic choice:

$$\begin{aligned} & \text{PRINT "goldfish"; fish1 := "goldfish"} \\ & \quad \text{fish2 := "piraña"} \\ & \frac{1}{4} \boxplus \quad \text{PRINT "piraña"; fish1} \in \text{"goldfish"} \frac{1}{3} \oplus \text{"piraña"} \\ & \quad \text{fish2 := "piraña"} \quad . \end{aligned} \quad (14.8)$$

With fragment (14.8), we can answer the original question very easily. A piraña is observed in the second branch —which occurs with probability $3/4$ — but in that case the observer deduces that the first fish was a piraña with probability $2/3$, corresponding to the posterior probability in this case. (Again, the `PRINT` statements make no difference.)

The question remains though “How do we prove such program equivalences?” In fact both this and the previous example are making use of a simple rule that can be expressed quite nicely using programs. When Exp has two possible values, a `PRINT Exp` statement corresponds to a channel with two possible observations.

And in that case the following equivalence holds between program fragments. Let the prior be π and suppose that the statement `PRINT Exp` leaks either \top or \perp , i.e. that Exp takes only those two values as x ranges over \mathcal{X} . Then let $\pi|\top$ be the prior π

conditioned on *Exp*'s value being \top , and similarly for \perp , and let p be the probability that *Exp* takes the value \top over π . Then we have the program equality

$$x \in \pi; \text{ PRINT } Exp = x := \pi^\top_p \boxplus x := \pi|_\perp ,$$

in which $x \in \pi$ is playing the role of a prior and `PRINT Exp` has the role of a two-column channel applied to that prior. It might seem odd that the left-hand side contains a `PRINT` statement but the right- does not: but in fact *both* sides leak (and by the same amount). On the right, the leak is caused by the external choice.

Indeed the posteriors π^\top and $\pi|_\perp$ are the two inners of the resulting hyper, and its outer is $(- \underset{p}{\boxplus} -)$. That also makes clear how the two-level structure of a hyper represents both external and internal probabilistic choice: the outer is external choice between the inners; and the inners are internal choice over the state.

The equality makes very clear the current status of knowledge about `fish1` in the original experiment, because the assignments are exactly the posteriors. If a goldfish is observed then `fish1` must also be a goldfish, but if a piraña is observed it is twice as likely also to be a piraña as a goldfish.

14.6.3 Third example: Repeated independent runs

We can also describe very straightforwardly the situation of repeated independent runs. Let x be a variable that is assigned uniformly one of three possible numbers p_1 , p_2 , or p_3 , which will later be interpreted as probabilities. Observations are made now based on the specific value assigned to x , so that 0 is observed with probability x and 1 is observed with probability $1-x$. A single observation can thus be carried out by executing the statement `PRINT 0 x⊕ 1`, and n independent executions of that is therefore the execution of `PRINT 0 x⊕ 1` composed sequentially n times.

Let us write $P^{(n)}$ for that composition. Observe that we have used sequential composition here rather than parallel composition, because parallel composition is defined only between channels (i.e. not between programs in general, which might make updates). But, as established in §13.4.1(2) and §14.2.2, the effect is the same when the programs are pure channels (as they are here).¹⁷

By definition $P^{(n+1)} = P^{(n)}; \text{ PRINT } 0 x \oplus 1$ and, since sequential composition is monotone (recall Def. 9.16) with respect to refinement, we have the following refinements:

$$\dots \sqsubseteq P^{(n+1)} \sqsubseteq P^{(n)} \sqsubseteq P^{(n-1)} \sqsubseteq \dots \sqsubseteq P^{(0)} = \text{SKIP} .$$

It can be shown that the greatest lower bound of the anti-refinement chain is the program `PRINT x` that leaks *everything* about x . (See Exercise 14.5.)

14.7 Underlying and unifying structures: a summary

The semantics of our *QIF*-aware programming language, its embedding of both markov transitions and channels, is based on quite general structures, as we now explain. An example of its utility is given for example by Exercise 15.5 to come.

The “type constructor” \mathbb{D} that we have used so extensively in this work is widely recognized in the more theoretical community as the *probability monad* of Lawvere and Giry, and (other than probabilistic) monads themselves are more general still. Any monad \mathbb{M} has (by definition) two polymorphic functions, natural transformations *unit*,

¹⁷ When we have loops in the language we will be able to write this very compactly.

written η , and *multiply* written μ , that interact in elegant and algebraically powerful ways. Indeed many of the equalities we exploit in these chapters are true in monads generally, and we are simply using their \mathbb{D} -instantiations.¹⁸ In this summary section, we show (or recall) how this general structure underlies and motivates what we have done.

- (1) The unit η of the probability monad \mathbb{D} takes an element x in some \mathcal{X} to the point distribution η_x in $\mathbb{D}\mathcal{X}$, which we have been writing $[x]$. Similarly, unit (being polymorphic) takes a distribution π in $\mathbb{D}\mathcal{X}$ to the point hyper $[\pi]$ in $\mathbb{D}^2\mathcal{X}$ –so that $\eta\pi = [\pi]$ — and thus the abstract channel corresponding to the “leak nothing” channel $\mathbb{1}$ is again η but now on the base type $\mathbb{D}\mathcal{X}$ which (making its polymorphism explicit) is sometimes for precision written $\eta_{\mathbb{D}\mathcal{X}}$.
- (2) The multiply μ of \mathbb{D} takes a distribution of distributions on \mathcal{X} , for us a hyper in $\mathbb{D}^2\mathcal{X}$, back to a simple distribution in $\mathbb{D}\mathcal{X}$. We have in some cases called it “squash” (§14.3.1). An example of its use is the fact that $\mu[\pi \triangleright C] = \pi$, i.e. that the prior π that made a hyper $[\pi \triangleright C]$ by being “pushed through” a channel C (Def. 4.6), whether abstract or concrete, can be recovered by taking the squash (multiply μ) of the hyper (Cor. 4.8).

It also underlies the definition of sequential composition: see (4) below.

- (3) The application of \mathbb{D} as a functor to some function $f: \mathcal{A} \rightarrow \mathcal{B}$, written $\mathbb{D}f$, is a general way of constructing a “lifted” function $\mathbb{D}f: \mathbb{D}\mathcal{A} \rightarrow \mathbb{D}\mathcal{B}$, and it is known as the “push forward” in the probability literature. One example of its use is that the “leak everything” channel \mathbb{O} is $\mathbb{D}\eta$ where (compare (1) above) here η is on base type \mathcal{X} (more precisely $\eta_{\mathcal{X}}$) — and thus \mathbb{O} takes a distribution on states x to the same distribution on point distributions $[x]$.
- (4) To compose two programs P, Q of type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$, we use lifting to form $\mathbb{D}Q$ of type $\mathbb{D}^2\mathcal{X} \rightarrow \mathbb{D}^3\mathcal{X}$, i.e. the push forward of Q from (3) just above, and then compose that with P to get a function $(\mathbb{D}Q) \circ P$ of type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^3\mathcal{X}$. Then we use multiply to squash the $\mathbb{D}^3\mathcal{X}$ back to $\mathbb{D}^2\mathcal{X}$ so that overall we have the correct type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$ for the sequential composition

$$\mu \circ \mathbb{D}Q \circ P$$

of abstract-HMM programs P and Q . It is a general construction known as *Kleisli composition*.

A simple example of Kleisli composition (but not well known as such) is the multiplication of Markov matrices when presented row-wise, i.e. as functions of type $\mathcal{X} \rightarrow \mathbb{D}\mathcal{X}$. To multiply matrices P, Q of type $\mathcal{X} \rightarrow \mathbb{D}\mathcal{X}$, convert the second to $\mathbb{D}Q$ of type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$, compose it functionally with the first to get $\mathbb{D}Q \circ P$ of type $\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$, and compose all of that with μ to get $\mu \circ \mathbb{D}Q \circ P$, recovering the original type $\mathcal{X} \rightarrow \mathbb{D}\mathcal{X}$ again.

Furthermore, we can use the above ideas to convert a Markov matrix $M: \mathcal{X} \rightarrow \mathbb{D}\mathcal{X}$ to a markov program-fragment $\llbracket M \rrbracket$ of type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$, i.e. an abstract HMM in our programming model. (We did that in more direct terms in §14.4.1.) First M is converted to $\mu \circ \mathbb{D}M$ of type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}\mathcal{X}$, and then the result of that is converted into a point hyper using unit, to give $\eta \circ \mu \circ \mathbb{D}M$ overall, of type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$ as required, and taking its distribution argument π to the hyper-distribution $[\pi']$ where π' is the final distribution that results from applying M as a Markov matrix to the initial distribution π .

¹⁸ Thus some of the direct proofs we have given in this text were included simply to be self-contained.

- (5) As an example of the benefits of generality, we note that all the above apply equally well to the *demonic* semantics that we will see in Chap. 17 — except that there we use the powerset monad \mathbb{P} instead of the distribution monad \mathbb{D} . Then η becomes the function that makes a singleton set, and μ squashes a set of sets back to a single set by taking the distributed union. Sequential composition of two demonic programs P, Q is then $\mu \circ \mathbb{P}Q \circ P$, again Kleisli composition; and Exercise 14.7 shows that in monadic form the definitions of \mathbf{O} and $\mathbf{1}$ are also the same as in the probabilistic case.¹⁹

Beyond the insights above, applying mainly to our general programming model, even when considering channels in isolation we find similar conceptual advantages. For example, many of the axiomatic properties of channels can be given very concise and elegant formulations in this general setting. Here are some examples, where we recall from Chap. 11 that we write $\mathbb{V}: \mathbb{D}\mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$ for a general prior vulnerability (11.1) and $\hat{\mathbb{V}}: \mathbb{D}^2\mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$ for a general posterior vulnerability (11.2).

- (6) The averaging axiom **AVG** from Def. 11.9 becomes $\hat{\mathbb{V}} = \mu \circ \mathbb{D}\mathbb{V}$.
- (7) The noninterference axiom **NI** from Def. 11.6 becomes $\hat{\mathbb{V}} \circ \eta = \mathbb{V}$. Assuming (6), that follows from the general monad laws $\mu \circ \eta = 1$ and $\mathbb{D}\mathbb{V} \circ \eta = \eta \circ \mathbb{V}$.
- (8) The convexity axiom **CVX** from Def. 11.2 becomes $\mathbb{V} \circ \mu \leq \mu \circ \mathbb{D}\mathbb{V}$.

And the proof of Prop. 11.12 for example can now be given in more abstract terms. It was that

If a pair of prior/posterior vulnerabilities $(\mathbb{V}, \hat{\mathbb{V}})$ satisfies **AVG** and **MONO**, then it also satisfies **CVX**.

We calculate

CVX	
is	$\mathbb{V} \circ \mu \leq \mu \circ \mathbb{D}\mathbb{V}$
iff	$(\mathbb{V} \circ \mu)(\Delta) \leq (\mu \circ \mathbb{D}\mathbb{V})(\Delta)$ for all Δ
iff	$(\mathbb{V} \circ \mu)(\Delta) \leq \hat{\mathbb{V}}(\Delta)$
iff	$(\mathbb{V} \circ \mu)[\pi \triangleright C] \leq \hat{\mathbb{V}}[\pi \triangleright C]$
iff	$\mathbb{V}\pi \leq \hat{\mathbb{V}}[\pi \triangleright C]$
is	MONO ,

“defined just above”
“pointwise (\leq)”
“**AVG** from (6) above”
“can always write Δ as $[\pi \triangleright C]$ for some π, C ”
“from (2) above”
“Def. 11.8”

which in fact establishes the stronger $\mathbf{AVG} \Rightarrow (\mathbf{CVX} \equiv \mathbf{MONO})$, and is indeed borne out by Fig. 11.1.

- (9) A consequence of accepting averaging (6) is that $\hat{\mathbb{V}}(\Delta^1_p + \Delta^2) = \hat{\mathbb{V}}(\Delta^1)_p + \hat{\mathbb{V}}(\Delta^2)$, i.e. linearity of $\hat{\mathbb{V}}$, where $_p+$ takes the p -weighted sum of its operands: on the left we sum over hypers; on the right we sum over scalars. That is more generally

$$\hat{\mathbb{V}}(\mu\Delta) = \mu((\mathbb{D}\hat{\mathbb{V}})\Delta) \quad \text{or more succinctly} \quad \hat{\mathbb{V}} \circ \mu = \mu \circ \mathbb{D}\hat{\mathbb{V}},$$

where $\underline{\Delta}$ is in $\mathbb{D}^3\mathcal{X}$, a distribution of hypers which –with multiply– thus generalizes linear combinations of hypers, another monad law when $\hat{\mathbb{V}} = \mu \circ \mathbb{D}\mathbb{V}$.

¹⁹ Another example of generality, of which we are taking advantage, is functional programming — where an example is the `list` type-constructor. Multiply μ is `concat` and unit η is “make a singleton list” (`:[]`). Lifting is `map`, i.e. the function that converts some $f: a \rightarrow b$ to `map f` of type `[a] → [b]`. Kleisli composition combines a function of type `a → [b]` with a function of type `b → [c]` to make a function of type `a → [c]` — the first “lifted” step gives a list of lists; and the multiply then `concat`s that to a single list again.

- (10) The space $\mathbb{D}^3\mathcal{X}$, introduced in (9) just above, as remarked a generalization of weighted sum, is a distribution on *hypers*, just as a hyper in $\mathbb{D}^2\mathcal{X}$ is a distribution on distributions. It allows us to give a “hyper native” definition of structural refinement (\sqsubseteq_\circ) on hypers, corresponding to our current definition that was induced from the structural definition on channel matrices Def. 9.6, i.e. that $\Delta^1 \sqsubseteq \Delta^2$ just when $\widehat{\mathbb{V}}\Delta^1 \geq \widehat{\mathbb{V}}\Delta^2$ for all $\widehat{\mathbb{V}}$ satisfying the axioms: it is that $\Delta^1 \sqsubseteq \Delta^2$ just when there is a $\underline{\Delta}$ such that $\Delta^1 = \mu\underline{\Delta}$ and $\Delta^2 = (\mathbb{D}\mu)\underline{\Delta}$. That formulation allows soundness of (\sqsubseteq), i.e. that it can only decrease g -vulnerability (increase ℓ -uncertainty), to be shown even for infinite state spaces \mathcal{X} and general measures.
- (11) Finally, the monadic structure coupled with the Kantorovich metric gives us continuity criteria not only for \mathbb{V} but also for $\widehat{\mathbb{V}}$. By giving the underlying \mathcal{X} the discrete metric, i.e. that $\text{dist}(x_1, x_2) = (0 \text{ if } x_1=x_2 \text{ else } 1)$, the Kantorovich-induced distance on $\mathbb{D}\mathcal{X}$ is equivalent to the Manhattan metric, the notion used in axiom CNTY.

But the great generality of the monadic construction then goes further, giving us that axiom AVG, i.e. that $\widehat{\mathbb{V}} = \mu \circ \mathbb{D}\mathbb{V}$, makes $\widehat{\mathbb{V}}$ continuous as well, but this time with respect to the Kantorovich metric on *hypers*. That in turn allows higher-order calculations that limit information flow in a very robust way.

14.8 Exercises

Exercise 14.1 Recall program fragments (14.7) and (14.8). Use the interpretation of the PRINT statement to compute the final hyper-distribution of (14.7). Use the semantics for internal and external probabilistic choice to compute the final hyper-distribution for (14.8), and hence deduce that the two program fragments are equivalent. \square

Exercise 14.2 In §13.4.3 (and later §14.5) we discussed the fact that conditionals “necessarily” leak their conditions, calling it *super-implicit flow*. (The same applies to iterations, since they can be unfolded.) But *why* is that necessary? The mathematical answer is that it is forced by the type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$ that we are using and the requirement that all constructs –thus IF in particular– are (\sqsubseteq)-monotonic.

But surely (one could argue) it is possible in real life to have IF’s that *don’t* leak, provided special measures are taken. Can we model those special provisions with our HMM’s while preserving our model’s essential mathematical property of monotonicity wrt. refinement?

Yes — it can be done by treating a whole block of code as “markov atomic” as far as QIF is concerned; semantically we would model that by taking the meaning of that whole block to be a markov only. But then the code *within* such a block can contain no PRINT statements, because we cannot give them semantics as (pure) markovs.

That leads to two questions: one general, and one specific. If we took the general approach above, i.e. interpreting an entire atomic block as a markov, how can we still have the monotonicity of IF that we are insisting on?

The specific question concerns the fact that the atomic “trick” suggested does not allow us to have a program in which a conditional does not leak its condition *but* one or both of its branches nevertheless does leak something: that is, maybe we don’t require that super-implicit flow be taken to leak the whole condition as it is evaluated, but we still recognize that sometimes “ordinary” implicit flow might do so in one of the subsequent branches. Is that not a limitation of our whole approach?

Maybe not: it might instead be that the mathematics here is in fact warning us that a construction like the above would be a bad idea. *Why* might it be a bad idea? \square

Exercise 14.3 Recall from §14.2.3 that it is not possible in general to reduce $(; M); (C;)$ to some $(C'; M')$, which –put informally– is because in a channel *followed* by a markov the observations and final states must be independently distributed (because the channel does not change the state), whereas when the channel *follows* the markov the observations and final states might not be independent.

But what about the other direction? Is it always possible to reduce $(C; M)$ to some $(; M'); (C';)$, i.e. to achieve the effect of revealing something about the *initial* state, then updating it, by instead updating the initial state first and then revealing something about the resulting *final* state?

Show first that if it were possible to do that then *any* sequence $(C^1; M^1); \dots (C^N; M^N)$ would be reducible to a single $(; M'); (C';)$.

Then take

$$C := \begin{pmatrix} 1 & 0 \\ 1/2 & 1/2 \end{pmatrix} \quad \text{and} \quad M := \begin{pmatrix} 1/2 & 1/2 \\ 1 & 0 \end{pmatrix}$$

and see whether you can find a C' and M' such that $(C; M) = (; M'); (C';)$.

[Hint: Remember that the equality must hold for all priors.] \square

Exercise 14.4 Consider the sequential composition $P; Q$, where the output from P is a hyper-distribution, which is then presented to Q as “input”. Using this intuition to prove that if $Q \sqsubseteq \text{SKIP}$ then $P; Q \sqsubseteq P$.

Use that observation to verify the refinement (anti-)chain in §14.6.3. \square

Exercise 14.5 Recall the refinement (anti-)chain in §14.6.3. Assuming that a limit P^* exists, and is a channel, and that semicolon is continuous with respect to the limit, show that $P^*; P^* = P^*$. \square

14.9 Chapter notes

Hidden Markov models (*HMM*'s) originated from a study by Baum and Petrie [1] on statistical prediction of Markov chains when the observations are only partial. Renewed interest in *HMM*'s grew from applications in computational linguistics and speech recognition [8, 19]. Typical in that setting is that the markov- and channel components are fixed, and the question is then to predict the value of the (hidden) variable given a sequence of observations.

HMM's have been used to analyze information flow in programs, where the leaks are caused by known side channels [4], and also as a basis for a semantic model for a small “while” language [5]. General logical properties for *HMM*'s have been explored, motivated by the desire to implement model-checking algorithms for computing quantitative properties [20]. The monadic view of them is based on the work of Lawvere [9] and Giry [7], and that together with work by van Breugel allows a connection to the Kantorovich metric [7, 3].

A *refinement order* for *HMM*-style models appeared first in McIver et al. [10] which itself is a combination and extension of earlier work in refinement for probabilistic programs [13] and, separately, for qualitative (possibilistic) refinement [17, 18]. The “hyper native” definition of refinement was given by McIver and Morgan [11, 12]. Detailed mathematical properties of the underlying partial order are worked out in McIver et al. [11] for internal and external probability, as well as demonic nondeterminism. The connection to the Giry monad [7] streamlined the presentation [14], and indeed that viewpoint nicely generalizes probabilistic concepts like the “push forward” that converts a distribution (more generally, measure) on one space to a distribution on

another space that is the image of some (measurable) function. Algebraic properties of the programming language have been developed relative to the hyper-distribution model [15], and these facilitate straightforward algebraic proofs of more complex ideas described in some of these small examples.

The model described here deals with the uncertainty of *current* values only, taking the point of view that an adversary will be interested in what a password is *now*, rather than what it was yesterday. As Chap. 10 makes clear, however, in some cases this is too weak as an attack model, ignoring correlations between variables' initial values and other variables not even mentioned in the program. More complex models that take that into account have been developed also [2, 16, 15], and make extensive use of the tools provided by the Giry monad.

The philosophy of our syntax's being only “a vehicle for the description of abstract mechanisms” is adopted from Dijkstra: a simple and feature-free programming language to facilitate the study of techniques for program correctness [6].

Bibliography

- [1] Baum, L.E., Petrie, T.: Statistical inference for probabilistic functions of finite state Markov chains. *Annals of Mathematical Statistics* **37**(6), 1554–1563 (1966)
- [2] Bordenabe, N.E., McIver, A., Morgan, C., Rabehaja, T.M.: Reasoning about distributed secrets. In: A. Bouajjani, A. Silva (eds.) *Proceedings of the 37th IFIP International Conference on Formal Techniques for Distributed Objects, Components, and Systems (FORTE 2017)*, *Lecture Notes in Computer Science*, vol. 10321, pp. 156–170. Springer, Berlin (2017)
- [3] van Breugel, F.: The metric monad for probabilistic nondeterminism (2005). Draft available at <http://www.cse.yorku.ca/~franck/research/drafts/monad.pdf>
- [4] Clark, D., Hunt, S., Malacaria, P.: Quantitative analysis of the leakage of confidential data. In: *Proceedings of the 1st Workshop on Quantitative Aspects of Programming Languages (QAPL 2001)*, *Electronic Notes in Theoretical Computer Science*, vol. 59, pp. 238–251 (2002)
- [5] Clark, D., Hunt, S., Malacaria, P.: Quantified interference for a while language. In: *Proceedings of the 2nd Workshop on Quantitative Aspects of Programming Languages (QAPL 2004)*, *Electronic Notes in Theoretical Computer Science*, vol. 112, pp. 149–166. Elsevier (2005)
- [6] Dijkstra, E.W.: *A Discipline of Programming*. Prentice Hall (1976)
- [7] Giry, M.: A categorical approach to probability theory. In: B. Banaschewski (ed.) *Categorical Aspects of Topology and Analysis*, *Lecture Notes in Mathematics*, vol. 915, pp. 68–85. Springer, Berlin (1981)
- [8] Jurafsky, D., Martin, J.H.: *Speech and Language Processing — An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall (2000)
- [9] Lawvere, F.W.: The category of probabilistic mappings, with applications to stochastic processes, statistics and pattern recognition (1962). Draft available at <http://ncatlab.org/nlab/files/lawvereprobability1962.pdf>
- [10] McIver, A., Meinicke, L., Morgan, C.: Compositional closure for Bayes risk in probabilistic noninterference. In: S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide, P.G. Spirakis (eds.) *Proceedings of the 37th International Colloquium on Automata, Languages, and Programming (ICALP 2010)*, *Lecture Notes in Computer Science*, vol. 6199, pp. 223–235. Springer, Berlin (2010)

Bibliography

- [11] McIver, A., Meinicke, L., Morgan, C.: A Kantorovich-monadic powerdomain for information hiding, with probability and nondeterminism. In: Proceedings of the 2012 27th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2012), pp. 461–470. IEEE, Los Alamitos (2012)
- [12] McIver, A., Meinicke, L., Morgan, C.: Hidden-Markov program algebra with iteration. Mathematical Structures in Computer Science **25**(2), 320–360 (2014)
- [13] McIver, A., Morgan, C.: Abstraction, Refinement and Proof for Probabilistic Systems. Monographs in Computer Science. Springer, Berlin (2005)
- [14] McIver, A., Morgan, C., Rabehaja, T.: Abstract hidden Markov models: a monadic account of quantitative information flow. In: Proceedings of the 2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2015), pp. 597–608 (2015)
- [15] McIver, A., Morgan, C., Rabehaja, T.: Program algebra for quantitative information flow. Journal of Logical and Algebraic Methods in Programming **106**, 55–77 (2019)
- [16] McIver, A., Morgan, C.C., Rabehaja, T.M.: Algebra for quantitative information flow. In: P. Höfner, D. Pous, G. Struth (eds.) Proceedings of the 16th International Conference on Relational and Algebraic Methods in Computer Science (RAMiCS 2017), *Lecture Notes in Computer Science*, vol. 10226, pp. 3–23. Springer (2017)
- [17] Morgan, C.: *The Shadow Knows*: Refinement of ignorance in sequential programs. In: T. Uustalu (ed.) Proceedings of the International Conference on Mathematics of Program Construction (MPC 2006), *Lecture Notes in Computer Science*, vol. 4014, pp. 359–378. Springer, Berlin (2006)
- [18] Morgan, C.: *The Shadow Knows*: Refinement of ignorance in sequential programs. Science of Computer Programming **74**(8), 629–653 (2009)
- [19] Rabiner, L.R.: A tutorial on hidden Markov models and selected applications in speech recognition. Proceedings of the IEEE **77**(2), 257–286 (1989)
- [20] Zhang, L., Hermanns, H., Jansen, D.: Logic and model checking for hidden Markov models. In: F. Wang (ed.) Proceedings of the 25th IFIP Formal Techniques for Networked and Distributed Systems (FORTE 2005), *Lecture Notes in Computer Science*, vol. 3731, pp. 81–112. Springer, Berlin (2005)



Chapter 15

Program algebra for QIF

One of our principal aims in this work is to extend the reliable derivation of implementations from specifications – so long pursued in other areas like functionality, concurrency and probability – to include security as well, and in particular *quantitative* security. And in doing so, we are attempting to follow the pattern of those earlier successes. For that we will need semantics, logic... and program algebra.

15.1 Semantics, logic, and program algebra

Once a programming language has a semantics, i.e. a mathematical model, with a refinement relation defined, it is in principle possible to show that $\text{Spec} \sqsubseteq \text{Imp}$ simply by determining the semantics of Spec and of Imp and then comparing them using the model-defined refinement relation. (For us here that corresponds to our definition of (\sqsubseteq) , that is of structural refinement.) In practice however that can be very difficult to do, especially for large programs. And as a vehicle for teaching people how to reason about everyday – even quite small – programs, it is not very good at all.

Defining and then using a programming *logic* can get around the practicality issue mentioned just above. Typically the logic makes statements about properties a program might have, and the *language* of the logic, which must itself be precisely defined, determines exactly which program properties are within the “terms of reference” of a refinement proof.

Thus for example “sequential programs with demonic choice” has a simple relational semantics and, in principle, refinement can be simply determined merely by checking that the denotation of the implementation, as a relation (set of initial-state/final-state pairs), is a *subset* of the denotation of the specification. Hoare logic however expresses properties of programs and of combinations of programs in terms of properties of their components, and its proof rules are themselves legitimized by their being proved in the relational logic in a general way. But a significant advantage of the logic is that for *specific* programs, one no longer has to “descend to the semantics”, i.e. to relations over a state space comprising a huge Cartesian product of variables’ types — instead, the reasoning can be done at the source level (i.e. of the programming language and its logic).

Finding such a logic, however, is not necessarily straightforward. Although Hoare logic might seem obvious now, it was a large step from Floyd’s enabling idea of attaching assertions to flowcharts — so much so that it is sometimes forgotten how the two ideas supported each other.

Taking the next step towards *QIF*, Kozen’s logic of probabilistic programs (recall §13.3.1, and its later extension to accommodate demonic choice and probability together) changes the type of Hoare-logic formulae from Boolean- to real-valued. But the extension from that to a *QIF* logic is another big step and –at least for us here– it is not yet obvious how best to do it.

Thus we consider a third level: not another extension of logic, but rather not to rely so much on logic at all. Instead we reason about programs directly, proving general equalities (i.e. of meaning) between program fragments, and even refinement inequalities, and then from them making a “toolkit” that can be used to manipulate the source code of a specification until it becomes the source code of an implementation. (The “meta-proofs”, i.e. proofs of things used for proof so that they can safely be included in the toolkit, can be done in a logic, or at the lower level of semantics: either way, they need be done only once.) Currently, in our style at least, that seems to be the most effective way of reasoning about *QIF* in the framework we have built here.

In this chapter therefore we will explain the program-algebraic approach to *QIF* programs, as it has developed so far. Although many of the example program derivations below have already been published, those earlier derivations were done in a *non-quantitative*, i.e. qualitative setting; but here we do them in a quantitative setting. As we explain in §15.3.1 below, most of the derivations carry across unchanged into the quantitative setting provided we replace the possibilistic demonic choice of the earlier examples with *uniform* probabilistic choice in the quantitative setting. That turns out to be a startling advantage of using program algebra.

15.2 Static visibility declarations; multiple variables

We begin with some adjustments that will make it easier to write our example programs and to present the equalities between them. The first concerns “visible” and “hidden” variables.

Earlier our entire state space \mathcal{X} was hidden, all of it, and the only leaks were those indicated explicitly by PRINT statements, i.e. channels from our program’s state to the outside world. It is common however in (quantitative) information flow, especially in noninterference or dependence analyses, to designate statically some variables as “high security” and others as “low security” and then to search for information flows from high- to low-security variables. Here we will call high-security variables “hidden” and low-security variables “visible”.¹

As part of this adaptation, we note that in many earlier examples we have had just one variable x representing the whole state space \mathcal{X} ; but here –as in programming generally– we might have many. As usual, we then consider the state space to be the Cartesian product of the individual types: thus $\mathcal{X} = \mathcal{V} \times \mathcal{H}$ is the simple case of one visible and one hidden variable.

Assignments and projections are handled in the normal way for state-based program semantics.

Also local variables (§14.4.8) can be visible or hidden, and they (too) are treated as normal: within the scope of a visible local-variable declaration like

$\{ \text{VIS } v'; \text{ HID } h'; \dots \}$

the state space is extended with further Cartesian multiplications at the beginning that are then “projected away” at the end.

¹ This slight abstraction is useful because security is not the only place where issues of visibility arise: another is in encapsulation of data-types.

As suggested earlier (§13.4.4), we can map this static noninterference style into our own leaner “everything is hidden” style by making all variables hidden (as our semantics does), but adding automatically a `PRINT v` statement after any assignment to any visible variable `v`. Thus the canonical, blatantly leaking program would be

<code>VIS v</code> <code>HID h</code> <code>v := h</code>	<code>- Left-justified variable declarations are global.</code>
---	---

(15.1)

which copies hidden `h` into visible `v`. By the “implicit `PRINT`” convention just above, the program is actually

<code>VAR v,h</code> <code>v := h</code> <code>PRINT v</code>	<code>- \mathcal{X} is $\mathcal{V} \times \mathcal{H}$.</code>
---	---

(15.2)

In the de-sugared version, the global variables are *all* hidden (as in our usual state space \mathcal{X}), so we declare them simply `VAR`. But the treated-as-visible `v` has its value leaked whenever it is updated. Thus the adversary knows `v` because she observes a leaked value that, from the program code, she knows is the value of `v` at that point; and –also from her knowledge of the code– she knows that it is as well the value of `h`.

But more resolution is possible: the above handles only one point of view, distinguishing only between “us”, the defenders who can see everything, and “her”, the adversary who can see only variables declared visible. Recall from Def. 2.1 that there might be more than one adversary, and each of them might have her own point of view. For that, a multi-party extension is possible, and we use it below: declarations `VIS` and `HID` can be subscripted with a list of agent (i.e. potential adversary) names (often only one, sometimes none) and then from the point of view of that agent –call her `A`– what is visible to her are:

- all variables declared `VIS`, but no variables declared `HID`;
- all variables declared `VISlist` where her name `A` appears in (nonempty) `list`; and
- all variables declared `HIDlist` where `A` does not appear in nonempty `list`.

A program with subscripted visibility annotations can then be “projected” onto a single agent’s point of view, resulting in a program with just unsubscripted `VIS` and `HID` declarations. And that can, if desired, be reduced further to simply `VAR` declarations with `PRINT` statements added in the right places.

The effect of all those conventions is that refinements, and thus chains of them that make a development from specification to implementation, sometimes have to be considered from several agents’ points of view; and sometimes those different points of view require different proofs. But often the same proof will do “generically” for many different but similar views at the same time.

15.3 Simple examples of program derivations in QIF

15.3.1 The Encryption Lemma

The *Encryption Lemma* is that in the context of a declaration $\text{HID } h$ the following program equality holds:

$$\text{SKIP} = \{ \text{HID } e \in \text{TRUE}_{1/2} \oplus \text{FALSE}; \text{ PRINT } h \nabla e \} ; \quad (15.3)$$

that is, if you initialize privately a fresh *local*, i.e. between $\{\dots\}$ Boolean variable uniformly to *true* or *false*, and leak its exclusive-or (∇) with some global hidden variable, then you have actually done nothing: you have changed no state (because the introduced variable was local) and you have leaked nothing (due to the properties of exclusive-or). That fact is the basis for the security of the “one-time pad”, as §15.3.3 will show.

Here, as our first example we *derive* the encryption lemma using program algebra; and it will be much used in the rest of the chapter. Our specification is

$$\begin{array}{c} \text{HID } h \\ \text{SKIP} , \end{array} \quad - \text{ Specification: } h \text{ is global.} \quad (15.4)$$

in which –we argue– it is obvious that h is not leaked. Our first step is to replace SKIP with a program fragment that is trivially equal to it, an assignment to a local visible v that has nothing to do with h : that gives

$$\begin{array}{c} \text{HID } h \\ \{ \text{VIS } v \in \text{TRUE}_{1/2} \oplus \text{FALSE} \} . \end{array} \quad - \text{ Equal to SKIP.}$$

We continue by introducing another local variable, but this time the hidden e , which –in spite of its “risky” initialization– in fact also has no effect, because it is nowhere used:

$$\begin{array}{c} \text{HID } h \\ \{ \text{VIS } v \in \text{TRUE}_{1/2} \oplus \text{FALSE} \quad - v \text{ is visible, but rhs is not secret.} \\ \quad \text{HID } e := h \nabla v \quad - \text{rhs is secret, but } e \text{ is hidden.} \\ \} . \end{array} \quad (15.5)$$

Now the next step, i.e. from (15.5) to (15.6), is the important one: the two assignments are rewritten equivalently: note that this is classical reasoning (over probabilistic programs), having nothing to do with *QIF*. We get

$$\begin{array}{c} \text{HID } h \\ \{ \text{HID } e \in \text{TRUE}_{1/2} \oplus \text{FALSE} \\ \quad \text{VIS } v := h \nabla e \\ \} . \end{array} \quad \begin{array}{l} - \text{ Secret message.} \\ - \text{ One-time pad.} \\ - \text{ Transmitted message.} \end{array} \quad (15.6)$$

That the last step (15.6) is equivalent to the original (15.4) is what we call the *Encryption Lemma* — that an instance of (15.6) can be introduced into a program wherever SKIP can be, i.e. almost anywhere. It’s a key component of the way we develop protocols below.

But finally, if we wish, we can remove the visible v , thus showing that

$$\begin{array}{c} \text{HID } h \\ \{ \text{HID } e \in \text{TRUE}_{1/2} \oplus \text{FALSE} \\ \quad \text{PRINT } h \nabla e \\ \} \end{array}$$

is equal to SKIP — and that was initially our goal.

15.3.2 From qualitative proofs to quantitative proofs

In our earlier non-quantitative work (for which, see the Chapter Notes) the Encryption Lemma in §15.3.1 just above appeared with demonic choice (\sqcap) rather than probabilistic choice: it was

$$\text{SKIP} = \{ \text{HID } e :: \text{TRUE} \sqcap \text{FALSE}; \text{ PRINT } h \nabla e \} , \quad (15.7)$$

where (\sqcap) represents demonic choice and “demonic” was interpreted as “bad for the user”, just as in Dijkstra’s work (and many others’ both before and since). That is, the (\sqcap)-choice means “good for the adversary”. For program-algebraic proofs in that probabilistic model, in most cases the introduction of demonic choice (into the implementation) was done by replacing `SKIP` by an instance of the *rhs* of Prog. (15.7) — and so to synthesize a *quantitative* proof from that, we need only replace that by the *rhs* of Prog. (15.3) from §15.3.1 instead.

15.3.3 The One-Time Pad

We now treat the One-Time Pad example with three points of view instead of only the simple “us” and “her” as in §15.3.1 just above: there are two agents A, B who wish to exchange a message, and an observer X from whom the message must be concealed. With multiple agents, we will use neutral gender for them all.

The two agents A, B are in different locations, with Boolean-valued variables a, b respectively at their respective locations: thus A can read and write a without observation by X , and similarly B . But Agent A wants to send the value a of its a to variable b at Agent B without the message’s being intercepted and read by external Agent X : and that cannot be done wholly at A ’s location or at B ’s. In spite of those complexities, the *specification* is simple — merely

$$\begin{aligned} & \text{VIS}_A a \\ & \text{VIS}_B b \\ & b := a \end{aligned} , \quad (15.8)$$

which captures both the functional behavior (that $b := a$ finally), and the *QIF* behavior that prevents X ’s intercepting the message: it cannot see a or b , because they are visible only to A and B respectively; and the single assignment statement $b := a$ is atomic, in particular not leaking any information to X (except whether or not it has executed).

As noted, however, the issue with this specification (for A, B), what makes it not immediately an implementation, is that it is not directly executable: the statement $b := a$ might indeed be atomic from a *QIF* perspective, but its source a and target b lie in different locations. There is no single “machine instruction” that can achieve an assignment like that. Neither A nor B can execute $b := a$ on its own.

If we ignore *QIF* issues for a moment, however, the solution to the non-executability of $b := a$ is well known and straightforward: the value a of a is *sent* to B in a message we call m and that is visible to everyone (which we assume because the network is insecure). That is written

$$\begin{aligned} & \text{VIS}_A a \quad \text{in transit} \\ & \text{VIS}_B b \quad \downarrow \\ & \{ \text{VIS } m; m := a; b := m \} \end{aligned} , \quad (15.9)$$

in which the value a of a is copied into a message m that is sent across the network. In contrast to $b := a$, the two atomic commands $m := a$ and $b := m$ are separately

executable by *A* and *B* respectively: in the first *A* sends the message, and in the second *B* receives it. Both *a* and the output buffer are with *A*; and both *b* and the input buffer are with *B*. And when execution lies between the two statements, the message is still in transit (and can be read by anyone).

The reason we encapsulate *m* in a declaration is that it is an implementation artifact: the specification concerns *a* and *b* only, and does not mention *m* at all.² Recall however that a visible variable's being declared in a local block does not affect whether it can be viewed by the adversary: as long as she looks while control is in its scope, she will obtain its value and, from that, might be able to deduce other variables' values on which –from knowing the code– she knows the local, visible variable depends. That obtains even if those other variables are outside the local declaration block.

The problem now however is that (15.9) does not refine (15.8) from the *QIF* perspective of Agent *X*: from its point of view the specification program (15.8) is

```
HID a
HID b
b := a ,
```

(15.10)

and the program (15.9) is

```
HID a
HID b
{ VAR m; m := a; PRINT m; b := m } ,
```

(15.11)

where to be clear we de-sugared the *VIS* and included the automatically inserted *PRINT m* that is implied by variable *m*'s being visible. The program (15.11) leaks *a* and *b*, but the specification (15.10) leaks neither. (Note that (15.8) and (15.10) are not different programs: they are the same program from different agents' point of view. The same applies to (15.9) and (15.11).)

Thus (15.10) $\not\sqsubseteq$ (15.11), and we must do something more elaborate: we set out the steps below. We will introduce not only a visible-to-all message *m* as we did just above, but also a hidden-from-*X* variable *e* that will be used to encrypt the message. Here are the steps, from Agent *X*'s point of view, beginning with (15.8):

<pre>VIS_A a VIS_B b b := a .</pre>	<ul style="list-style-type: none"> - This is (15.8) again, our specification. - Agent <i>X</i> can see neither <i>a</i> nor <i>b</i>. (15.12)
-------------------------------------	--

Using the Encryption Lemma from the previous section §15.3.1 we introduce the encryption key and the message:

<pre>VIS_A a VIS_B b b := a { VIS_{AB} e :∈ TRUE_{1/2} ⊕ FALSE; VIS m := a ∨ e }</pre>	<ul style="list-style-type: none"> - Introduce <i>e</i> and <i>m</i>. - Equivalent to SKIP. (15.13)
--	--

That step must be justified for all three agents: that each of the following programs is equal to *SKIP*. The functional equality is clear, since the introduced variables are local. For *QIF* equality, we look at the three points of view separately:

² The technical issue is that, with full rigor, making *m* global would prevent (15.9)'s being a functional refinement of (15.8), since the latter does not change *m* but the former does.

point of view		justification
Agent	projected program equal to SKIP	
A	VIS a { VIS e::= TRUE _{1/2⊕} FALSE; VIS m:= a∇e }	no hidden variables
B	HID a { VIS e::= TRUE _{1/2⊕} FALSE; VIS m:= a∇e }	see below
X	HID a { HID e::= TRUE _{1/2⊕} FALSE; VIS m:= a∇e }	Encryption Lemma

For the second case, “see below”, we note that in

$$\begin{array}{ll}
 \text{HID a} & - \text{ } B \text{'s point of view.} \\
 \text{VIS b} & \\
 \text{valid} & \dots \quad \leftarrow \text{ Statement } b := a \text{ is here} \\
 \text{here} \rightarrow & \{a = b\} \\
 & \{ \text{VIS e::= } \text{TRUE}_{1/2⊕} \text{FALSE; VIS m:= a∇e } \} \quad \Leftarrow \text{not SKIP}
 \end{array}$$

the \Leftarrow -indicated statement is *not* a QIF-refinement of SKIP, at least not in isolation, because it leaks hidden a and SKIP does not. However the refinement (actually equality) can be justified by context: the statement $b := a$ immediately before establishes $a = b$ and so, in the \Leftarrow -block following, we can replace occurrences of a on assignments’ right-hand sides by b , giving

$$\begin{array}{c}
 \text{was a} \\
 \downarrow \\
 \{ \text{VIS e::= } \text{TRUE}_{1/2⊕} \text{FALSE; VIS m:= b∇e } \} \quad - \Leftarrow \text{is now SKIP.}
 \end{array}$$

which –from B ’s point of view– now contains only visible variables and so is indeed equivalent to SKIP.

What we have just seen is an example of a derivation’s progress through successively more implementation-like programs whose code can be organized to be the same for all agents, although occasionally the *reason* for a step might depend on which agent is looking.

Now we rearrange the declarations, using the classical rules of “no variable capture”. The encryption key is generated first; then the message m is introduced; and then the message is constructed. That gives

$$\begin{array}{ll}
 \text{VIS}_A a & \\
 \text{VIS}_B b & \\
 \{ \text{VIS}_{AB} e::= \text{TRUE}_{1/2⊕} \text{FALSE} & \quad - \text{ Introduce key } e. \\
 \{ \text{VIS } m & \quad - \text{ Declare message } m. \\
 b := a & \\
 m := a∇e & \\
 \} & \\
 \} & . \quad (15.14)
 \end{array}$$

Next we deal with the fact that the two interior statements are out of order: the statements $b := a$; $m := a∇e$ are replaced by $m := a∇e$; $b := m∇e$, which is a classical equality given that the assignment to m establishes

$$m∇e = (a∇e)∇e = a∇(e∇e) = a∇\text{FALSE} = a .$$

The algebraic rules for reordering in general are explained in §15.4 just below. And so finally we have

- One-time pad implementation.

```
VISA a
VISB b
{ VISAB e::= TRUE1/2⊕ FALSE
  { VIS m
    m:= a⊸ e
    b:= m⊸ e
  }
}
} ,
```

(15.15)

- Encode message and send.
- Receive message and decode.

where establishment of the one-time pad, the statement $e ::= \text{TRUE}_{1/2} \oplus \text{FALSE}$, is implemented either by A, B 's meeting beforehand, or by a trusted third party. Note that e is visible only to A, B ; but m is visible to everyone.

15.4 Algebraic rules for reordering statements

Many of the derivations in this chapter use statement reordering in order to move from the specification to an implementation; we now look at them more closely. We continue to use Exp etc. within programs to indicate “any expression”.

Classical rules for statement reordering are well known, for example that if y does not occur in $\text{Exp}X$ then

$$x := \text{Exp}X ; y := \text{Exp}Y \\ = y := \text{Exp}Y[x \setminus \text{Exp}X] ; x := \text{Exp}X , \quad \text{“y not free in } \text{Exp}X\text{”}$$

where $\text{Exp}Y[x \setminus \text{Exp}X]$ is $\text{Exp}Y$ with all (free) occurrences of x replaced by $\text{Exp}X$ (after renaming if necessary to avoid capture). Generalizations of those apply to probabilistic programming, as is also well known.

But in the context of *QIF* we must be sure that obviously functionally correct reorderings do not affect the escape of information. For primitive statements, i.e. assignments and PRINT's, that is surprisingly straightforward if we operate in our principal model, where all variables are hidden.

swap two PRINT's As mentioned in §13.4.1(2), sequential composition between PRINT statements is commutative: they can simply be swapped. Since they don't affect the state, there is no need to check for “this not free in that”.

swap two assignments For that, the classical rules suffice.

swap assignment and PRINT This is essentially the same as swapping two assignments, as we illustrate here. Note that we use VAR for variable declaration, as we are reasoning in the “all-hidden” model. We reason

$$x := \text{Exp}X ; \text{PRINT } \text{Exp}Y \\ = x := \text{Exp}X ; \{ \text{VAR } y ; y := \text{Exp}Y ; \text{PRINT } y \} \\ = \{ \text{VAR } y ; x := \text{Exp}X ; y := \text{Exp}Y ; \text{PRINT } y \} \quad \text{“fresh } y \text{ cannot occur in } \text{Exp}X \text{ (used also at ‘idem’ below)’}$$

$$\begin{aligned}
 &= \{ \text{VAR } y; y := \text{Exp}Y[x \setminus \text{Exp}X]; x := \text{Exp}X; \text{PRINT } y \} \quad \text{"(idem)"} \\
 &= \{ \text{VAR } y; y := \text{Exp}Y[x \setminus \text{Exp}X]; \text{PRINT } y; x := \text{Exp}X \} \quad \text{"two statements disjoint: see Exercise 15.5"} \\
 &= \{ \text{VAR } y; y := \text{Exp}Y[x \setminus \text{Exp}X]; \text{PRINT } y \}; x := \text{Exp}X \quad \text{"(idem)"} \\
 &= \text{PRINT } \text{Exp}Y[x \setminus \text{Exp}X]; x := \text{Exp}X \quad .
 \end{aligned}$$

Thus in swapping an assignment and a PRINT, one can treat the PRINT as an assignment to a fresh variable.

15.5 Larger example 1: Oblivious Transfer

We now turn to a larger –and more intriguing– example. The Oblivious Transfer protocol allows one agent A holding two secrets $m_{1,2}$ to supply one of them to another agent B in a way that allows B to choose which of the two secrets it wants, but without revealing to A which secret that is. In the process however B discovers nothing about the secret it doesn't want. As we will see later, Oblivious Transfer is a useful building-block for other protocols; in this section however we develop it in isolation.

We use type `Bit` for the set $\{0, 1\}$, but will use traditional Boolean operators if convenient: thus $\neg 0 = 1$ etc.

Specification We assume that Agent A has two messages $m_0, m_1 : \text{Bit}$. The protocol is to ensure that Agent B will obtain either m_0 or m_1 according to its choice of $c : \text{Bit}$ — that is, B will obtain m_c . But A must not learn c , and B must not learn $m_{\neg c}$. The specification is thus

$$\begin{array}{ll}
 \text{VIS}_A m_0, m_1 & - B \text{ to receive one of these from } A. \\
 \text{VIS}_B c & - B \text{'s choice of which one it wants.} \\
 \text{VIS}_B m & - What B receives from A. \\
 m := m_c & . \tag{15.16}
 \end{array}$$

Note that m above is *not* (i.e. will turn out not to be) the actual message sent across the network from A to B . The specification cannot refer to that, in any case, since “the network” is not in its vocabulary: that's an implementation artifact. But if m indeed were the actual message sent, then A would know what it is (since it is the sender) and then it could easily learn B 's variable c by comparing that m with its own $m_{0,1}$. That is why the declaration of m is only VIS_B , not VIS_{AB} .

Setup Trusted third party Agent T privately gives A two random bits $r_{0,1}$. Then it chooses a random bit d and gives both d and r_d privately to B .

Agent T is now done, and can go home.

One might think that with a trusted third party there is no need for a complicated protocol at all. But the third party used here (1) does not learn anything about the agents' data and (2) does not need to take part in the protocol itself as it runs. That latter is the significance of “and can go home”.

Implementation protocol sketch Agent B has chosen privately a bit c , and it wants to obtain m_c from A . Using the d it received from T , it sends A a bit $e = c \nabla d$. Agent A sends back to B the values $f_0, f_1 = m_0 \nabla r_e, m_1 \nabla r_{\neg e}$. Then Agent B computes $m = f_c \nabla r_d$.³

Implementation derivation Given that there are three agents A, B, T we have three potential points of view; for this example we will take the point of view of B , which is

HID m_0, m_1	- B cannot see A 's messages.
VIS c	- B can see his own choice.
VIS m	- B can see the message he receives.
$m := m_c$.	(15.17)

As a first step towards implementation, we introduce the variables prepared by Agent T . To reduce the clutter of nesting (which would be more accurate, since they are implementation-local), we declare them global as well: they are

HID r_0, r_1	- B cannot see the bits T gives to A .
VIS d, r	- B can see what T gives it.
VIS f_0, f_1 .	B can see the messages from A .

For comparison (although we will continue with B), Agent A 's view is

VIS m_0, m_1	- A sees his own messages.
HID c	- A cannot see B 's choice.
HID m	- A cannot see the message B constructs.
VIS r_0, r_1	- A sees the bits he received from T .
HID d, r	- A cannot see the bits B received from T .
VIS f_0, f_1 .	- A can see the messages he sent to B .

³ An even more informal sketch is this fairy tale.

An apprentice magician is about to graduate from the Academy, and according to tradition must now choose a path for the future: will she follow the Black Arts, or the White Arts? Since this is an issue of morality, her choice must be her own, without coercion, and is never revealed (except implicitly by her later actions). To graduate, she will read either the Black Tome or the White Tome — but she may not read both.

The ancient protocol for this rite of passage is that a Djinn is summoned who gives the Master Sorcerer two locks, one black and one white. To the Apprentice the Djinn gives a single, golden key, whispering to her “black” or “white” as he does so: the key will open only the lock of that color. The Djinn then returns to his own dimension.

Now the Apprentice tells the Sorcerer whether the Djinn’s whisper matched the choice she is about to make: if she says “match”, the Sorcerer secures each Tome with the lock of its color; if the Apprentice says “no match”, the locks are reversed. The Sorcerer leaves... and the Apprentice is alone.

The Apprentice applies her key to the matching lock, and reads Ancient Lore within the Tome it secured. When finished, she replaces the lock and throws the key into the moat. She now knows the secrets of her chosen Art; but no-one but she knows which Art it is; and she remains forever ignorant of the secrets of the other.

Note that the difference is not a simple swap of VIS and HID, since some variables are visible to *both* A, B . Finally, Agent T 's view is

$\text{HID } m_0, m_1$ $\text{HID } c$ $\text{HID } m$	$- \text{ View of Agent } T.$
$\text{VIS } r_0, r_1$ $\text{VIS } d, r$ $\text{HID } f_0, f_1$	$. \quad - T \text{ does not see the exchanged messages.}$

(15.18)

That last assumption, that T does not see the exchanged messages, can be implemented by a separate protocol between A, B that ensures secure message exchange (for example the one-time pad as in §15.3.3). We have expressed that abstraction, i.e. a “layering” of protocols, by the simple declaration $\text{VIS}_{AB} f_0, f_1$ which projects onto HID f_0, f_1 for Agent T (and indeed other agents X , for whom *all* declarations in (15.18) would be HID).

Here then is the derivation of the protocol from Agent B 's point of view. We omit the global declarations, and begin with

$m := m_C$	$. \quad - \text{ specification.}$
------------	------------------------------------

We introduce code that is trivially SKIP, for B at least, since all of c, d, e are visible to it:

$\{ \text{VIS } d := 0_{1/2} \oplus 1, e := c \nabla d \}$	$. \quad - d \text{ is initialized by Agent } T.$
$m := m_C$	$.$

And we introduce more code, this time justified by the Encryption Lemma:

$\{ \text{VIS } d := 0_{1/2} \oplus 1, e := c \nabla d \}$	$. \quad - d \text{ is supplied by Agent } T.$
$\{ \text{VIS } f_{0,1}; \text{ HID } r_{0,1}$	
$r_0, r_1 := 0_{1/2} \oplus 1, 0_{1/2} \oplus 1$	$. \quad - \text{Independent internal choices by } T.$
$f_0, f_1 := m_0 \nabla r_e, m_1 \nabla r_{\neg e}$	$. \quad - \text{Message transmission by } A \text{ to } B.$
$\}$	
$m := m_C$	$.$

Now we adjust the variable scopes:

$\{ \text{VIS } d, e$	$. \quad - \text{Move declarations to outside.}$
$\text{VIS } f_{0,1}$	
$\text{HID } r_{0,1}$	
$d := 0_{1/2} \oplus 1$	
$e := c \nabla d$	
$r_0, r_1 := 0_{1/2} \oplus 1, 0_{1/2} \oplus 1$	
$f_0, f_1 := m_0 \nabla r_e, m_1 \nabla r_{\neg e}$	
$m := m_C$	
$\}$	$,$

and we introduce \mathbf{r} , which is visible-only reasoning for Agent B :

```
{ VIS d,e,r
  VIS f0,1
  HID r0,1

  d::= 01/2⊕1
  e:= c∇d
  r0,r1::= 01/2⊕1,01/2⊕1
  f0,f1::= m0∇re,m1∇r-e
  m:= mc
  r:= fc∇m
}
. - Introduce  $\mathbf{r}$ , declared above.
```

(15.19)

Now we use classical reasoning to change the assignment to \mathbf{r} , based on equalities established by the statements above it:

```
{ VIS d,e,r
  VIS f0,1
  HID r0,1

  d::= 01/2⊕1
  e:= c∇d
  r0,r1::= 01/2⊕1,01/2⊕1
  f0,f1::= m0∇re,m1∇r-e
  m:= mc
  r:= rd
}
. - ⇐ Classical reasoning.
```

(15.20)

Those “replacements due to classical reasoning” are an important part of the technique, based ultimately on Referential Transparency. In the previous step (15.19) we were able to introduce \mathbf{r} and the assignment to it because (from B 's point of view) only visible variables were involved, so that there could be no *QIF* implications. From a functional point of view, its introduction is justified by its being local.

However the replacement right-hand side at \Leftarrow in (15.20) involves hidden variables, and one of the $r_{0,1}$ (which one depends on d) is visible only to A (and T) and thus not to B .⁴ But the equality of those two *rhs*'s is established by simple (Hoare-style) assertional reasoning over the code above, and the referential-transparency principle allows such replacements *irrespective of visibility* — equals may be replaced by equals in all cases, no matter “who can see what”.

⁴ Although both \mathbf{r} and \mathbf{d} are visible, the variable r_d is not — it is because r_d is the *name* of a (hidden) variable, not an indexing operation where \mathbf{r} is indexed by \mathbf{d} . More exactly (but less concisely) it could be written

$r := \text{IF } d=0 \text{ THEN } r_0 \text{ ELSE } r_1 ,$

where it is clear that \mathbf{r} , r_0 and r_1 are three different variables. The first variable is visible; the other two variables are hidden.

```

VISA m0,1                                - A's messages.
VISB c                                     - B's choice.

{ VISAT r0,1                         - Given to A by T.
  r0,r1 ∈ 01/2⊕1,01/2⊕1      - Independent secret random bits.

  VISBT d,r                           - Given to B by T.
  d ∈ 01/2⊕1                          - Another secret random bit.
  r := rd

  VISAB e                            - Calculated by B, sent to A.
  e := c∇d                            - B encrypts c with d.

  VISAB f0,1;                      - Calculated by A, sent to B.

  f0,f1 := m0∇re,m1∇r-e    - A sends both m's, encrypted;
                                             - but B can decrypt only the one it asked for.

  m := fc∇r                         - Decrypted by B.
}
.

```

The *functional* correctness of the above, that at the end $m = m_c$, is not in doubt — classical reasoning through the program text establishes it easily. It is however the derivation above that ensures that this code leaks no more than the specification (15.16).

Figure 15.1 Implementation of Rabin's *Oblivious Transfer protocol*

We continue by reordering the imperative code, giving

```
{ VIS d,e,r
  VIS f0,1
  HID r0,1

  d := 01/2⊕1
  e := c∇d
  r0,r1 := 01/2⊕1, 01/2⊕1
  r := rd
  f0,f1 := m0∇re, m1∇r¬e
  m := mc
}
```

And finally we use classical reasoning (and Referential Transparency) to replace the *rhs* of the final assignment, giving the implementation:

```
{ VIS d,e,r
  VIS f0,1
  HID r0,1

  d := 01/2⊕1
  e := c∇d
  r0,r1 := 01/2⊕1, 01/2⊕1
  r := rd
  f0,f1 := m0∇re, m1∇r¬e
  m := fc∇r
}
```

(15.21)

We have thus shown that (15.17) = (15.21). Of course, to complete the full proof we would have to reason over *A*'s and *T*'s views as well.

Put back into its original form, i.e. “un-projecting” from *B*'s point of view, gives the implementation shown in Fig. 15.1.

15.6 Larger example 2: Two-party conjunction, or *The Lovers' protocol*

Two lovers *A,B* want to discover whether they indeed love each other, but without risking embarrassment if one loves but the other does not: thus if *A* loves *B* but the love is not returned, then they should both learn “no, not mutual” — but that is all. *A* for example should not be embarrassed by having his unrequited feelings revealed to *B*.⁵

The specification of this protocol is

```
VISA a
VISB b
PRINT a ∧ b ,
```

(15.22)

⁵ Here we'll use male for *A* and female for *B* to make pronouns clear, but without our usual convention that therefore *B* is the adversary.

and the reason we need an implementation is that $a \wedge b$ cannot be executed directly: we must therefore develop one. Note that—as for Oblivious Transfer (§15.5) just above—for some steps the *justification* will vary depending on the agent, although we have arranged that the claimed equality—i.e. the step actually taken—is valid for all of them.

Our first step is to use the Encryption Lemma. For neatness of presentation, we introduce the special syntax $(x \nabla y) : \in \text{Exp}$ to mean “Choose Booleans x, y uniformly and independently so that $x \nabla y = \text{Exp}$.” It is easily implemented by e.g. the asymmetric $x : \in \text{TRUE}_{1/2} \oplus \text{FALSE}; y := \text{Exp} \nabla x$.

In fact we use a “double” version of the Encryption Lemma (Exercise 15.11) to show that (15.22) is equal to

$$\begin{array}{ll} \text{VIS}_A a & \text{— Encryption Lemma ensures } a \text{ not leaked.} \\ \text{VIS}_B b & \\ \{ \text{VIS}_A a_0, a_1; (a_0 \nabla a_1) : \in a; \text{PRINT } a_0 \} & (15.23) \\ \text{PRINT } a \wedge b & . \end{array}$$

From B ’s point of view a is hidden; and it is the Encryption Lemma that ensures that no information about a (at all) is leaked to B by the code inside the introduced block, in spite of the $\text{PRINT } a_0$. (For A all of the introduced code involves only visible-variable assignments anyway.)

Now we do some reordering of statements, and move the declarations to the outside:

$$\begin{array}{ll} \text{VIS}_A a & \text{— Reorganize.} \\ \text{VIS}_B b & \\ \{ \text{VIS}_A a_0, a_1 & \\ (a_0 \nabla a_1) : \in a & \\ \text{PRINT } a_0 & \\ \text{PRINT } (\text{IF } b \text{ THEN } a_1 \text{ ELSE } a_0) & \text{— Expression equals } a \wedge b. \\ \} & . \end{array}$$

Recall that §15.4 gives the rules for such reorganizations. The *internal* choice in $\text{PRINT } (\text{IF } b \text{ THEN } a_1 \text{ ELSE } a_0)$ contains a condition whose resolution is not visible (to A).⁶

Recall that we can replace the second PRINT by an explicit assignment (as mentioned in Prog. (13.13) on p. 243), to give

$$\begin{array}{ll} \text{VIS}_A a & \text{— Reorganize.} \\ \text{VIS}_B b & \\ \{ \text{VIS}_A a_0, a_1 & \\ (a_0 \nabla a_1) : \in a & \\ \text{PRINT } a_0 & \\ \{ \text{VIS}_B b_0 := (\text{IF } b \text{ THEN } a_1 \text{ ELSE } a_0) & \\ \text{PRINT } b_0 & \\ \} & \\ \} & , \end{array}$$

where the internal choice is now within an assignment statement.

⁶ An external choice $\text{IF } b \text{ THEN } \text{PRINT } a_1 \text{ ELSE } \text{PRINT } a_0$ on the other hand *would* reveal b to A (and everyone else).

And finally, we rearrange again to get

```

V1SA a
V1SB b
{ V1SA a0,a1; V1SB b0
  (a0 ∇ a1) :∈ a; PRINT a0           - Reorganize.
  b0 := (IF b THEN a1 ELSE a0) - Done by A.      (15.24)
  PRINT b0           - Oblivious Transfer A→B.
}
,
```

in which we have introduced a *sub-protocol* –Oblivious Transfer– as developed in §15.5 just above.

15.7 Sub-protocols and declassification

We stress the importance of the use of sub-protocols: it is essential for developments of any size, whether for *QIF* or not. But *also* essential is the legitimacy of the technique. The Oblivious Transfer step above was not “an abbreviation”, reasoned about informally: it was actual code, with precisely defined semantics and so, *as it stands*, guarantees the correctness of (15.24) with respect to the specification (15.22). We absolutely do *not* need to dig into the code in §15.5 to see whether (15.24) is correct. What §15.5 tells us is that if we make the substitution of implementation-for-specification that §15.5 proved in isolation, then the resulting program, much longer and more complex, is *still* correct. It is an example of Referential Transparency or, more generally, of monotonicity of refinement (\sqsubseteq) — that specifications, as the single-line $b_0 := (\text{IF } b \text{ THEN } a_1 \text{ ELSE } a_0)$ is, can be replaced by their implementations, as the much more complicated (15.21) is.

The overall program, after the substitution, is shown in Fig. 15.2.

Finally, we make a remark about “declassification”, where information is explicitly allowed to leak. In some approaches that is done by explicit statements, or annotations, in a source program. Our corresponding approach here is that information is allowed to leak in an *implementation* precisely when the *specification* allows it. That can, for example, be done by including PRINT statements explicitly in the specification; but it can occur also when the specification’s functional properties imply a leak, for example in the Dining Cryptographers’ protocol (Exercise 21.4 to come) where the leak “Did the NSA pay?” is implied in the specification.

In our example here, we note that it could be that b is *true* but a is *false* and thus that B learns a by noting that *false* is revealed overall; note that this is a property of the *specification*. Now, in the implementation, we can see how this happens: when b is true the local variables $V_{0,1}$ will be complementary and so –in spite of not learning a while the Oblivious Transfer is carried out– Agent B will still learn a afterwards by comparing a_0 with her own $b_{0,1}$.

15.8 Refinement and quantitative analyses

We now consider the protocol of Fig. 15.2 quantitatively, viz. with explicit numbers, recalling the discussion in §13.2.2. Suppose Agent X knows that, in the population generally, for each A (resp. B) there is probability $2/3$ that he (resp. she) loves a randomly chosen B (resp. A). Thus X knows the prior distribution of (a,b) , where a is “ A loves B ” and b is “ B loves A ”, and we suppose that it wants to find out whether a randomly chosen A,B have matching feelings, i.e. either they both love or both do

Specification from (15.22):

```
VISA a
VISB b
PRINT a  $\wedge$  b .
```

Implementation, with Oblivious Transfer inlined, where one line of specification code becomes fifteen lines of implementation:

```
VISA a
VISB b
{ VISA a0, a1; - Variables for A (and T).
  VISAT r0,1 - initialized for A by trusted third party T.
  r0, r1  $\in$  {0,1} - Independent secret random bits.

  VISB b0 - Variables for B (and T).
  VISBT d, r - initialized for B by trusted third party T.
  d  $\in$  {0,1} - Done for B by T.
  r := rd

  VISAB e
  VISAB f0,1; - Message B  $\rightarrow$  A.
  - Message A  $\rightarrow$  B.

  (a0  $\nabla$  a1)  $\in$  a; - Done by A.
  PRINT a0

  e := b  $\nabla$  d - B encrypts b with d, sends to A.
  f0, f1 := a0  $\nabla$  re, a1  $\nabla$  r $\neg$ e - A encrypts a's with r's, sends to B.
  b0 := fb  $\nabla$  r - Decrypted by B.

  PRINT b0 - Done by B.
}
```

Note that whereas the specification has a single leak `PRINT a \wedge b`, the implementation has effectively two leaks `PRINT a0` and `PRINT b0`, one from A to B and the other from B to A and, in both cases, to X. That illustrates two things: the first is a typical pattern of multi-party computation, that the overall result is computed in “shares” by the parties separately, and afterwards the shares are published (leaked, in our terms). By combining the shares the desired result is obtained. In this case classical reasoning shows that the specification’s `a \wedge b` is the implementation’s `a0 ∇ b0`.

The other aspect is that the abstraction we achieve –that we emphasize– follows from our policy of not depending on the precise values released, but only on what those values tell us about the hidden state — in this case about `a` and `b`. The equivalence of the two programs above tells us that what is learned from the single value `a \wedge b` is the same as what is learned from the separate values `a0` and `b0` *however* one might choose to combine them.⁷

Figure 15.2 Complete code for the Lovers’ protocol

not. Its best guess with respect to the prior is that they do have matching feelings, since that has probability $2/3 \cdot 2/3 + 1/3 \cdot 1/3 = 5/9$.⁸

Now suppose X observes a run of the Lovers' protocol between A and B and that, as usual, it knows the code: it is open source, and is given in Fig. 15.2. How would A and B , by analyzing that code, its 15 statements, calculate X 's posterior chances of correctly guessing whether they have matching feelings once it has overheard the declarations `PRINT a0` and `PRINT b0` that they make to each other?

The answer is that A and B would not analyze that code beyond the purely functional observation that $a_0 \nabla b_0 = a \nabla b$. They would instead analyze the specification (15.22), reasoning

Agent X observed $a_0 \nabla b_0$, which we know equals $a \nabla b$. The specification tells us that we can reason as follows (about X).

If X learns that they do love each other, then it will know definitely that they have the same feelings. If however they don't love each other, then the *a posteriori* probability that they have the same feelings will be known by X (from its knowledge of the prior, and its eavesdropping) to be only $1/5$. Thus it will guess “Same” if it learns that they do love each other (probability $4/9$) and “Different” (probability $1 - 4/9 = 5/9$) if it learns they do not. Agent X 's chance of success is therefore no more than

$$4/9 \cdot 1 + 5/9 \cdot (1 - 1/5) = 8/9 ,$$

even if it overhears the conversation. (The “no more than” comes from the fact that refinement is guaranteed not to increase the vulnerability of the implementation beyond that of the specification. In this case A, B could reason “is exactly”, because the derivation of the implementation contained no proper refinements — it was equality all the way.) Of course if $8/9$ is too high for A and B to tolerate, they can implement the `PRINT a0` and `PRINT b0` by encrypted messages (in this case probably note-passing, since they are sitting next to each other), a protocol layering as we suggested earlier for the Oblivious Transfer (just after (15.18) on p. 293). But we recall that the point of the Lovers' protocol is mainly to conceal information from each other, not from X .

The *soundness* of A and B 's decision to analyze the specification rather than the implementation derives from the soundness of the rules we used to do the derivation in §15.6. Of course it does:

That is the point of refinement: its utility in replacing complicated calculations by simpler ones.

Our program equalities, the “axioms” of the program algebra, all have the property that for a given prior the conditional posterior uncertainty (or vulnerability) of the specification and the implementation are equal *for any uncertainty (or vulnerability) whatsoever*.⁹ Thus for example the Shannon entropy of the posterior distribution on (a, b) in this situation, after running the code in Fig. 15.2 just above, is again available by imagining that she has run the specification (15.22).¹⁰ That gives

$$4/9 \cdot \log_2 1 + 5/9 \cdot (1/5 \log_2 5 + 2 \cdot 2/5 \log_2 5/2) \approx 0.85 .$$

⁷ For example, you might worry that perhaps a_0 and b_0 could be combined in an unexpected, different way, not exclusive-or'd, and that extra information beyond $a \wedge b$ would leak as a result. The derivation ensures automatically that that cannot happen.

⁸ Note that here A is male and B is female and X is neuter.

⁹ Recall however from §11.2.1 that by “any” we mean any that are continuous and convex/concave for gain/loss respectively.

¹⁰ The prior Shannon entropy on (a, b) is $1/9 \cdot \log_2 9 + 2 \cdot 2/9 \log_2 9/2 + 4/9 \log_2 9/4 \approx 1.84$.

More generally, we can allow program-development steps that increase the uncertainty (decrease the vulnerability); but we do not see any of those in this chapter. In probabilistic programming without demonic choice, strict increase in uncertainty is not so common: it occurs mainly as a result of ensuring program termination with higher probability (Chap. 16) or replacing external choice by internal choice. The latter would occur for example if we replaced a conditional between assignments with an assignment of a conditional, or a probabilistic choice between assignments with a choice from a distribution.

In a setting where we did have demonic choice, proper refinement would occur also if the demonic choices were strictly reduced during the development process. That is very common in data refinement, for example, where abstract structures are replaced by concrete ones: a set-based abstract program might be implemented by representing the sets as lists. The development is likely to be indifferent to (demonic with respect to) the order of elements in those lists; but in the implementation e.g. ascending order could be chosen for efficiency of searching. The refinement would still be correct.

In all those cases, we could still reason about the utility of our implementation by ensuring that the uncertainty of its specification was no less (its vulnerability no greater) than what we as customers could tolerate. For

$$\begin{aligned} &\text{uncertainty of specification} \geq \text{at least what we require} \\ \text{and } &\text{uncertainty of specification} \leq \text{uncertainty of implementation} \end{aligned} \quad 11$$

$$\text{implies} \quad \text{uncertainty of implementation} \geq \text{at least what we require}.$$

Indeed, with some luck (and good management) we might never have to analyze the conditional entropy (of any kind) for code like Fig. 15.2 (or e.g. the forty lines of Fig. 21.3 in Chap. 21 below). We don't need to know what that entropy *is* exactly: we need only know that it is *enough*.

15.9 Exercises

Exercise 15.1 (Uniform choice essential) Would the Encryption Lemma from Prog. (15.3) hold for *any* initialization $e \in \text{TRUE}_p \oplus \text{FALSE}$ of the encryption key e , i.e. even those for which $p \neq 1/2$?

If so, prove it. If not, give an informal *but rigorous* counterexample, and then find where the derivation of the lemma depends on $p = 1/2$. \square

Exercise 15.2 (Repeated independent runs: probabilistic vs. quantitative)

Consider the Program P that is $\text{PRINT } (b_p \oplus \neg b)$ for some Boolean b and constant probability $0 < p < 1$, and its demonic abstraction Q that is $\text{PRINT } (b \sqcap \neg b)$. (Demonic information flow is treated in Chap. 17.) Write $P^{(n)}$ for $P; \dots; P$ executed n times in a row, i.e. an example of repeated independent runs (§14.6.3). It was suggested that as n grows larger the program $P^{(n)}$ approaches $\text{PRINT } b$ — call it P^∞ . Thus P^∞ reveals b exactly.

¹¹ Equivalently, this is

$$\begin{aligned} &\text{vulnerability of specification} \leq \text{what we can tolerate} \\ \text{and } &\text{vulnerability of specification} \geq \text{vulnerability of implementation} \end{aligned}$$

$$\text{implies} \quad \text{vulnerability of implementation} \leq \text{what we can tolerate}.$$

Use your intuition to examine the same experiment for Q . Given that (\sqcap) means “bad for us” (and good for the adversary), is it reasonable to expect that Q^∞ is PRINT b in the possibilistic model as well?

[Hint: Even in the “ordinary” world of probabilistic programs without information flow (of either kind), one can find similar questions. What is the difference between

WHILE $b \{ b \in \text{TRUE } p \oplus \text{FALSE} \}$

and its demonic abstraction WHILE $b \{ b \in \text{TRUE } \sqcap \text{FALSE} \}$?
]

\square

Exercise 15.3 (Reuse of random bits) Suppose Agent A wanted to send *two* messages to Agent B , for which a specification would be

```
VISA a1,a2
VISB b1,b2
b1:= a1
b2:= a2 .
```

(15.25)

Explain informally *but rigorously* why the following implementation would not be a good idea:

```
VISA a1,a2
VISB b1,b2
{ VISAB e::= TRUE  $_{1/2} \oplus$  FALSE
  { VIS m
    m:= a1  $\nabla$  e; b1:= m  $\nabla$  e
    m:= a2  $\nabla$  e; b2:= m  $\nabla$  e
  }
}
```

- Single key.
- Use key once.
- Use it again.

(15.26)

By “rigorously” is meant that just “We are using the same key e twice.” is not enough. Required is a concrete example of a leak that the supposed implementation has but the specification does not.

Then try to adapt the earlier proof that (15.12) \sqsubseteq (15.15) so that it *does* prove (15.25) \sqsubseteq (15.26) anyway. Where does it go wrong?

\square

Exercise 15.4 (The Encryption Lemma) By specializing the implementation of the one-time pad of §15.3.3 to the case where Agents A, B are the same, with just one hidden variable h , derive the Encryption Lemma, thus showing that the two refinements are equivalent.

[Hint: Recall that $a := a$ is equal to SKIP.]

\square

Exercise 15.5 (Proof of “axioms”) In the swap assignment and PRINT case in §15.4, one of the steps was justified by the comment “statements disjoint”, as if its soundness were self-evident. But it does need proof, at a lower level. What is that level? Can you prove at that level that indeed with declarations VAR x, y we have

$$x := \text{Exp}X; \text{PRINT } y = \text{PRINT } y; x := \text{Exp}X , \quad (15.27)$$

when x and y are different variables? Does it matter if y appears in $\text{Exp}X$?

\square

Exercise 15.6 (Bolstering the intuition) Match the variables, agents and visibility declarations in Fig. 15.1 to the objects and participants featuring in the informal sketch of Footnote 3 on p. 292. \square

Exercise 15.7 (A fourth party) How would you express that in Oblivious Transfer (§15.5) there was an Agent X who could not break the separate security of A, B, T but nevertheless has broken the encryption on the network between A and B , so that it could see the messages they exchanged? Would it learn anything that it shouldn't? \square

Exercise 15.8 (A nosy third party) Recall Exercise 15.7. How would you express that in Oblivious Transfer (§15.5) the Trusted Third Party T could see the unencrypted messages between A and B ? Would it then learn anything that it shouldn't? \square

Exercise 15.9 (Completing the proof) Do the derivation of Oblivious Transfer (§15.5) for the other two points of view, that is for A and for T . (There's no need for an Agent- X proof, because for X all variables are hidden — except in Exercise 15.7.) \square

Exercise 15.10 (Collaboration) How would you investigate whether collaboration between A and T , or B and T , would break the Oblivious-Transfer protocol?

If it does break, can you figure out who precisely learns something they should not know, and what precisely that is? (That is, try to be more precise than simply “It is broken.”)

What are the general rules for visibility projections (as in §15.2) when agents are collaborating? \square

Exercise 15.11 (The double encryption lemma) Formulate and prove the “double” Encryption Lemma used in the step from (15.22) to (15.23) in §15.6. \square

Exercise 15.12 (Partial leaks) In the Lovers’ protocol (both specification and implementation, the PRINT’s are to the public generally. But you could imagine that although A and B want to know whether they are both in love, they don’t want anyone else to know. That suggests that there should be statements like $\text{PRINT}_B \text{ a}_0$ and $\text{PRINT}_A \text{ b}_0$ that leak only to some agents.

How would you define that in terms of constructs we have already? \square

Exercise 15.13 The Lovers’ protocol specification does not update the state, and thus is a pure channel even though it is written in our programming notation. The corresponding channel matrix is

$$\text{Specification: } (a, b) \left\{ \begin{array}{c} \overbrace{\quad}^{a \wedge b} \\ \downarrow 0 \\ \downarrow 1 \\ \begin{array}{c} (0, 0) \\ (0, 1) \\ (1, 0) \\ (1, 1) \end{array} \end{array} \right\} \left(\begin{array}{cc} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{array} \right),$$

where it is clear (from its containing only 0’s and 1’s) that it is a *deterministic* channel, a function from its inputs to its (sole) observable. (Note the distinction from saying that it is a function from its initial to its final state: it is indeed that too; but its final state is equal to its initial state.)

The Lovers' protocol *implementation* is also a pure channel — as it must be if it is a functional refinement of its specification: if the specification is the identity as a state-to-state function, then so must the implementation be. But it is not the same channel, at least not in the matrix formulation. The visibility declarations, that A , or B , or T (or perhaps X) can see assignments to certain variables, affects the observation set available to the adversary. If we concentrate on B as an adversary, e.g. trying to figure out whether A loves him even when he does not love her, then there are other data he has available to formulate his a posteriori conclusions.

From B 's point of view, what are the possible observations? Can you give the channel matrix from the initial state to those observations? (To do that you must refer to the code of Fig. 15.2.) Is the resulting matrix deterministic? Is it a refinement of the matrix above? \square

15.10 Chapter notes

Algebra is a powerful technique for expressing behavioral relationships between language constructors in programs. Important for algebras is the idea that for any expression E containing a subexpression F , if $F \sqsubseteq F'$ and F' replaces F in E , then the result E' is a refinement of E . This *monotonicity* property requires each program operator to be similarly monotone with respect to (\sqsubseteq) .

Such algebras have been used to describe Dijkstra's minimalist sequential programming language [1, 2], its extension for probability (but no non-determinism) [3], and its generalization to probability and non-determinism together *pGCL* [4]. A recent example of algebras for noninterference properties in sequential programs appeared in Morgan's Shadow semantics [7, 8]. General algebraic rewrite rules have been proven for quantitative interpretations as *HMM*'s [6]. These rewrite rules are proven, as sketched in §14.6.2, by interpreting the expressions (in the programming language) as functions from priors to hyper-distributions. Expressions that are deemed equivalent always produce the same output hyper-distribution for any given prior.

A more detailed discussion of the influence of Hoare, Dijkstra and others is given in the Notes for Chap. 9; and the evolution of those ideas towards probabilistic programming is discussed in Chap. 13 (§13.2.3 p. 230) and this chapter (§15.1 p. 283 and §15.10 p. 304).

The Oblivious Transfer protocol was first described by Rabin [9], and the implementation –verified algebraically in this chapter– was suggested by Rivest [10].

The Lovers' protocol is the simplest of a class of multi-party computations that can be implemented using the Oblivious Transfer protocol [11, 5].

The “earlier qualitative work” we referred to in §15.3.2 is the topic of Chap. 17.

Bibliography

- [1] Dijkstra, E.W.: A Discipline of Programming. Prentice Hall (1976)
- [2] Hoare, C.A.R., Hayes, I.J., Jifeng, H., Morgan, C.C., Roscoe, A.W., Sanders, J.W., Sorensen, I.H., Spivey, J.M., Sufrin, B.A.: Laws of programming. Communications of the ACM **30**(8), 672–686 (1987)
- [3] Kozen, D.: A probabilistic PDL. In: Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC 1983), pp. 291–297. ACM, New York (1983)
- [4] McIver, A., Morgan, C.: Abstraction, Refinement and Proof for Probabilistic Systems. Monographs in Computer Science. Springer, Berlin (2005)
- [5] McIver, A., Morgan, C.: *Sums and Lovers*: Case studies in security, compositionality and refinement. In: A. Cavalcanti, D.R. Dams (eds.) Proceedings of the 16th International Symposium on Formal Methods (FM 2009), *Lecture Notes in Computer Science*, vol. 5850, pp. 289–304. Springer, Berlin (2009)
- [6] McIver, A., Morgan, C., Rabehaja, T.: Program algebra for quantitative information flow. Journal of Logical and Algebraic Methods in Programming **106**, 55–77 (2019)
- [7] Morgan, C.: *The Shadow Knows*: Refinement of ignorance in sequential programs. In: T. Uustalu (ed.) Proceedings of the International Conference on Mathematics of Program Construction (MPC 2006), *Lecture Notes in Computer Science*, vol. 4014, pp. 359–378. Springer, Berlin (2006)
- [8] Morgan, C.: *The Shadow Knows*: Refinement of ignorance in sequential programs. Science of Computer Programming **74**(8), 629–653 (2009)
- [9] Rabin, M.O.: How to exchange secrets by oblivious transfer. Tech. Rep. TR-81, Harvard University (1981). Available at <https://eprint.iacr.org/2005/187>
- [10] Rivest, R.L.: Unconditionally secure commitment and oblivious transfer schemes using private channels and a trusted initializer. Tech. rep., MIT (1999). Available at <http://theory.lcs.mit.edu/~rivest/Rivest-commitment.pdf>
- [11] Yao, A.C.: Protocols for secure computations. In: Proceedings of the 1982 IEEE 23rd Annual Symposium on Foundations of Computer Science (FOCS 1982), pp. 160–164. IEEE Computer Society, Washington, DC (1982)



Chapter 16

Iteration and nontermination

Iteration in programming-language semantics requires a more sophisticated approach than the other constructs we have seen so far: once loops are introduced, we are forced to describe the meaning of a loop's failing to terminate, in the most extreme case for example `WHILE TRUE DO SKIP`. And we must do that even if our primary concern is loops that *do* terminate. That is the purpose of this chapter.

16.1 Why iteration is “different”

We take the conventional –and simplest– view for non-reactive sequential programs that an infinite loop is a “failure to terminate cleanly” and, now that we admit that possibility, we identify it with all other such failures — things like division by zero, or subscript out of range etc. We will use the word “nontermination” to refer to all of those, and in terms of program correctness we will not distinguish between them.

Many decades' worth of experience can be drawn on in order to describe the semantics of nontermination, and it usually involves a special nontermination state written \perp for “bottom”, a refinement order and the use of least fixed points in that order. We will begin with that in the next section. As a final word here, though, we stress that in this text we are introducing those mechanisms mainly in order to describe programs that *do* terminate; we are not (at this point) attempting to describe for example the *QIF* behavior of programs that loop forever in a useful way (such as operating systems).

16.2 Classical nontermination

In sequential non-reactive programs, the most elementary view of nontermination is that it is catastrophic: that is because from a user's perspective the computation delivers no useful result. Either it delivers no result at all (loops forever), or it delivers a result that cannot be trusted — for example an infinite loop that turned out not to be infinite after all (a space leak exhausted the program's storage, or the maximum stack-depth was exceeded), or a division by zero — or even an index out-of-bounds that resulted in overwriting code with data. That is why, in the simplest practical terms, nontermination –however caused– is equated with “arbitrary behavior from that point on”.

Most of the time that will be only mildly frustrating –e.g. having to reboot the system– but sometimes it will indeed *be* catastrophic, such as an external interrupt that leads to data’s being destroyed. Moreover, for sequential non-reactive programs there is no way to recover from nontermination except by external intervention.

The intuitive justifications for the above are based on theoretical steps (beginning in the 1970’s) in which to model nontermination the special \perp nontermination state indicated program failure, a mathematical abstraction from all the things mentioned above, like division by zero, subscript out-of-range and infinite loop... and other things too. (More specialized semantics nowadays distinguishes between those types of failure –the abstraction is less severe– but initially that was not so.) And from there, quite general and elegant theory –but applied in this simple context– led to a sequential semantics in which \perp was the “least” outcome in a refinement order, and loops’ semantics were given as least fixed-points, in that order, of a program-to-program function that represented the loop body. The refinement-least *program* was called **ABORT**, and was made semantically equal to all the catastrophic possibilities mentioned above.

Subsequent work showed how to extend that to programs allowing demonic choice (Dijkstra, 1970’s), probabilistic choice (Kozen, 1980’s; then more generally Jones and Plotkin, 1990’s), and later both demonic and probabilistic choice together.

Most relevant for us is the probabilistic case, where the \perp state for nontermination can happen with a certain probability. The model for that turned out in fact to be *subdistributions* over the final state, with \perp not explicitly mentioned, and in which nontermination’s probability was instead given implicitly by the “deficit”, the amount by which the weight of the subdistribution of final states was less than 1. This was shown to be an instance of the probabilistic powerdomain of Jones and Plotkin: their construction for probabilistic but possibly nonterminating programs generated subdistributions for the model — just as foreseen by Kozen. Later, this was extended to include demonic choice, probabilistic choice and nontermination all together, giving a model with sets and (sub-)distributions together: up-closed sets of subdistributions.

But none of the above (to our knowledge) dealt with probabilistic choice, nontermination and *quantitative information flow* all at once in sequential programs. That is what we explore in this chapter. (Demonic choice however is not included — but see the Chapter Notes.)

16.3 Nontermination for markovs and channels

In §13.1 we introduced markovs and channels as the primitives from which more complex *QIF*-programs could be built. To add nontermination, we will at first concentrate on markovs, because in §16.2 just above we recalled a “starting point” for that: probabilistic programs (without *QIF*) that might fail to terminate.

16.3.1 Nontermination for markovs

Repeating and adapting the trajectory of §13.1, a possibly nonterminating probabilistic state-update could be modeled abstractly as a function of type $\mathcal{X} \rightarrow \underline{\mathbb{D}}\mathcal{X}$, where $\underline{\mathbb{D}}\mathcal{X}$ is the set of discrete distributions on \mathcal{X} with weights *no more than* 1, rather than exactly one (as \mathbb{D} without the underline would indicate). Concretely, i.e. as Markov matrices, they correspond to the *sub*-stochastic property of having rows that sum to no more than 1. To lift this into our type of programs, we imagine such a matrix \mathbf{M} applied to a prior π where first we determine the final distribution $\pi \cdot \mathbf{M}$ by matrix multiplication.

As our theory has so far developed, a distribution π is converted to a hyper by making it a point hyper on π , that is a distribution Δ over $\mathbb{D}\mathcal{X}$ whose sole inner is π and whose outer assigns probability 1 to π . But we are now dealing with the case where π is in $\mathbb{D}\mathcal{X}$, i.e. might have weight strictly less than one. We use the same notation $[-]$ to convert that to a “sub-hyper”, which is a hyper whose inners are still one-summing, i.e. total distributions but whose outer might sum to less than one.

Definition 16.1 (Sub-point hyper) For sub-probability distribution π the sub-point hyper $[\pi]$ is the hyper whose inner is $[\pi]$, the normalization of π , and whose outer is $\langle \pi \rangle$, the weight of π . In the special case where $\langle \pi \rangle$ is zero (and so $[\pi]$ is undefined) the result is the zero-weight sub-point hyper. \square

Using Def. 16.1, the result of our possibly nonterminating markov M acting on the prior π is converted into a sub-point hyper $[\pi \triangleright M]$. We note in passing that the sub-point–hyper construction gives us a particularly succinct formulation of the function $[-]$ mentioned in Def. 4.6 that makes a hyper from a joint distribution J between \mathcal{X} and \mathcal{Y} — it is simply $\sum_y [J_{-,y}]$, in which each column $J_{-,y}$ is likely a subdistribution and is made into “its own” sub-point hyper, and then all those sub-point hypers are summed into a “normal”, i.e. likely-to-be full hyper. Removal of all-zero columns (which become all-zero inners), and aggregation of similar inners, therefore occurs automatically. Thus for example we have

$$[\pi \triangleright C] = \sum_y [(\pi \triangleright C)_{-,y}] . \quad (16.1)$$

As an example, let us write **ABORT** for the everywhere nonterminating program, and consider program **SKIP** $\frac{1}{2} \boxplus \text{ABORT}$ on the two-element state-space $\mathcal{X} := \{0, 1\}$. We have these four, equivalent ways of looking at it:

- As a classical program of type $\mathcal{X} \rightarrow \mathbb{D}\mathcal{X}$ it takes initial x to the subdistribution “ x with probability $1/2$ and $1-x$ with probability 0”.
- As a program of type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}\mathcal{X}$ it takes prior $(p, 1-p)$ to final subdistribution $(p/2, (1-p)/2)$, which sums to $1/2$ only.
- As a sub-stochastic matrix, it is

$$M: \underbrace{\mathcal{X}}_{\downarrow 0} \xrightarrow{\quad} \underbrace{\mathcal{X}'}_{\downarrow 1} .$$

$$\begin{matrix} & \mathcal{X}' \\ & \downarrow 0 \\ \mathcal{X}: & \left\{ \begin{array}{c} 0 \\ 1 \end{array} \right. & \left(\begin{array}{cc} 1/2 & 0 \\ 0 & 1/2 \end{array} \right) \end{matrix} .$$

- And finally, as a markov of type $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}\mathcal{D}\mathcal{X}$ it takes prior $(p, 1-p)$ to the sub-point hyper $\frac{1}{2} \cdot [(p, 1-p)]$, equivalently $[(p/2, (1-p)/2)]$ if we use the sub-point hyper notation from Def. 16.1.

16.3.2 Nontermination for channels

The above dealt with markovs; we now turn to possibly nonterminating channels. Intuition suggests that as matrices they, too, should be sub-stochastic: missing weight in a row indicates that the channel might “break” on that input. But does it then reveal nothing — or everything? We shall see.

We define a nonterminating channel’s behavior to be given by normalizing each of its rows, to make it terminating,¹ but running it *after* a markov whose termination probability for each initial state is the original weight of the corresponding channel row, i.e. before it was normalized. In other words, a channel that can **ABORT** with some probability $1-p_x$ on input x is modeled as a channel that is scaled up, for each x , so that its termination is certain, but as a whole is preceded by a **SKIP**-markov whose termination however for any x is only p_x . (The corresponding Markov matrix has p_x ’s on the diagonal and zeroes elsewhere.)

16.3.3 Applying abstract channels and markovs to sub-hypers

We saw in §13.4.1 how Kleisli composition defines sequential composition and, in particular, how the outers of the intermediate hyper are carried through as weights when the output hypers are combined into one. We now have the possibility that the intermediate hyper might be a *sub*-hyper; but the treatment is just as before — the weights are carried through.

A surprising consequence of that construction is that a partial channel *can* leak information as a result of its failure to terminate, and —even worse— might behave in a way that the adversary knows is inconsistent with her (certain) knowledge of the prior — for example, the final hyper might suggest that the final state has some value that the adversary knows was not in the support of the prior and “therefore”, because a channel does not change the state, should not be in the final state either, no matter how small its probability. The reason that can happen is that a partial channel actually *can* change the state — if it fails to terminate cleanly.

A second surprise is that the following alternative interpretation method is equivalent to the above: declare a new fresh variable `{ VAR y:γ ... }` and interpret the (partial) channel as a partial *assignment* to that y , for whose semantics we can refer to the partial-distribution construction in §16.3.1. Follow that by `... ; PRINT y }` and close the scope (of y). That gives the same result as the method above.

In the second method it looks at first as if x cannot be affected, since the probabilistic assignment targets y only. (In the first method x is assigned-to explicitly by the pre-executed partial-identity markov.) But what has happened is that we have exposed another (simplifying) assumption we are making, which is that an assignment to y that fails can affect other variables as well (in this case x) as the system crashes.² Thus if we want a semantics for partial channels that *cannot* affect the input in the case of failure, it’s not in fact at *channel* semantics that we should be looking: instead we must (re-)examine sequential-program semantics itself, to find a more discriminating view in which e.g. `x,y:= 1,1/0` cannot affect the final value of x .³ But we do not do that here.

¹ If the channel row is completely zero, it can be set arbitrarily: the preceding markov will never activate that row.

² This effect occurs also in Hoare/Dijkstra semantics of conventional sequential programs, where for example the forever looping program `x:= 0; WHILE true { SKIP }` can still terminate and set the variable x to 1.

³ And that is not an unreasonable thing to consider: one could assume that an adversary’s side-channel radio receiver could fail without affecting the system she is observing. That is an encouragement to investigate the more discriminating semantics.

16.3.4 The semantic model for nontermination: sub-hyper-distributions

Our conclusion from the above is that our model for nontermination and hyper-distributions will be *sub-hyper-distributions*, that is hyper-distributions whose inners are full distributions $\mathbb{D}\mathcal{X}$ but whoseouters may sum to less than one. We write that $\underline{\mathbb{D}}\mathcal{X}$ and call them “sub-hypers”, remembering however that only the outer is possibly “sub” — the inners remain full distributions in $\mathbb{D}\mathcal{X}$.

16.4 The algebra of nontermination in QIF

A theme of our program semantics and its algebra for *QIF* (earlier Chapters 13 & 14 and Chap. 21 to come) is “*gedanken experiments*” based on reasoning principles applied to very small examples.⁴

Below is a selection of the experiments and analogies on which we have relied for our approach to nontermination.

All ABORT's are the same. This is an analogy — we do not (cannot) prove it, but rather adopt from conventional sequential programming in the early stages of its formalization (as we are here in an early stage of our formalization of *QIF*-aware programs). And it could be regarded however as characterizing non-reactive semantics, and is informally justified by the remarks in §16.2 above.

We realize that in many computational models **ABORT**'s are not all the same; those models typically are historically later in the development of semantics; and perhaps the same path later will be open to us. For now, though, our informal justification is that our program semantics is non-reactive, i.e. end-to-end. It is not possible in our approach for the adversary to react to an intermediate leak and somehow influence the program's further behavior from that point, based on what she saw.

Thus we write **ABORT** for any infinite loop, irrespective of its body, and for any (other) failure: division by zero, array-subscript error. They are all equal.

ABORT is refined by anything⁵ Like the previous item, this is by analogy — and to make the analogy we temporarily broaden our scope to include demonic choice (\sqcap). (Demonic choice is discussed in its own right in Chap. 17.) In sequential programs, refinement (\sqsubseteq) and demonic choice (\sqcap) are aspects of the same partial order, that is that (\sqcap) is the greatest lower bound wrt. refinement (\sqsubseteq) or, equivalently that $S \sqsubseteq I$ just when $S = S \sqcap I$.

Thus **ABORT** is the (\sqsubseteq)-least program.

ABORT is a right-zero of sequential composition This is *not* an assumption, but rather a *gedanken*-style consequence of the two assumptions above — i.e. if we make those assumptions, and expect a reasonable program algebra to ensue, what equalities “must” we admit? We reason

⁴ The allusion to *gedanken* experiments is made to highlight our approach to the design of the semantics: we ask “If we demand that the program algebra have these reasonable properties *then* what would the semantics have to be?” with the analogy being “If light's speed were the same in all inertial frames, what would the behavior of time and length have to be?”

⁵ The (\sqcap) in this example is demonic choice, introduced under the assumption that eventually we *will* have a semantics that accommodates it and *QIF* as well. Thus we should not make any steps now that would lead to inconsistencies later, when that further extension is achieved.

	ABORT	
=	WHILE true { <i>Prog</i> }	“for any <i>Prog</i> ”
=	<i>Prog</i> ; WHILE true { <i>Prog</i> }	“unfold once”
=	<i>Prog</i> ; ABORT	“as above”

ABORT leaks potentially everything This again is a consequence –of the previous item– noting that “*Prog*” there includes the program PRINT *Exp* for any expression *Exp*.

Thus we have ABORT \sqsubseteq PRINT *Exp* directly.

This last deserves special attention: it means that we are taking the view that the program

$$\text{WHILE true } \{ \text{Prog} \} \quad (16.2)$$

does *not* remain safely humming away, producing heat, but no information other than that its loop guard evaluates to *true*. That is, more generally we might have the program WHILE *x*=0 { SKIP }, and although an adversary could infer from nontermination that *x*=0 at least while the program is running, she could not however make that inference about the final state (if there is one) — because if the program aborts then it could set *x* to anything. That is, if *x* is initially 0 then the program acts as ABORT and, as such, it might leak anything, and change the state in any way.

As a second example, we note that observing the program

$$\begin{array}{ll} b := 0 \oplus 1 \oplus 2 & - \text{ Uniform choice (internal) from } \{0,1,2\}. \\ \text{WHILE } b=0 \{ \text{SKIP} \} & \end{array} \quad (16.3)$$

does *not* allow the adversary to deduce that “If the program terminates with *b* \neq 0 then *b* is equally likely to be 1 or 2.” Instead, it produces the (sub-)hyper $[(0, 1/3, 1/3)]$, equivalently $2/3 \cdot [(0, 1/2, 1/2)]$, from which she can be sure that on termination (if it occurs) variable *b* will be 1 with probability at least $1/3$, and similarly for 2. But that is all: in particular, she cannot reason “The program terminated, therefore *b* cannot be 0.”

For we must remember that one possible consequence of *non-termination* is *apparent* termination but with corrupt results: that kind of “catastrophe” –an incorrect answer masquerading as a correct one– is just as it is in classical Hoare/Dijkstra semantics for sequential programs. However one ABORT-induced deception that our semantics does *not* allow is for the program’s leaks to suggest to the adversary that a variable *x*’s final value *x* is according to some posterior distribution δ where in fact *x* is not in the support $[\delta]$ of that posterior — for that would be inconsistent, rather than merely surprising: for example a leak leading to the inference (the posterior) “The value of *x* is uniformly distributed between 1 and 2.” cannot occur if *x* is actually 0.

An interesting contrast with the program Prog. (16.3) above is the program

$$\begin{array}{l} - \text{ Uniform choice (external) between the three statements.} \\ b := 0 \boxplus b := 1 \boxplus b := 2^6 \\ \text{WHILE } b=0 \{ \text{SKIP} \} , \end{array} \quad (16.4)$$

which produces the partial hyper $1/3 \cdot [(0, 1, 0)] + 1/3 \cdot [(0, 0, 1)]$, meaning that with probability $1/3$ the adversary will know that *b*=1, and similarly for 2. With the

⁶ We informally use a “row” of \boxplus ’s for uniform (external) choice. However we note that as a binary operator the unadorned \boxplus has no reasonable algebra.

remaining (outer) probability $1/3$ the adversary could learn that the posterior is e.g. $(1/3, 1/3, 1/3) \dots$ or indeed anything else.

To see the difference more clearly, consider an adversary who is trying to guess b . For Prog. (16.3), her best strategy is to guess 1 or 2 – it does not matter which – and with probability at least $1/3$ she will be right. But we – as defenders – must assume the worst if b is set to 0, which will (also) happen $1/3$ of the time: the adversary could be right every time.

For Prog. (16.4) the situation is different, because the probabilistic choice of b is external. (In Prog. (16.3) it was internal.) Once b has been set, she will know for sure what it is and, if it's 1 or 2, she should of course guess that: her probability of being right is thus $2/3$, which comprises $1/3$ of her knowing b is 1 and guessing “1”, and the same for $b=2$. But again, if b is set to 0, we defenders should assume the worst.

We devote a final remark to “assume the worst”. It is a difficult situation to describe with gain functions, as it suggests a gain of ∞ in that case. This issue of “worst” as it arises from nontermination is precisely the reason that we use loss functions for programs, instead of gain functions: the worst is now easily characterized as “loss of 0”, which is indeed worst from the defender’s point of view, since loss functions are usually constrained to be non-negative.

16.5 A refinement order on *sub*-hyper-distributions

In this section we extend our existing refinement order (\sqsubseteq) on *total* hypers to one on the sub-hypers $\mathbb{DD}\mathcal{X}$ (from §16.3.4) that takes possible nontermination into account, and makes ABORT the least element as foreshadowed above: for clarity in this section only we will call that extended order (\sqsubseteq_P) for “partial”, and we will for clarity temporarily use (\sqsubseteq_T) instead of (\sqsubseteq) for our existing order on “total” distributions. Further, we introduce temporarily a supplementary order (\sqsubseteq_\perp) on (partial) hyper-distributions that concerns their termination probability only.

Thus our semantic space is *partial* hyper-distributions, that is hypers whoseouters can sum to less than one. The inners remain total, which is why we write it $\mathbb{DD}\mathcal{X}$. Our first step is to define the special “termination only” order (\sqsubseteq_\perp) as follows.

Definition 16.2 (Termination order) Let Δ, Δ' be sub–hyper-distributions in $\mathbb{DD}\mathcal{X}$. Then

$$\Delta \sqsubseteq_\perp \Delta' \quad \text{iff} \quad \Delta_\delta \leq \Delta'_\delta \quad \text{for all } \delta \text{ in } \mathbb{D}\mathcal{X}.$$

□

Thus (\sqsubseteq_\perp) on sub-hypers is just pointwise lifted (\leq) on the \mathbb{D} , i.e. the outer part, which is consistent with the termination order for non-*QIF* but still probabilistic programs. That is, Def. 16.2 orders (partial) hyper-distributions in $\mathbb{DD}\mathcal{S}$ so that $\Delta \sqsubseteq_\perp \Delta'$ only when the probability of any posterior's occurring is at least as great in Δ' as it is in Δ . This means that *non* termination in Δ' is overall less likely than in Δ .

It is straightforward to see that (\sqsubseteq_\perp) is a complete partial order, by which we mean that any (\sqsubseteq_\perp)-chains have well defined limit.

Our second step is to extend our existing order on $\mathbb{D}^2\mathcal{S}$, temporarily called (\sqsubseteq_T), to an order (\sqsubseteq_P) on $\mathbb{DD}\mathcal{S}$, but effectively only between sub–hyper-distributions whoseouters sum to the same probability. (Since total hypers, those in $\mathbb{D}^2\mathcal{S}$, all have the same weight 1, this extended order (\sqsubseteq_P) agrees with the existing (\sqsubseteq_T) on their common domain.)

Write $\Delta \sim \Delta'$ just when their (outers') weights are equal, that is $\langle \Delta \rangle = \langle \Delta' \rangle$, where in general we write $\langle \Delta \rangle$ for the sum of the outer probabilities in Δ (just as in general we use $\langle - \rangle$ for the sum of the probabilities in any (sub-)distribution).

In such cases (\sqsubseteq) can be extended easily because we can normalize the outers of both Δ and Δ' by the same factor, and apply (\sqsubseteq_T) directly to the normalizations.

Definition 16.3 (Partial security refinement order) Let Δ, Δ' be (nonzero) hyper-distributions in $\mathbb{DD}\mathcal{X}$. We extend the total-hyper order (\sqsubseteq_T) to partial hypers as follows. We have

$$\Delta \sqsubseteq_P \Delta' \quad \text{iff} \quad \Delta \sim \Delta' \quad \text{and} \quad \Delta/p \sqsubseteq_T \Delta'/p ,$$

where $p > 0$ is their shared normalizing factor. (If the common weight p of Δ, Δ' is zero, then they are equal and thus can be made to (\sqsubseteq_P) -refine each other by definition.) \square

Finally we combine (\sqsubseteq_P) and (\sqsubseteq_\perp) to generalize both orders to a single (unsubscripted) refinement order (\sqsubseteq) that handles both increased termination and increased security.

Definition 16.4 (Refinement and Termination) Let Δ, Δ' both be (sub-)hyper-distributions in $\mathbb{DD}\mathcal{S}$. We say $\Delta \sqsubseteq \Delta'$ just when there is some intermediate Δ'' such that $\Delta \sqsubseteq_\perp \Delta''$ and $\Delta'' \sqsubseteq_P \Delta'$. (Note that the latter implies $\Delta'' \sim \Delta'$.) That is, we define

$$(\sqsubseteq) := (\sqsubseteq_\perp) \circ (\sqsubseteq_P) ,$$

where (\circ) is composition of relations. ⁷ \square

Note that if Δ, Δ' are terminating, then necessarily $\Delta \sim \Delta'$ already, so that our new order agrees with our old order in that case.

Before we move on, let us collect together the steps we have taken so far. We began with (\sqsubseteq) , the partial order of refinement defined on (total) hyper-distributions that has been our principal order to this point. (In earlier chapters we have two apparently independent definitions of it, the structural definition (\sqsubseteq_s) and the testing definition (\sqsubseteq_G) . A major result (Thm. 9.13) was then that they were the same in meaning, that is $(\sqsubseteq_s) = (\sqsubseteq_G)$ — and so when dealing with meaning alone we use (\sqsubseteq) for both.)

Then, for this chapter, we renamed our earlier (\sqsubseteq) to (\sqsubseteq_T) , to remind us that its domain is (only) total hypers. We then introduced (\sqsubseteq_P) which extends to sub-hypers (including total hypers as a special case) but holds only between hypers of the same weight; and we introduced (\sqsubseteq_\perp) which mimics increased-termination refinement from classical probabilistic (but non-*QIF*) programming.

And finally we (re)-defined (\sqsubseteq) to generalize it from its earlier meaning (\sqsubseteq_T) on total hypers.

Now we continue, from Def. 16.4 of (\sqsubseteq) , with the observation that with it we have captured the point of view that an increase of security *or* of termination for the user are both improvements, because in both cases their consequences are more tolerable (for the defenders of a secret or the users of a program). For semantics however we would like to show that Def. 16.4 determines not only a partial order, but one that is complete, i.e. having the property that limits of increasing chains exist. The next lemma describes some basic properties of (\sqsubseteq) that we will need.

⁷ The opposite composition $(\sqsubseteq_P) \circ (\sqsubseteq_\perp)$ is not the same: in fact $(\sqsubseteq_P) \circ (\sqsubseteq_\perp) \subset (\sqsubseteq_\perp) \circ (\sqsubseteq_P)$. See Exercise 16.2.

Lemma 16.5 (Partial-order properties) The following properties hold for all hyper-distributions $\Delta, \Delta' : \mathbb{DD}S$:

1. If $\Delta \sqsubseteq_{\perp} \Delta'$ then $\Delta \sqsubseteq \Delta'$ — more succinctly $(\sqsubseteq_{\perp}) \subseteq (\sqsubseteq)$.
2. If $\Delta \sqsubseteq_P \Delta'$ then $\Delta \sqsubseteq \Delta'$ — more succinctly $(\sqsubseteq_P) \subseteq (\sqsubseteq)$.
3. $(\sqsubseteq_P) \circ (\sqsubseteq_{\perp}) \subseteq (\sqsubseteq_{\perp}) \circ (\sqsubseteq_P)$, where the parentheses (\cdot) indicate taking the binary relation within it as a set of pairs which, therefore, can be compared (with (\supseteq)) and composed (with (\circ)). (See Exercise 16.1.)
4. Refinement (\sqsubseteq) can't decrease probabilistic weight: if $\Delta \sqsubseteq \Delta'$ then we have $(\mu\Delta)_x \leq (\mu\Delta')_x$ for all x in \mathcal{X} .

Proof. Items (1) and (2) are trivial because both (\sqsubseteq_P) and (\sqsubseteq_{\perp}) are reflexive.

Item (3) holds because if $\Delta((\sqsubseteq_P) \circ (\sqsubseteq_{\perp}))\Delta'$ then there must be some hyper Λ such that $\Delta \sqsubseteq_P \Lambda \sqsubseteq_{\perp} \Delta'$. Observe that obtaining Δ' from Λ requires the increase of Λ by some sub-hyper-distribution Λ' . But we can also obtain Δ' by first increasing Δ by that same Λ' (thus a (\sqsubseteq_{\perp}) -step) and then, in a subsequent (\sqsubseteq_P) -step, redistributing *only* the inners belonging to Δ (in order to make Λ) leaving the inners added by Λ' alone. All of this can be arranged by simple block matrix manipulation.

Finally (4) holds because (\sqsubseteq_{\perp}) only increases probability weight, and although (\sqsubseteq_P) redistributes it, it does not change the total weight. \square

A corollary of those basic properties is that (\sqsubseteq) is a partial order.

Corollary 16.6 (Partial order) (\sqsubseteq) is symmetric, reflexive and transitive on $\mathbb{DD}S$.

Proof. Reflexivity of (\sqsubseteq) holds because it inherits that property from its constituents. Similarly transitivity follows by transitivity of (\sqsubseteq_P) and (\sqsubseteq_{\perp}) together with Lem. 16.5(3).

For antisymmetry we reason that if both $\Delta \sqsubseteq \Delta'$ and $\Delta' \sqsubseteq \Delta$ hold, then there must be Λ and Λ' such that $\Delta + \Lambda \sqsubseteq_P \Delta'$ and $\Delta' + \Lambda' \sqsubseteq_P \Delta$. We can now reason

$$\begin{array}{lll} \sqsubseteq & \Delta + \Lambda & \\ \sqsubseteq & \Delta' & \text{"}(\sqsubseteq_P) \subseteq (\sqsubseteq)\text{"} \\ \sqsubseteq & \Delta & \text{"assumption"} \end{array}$$

and thus Λ must be zero, since (\sqsubseteq) never reduces strictly the overall weight of hyper-distributions. A similar calculation shows that Λ' is also zero. Thus we must have that $\Delta \sim \Delta'$ and the result follows by antisymmetry of (\sqsubseteq_P) . \square

We now look at the fixed-point properties of (\sqsubseteq) .

Theorem 16.7 (Equivalence of fixed points) Let partial orders (\sqsubseteq_{\perp}) and (\sqsubseteq) be as defined above over $\mathbb{DD}\mathcal{X}$, and let \mathcal{L} be a function of type $\mathbb{DD}\mathcal{X} \rightarrow \mathbb{DD}\mathcal{X}$. If a (\sqsubseteq_{\perp}) -least (resp. greatest) fixed point of \mathcal{L} exists, then also a (\sqsubseteq) -least (resp. greatest) fixed point of \mathcal{L} exists, and in fact they are equal.

Proof. Observe that $(\sqsubseteq_{\perp}) \subseteq (\sqsubseteq)$, and that it is an elementary property of any two partial orders with $(\sqsubseteq_1) \subseteq (\sqsubseteq_2)$ that a least/greatest fixed point wrt. (\sqsubseteq_1) , if it exists, is also a least/greatest resp. fixed point wrt. (\sqsubseteq_2) . \square

The significance of Thm. 16.7 is that it exploits the fact that when loop semantics is given by a least fixed-point of partial iterates, the chain (of which the loop's semantics

is the limit) is actually a (\sqsubseteq_{\perp}) -chain: that is, that successive iterates improve (refine) termination but do not refine secrecy. (The same happens in classical sequential-program loop semantics in a setting including demonic choice, where the iterates improve termination but do not reduce the demonic nondeterminism otherwise.) Thus the theorem shows that existence of the (\sqsubseteq_{\perp}) -fixed point suffices for well-definedness of loop semantics in this more general (\sqsubseteq) -setting.

Thm. 16.7 says that the predictable (to the user) information leaks caused by programs are the accumulation of the information leaks during the execution provided the program terminates. If the program does not terminate (or “has not terminated yet”) then the amount of total information flow on termination cannot be predicted, and so the user should assume the worst: i.e. that *all* his information could be leaked.

We now confirm that our new order (\sqsubseteq) on sub-hypers, when restricted to total hypers, is indeed the same as the temporarily renamed refinement (\sqsubseteq_T) that we have used earlier, until this chapter: we extend the Coriaceous theorem (§9.5) to sub-hypers.

Theorem 16.8 (Refinement (\sqsubseteq) equals loss order)

Let $\Delta, \Delta' : \mathbb{DDS}$ be sub-hyper-distributions. Then

$$\Delta \sqsubseteq \Delta' \quad \text{iff} \quad U_\ell(\Delta) \leq U_\ell(\Delta') \quad \text{for all loss functions } \ell.$$

Proof. This follows because the proof of the Coriaceous Theorem (Thm. 9.12 on page p. 155) relies on the construction of a convex and closed Euclidean subspace based on Δ . The same construction produces a convex-closed subset even when Δ is a sub-hyper-distribution. \square

Thm. 16.8 says that the only ways to increase the loss (or decrease the gain) to the attacker are to terminate in a more predictable state or to increase security. However by including nontermination we have now introduced an asymmetry between loss and gain functions. In effect we are measuring the loss to the attacker as *zero* if a program does not terminate — in other words we are modeling a possibility that nontermination could be to the attacker’s advantage, and therefore a disaster for the user. A mathematical consequence is that the duality between loss- and gain functions operates only when reasoning about programs that are guaranteed to terminate. If they are not, then we must use loss functions.⁸

16.6 From nontermination to termination

As we mentioned in the introduction to this chapter, we are forced to consider nontermination (of loops), and to build semantic apparatus for it, even though our primary interest here is in loops that *do* terminate. The next section gives an example.

Still it is a legitimate question at this point to ask how we prove loop termination in the *QIF* context. Luckily, mere termination of the loop is a functional property — and so the existing techniques for proving termination of probabilistic programs (without *QIF*) suffice for that. For loops such as the one below, a standard variant argument suffices because the termination is certain. In properly probabilistic cases, where termination occurs only “almost surely” (i.e. with probability one), there are many techniques based on probabilistic variants that suffice. (See the Chapter Notes.)

⁸ This asymmetry is not new in theorems of programming: in models that also include angelic and demonic nondeterminism, there is a duality similar to loss and gain functions, but only when the distributions over states are normalized.

16.7 Example of (certain) termination: how to design a password checker

We now consider a simple password-checking algorithm: it iterates over two arrays `pw` and `gs` containing the password and the guess respectively. (And its termination is certain.)

The two string inputs `pw` and `gs` are of length N . The string `pw` is a stored password and `gs` is a guess — the password checker compares `pw` and `gs` character-by-character, and returns *false* at the first opportunity:

```
i:= 0
ans:= TRUE
WHILE ans  $\wedge$  i<N {
    IF pw[i] $\neq$ gs[i] THEN ans:= FALSE - Side channel here.      (16.5)
    i:= i+1
}
PRINT ans - Superfluous: the side channel has leaked ans already.
```

The output of this program is a hyper-distribution that takes into account all of the observations during program execution — in this case, as explained earlier, there is an information leak both at the loop guard and at the conditional statement and, taken together, those leaks constitute a so-called “timing attack”.

For example, for a given guess `gs` the observations could be that $pw[i]\neq gs[i]$ returns *true* twice, and then *false* — and that means that the first two characters of `gs` matched `pw` but the third did not. From that the adversary is able to use that information to reduce her effort in a brute-force guessing strategy.

Suppose that the adversary knows that the password is some permutation of the string "abc" so that the prior is a uniform distribution over those 6 permutations. Suppose further that the adversary's guess is "abc"; in this case (and in fact for any guess) there are three observations: the loop terminates after either one, two or three iterations. Notice that it is not possible for the loop to terminate after returning *true* twice and then *false* once in this scenario — because if two of the characters match then the third must match as well. This hyper-distribution summarizes the complete situation, where the comments summarize the observations that led to the adjacent inner:

$1/6$	1	"abc"	3 <i>true</i> observations
$1/6$	1	"acb"	1 <i>true</i> and 1 <i>false</i> observation
$2/3$	$1/4$	"bac"	1 <i>false</i> observation
	$1/4$	"bca"	
	$1/4$	"cab"	
	$1/4$	"cba"	.

We see that the side channel has had a significant impact, because $1/6+1/6 = 1/3$ of the time the adversary knows the secret completely, rather than simply knowing that the guessed password is incorrect but nothing else. A more secure version of Prog. (16.5) would report the result only after every character has been tested. The following, rather more cryptic design does just that: only if `ans` is false after the complete execution of the loop does it mean that the guess and the password differ in at least one position — but this time the adversary receives no information about which character(s) differ:

```

i:= 0
ans:= true
while i<N {
    ans:= (pw[i]=gs[i]) $\wedge$  ans           - Hidden comparison.      (16.6)
    i:= i+1
}
PRINT ans .                         - No longer superfluous.

```

The output from Prog. (16.6) with the password known to be some permutation of "abc" is as follows, now with only two possible observations:

1/6	1	"abc"	ans = true
5/6	1/5	"acb"	ans = false
	1/5	"bac"	
	1/5	"bca"	
	1/5	"cab"	
	1/5	"cba"	.

In terms of information flow Prog. (16.6) releases the least information possible. However if a user mistypes his password he might prefer not to wait until all the characters have been checked, but rather have the checking terminate early. The next design attempts to allow an early exit the first time that a character does not match, while reducing the severity of the timing attack by randomizing the order in which the characters are tested. The resulting design is as follows.

```

i:= 0
ans:= TRUE
leftToCheck:= {0, ..., N-1}
WHILE ans  $\wedge$  i < N {
    i:= uniform(leftToCheck)           - Choose i randomly.      (16.7)
    IF pw[i] $\neq$ gs[i] THEN ans:= FALSE - Side channel here.
    leftToCheck:= leftToCheck - {i}
}
PRINT ans ,

```

where `uniform(leftToCheck)` is a uniform distribution over the set `leftToCheck`.

Rather than checking the characters in order, the next character to check is selected uniformly from those that remain: the set-valued variable `leftToCheck` keeps a record of the indices left to test. Assuming again that the password is a permutation of "abc", we can see that an adversary using guess "abc" again can make three possible observations as for Prog. (16.5), but what she is able to deduce is quite different — as this hyper-distribution indicates:

1/6	1	"abc"	3 true observations
1/6	1/3	"acb"	1 true and 1 false observation
	1/3	"bac"	
	1/3	"cba"	
2/3	1/6	"acb"	1 false observation
	1/6	"bac"	
	1/4	"bca"	
	1/4	"cab"	
	1/6	"cba"	.

This time we see that the only case in which the adversary can determine the password exactly is if her guess is correct, and that occurs with probability $1/6$. In both the other two cases uncertainty remains — in the case of one *true* and one *false* observation, the adversary deduces that exactly one position is correct, and so the remaining uncertainty is equally distributed amongst "acb", "bac" and "cba". In the only other observation, all the adversary can deduce is that *at least* one of the characters in the guess is incorrect, and it is just as likely (probability $1/2$) that all three characters are incorrect or exactly two characters are incorrect.

To decide whether Prog. (16.7) is “more secure” than Prog. (16.5) we could consider computing the Bayes vulnerability with respect to the uniform prior — in the former case we see that it is $7/18$ but in the latter it is $1/2$. However in some scenarios, as defined by a gain function, an adversary would indeed be able to gain more with the information released by Prog. (16.5) (Exercise 16.5 below), so in fact Prog. (16.7) is not more secure in *all* situations.

16.8 A taxonomy of refinement orders

We now draw all the above together: happily, beyond this point we will not be introducing further variations on refinement (except (\preccurlyeq) briefly again in §17.8). The refinement orders we have considered are as follows:

General –

structural refinement (\sqsubseteq_o)

Def. 9.5 p. 150

This was originally defined between concrete channel/prior pairs interpreted as matrices, and uses a refinement matrix. Its definition for abstract channels is based on that; and it extends to HMM’s (i.e. with markov updates) as well. Conceptually it captures the idea of *systematic modification of a specification program* in ways that preserve its correctness where, for us here, “correctness” includes quantitative security. It’s what developers do.

testing refinement (\sqsubseteq_G)

Def. 9.10 p. 152

This is defined with respect to the *results* that a program is intended to achieve, and is instrumented by gain- or loss functions that capture in an informal way the “economic” benefit or cost that an adversary can on average expect when using a particular kind of attack. Here the main concept is that “refinement means more loss (less gain) for an adversary”. It’s what customers rely on.

refinement for terminating programs (\sqsubseteq)

Remark (9.1) p. 157

A major result is that the two refinement definitions above, that is structural and testing refinement, are equivalent within the framework we have defined. That is, a program S is structurally refined by a program I means not only that I has been developed systematically but as well that the *result* of that development is guaranteed not to impair security (with respect to the security measures we consider). Furthermore, if I is guaranteed to be at least as secure as S (as we define that), then it *must* in principle be able to be reached from S via a systematic development.

That $(\sqsubseteq_o) = (\sqsubseteq_G) = (\sqsubseteq)$ is what allows developers and customers to work together.

For compositional closure –**primitive refinement** (\preccurlyeq)

Def. 9.17 p. 162

This was introduced temporarily, as a stepping stone to understanding why general refinement (\sqsubseteq) is defined the way it is, in particular that it cannot really be any simpler. The argument is that primitive refinement is a “man in the street” concept in the sense that it’s easy to argue that it must at least be that $S \preccurlyeq I$ if I is to be considered seriously at all. It’s helpful in fact to look negatively — and in our case, because (\preccurlyeq) was defined to be comparison of Bayes vulnerability, the criterion became “If the chance of guessing the secret in one try after running I is *more* than after running S , then I cannot be a (primitive) refinement of S . ”

The key follow-up to that beginning was to show that if $S \not\sqsubseteq I$ according to our “complicated” definition of (\sqsubseteq), then in fact there would be circumstances (a context) in which $\mathcal{C}S \not\leq \mathcal{C}I$. That approach was called *compositional closure*.

For possible nontermination and the definition of iteration –**partial refinement** (\sqsubseteq_P)

(used in this chapter only)

This was an extension of refinement (\sqsubseteq) for terminating programs, allowing the comparison of possibly nonterminating programs as well, provided their probabilities of terminating were equal.

total refinement (\sqsubseteq_T)

(used in this chapter only)

This was merely a temporary renaming of (\sqsubseteq) from above, so that we could talk about “the earlier definition” unambiguously while formulating a more general replacement for it (as below).

termination refinement (\sqsubseteq_{\perp})

(used in this chapter only)

This refinement improves termination behavior *only*, while leaving security characteristics otherwise the same. It was pointed out that loop semantics generates a (\sqsubseteq_{\perp})-chain, whose properties are substantially simpler than they would be for a (\sqsubseteq_T)-chain.

All brought together –**refinement** (\sqsubseteq)(combines (\sqsubseteq_P) and (\sqsubseteq_{\perp}) and generalizes (\sqsubseteq_T))

This, our final version, is a combination of termination refinement and then partial refinement. It allows programs to be improved both in their termination and in their security, and generalizes the (\sqsubseteq) we had been using until this chapter. Within this chapter the earlier (\sqsubseteq) was renamed to (\sqsubseteq_T) so that we could state the generalization, i.e. that on terminating programs our new (\sqsubseteq) agrees with the (\sqsubseteq_T) from Chap. 9 (Refinement), which unified (\sqsubseteq_{\circ}) and (\sqsubseteq_G).

Thus, specialized to terminating programs only, our (\sqsubseteq) here is the same as (\sqsubseteq_T) earlier, i.e. the refinement order we developed in earlier chapters. And so now we drop the (\sqsubseteq_T), and return to the simple (\sqsubseteq) again — but we apply it over a wider domain.

16.9 Exercises

Exercise 16.1 Prove Lem. 16.5(3), that $(\sqsubseteq_P) \circ (\sqsubseteq_{\perp}) \subseteq (\sqsubseteq_{\perp}) \circ (\sqsubseteq_P)$.

[Hint: Write the hypers as (possibly unreduced) joint-distribution matrices, and use the equivalent matrix operation for the hyper-operation $(+\Lambda)$ from the proof of Cor. 16.6.] \square

Exercise 16.2 Give an example to show that the inclusion $(\sqsubseteq_P) \circ (\sqsubseteq_{\perp}) \subset (\sqsubseteq_{\perp}) \circ (\sqsubseteq_P)$ is indeed strict. \square

Exercise 16.3 What is the output hyper-distribution of Prog. (16.5) if the password is some permutation of "abc" but the adversary guesses "abx"? \square

Exercise 16.4 Compute the multiplicative Bayes leakages for Programs (16.5) and (16.7); hence or otherwise show that the multiplicative capacity of Prog. (16.5) is more than that of Prog. (16.7). \square

Exercise 16.5 Show that there is a gain function g such that the corresponding g -leakage is strictly greater for Prog. (16.7) than for Prog. (16.5).

[Hint: compare relative frequencies of the secrets within each observation.] \square

16.10 Chapter notes

In semantic models for nondeterministic sequential programs, nontermination is often handled through limits in powerdomains. In the Smyth powerdomain [13] for example one subset X of results is dominated by another Y provided that any result in Y can be approximated by some result in X . In relational models for (nondeterministic) sequential programs, nontermination is often considered to be the worst outcome, and therefore in the Smyth ordering, termination in *any* proper state is preferred. That is the approach taken in simple relational models, e.g. in Hoare Logic and in the weakest preconditions of Dijkstra's guarded commands [1], and leads to the conclusion that for example an infinitely looping program can terminate and set variables arbitrarily.

Probabilistic powerdomains similarly allow an underlying order on states to induce an order on probabilistic behavior. Jones and Plotkin [2, 3] applied that idea to Kozen's probabilistic PDL [4]. The result was to describe probabilistic results in terms of subdistributions where the deficit from 1 is deemed to be the probability of nontermination. Like the Smyth order on nondeterministic result-sets, that allows any terminating behavior to be considered an improvement to nonterminating behavior. The further addition of demonic choice to the mix [12, 7] also takes that view.

Techniques for proving termination of loops like

```
b := TRUE; WHILE b DO b := TRUE  $\frac{1}{2} \oplus$  FALSE
```

rely on a relaxed definition of termination: the loop above could iterate forever, but the probability of its doing so is zero. In that case one says that it “terminates almost surely”. McIver et al. summarize techniques for doing that [8].

The use of “*gedanken experiments*”, i.e. “thought experiments” is intended to expose features of the semantics that we expect to be essential for a practical development method within our terms of reference — that is, taking into account the program-behavior criteria that we have carefully circumscribed in advance. Used in that way, the algebra can be explored *before* the semantics rather than derived afterwards: that is, algebra guides the design of semantics, rather than the other way around.

That approach was earlier used in [9, 10, 11], whose “sets of (shadow) sets” for probabilistic information flow contributed to the idea of hypers as we use them here: the earlier sets of sets, once quantified, they became “distributions of distributions”. The correspondingly generalized “demonic collections” of expectations, considered for the *QIF* program logic and which (remarkably) turned out to be the same as loss functions.

Although this chapter brings together the ideas of information flow as a consistent extension of probability and nontermination (only) [6], external demonic nondeterminism can also be included at the cost of some complication [5].

Bibliography

- [1] Dijkstra, E.W.: A Discipline of Programming. Prentice Hall (1976)
- [2] Jones, C.: Probabilistic nondeterminism. Monograph ECS-LFCS-90-105, The University of Edinburgh (1990). (Ph.D. Thesis)
- [3] Jones, C., Plotkin, G.: A probabilistic powerdomain of evaluations. In: Proceedings of the 1989 4th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 1989), pp. 186–95. IEEE, Los Alamitos (1989)
- [4] Kozen, D.: A probabilistic PDL. In: Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC 1983), pp. 291–297. ACM, New York (1983)
- [5] McIver, A., Meinicke, L., Morgan, C.: A Kantorovich-monadic powerdomain for information hiding, with probability and nondeterminism. In: Proceedings of the 2012 27th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2012), pp. 461–470. IEEE, Los Alamitos (2012)
- [6] McIver, A., Meinicke, L., Morgan, C.: Hidden-Markov program algebra with iteration. Mathematical Structures in Computer Science **25**(2), 320–360 (2014)
- [7] McIver, A., Morgan, C.: Abstraction, Refinement and Proof for Probabilistic Systems. Monographs in Computer Science. Springer, Berlin (2005)
- [8] McIver, A., Morgan, C., Kaminski, B.L., Katoen, J.P.: A new proof rule for almost-sure termination. Proceedings of the ACM on Programming Languages **2**(POPL), 33:1–33:28 (2017)
- [9] Morgan, C.: *The Shadow Knows*: Refinement of ignorance in sequential programs. In: T. Uustalu (ed.) Proceedings of the International Conference on Mathematics of Program Construction (MPC 2006), *Lecture Notes in Computer Science*, vol. 4014, pp. 359–378. Springer, Berlin (2006)
- [10] Morgan, C.: *The Shadow Knows*: Refinement of ignorance in sequential programs. Science of Computer Programming **74**(8), 629–653 (2009)
- [11] Morgan, C.: Compositional noninterference from first principles. Formal Aspects of Computing **24**(1), 3–26 (2012)
- [12] Morgan, C., McIver, A., Seidel, K.: Probabilistic predicate transformers. ACM Transactions on Programming Languages and Systems **18**(3), 325–353 (1996)
- [13] Smyth, M.B.: Power domains. Journal of Computer and System Sciences **16**(1), 23–36 (1978)



Chapter 17

A demonic lattice of information

The “double \mathbb{D} ”, the distribution-of-distributions construction that makes our hyper-distributions, has its origins in an earlier “double \mathbb{P} ” construction where \mathbb{P} is for powerset — that is, where we doubled over powersets rather than over distributions. It was used for qualitative, that is *possibilistic* security — and thus our focus here on *quantitative* measures can be seen as having taken us from sets of sets to distributions of distributions. But the richer structure that \mathbb{D}^2 allows, together with all the insights from existing research on quantitative security, has fed back in a way that suggests a re-presentation of the earlier \mathbb{P}^2 work — it is now much clearer what was going on, and how it all fits together.

And that is the subject of this chapter, a retrospection. We argue that conceptually there are in fact *three* approaches —of increasing sophistication— to security in this style: there are the deterministic channels of Landauer and Redmond, the demonic (*qualitatively nondeterministic*) channels of this chapter, and then the probabilistic (*quantitatively nondeterministic*) channels of this text as a whole, i.e. for *QIF*.

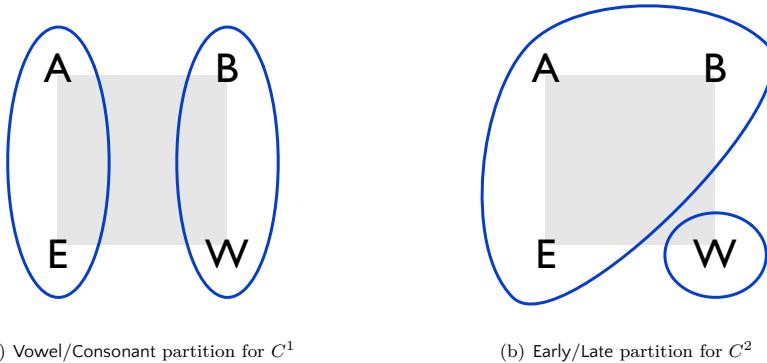
We begin by revisiting the original qualitative, deterministic case of abstract information leakage. (To make this chapter more self-contained, we will repeat some of our earlier definitions.)

17.1 A *deterministic* lattice of information — the original

17.1.1 Historical introduction, intuition and abstraction

The idea of abstracting from leaked values, retaining however the information they convey, is found already in the work of Landauer and Redmond; and it is the very same abstraction —so much later— that is inherent in our hyper-distributions’ “forgetting” the \mathcal{Y} -values an adversary observes, retaining only the “inners” (conditional posterior distributions) that they induce. We now explain the connection.

Recall from the beginning of Chap. 4 that a deterministic channel can be modeled as a total function from some input/hidden type \mathcal{X} to output/observable type \mathcal{Y} . That is, for a deterministic channel an unobservable input x of type \mathcal{X} produces an observable output y of type \mathcal{Y} that (perhaps) reveals something about the value of x , where by “deterministic” we mean that for any x the channel always produces the same observation. Thus we can regard a deterministic channel as a function $C: \mathcal{X} \rightarrow \mathcal{Y}$ and write $y = C(x)$.



Each of these two partitions of $\mathcal{X} = \{\text{A}, \text{B}, \text{E}, \text{W}\}$ has two cells: on the left they are $\{\text{A}, \text{E}\}$ and $\{\text{B}, \text{W}\}$; and on the right they are $\{\text{A}, \text{B}, \text{E}\}$ and $\{\text{W}\}$.

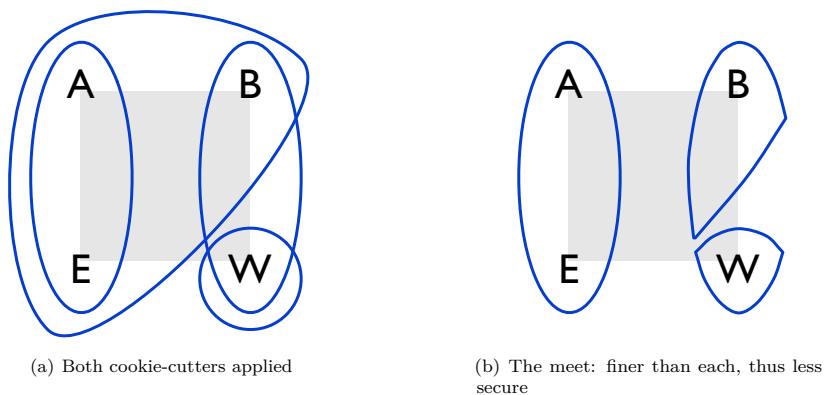
Figure 17.1 Partitions induced on \mathcal{X} by deterministic channels C^1 and C^2

As an example of a deterministic channel, take for the input space \mathcal{X} the letters $\{\text{A}, \text{B}, \text{E}, \text{W}\}$, and for the observables' space \mathcal{Y}^1 the two constants $\{\text{Vowel}, \text{Consonant}\}$. Then we can define a channel $C^1: \mathcal{X} \rightarrow \mathcal{Y}^1$ as a function in the obvious way: for example $C^1(\text{A}) = \text{Vowel}$. And we can take a second channel $C^2: \mathcal{X} \rightarrow \mathcal{Y}^2$ with observables instead in the set $\mathcal{Y}^2 := \{\text{Early}, \text{Late}\}$, indicating position in the alphabet so that $C^2(\text{A}) = \text{Early}$. Those two channels C^1 and C^2 have the same input space, but different output spaces $\mathcal{Y}^{1,2}$ because they are reporting different properties; but in spite of $\mathcal{Y}^{1,2}$'s being different, each channel induces separately its own “posterior” partition on *the same \mathcal{X}* via the kernels of those two functions, that is $C^{1,2}$.¹ We illustrate that in Fig. 17.1, where the partitions' cells determine just which elements of \mathcal{X} can be distinguished by an observer in each case: two input elements can be distinguished just when they are *not* in the same cell. Of course the partitions induced by $C^{1,2}$ are different, and so Fig. 17.1(a) shows that B, W cannot be distinguished by an observer of C^1 's output, because they are both consonants; but Fig. 17.1(b) shows that B, W can be distinguished by C^2 , because B is early but W is late.

In general we write $\mathbb{E}\mathcal{X}$ (equivalence relations on \mathcal{X}) for the set of partitions of \mathcal{X} ; each partition is a set of subsets of \mathcal{X} and each of those subsets is called a *cell*. But clearly $\mathbb{E}\mathcal{X}$ is only a subset of all sets of subsets $\mathbb{P}^2\mathcal{X}$ of \mathcal{X} — a particular set of subsets E is in $\mathbb{E}\mathcal{X}$ just when the elements of (i.e. sets in) E are nonempty, pairwise disjoint and collectively cover \mathcal{X} .

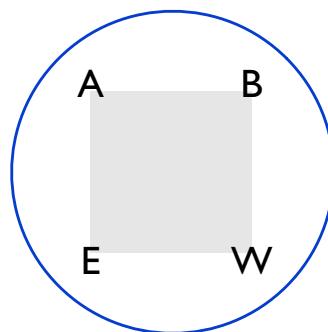
The refinement partial order defined by Landauer and Redmond relates two partitions in $\mathbb{E}\mathcal{X}$ just when one can be formed from the other by merging cells: in this text we will continue to call that relation (\sqsubseteq) . Landauer and Redmond's refinement order is a lattice, because both meet (greatest lower bound) and join (least upper bound) are defined for it; and, with our examples, we can show how that works. The *meet* of two deterministic channels is the most secure (deterministic) channel that reveals at least as much about the input as each channel does on its own: it can be visualized by

¹ Recall that in general the *kernel* of a function f is the equivalence relation (\equiv) induced on its input type by equality on its output: thus $x \equiv x'$ just when $f(x) = f(x')$.



On the right is the meet, a partition with three cells $\{A, E\}$ and $\{B\}$ and $\{W\}$.

Figure 17.2 Induced partitions



The join's partition comprises only the single cell $\{A, B, E, W\}$.

Figure 17.3 The join, coarser (more secure) than each of $C^{1,2}$ separately

thinking of their partitions as “cookie cutters”, with set \mathcal{X} being the “dough” and *both* partitions applied, one on top of the other: then the pieces formed by both cookie-cuts together determine the cells of the meet, as is shown for C^1 and C^2 in Fig. 17.2. Their meet must distinguish A, B because C^1 does, and it must distinguish B, W because C^2 does; but the meet does not distinguish A, E because neither of $C^{1,2}$ does.

Complementarily, the *join* of two deterministic channels is the least secure channel that reveals no more than either does separately, and is shown in Fig. 17.3 for $C^{1,2}$ — in that case the result is the single-column, equivalently the constant-function channel 1 that reveals nothing. Although both C^1 and C^2 distinguish A, W , their join cannot: since C^2 does not distinguish A, B , the join cannot; and the join cannot distinguish B, W because C^1 does not; thus by transitivity the join cannot distinguish A, W either. The same applies to E, W .

A crucial point about both constructions, i.e. meet and join, is that they are *abstract* — they work on the partitions directly, without needing details of $C^{1,2}$ ’s output values — and so we begin to see already how partitions ignore the actual output values in the same way that hyper-distributions do.

17.1.2 Structural definition of refinement for deterministic channels

We now recall further material on deterministic channels from Chap. 4, where they were identified as special cases of probabilistic channels. Although we used separate symbols “ \sqsubseteq_{\circ} ”, and later “ \sqsubseteq_G ” for structural and testing refinement resp., we did go to some trouble in Chap. 9 to show that they are equivalent (given our general assumptions) — and so we will continue here to use the same symbol “ \sqsubseteq ” for both.

Our definition of refinement (\sqsubseteq) for deterministic channels is the same chosen by Landauer and Redmond: that for two channels $S: \mathcal{X} \rightarrow \mathcal{Y}^S$ and $I: \mathcal{X} \rightarrow \mathcal{Y}^I$ we have $S \sqsubseteq I$ just when there is a “refinement function” $R: \mathcal{Y}^S \rightarrow \mathcal{Y}^I$ such that I is R functionally composed with S , that is $I = R \circ S$. That relation (\sqsubseteq) is a partial order: reflexive, transitive and antisymmetric. It is *structural* refinement in the sense of Chap. 9 because it is carried out on the channels’ definitions (here, as functions), and a successful refinement has a witness — the function R .

The intuition for refinement’s definition is that having such an R means that I ’s output would not tell an adversary anything she didn’t already know from the output of S . That is, if for a given x she knew y^S (i.e. $S(x)$), then she would not need to run I to know what y^I would be (i.e. $I(x)$) — since she knows R she can simply calculate it herself. That is, she calculates $y^I = R(y^S)$.

Again reprising our earlier chapters, we know also that an equivalent formulation of deterministic refinement is to see S, I, R as Boolean matrices, with for example $S_{x,y} = \text{true}$ just when $S(x) = y$ as in Fig. 17.4.² Because the channels S, I are deterministic (and total) the corresponding matrices have exactly one *true* in each row, and the induced cells are given by the matrix columns: the *true* entries identify members of the cell corresponding to that column. Similarly because R represents a function, it too has exactly one *true* in each row. With matrices therefore the formulation of refinement is that if channel S is a Boolean $\mathcal{X} \times \mathcal{Y}^S$ matrix and channel I is a Boolean $\mathcal{X} \times \mathcal{Y}^I$ matrix then $S \sqsubseteq I$ just when there is a Boolean $\mathcal{Y}^S \times \mathcal{Y}^I$ matrix R such that $I = SR$.³

Fig. 17.5 shows how the meet of $C^{1,2}$ in Fig. 17.2(b) is refined in that style, i.e. by a matrix R^1 to the more secure C^1 . (Remember that a meet is less secure, and

² Recall that for matrix M indexed by r, c we write $M_{r,c}$ for the value in row r and column c . Note that in this chapter we will not be writing matrices in sans-serif as we do elsewhere; that is to avoid confusion with the A, B, E, W of our running example.

³ As usual we write SR for the matrix multiplication of S by R .

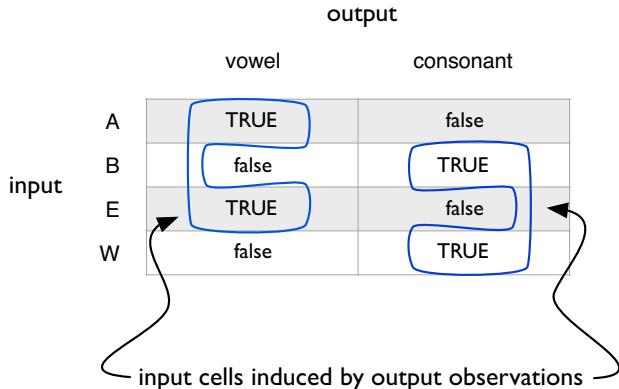


Figure 17.4 Matrix representation of a deterministic channel

$$\begin{array}{c}
 \begin{array}{ccc}
 0 & 1 & 2 \\
 \hline
 A & \text{TRUE} & \text{false} & \text{false} \\
 B & \text{false} & \text{TRUE} & \text{false} \\
 E & \text{TRUE} & \text{false} & \text{false} \\
 W & \text{false} & \text{false} & \text{TRUE}
 \end{array}
 \times
 \begin{array}{cc}
 \text{vowel} & \text{consonant} \\
 \hline
 0 & \text{TRUE} & \text{false} \\
 1 & \text{false} & \text{TRUE} \\
 2 & \text{false} & \text{TRUE}
 \end{array}
 =
 \begin{array}{cc}
 \text{vowel} & \text{consonant} \\
 \hline
 A & \text{TRUE} & \text{false} \\
 B & \text{false} & \text{TRUE} \\
 E & \text{TRUE} & \text{false} \\
 W & \text{false} & \text{TRUE}
 \end{array}
 \end{array}$$

$$C^1 \sqcap C^2 \quad R^1 \quad C^1$$

The column labels 0,1,2 for the matrix representing the meet $C^1 \sqcap C^2$ are chosen arbitrarily. (The (\times) here is matrix multiplication.)

 Figure 17.5 Refinement of $C^1 \sqcap C^2$ to the more secure C^1

a join is more secure.) Notice that the column-labels 0, 1, 2 of the meet (and the rows of the refinement matrix, in the middle) have no external significance, again supporting our abstract view: it is the partition of the input cells, alone, that indicates the information flow induced by a channel. Fig. 17.6 uses a different matrix R^2 to refine the same meet to C^2 instead.

17.1.3 Testing, soundness and completeness: deterministic

In §17.1.2 just above, we saw that the refinement function, or matrix R , is a witness to the refinement $S \sqsubseteq I$, which shows not only that the partition cells induced by S can be merged to form the cells of I , but actually how to do it; in fact the existence of such an R is (structural) refinement's definition. In principle that gives a method for the systematic development of implementations I from specifications S , a “security by design” approach where suitable matrices R guide the programmer's efforts.

$$\begin{array}{c}
 \begin{array}{ccccc}
 & & 0 & 1 & 2 \\
 \begin{array}{c} A \\ B \\ E \\ W \end{array} & \times & \begin{array}{ccccc}
 & & \text{early} & \text{late} & \\
 \begin{array}{c} 0 \\ 1 \\ 2 \end{array} & \times & \begin{array}{cc}
 \begin{array}{cc} \text{TRUE} & \text{false} \\ \text{true} & \text{false} \end{array} & \begin{array}{cc} \text{false} & \text{true} \\ \text{false} & \text{true} \end{array} \\
 \end{array} & = & \begin{array}{ccccc}
 & & \text{early} & \text{late} & \\
 \begin{array}{c} A \\ B \\ E \\ W \end{array} & \times & \begin{array}{cc}
 \begin{array}{cc} \text{TRUE} & \text{false} \\ \text{true} & \text{false} \end{array} & \begin{array}{cc} \text{false} & \text{true} \\ \text{false} & \text{true} \end{array} \\
 \end{array} & C^2 &
 \end{array}
 \end{array}$$

Figure 17.6 Refinement of $C^1 \sqcap C^2$ to the more secure C^2

The complementary problem however is how to convince a court that $S \not\sqsubseteq K$, i.e. that when we bought S , but were delivered K instead, we were cheated. (Recall §9.8.3.)⁴ But it's not practical to go through all the potential R matrices, hoping to show for each and every one that $K \neq SR$. Just as R provides a witness for (\sqsubseteq) , we need a witness for $(\not\sqsubseteq)$ too. That leads us again (recall §9.3) to *testing* refinement.

In this deterministic setting, a test that provides a witness for non-refinement $(\not\sqsubseteq)$ is a subset χ of \mathcal{X} that we regard as an attack: the adversary is trying to determine whether the secret x is an element of χ . Equivalently we could regard χ as a property, so that the adversary is therefore trying to discover whether the secret x has property χ . If that attack can succeed against K but cannot succeed against S – that is if K could ever reveal to her that x has property χ , but S could never do that – then indeed test χ is a witness for $S \not\sqsubseteq K$. For the test possibly to succeed against K there must be a cell κ of K (which K , remember, induces a partition of \mathcal{X}) that is a subset of χ , that is $\kappa \subseteq \chi$ — for if the adversary ever sees an observation from which she can deduce that $x \in \kappa$ she will deduce that therefore $x \in \chi$ as well. Correspondingly, for test χ never to succeed against S , there must be no cell σ of S with $\sigma \subseteq \chi$.

Those two complementary witnesses, matrix R for (\sqsubseteq) and subset χ of \mathcal{X} for $(\not\sqsubseteq)$, are related by “soundness” and “completeness”, as we saw before in Chap. 9. *Soundness* says that whenever $S \sqsubseteq I$, i.e. there exists a suitable structural-witness R for refinement, then there cannot be any refuting test-witness χ that (inconsistently with $S \sqsubseteq I$) would establish $S \not\sqsubseteq I$ instead. In intuitive terms, it says that if a software engineer follows sound practices then he will never lose a court case. *Completeness* says that whenever $S \not\sqsubseteq K$, i.e. there is no structural-refinement witness R , then there exists a refuting test-witness χ for that; we do not have to try (and reject) every single refinement matrix R . Thus for deciding whether $C^1 \sqsubseteq C^2$, we always have either a refinement-certifying R or a refinement-refuting χ , but never both.

17.2 Our *probabilistic* partial order of information – the most recent, but not a lattice

The probabilistic analog of the deterministic case, just above, is communication channels with probabilistic transmission, as we have explained earlier at length, particularly in Chap. 4. In communication channels the input is a message to be sent and the output is the message received; and the channel is analyzed wrt. a known

⁴ The mnemonics are S for Specification and I for an Implementation that is supposed to refine S , and K for a “Kludge” that, as an example, in fact does not refine S . In uncommitted cases, neither I nor K , we will use P for Program.

distribution of input messages in the style of Shannon. The traditional representation of such channels is stochastic matrices, where real numbers in each row x give for each column y the probability that input message x will result in output message y . Deterministic channels §17.1 are of course special cases of probabilistic channels, where *true* is probability 1 and *false* is probability 0.

As explained earlier in §9.2.2, the probabilistic analog of refinement $S \sqsubseteq I$ can now be seen as a generalization of §17.1.2: the refinement matrix R can be stochastic (instead of deterministic), representing a “probabilistic merge” of S -outputs to make I -outputs mediated by R such that again $I = SR$. That relation is reflexive and transitive (obviously). But this time it is not antisymmetric: for that either we quotient to *reduced channels* (Def. 4.9 on p.59) —where all-zero columns are removed, similar columns are merged and the order of (the remaining) columns is ignored— or we pass to hyper-distributions.⁵

Indeed probabilistic refinement shares many structural properties with deterministic refinement. In particular, as already remarked, there are probabilistic analogs of soundness and completeness (§§ 9.4, 9.5), with tests based on gain/loss functions over \mathcal{X} . They were the subject of Chap. 3, and are considerably more general than the subsets χ of \mathcal{X} that sufficed for deterministic channels in §17.1.⁶

Although (\sqsubseteq) for probabilistic channels appears not to be a lattice (§12.4), it seems possible that channels allowing *both* probabilistic and demonic nondeterminism could be — see the Chapter Notes.

17.3 Basic structure of the demonic lattice

With §17.1 and §17.2 as motivating examples, we now turn to the main topic of this chapter. The *demonic* lattice of information is more expressive than deterministic but more abstract than probabilistic. Here observations are not necessarily wholly determined by the inputs, but still we have no quantitative information about possible variation.

Our approach here is based on the earlier “Shadow Model” for demonic noninterference; the probabilistic model of §17.2 was not known (we believe) at that time. Yet the similarities shared by all three models are striking.

We begin by defining a demonic channel.

Definition 17.1 (Demonic channel: matrix formulation) A demonic channel from \mathcal{X} to \mathcal{Y} is a Boolean matrix with \mathcal{X} -indexed rows and \mathcal{Y} -indexed columns in which each row has at least one *true* element. \square

Whereas deterministic channels induce partitions on their input-space \mathcal{X} , as we saw in §17.1, demonic channels induce more generally simply sets of subsets of \mathcal{X} , i.e. like partitions but allowing the cells to overlap. We continue to call them “cells” (in spite of their potential overlaps); and they are the qualitative analog (in fact the precursors) of hyper-distributions’ inners.

The overlaps occur just for those x -rows containing more than one *true*: those shared x ’s “belong” more to one y -column, i.e. to more than one cell. From here on,

⁵ The identity matrix is stochastic, and the product of two stochastic matrices is stochastic. Matrix columns are *similar* just when each is a multiple of the other. Column order can then be ignored by representing the matrix as a set of (the remaining) columns.

⁶ The completeness property was called “Coriaceous” (leathery, i.e. tough) by Palamidessi. It was proved (independently) by McIver and, it turns out, earlier by Blackwell. More recently, it has been studied as “majorization” by Dahl.

in this chapter, we use $\mathcal{X} \rightarrow \mathcal{Y}$ for the type of a demonic channel matrix from \mathcal{X} to \mathcal{Y} (rather than probabilistic, as elsewhere).

We can define a refinement relation between demonic channels as matrices. Remarkably, structural refinement is again post-multiplication by a matrix of the same kind — just as for the other two cases. That matrix R is our witness for refinement.

Definition 17.2 (Refinement for demonic channels: matrix formulation)

A demonic “specification” channel $S: \mathcal{X} \rightarrow \mathcal{Y}^S$ is refined by an “implementation” channel $I: \mathcal{X} \rightarrow \mathcal{Y}^I$ just when there is a demonic matrix $R: \mathcal{Y}^S \rightarrow \mathcal{Y}^I$ such that $I = SR$ where R is also a demonic channel. As usual, we write $S \sqsubseteq I$ for that. \square

Each column of the refinement matrix R creates a cell in the implementation I by taking the union of the specification cells in S that have *true* in that column. With that insight, we can formulate Def. 17.2 more abstractly as follows.

Definition 17.3 (Refinement for demonic channels: cells formulation)

For two subsets S, I of $\mathbb{P}\mathcal{X}$ (i.e. two sets of subsets of \mathcal{X}) we have that S is refined by I iff for every cell of I there is a set of cells of S of which it is the union. Put another way, every cell of the more secure I must be “justified” as being the union of some set of cells in the possibly less secure S .⁷ \square

Demonic refinement (in either formulation) is reflexive and transitive but, as we observe in the example below, and as in the probabilistic case §17.2, the relation is not antisymmetric: so far, we have only a pre-order. An example of antisymmetry’s failure is the two demonic channels $P^{1,2}$ defined abstractly as the sets of cells

$$\begin{array}{ll} P^1 &= \{ \{x_0\}, \{x_1\}, \{x_0, x_1\}, \{x_0, x_1, x_2\} \} \\ \text{and} \quad P^2 &= \{ \{x_0\}, \{x_1\}, \{x_0, x_1, x_2\} \} , \end{array}$$

so that each cell in one of them is a union of cells in the other — but they are not equal. Using 1 for *true* and 0 for *false*, with matrices $P^1 \sqsubseteq P^2$ we have the witness R and the matrix multiplication

$$\begin{array}{c} P^1 \qquad \qquad \qquad R \qquad \qquad \qquad P^2 \\ \hline x_0 \quad \left(\begin{array}{cccc} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{array} \right) \quad \left(\begin{array}{cccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{array} \right) \quad = \quad \left(\begin{array}{ccc} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{array} \right) \quad , \\ x_1 \\ x_2 \end{array}$$

where for example the third column of R shows that P^1 ’s cells $\{x_0, x_1\}$ (its third column) and $\{x_0, x_1, x_2\}$ (fourth column) are merged into a single cell $\{x_0, x_1, x_2\}$ in P^2 , and so $\{x_0, x_1\}$ is effectively discarded. For the other direction $P^2 \sqsubseteq P^1$ we have the witness R' and the multiplication

$$\begin{array}{c} P^2 \qquad \qquad \qquad R' \qquad \qquad \qquad P^1 \\ \hline x_0 \quad \left(\begin{array}{ccc} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{array} \right) \quad \left(\begin{array}{cccc} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right) \quad = \quad \left(\begin{array}{cccc} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{array} \right) \quad , \\ x_1 \\ x_2 \end{array}$$

⁷ This is like the Smyth powerdomain construction that promotes an underlying partial order (\sqsubseteq_X) on some X to a partial order $(\sqsubseteq_{\mathbb{P}X})$ on $\mathbb{P}X$, where each element in a $(\sqsubseteq_{\mathbb{P}X})$ -more-refined subset of X must be (\sqsubseteq_X) -above some element in the less refined subset. A difference however is that in this case *several* elements in the less-refined subset might be required.

where the third column of R' restores $\{x_0, x_1\}$ in P^1 as the union of cells $\{x_0\}$ and $\{x_1\}$ from P^2 .

We can however achieve antisymmetry of (\sqsubseteq) via the usual closure construction.

Definition 17.4 (Union closure) Say that a set of cells, i.e. a subset of $\mathbb{P}\mathcal{X}$, is *union closed* just when the union of each of its subsets (which are sets of sets themselves) is also an element (a set) of it. Define the *union closure* P^\cup of some subset P of $\mathbb{P}\mathcal{X}$ to be the smallest union-closed subset of $\mathbb{P}\mathcal{X}$ that contains P ; this is well defined because $\mathbb{P}\mathcal{X}$ is union closed and any intersection of union-closed sets is again union closed. \square

Definition 17.5 (Demonic-refinement domain for information hiding)

Let $\mathbb{U}\mathcal{X}$ be the set of subsets of $\mathbb{P}\mathcal{X}$ that are union closed and that cover \mathcal{X} : it will be our abstract model for demonic information hiding. \square

Note that all elements of $\mathbb{U}\mathcal{X}$, that is union-closed subsets of $\mathbb{P}\mathcal{X}$, contain \emptyset and so are nonempty.⁸ We now observe that on union-closed sets, refinement (\sqsubseteq) is simply (\supseteq) , as is common for up-closure constructions.

Lemma 17.6 (Refinement (\sqsubseteq) over $\mathbb{U}\mathcal{X}$ is (\supseteq) over $\mathbb{P}^2\mathcal{X}$) For any two elements S, I of $\mathbb{U}\mathcal{X}$ we have $S \sqsubseteq I$ iff $S \supseteq I$ when they are considered as subsets of $\mathbb{P}\mathcal{X}$.

Proof. Recall that if S, I are elements of $\mathbb{U}\mathcal{X}$ then they are both subsets of $\mathbb{P}\mathcal{X}$, and thus their own elements, i.e. their cells, are themselves subsets of \mathcal{X} .

First, if $S \supseteq I$ then any i in I is in S also, and is therefore the union of (the singleton) subset $\{i\}$ of S ; thus $S \sqsubseteq I$. And second, if $S \sqsubseteq I$ then any cell in I must be the union of some cells in S ; but since S is union closed that union of cells must itself be in S ; thus $I \subseteq S$. \square

Lemma 17.7 (Refinement (\sqsubseteq) is a partial order on $\mathbb{U}\mathcal{X}$) Refinement (\sqsubseteq) as in Def. 17.2 is a partial order on $\mathbb{U}\mathcal{X}$.

Proof. Immediate from Lem. 17.6 and (\supseteq) 's being a partial order on $\mathbb{P}^2\mathcal{X}$. \square

More than being a partial order, refinement on $\mathbb{U}\mathcal{X}$ is in fact a lattice. That cannot be inherited directly from (\supseteq) 's being a lattice on $\mathbb{P}^2\mathcal{X}$; but it is nevertheless straightforward.

Lemma 17.8 (Demonic refinement (\sqsubseteq) over $\mathbb{U}\mathcal{X}$ is a lattice) For $P^{1,2}$ in $\mathbb{U}\mathcal{X}$ and (\sqsubseteq) as in Def. 17.3, both $P^1 \sqcup P^2$ and $P^1 \sqcap P^2$ are well defined.

Proof. We prove first that $P^1 \sqcup P^2$ exists for any $P^{1,2}$ in $\mathbb{U}\mathcal{X}$: it is in fact $P^1 \cap P^2$, because (\sqsubseteq) is (\supseteq) , and (\supseteq) is a lattice on $\mathbb{P}^2\mathcal{X}$, and $P^1 \cap P^2$ is union closed if $P^{1,2}$ are.

For $P^1 \sqcap P^2$ however we do not have that $P^1 \cup P^2$ is union closed if $P^{1,2}$ are; therefore we define $P^1 \sqcap P^2$ to be $(P^1 \cup P^2)^\cup$.

Clearly for any P in $\mathbb{U}\mathcal{X}$ if $P \sqsubseteq P^1 \sqcap P^2$ then $P \sqsubseteq P^{1,2}$; and if $P \sqsubseteq P^{1,2}$ then by definition $P \supseteq P^{1,2}$ and so $P \supseteq P^1 \cup P^2$, whence $P \supseteq (P^1 \cup P^2)^\cup = P^1 \sqcap P^2$ because P is union closed. \square

In the next section we give some examples of properly demonic channels.

⁸ The subset \emptyset of \mathcal{X} is the union of the empty subset of subsets of $\mathbb{P}\mathcal{X}$.

17.4 Examples of demonically nondeterministic channels: the Alphabet Spies in action

Recall the deterministic channels $C^{1,2}$ from §17.1, the “Alphabet Spies”. We can see that the union closure of C^1 from Fig. 17.1(a) is $\{\emptyset, AE, BW, ABEW\}$, where we write AE for $\{A, E\}$ etc. The union closure of C^2 is $\{\emptyset, W, ABE, ABEW\}$. Therefore from Lem. 17.8 their join $C^1 \sqcup C^2$ is their intersection $\{\emptyset, ABEW\}$ as in Fig. 17.3, and their meet $C^1 \sqcap C^2$ is

$$\{\emptyset, W, AE, BW, ABE, \underline{AEW}, \underline{ABEW}\} , \quad (17.1)$$

where the underlined \emptyset and \underline{AEW} have been added by union closure (as explained in Lem. 17.8). We note however that (17.1) differs (surprisingly?) from the meet Fig. 17.2(b) we gave in the deterministic lattice: when union closed, it would be $\{B, W, AE\}^\cup$, that is

$$\{\emptyset, B, W, AE, \underline{BW}, \underline{ABE}, \underline{AEW}, \underline{ABEW}\} . \quad (17.2)$$

In fact in $\mathbb{U}\mathcal{X}$ we have the strict refinement (17.2) \sqsubset (17.1) by discarding $\{B\}$ from the left-hand side.

That is indeed an interesting phenomenon: when restricted to deterministic channels, we were obliged to take a less secure meet of $C^{1,2}$ because at that point we could not express the nuance that demonic behavior offers. But now we can — in the demonic case $\mathbb{U}\mathcal{X}$ there is a more refined, that is a more *secure* meet (17.1) than the (17.2) admitted by the deterministic case, i.e. in Landauer and Redmond’s lattice of information $\mathbb{E}\mathcal{X}$; and that is because (17.1) takes into account behaviors that no *deterministic* channel could realize, as we now discuss.

Suppose that $C^{1,2}$ are *real* spies, with real names *Ms. Vowel* and *Mrs. Early*, and our adversary M will use them to try to discover the value of our hidden letter x . However she is only allowed to send one spy, not both. From our point of view (as defenders) that choice is demonic: although we know she will send only one spy, we don’t know beforehand which it will be and so we must assume the worst. How do we describe that situation in $\mathbb{U}\mathcal{X}$?

In $\mathbb{U}\mathcal{X}$ that mission is in fact $C^1 \sqcap C^2$, as in (17.1) and, as we remarked above, it is a proper refinement of the deterministic (17.2) where both spies are sent. After all, whichever spy she sends, from our point of view it can’t be worse than her sending both. The following lemma shows that (17.1) cannot be deterministic, and that is why —because (17.2) is deterministic— the refinement is strict.

Lemma 17.9 (Characterization of determinism within $\mathbb{U}\mathcal{X}$) For input space \mathcal{X} , the (union closures of the) *deterministic* subset $\mathbb{E}\mathcal{X}$ of its demonic channels $\mathbb{U}\mathcal{X}$ comprise exactly those channels that are *also* complement closed. That is, any U in $\mathbb{U}\mathcal{X}$ is in fact E^\cup for some E in $\mathbb{E}\mathcal{X}$ just when U is not only union closed but also complement closed (and therefore intersection closed as well).

Proof. “Only if” is easy, because any element of E^\cup is a union of some of E ’s cells, and its complement is thus the union of E ’s remaining cells.

For “if” we note that the minimal nonempty elements of a complement- and union-closed U are pairwise disjoint, because U is also intersection closed — and so if two such cells overlapped, they would not be minimal. Those are the E such that $E^\cup = U$. \square

Lem. 17.9 shows that (17.1) cannot be deterministic, because it can reveal BW in the case that Ms. Vowel returns and says **Consonant**; and it can also reveal ABE if

Mrs. Early returns and says **Early** — and as defenders we must accept that either could happen, because the choice is demonic from our point of view. But it cannot reveal **B**, that is the intersection $\text{BW} \cap \text{ABE}$, and so it is not intersection closed.

A more intuitive explanation of the strictness $(17.2) \sqsubset (17.1)$ is as follows, concentrating on the case where the input x is **B**. Any meet of C^1 and C^2 , deterministic or demonic, must make all distinctions that either of $C^{1,2}$ separately could make. But if it's deterministic, it must return a fixed, agreed-beforehand value for each x . Now the value it returns for **B** must be different from the value it returns for **A** and for **E**, because otherwise it is not making a distinction that C^1 could make; and the value it returns for **B** must *also* be different from the value it returns for **W**, because otherwise it is not making a distinction that C^2 could make. Thus the value it returns for **B** must be different from *all* other values — and so if x is in fact **B**, the deterministic meet will reveal it.

If however the meet is demonic, its ability to reveal whatever C^1 or C^2 might reveal is due to its ability to behave like either C^1 or C^2 , as M desires. But no matter what she desires, the meet cannot reveal for sure that x is **B** — for if the meet behaves as C^1 , it will confuse **B** with **W**; and if it behaves as C^2 it will confuse **B** with **A** and **E**. In neither case does M learn for sure that x is **B**, as she would have in the deterministic-meet case.

Thus properly demonic nondeterminism really does make a difference.

Now we consider an intriguingly different scenario, where M sends *both* spies, and the spies report by radio instead of in person — they use Booleans agreed beforehand (i.e. a one-time pad), so that for Ms. Vowel “*true*” encodes **Vowel** and for Mrs. Early it encodes **Early**.⁹ On this even more dangerous mission M knows that both spies will be captured, but she knows also that —in spite of their capture— exactly one will manage to send a report back to her by radio, either *true* or *false*. But she won't know which spy it was. Here the demonic channel is

$$\{\emptyset, \text{BW}, \text{ABE}, \underline{\text{ABEW}}\} \quad (\text{See Exercise } 17.4.) \quad (17.3)$$

which, by Lem. 17.9 again, is also properly demonic (because there's no **B**).

Putting all those scenarios together, we see that $(17.2) \supset (17.1) \supset (17.3)$, that is

$$(17.2) \quad \sqsubset \quad (17.1) \quad \sqsubset \quad (17.3) \quad ,$$

which illustrates nicely the difference between deterministic, internally demonic and externally demonic. (Recall that we have seen internal and external *probabilistic* choice in several places already: in §§8.1.4, 8.1.5 it was discussed as it applied to probabilistic channels; and in §13.5 it was discussed for PRINT-statements in probabilistic programs. Here we are seeing a conceptually earlier manifestation of the same distinction.)

Here is a summary of what we have discovered:

In the deterministic-meet case (17.2) spymistress M deploys both spies Vowel and Early, and they both return. She combines their reports to draw her conclusion about the hidden x , and that is the worst situation from the defender's point of view. In the deterministic case, meet is equivalent to parallel composition (Def. 17.17 below).

⁹ Compare the intuitive explanation of internal choice given by Boris and Natasha's use of a typewriter in §4.6.1 and Warner's protocol in §8.1.5.

In the external-demonic-choice case, the demonic meet (17.1) spymistress M can control which spy will report, and so will know which one it was; but she can access the report only of the one she chose.¹⁰ (Later, in Def. 17.17 we will see the case where M can access both reports — that is demonic-channel parallel composition.)

In the internal-demonic-choice case (17.3) spymistress M gets a report from Vowel or Early, but she cannot control which one it will be and, worse from her point of view, she doesn't know even once she's received the report which spy it was that sent it. Note that this case *cannot* be the demonic meet, because (17.3) is not refined by $C^{1,2}$. (See Exercise 17.4.)

17.5 Testing, soundness and completeness: demonic

In this section –because we are concerned with soundness and completeness– we will use the symbol (\sqsubseteq_\circ) for structural refinement (rather than simply (\sqsubseteq) on its own); for testing refinement we continue to use (\sqsubseteq) . As a result we can distinguish the two.¹¹

The methodological concerns of §17.1.3, that is testing, soundness and completeness, apply to demonic channels too: if we suspect that $S \not\sqsubseteq_\circ K$, how can we prove the refinement's failure in court?

Our earlier technique, for testing deterministic channels, does not work for demonic channels. Let S be $\{ \emptyset, \{x_0, x_1\}, \{x_2\}, \{x_0, x_1, x_2\} \}$ and K , not a refinement, be $\{ \emptyset, \{x_0, x_1\}, \{x_1, x_2\}, \{x_0, x_1, x_2\} \}$, where we know that $S \not\sqsubseteq_\circ K$ because $\{x_1, x_2\}$ in K is not the union of any cells in S . But no deterministic test χ in the style of §17.1.3 shows $S \not\sqsubseteq_\circ K$, because every nonempty cell of K is a superset of some nonempty cell of S — that is, any test χ for which some nonempty cell κ of K satisfies $\kappa \subseteq \chi$ implies there is a nonempty cell σ of S with $\sigma \subseteq \chi$ also, because there is a cell with $\sigma \subseteq \kappa$. Thus deterministic tests are too weak, not discriminating enough: it's an example of their incompleteness for demonic channels.¹² But it does seem strange that every cell of K 's being a superset of some cell of S , in a sense more demonic, is still *not* sufficient for refinement. We return to that issue in §17.8. In the meantime, we continue by synthesizing a test suite for demonic channels for which structural refinement is sound and complete.

By definition we have $S \not\sqsubseteq_\circ K$ just when there is some cell κ in K that is not the union of any set of cells σ_1, \dots, N drawn from S — which, in turn, is just when there is some single element x of \mathcal{X} such that every x -containing cell σ of S is *not* a subset of κ . That is we have $S \not\sqsubseteq_\circ K$ just when

$$\begin{aligned} &\text{there is an element } x \text{ of } \mathcal{X} \text{ and a cell } \kappa \text{ of } K, \text{ with } x \in \kappa \in K, \\ &\text{such that for every cell } \sigma \text{ in } S \text{ we have } x \in \sigma \Rightarrow \sigma \not\subseteq \kappa. \end{aligned} \quad (17.4)$$

Our preliminary definition of the “suite” of demonic tests is therefore that they are pairs (x, χ) with $x \in \chi \subseteq \mathcal{X}$ — and those are our (preliminary) witnesses for non-refinement in the demonic case.

¹⁰ An intriguing difference between external demonic and external probabilistic choice is that in the former case the choice is controlled by M , but in the latter case it is not — because probabilistic choice is not controlled by any agent, whether attacker or defender. In the probabilistic case, the external/internal issue is only whether the outcome of the choice is known.

¹¹ We use (\sqsubseteq_G) for testing refinement elsewhere in the text, and that is because those tests are done with gain- or loss functions. Here however they are not.

¹² They are trivially sound, however, since weakening a test suite trivially preserves its soundness: with fewer tests, there will be fewer failures.

A demonic channel P passes such a test just when every cell π in P with $x \in \pi$ satisfies $\pi \not\subseteq \chi$, i.e. fails the test (x, χ) just if there is a cell π in P with $x \in \pi \subseteq \chi$, and we define (demonic) testing refinement $S \sqsubseteq I$ to hold just when I passes every such test that S does. (This testing refinement is the demonic analogy of $(\sqsubseteq_{\mathbb{G}})$ for probabilistic channels; but we will not bother to invent another symbol for it.)¹³

For *soundness* of structural refinement (\sqsubseteq_{\circ}) wrt. testing refinement (\sqsubseteq) , we argue the contrapositive by assuming that we have $S \sqsubseteq_{\circ} I$ and a test (x, χ) that I fails, so that there is some cell ι in I with $x \in \iota \subseteq \chi$. But because $S \sqsubseteq_{\circ} I$ we know that $\iota = \cup_n \sigma_n$ for some $\sigma_1, \dots, \sigma_N$, and so $x \in (\cup_n \sigma_n) \subseteq \chi$ whence, for some n , we have $x \in \sigma_n \subseteq \chi$ with $\sigma_n \in S$. That is, there is a cell σ_n of S that fails the test, and so S fails as a whole: that is, if I fails (x, χ) then so does S — just as required (contrapositively) for the refinement $S \sqsubseteq I$.

For *completeness*, suppose $S \not\sqsubseteq_{\circ} K$. We choose (x, κ) as in (17.4) to be our test, which K fails trivially, because $\kappa \not\subseteq \kappa$ does not hold, but S passes. For example the test that shows

$$\not\sqsubseteq \quad \begin{array}{l} \{ \emptyset, \{x_0, x_1\}, \{x_2\}, \{x_0, x_1, x_2\} \} \\ \sqsubseteq \quad \{ \emptyset, \{x_0, x_1\}, \{x_1, x_2\}, \{x_0, x_1, x_2\} \} \end{array} , \quad (17.5)$$

the example from above, is $(x_1, \{x_1, x_2\})$ — the cells σ on the left that satisfy $x_1 \in \sigma$ are $\{x_0, x_1\}$ and $\{x_0, x_1, x_2\}$ and, for both, we have $\sigma \not\subseteq \{x_1, x_2\}$. The cell $\{x_1, x_2\}$ on the right however satisfies $x_1 \in \{x_1, x_2\}$ but does not of course satisfy $\{x_1, x_2\} \not\subseteq \{x_1, x_2\}$.

From the above, we have again the equivalence of structural and testing refinement.

Theorem 17.10 (Equivalence of structural and testing refinement for demonic channels) For any two demonic channels $P^{1,2}$ we have $P^1 \sqsubseteq_{\circ} P^2$ just when $P^1 \sqsubseteq P^2$. \square

As with probabilistic channels, in light of the equivalence we will from now on write just “ \sqsubseteq ” for demonic refinement, whether structural or testing, unless the distinction is important.

17.6 A reformulation of demonic testing

For our preferred definition of demonic testing we will reformulate (17.4) above in terms of two subsets of \mathcal{X} , rather than an element x and a subset χ , because that turns out to be more suitable for source-level reasoning over programs.¹⁴ Lem. 17.12 will show the reformulation to be equivalent to the definition in §17.5.

Definition 17.11 (Tests for demonic refinement) A test for demonic refinement over space \mathcal{X} is a pair (α, β) of subsets of \mathcal{X} . As usual, a demonic channel P passes the test (α, β) just when all its cells pass the test; and a cell π of P passes the test just when $\pi \subseteq \alpha \Rightarrow \pi \subseteq \beta$. \square

¹³ In fact $x \in \chi$ is not necessary, since a pair (x, χ) with $x \notin \chi$ would be a test passed by every channel, and thus of no practical use.

¹⁴ Subsets of \mathcal{X} , rather than individual elements, are more easily turned into predicates for source-level reasoning over a state-space of typed variables: if you add another variable, a subset remains a subset but a point is no longer a point.

Those (α, β) pairs will be our preferred witnesses for non-refinement (replacing (x, χ) from earlier). The top of the $\mathbb{U}\mathcal{X}$ lattice is the reveal-nothing channel $\{\emptyset, \mathcal{X}\}$, that is $\mathbb{1}$, and it passes every test; the bottom of the lattice is the reveal-everything channel $\mathbb{P}\mathcal{X}$, that is $\mathbb{0}$, which fails all non-trivial tests. ¹⁵

Lemma 17.12 (Equivalence of testing suites) The test suite of Def. 17.11 is equivalent in power to the preliminary test suite (x, χ) discussed at (17.4) further above.

Proof. We show that $S \sqsubseteq K$ can be refuted by an (α, β) -test just when it can be refuted by an (x, χ) -test.

if — Any (x, χ) -test can be expressed as a single (α, β) -test by setting $\alpha := \chi$ and $\beta := (\mathcal{X} - \{x\})$. To see that, let π be an arbitrary cell and reason

$$\begin{array}{lll} x \in \pi \Rightarrow \pi \not\subseteq \chi & & \text{"}\pi \text{ passes } (x, \chi)\text{"} \\ \text{iff} & \pi \not\subseteq (\mathcal{X} - \{x\}) \Rightarrow \pi \not\subseteq \chi & \text{"rewrite lhs"} \\ \text{iff} & \pi \subseteq \chi \Rightarrow \pi \subseteq (\mathcal{X} - \{x\}) & \text{"contrapositive"} \\ \text{iff} & \pi \subseteq \alpha \Rightarrow \pi \subseteq \beta . & \text{"}\pi \text{ passes } (\alpha, \beta) \text{ with } \alpha, \beta := \chi, (\mathcal{X} - \{x\})\text{"} \end{array}$$

Thus (α, β) -tests are at least as discriminating as (x, χ) -tests — that is, any refinement that can be refuted by some test (x, χ) can be refuted as well by an (α, β) test by setting $\alpha, \beta := \chi, (\mathcal{X} - \{x\})$.

only if — The other direction is more involved, because we cannot find a *single* (x, χ) -test equivalent to any given (α, β) test. Yet we can show that if some K fails but S passes a given (α, β) -test then there must also be an (x, χ) -test that K fails but S passes — it is just that the choice of x and χ might depend on K , as well as on α, β of course. That is, (x, χ) -tests are as discriminating as (α, β) -tests, but which x, χ we use might depend on what cell we are testing.

Suppose therefore that $S \sqsubseteq K$ is refuted by (α, β) . Then for all cells σ in S we have $\sigma \subseteq \alpha \Rightarrow \sigma \subseteq \beta$, but for some cell κ in K we have $\kappa \subseteq \alpha \wedge \kappa \not\subseteq \beta$. Now reason

$$\begin{array}{lll} \kappa \subseteq \alpha \wedge \kappa \not\subseteq \beta & & \\ \text{iff} & \kappa \subseteq \alpha \wedge x \in \kappa & \text{"for some } x \notin \beta\text{"} \\ \text{hence} & \kappa \text{ fails test } (x, \alpha) . & \end{array}$$

That (x, α) will be the (simpler) test that K fails but S passes. Just above we saw that K fails it; we now show that S passes it, as required to reject $S \sqsubseteq K$.

For a contradiction, suppose that S fails that test; then $x \in \sigma \wedge \sigma \subseteq \alpha$ for some $\sigma \in S$, by definition of (x, χ) tests. Continue reasoning

$$\begin{array}{lll} \text{hence} & x \in \sigma \wedge \sigma \subseteq \beta & \text{"assumption above that all }\sigma\text{'s in }S\text{ pass test }(\alpha, \beta)\text{"} \\ \text{hence} & x \in \beta , & \end{array}$$

which contradicts the choice $x \notin \beta$ above. Thus no cell in S can fail test (x, α) , and so test (x, α) rejects $S \sqsubseteq K$, as required, given that (α, β) did. \square

¹⁵ Non-trivial tests are those that make at least one distinction. Tests (α, β) are trivial when $\alpha \subseteq \beta$ (passed by every cell), and when α, β are disjoint (passed only by cell \emptyset). In general (α, β) is equivalent to $(\alpha, \alpha \cap \beta)$. Also for example (α', β') is weaker than, i.e. rejects fewer cells than (α, β) when $\alpha' \subseteq \alpha$ and $\beta \subseteq \beta'$. Compare Footnote 22 on p. 344 below.

17.7 Reduced demonic channels

Although $\mathbb{U}\mathcal{X}$ is restricted to union closed subsets of \mathcal{X} , we can give a “reduced” representation of demonic channels in which union closure is taken implicitly. In reduced form the non-refinement example from (17.5) becomes

$$\{\{x_0, x_1\}, \{x_2\}\} \not\sqsubseteq \{\{x_0, x_1\}, \{x_1, x_2\}\} ,$$

and the (α, β) -test for that non-refinement is $(\{x_1, x_2\}, \{x_0, x_2\})$.

Lemma 17.13 (Testing reduced representations) For any subset P of $\mathbb{P}\mathcal{X}$ and subsets α, β of \mathcal{X} , we have that P passes the test (α, β) iff the channel P^\cup passes that same (α, β) .

Proof. If P^\cup passes the test then so does P , because $P \subseteq P^\cup$.

If P^\cup fails the test (α, β) then for some $\pi_{1, \dots, N}$ in P we have $\cup(\pi_{1, \dots, N}) \subseteq \alpha$ but $\cup(\pi_{1, \dots, N}) \not\subseteq \beta$. From the latter we have $\pi_n \not\subseteq \beta$ for some n ; but from the former we still have $\pi_n \subseteq \alpha$ for that n . Because that π_n from P fails the test, so does P itself. \square

From here on, we will use reduced representations if convenient. In fact, among reduced representations of a channel there is a smallest one where no cell is the union of any other cells (except itself). We call that “the” reduced representation of the channel, and note that all deterministic channels in $\mathbb{E}\mathcal{X}$ are reduced.

Definition 17.14 (Reduced demonic channels) A subset P of $\mathbb{P}\mathcal{X}$ is a *reduced* channel just when $\cup P = \mathcal{X}$ and no cell π in P is the union $\cup \pi_{1, \dots, N}$ of any other cells in P except trivially $\cup \{\pi\}$. Note that \emptyset is excluded from any reduced representation, since it is $\cup \{\}$. \square

Lemma 17.15 (Uniqueness of reductions) Any demonic channel P in $\mathbb{U}\mathcal{X}$ has a unique reduction, i.e. a unique reduced channel P^r in $\mathbb{P}\mathcal{X}$ such that $P = (P^r)^\cup$.

Proof. Existence of a reduction of P is trivial: keep removing superfluous cells in P until no more are superfluous.

For uniqueness we argue first from Lem. 17.13 and the soundness of testing that two reductions P', P'' of the same P must satisfy $P' \sqsubseteq P''$ and $P'' \sqsubseteq P'$.

Now take any π'' from P'' . From $P' \sqsubseteq P''$ we have that π'' is the union of some set of cells in P' ; but from $P'' \sqsubseteq P'$ we have that each of those cells π' is the union of some set of cells $P''_{\pi'} \subseteq P''$ again. By collapsing the double union we have thus constructed π'' as the union of (other) cells in P'' ; but since P'' is reduced, the only such union is of $\{\pi''\}$ itself — that is $P''_{\pi'} = \{\pi''\}$, and so $\pi' = \cup P''_{\pi'} = \cup \{\pi''\} = \pi''$, establishing that π'' is in P' . \square

17.8 Compositional closure

The tests of Def. 17.11 show how to reject $S \sqsubseteq K$ by exhibiting a test that S passes but K fails. The utility of a discriminating test is that, if you can find it, it proves the failure with a single witness. But the tests (α, β) are hardly an obvious, intuitive choice themselves.

To justify refinement’s definition to both client and vendor, we use the same technique as in §9.8.3 — we appeal to a more primitive notion of correctness that we take as self-evidently desirable for security (of demonic channels): that if K can ever reveal that its input is some x *exactly* but S never can, then K cannot be a refinement of S .

That is, our aim here is the same as in §9.8.3, where compositional closure was used to reduce general (\sqsubseteq) to the primitive-refinement criterion (\preccurlyeq) of comparing Bayes vulnerability, and to the construction in §10.2.2 that uses a Dalenius context for the same purpose.

Definition 17.16 (Primitive refinement of channels) We say that S is *primitively refined* by P just when there is no singleton cell $\{x\}$ in P that is not also in S . We write it $S \preccurlyeq P$. \square

Put more simply, Def. 17.16 says that $S \preccurlyeq P$ unless there is a particular, single x that P can reveal but S cannot. “I might not know any theory; but I know that if S guarantees never to leak my password, then P can’t either.”¹⁶ ¹⁷

As earlier in §9.8.3, it’s the simplicity of (\preccurlyeq), in everyday terms, that is its justification. But it is too simple for general use: Def. 17.16 does not justify (\sqsubseteq) directly. If S leaks the last character of a password, but K leaks the last two characters, then probably $S \not\sqsubseteq K$ — but we will still have $S \preccurlyeq K$ because neither leaks the password exactly.

Therefore to justify (\sqsubseteq) using Def. 17.16 we must do more: for that, we recall that channels will probably not be used alone: larger channels can be made from a collection of smaller ones.

Definition 17.17 (Demonic-channel parallel composition) The parallel composition of two demonic channels $C^{1,2}$ over the same input \mathcal{X} but outputs $\mathcal{Y}^{1,2}$ respectively is a new demonic channel of type $\mathcal{X} \rightarrow (\mathcal{Y}^1 \times \mathcal{Y}^2)$ defined for matrices as

$$(C^1 \| C^2)_{x,(y^1,y^2)} := C_{x,y^1}^1 \wedge C_{x,y^2}^2 .$$

For $P^{1,2} : \mathbb{U}\mathcal{X}$ the corresponding definition is

$$P^1 \| P^2 := \{\pi^1 \cap \pi^2 \mid \pi^1 : P^1, \pi^2 : P^2\} .$$

\square

In the deterministic model (\parallel) is meet (\sqcap), in effect sending both spies and getting them both back. In (\sqcap) for the demonic model it is not: instead the adversary gets only one of them back. But since the nondeterminism is demonic, we (the defenders) must assume that the one she *does* get back is the worse, from our point of view.

In any case, an adversary with access to two channels $C^{1,2}$ acting on the same input can be considered to be using a single channel $C^1 \| C^2$ that is their composition: she can potentially observe the composite output (y^1, y^2) just when she could have observed y^1 from C^1 and y^2 from C^2 .

We now use parallel composition to give (actually to recall from §9.8.3) two desirable principles that should apply to (\sqsubseteq) in general:¹⁸

safety If $S \sqsubseteq I$ then we should have primitive refinement even in the context of an arbitrary (other) channel C , that is $(S \| C) \preccurlyeq (I \| C)$.

¹⁶ Recall “Any fool can see.....” from §9.8.3.

¹⁷ Just to be clear: a security breach releasing some large number N of passwords usually means in our terms that there are N singleton cells, not that there is just one cell with N passwords in it. The former means that each of N people has his password leaked exactly. The latter means instead that someone’s password is learned to be one of those N .

¹⁸ Together they are an equivalence because $S \sqsubseteq P$ iff $(S \| C) \preccurlyeq (P \| C)$ for all C .

necessity If $S \not\sqsubseteq K$ then for there must be some (other) channel C that justifies the failure, i.e. such that $(S\|C) \not\leq (K\|C)$.

(Compare the probabilistic versions of those given earlier, in Def. 9.19.)

From the two principles above we see easily that if $S \sqsubseteq P$ then $S \preccurlyeq P$, from applying *safety* with the identity context. And we also have that

monotonicity If $S \sqsubseteq P$ then $(S\|C) \sqsubseteq (P\|C)$ for any (other) channel C — for, if not, by *necessity* there would be (still another) channel D such that $(S\|C)\|D \not\leq (P\|C)\|D$, that is by associativity $S\|(C\|D) \not\leq P\|(C\|D)$; and that, by safety wrt. channel $C\|D$, implies $S \not\sqsubseteq P$.

(There are further properties listed at the end of §9.8.3.)

We note that the basic principles rest on two informal notions: that $S \not\leq K$ reasonably captures “ K is clearly broken” in the sense a layman expecting S might understand it, and that $(\|C)$ describes “contexts” in which laymen would expect our channels reasonably to be able to operate. In particular, robustness emphasizes that checking channels’ security properties individually is not enough: two adversaries could have one channel each and, if they combined their results, they would in fact be acting like a single adversary using the channels’ composition, probably a more powerful attack than is possible with either channel alone.

Once those notions (\preccurlyeq) and $(\|C)$ are fixed, safety and necessity determine refinement (\sqsubseteq) uniquely, as was shown in Chap. 9 for probabilistic channels. That is, justification of (\preccurlyeq) and $(\|C)$ and safety and necessity are collectively a justification of our definition of (\sqsubseteq) and, further, it is the only refinement relation that can be justified that way. As we saw in §9.8.3, that justification procedure is called *compositional closure*, i.e. that (\sqsubseteq) is the compositional closure under $(\|)$ of (\preccurlyeq) .

The derived principles have direct significance for everyday use when a system $C_1 \| \dots \| C_N$ might comprise many subsystems C_n : if a vendor establishes $S \sqsubseteq I$ through his software-development practices then, because (as well) he has established $S \preccurlyeq I$, his client will be happy; and monotonicity says that the vendor can use stepwise refinement (in the sense of Wirth) on his C_n ’s separately to modularize his software-development process that ultimately produces the whole system $C_1 \| \dots \| C_N$.

Theorem 17.18 (Refinement for demonic channels is justified) Def. 17.3 of refinement satisfies safety and necessity wrt. Def. 17.16 (primitive refinement) and Def. 17.17 (parallel composition).

Proof. We treat safety and then necessity.

Safety

Assume that $S \sqsubseteq I$ but suppose for a contradiction that $S\|C \not\leq I\|C$. In that case there must be ι, γ from I, C respectively and input x such that the intersection $\iota \cap \gamma$ is $\{x\}$ for some x in \mathcal{X} , indicating that when the observation of $I\|C$ is (ι, γ) an adversary would know that the input was x exactly — furthermore we must have that that does not occur with any σ from S . Now because $S \sqsubseteq I$ we have $\iota = \cup \sigma_{1,\dots,N}$ for some $\sigma_{1,\dots,N}$ each in S , so that

$$(\sigma_1 \cap \gamma) \cup \dots \cup (\sigma_N \cap \gamma) = \{x\} \text{ also,}$$

and so for at least one n we must have $\sigma_n \cap \gamma = \{x\}$, contradicting “furthermore” at (†) just above. Thus by contradiction we have $S\|C \leq I\|C$ as desired.

Necessity

If $S \not\subseteq K$ then by §17.5 (completeness and Def. 17.11) there is an (α, β) test that S passes but K fails, that is

$$\begin{array}{ll} \text{for all } \sigma: S & \sigma \subseteq \alpha \Rightarrow \sigma \subseteq \beta \\ \text{for some } \kappa: K & \kappa \subseteq \alpha \wedge \kappa \not\subseteq \beta \end{array}, \text{ and}$$

Choose therefore an element x in $\kappa - \beta$: we will define C so that $K \| C$ contains cell $\{x\}$ but $S \| C$ does not. Let channel $C: \mathbb{U}\mathcal{X}$ have cells

$$\{x\} \cup (\mathcal{X} - \alpha) \quad \text{and} \quad \alpha - \{x\} \quad . \quad ^{19} \quad (17.6)$$

From Def. 17.17 the parallel composition $K \| C$ contains the cell

$$\kappa \cap (\{x\} \cup (\mathcal{X} - \alpha)) = (\kappa \cap \{x\}) \cup (\kappa \cap (\mathcal{X} - \alpha)) = \{x\} \cup \emptyset = \{x\} ,$$

because $x \in \kappa \subseteq \alpha$, and so $K \| C$ can reveal that x .

Now suppose for a contradiction that $S \| C$ also contains $\{x\}$, i.e. that either $\sigma \cap (\{x\} \cup (\mathcal{X} - \alpha)) = \{x\}$ or that $\sigma \cap (\alpha - \{x\}) = \{x\}$ for some σ in S . The second is clearly not possible. From the first we see immediately that $x \in \sigma$ (since the equality “ $= \{x\}$ ” establishes that x is an element of the left-hand side) and that $\sigma \subseteq \alpha$ (since x is the *only* element of the left-hand side). From those two facts we have also that $x \in \beta$, because σ satisfies (α, β) . But we chose x from $\kappa - \beta$ above, the contradiction that establishes $\{x\} \notin S \| C$.

So we conclude that if $S \not\subseteq K$ then there is an input x and a channel C such that $K \| C$ can reveal x exactly but $S \| C$ can never reveal x exactly, and so indeed we have $S \| C \not\leq K \| C$. \square

17.9 “Weakest pre-tests” and source-level reasoning

We now turn to the possibility of source-level reasoning, which was our motivation in §17.6 for adopting the (α, β) test in favor of the (x, χ) test.

For eventual source-level reasoning, where e.g. leakage via channels is made a primitive imperative-programming statement (as we have explored earlier in this Part IV), we can imagine asking what security guarantees we must have *before* a program runs in order to be sure that running the program has not leaked “too much” information *afterwards*.

Suppose that in our letters example it’s especially important that the spies never learn that our x is exactly **A**, because **A** is information about a *VIP*. For us, the other people **B**, **E**, **W** are not so important.

Let our program (i.e. channel) be P with typical cell names π . To express “ P never reveals that x is **A**” using a test in the style of Def. 17.11, we could write $\pi \subseteq \{\mathbf{A}\} \Rightarrow \pi \subseteq \emptyset$ for all cells π in P . We can see by inspection from Fig. 17.7 that both the “one spy returns” channel and the “radio spies” channel pass that test (because all of their cells π do).

So now we complicate things by imagining that, as a result of previous missions, M has some *a priori* knowledge about our x , knowledge that we would also like to express as a test. For example we might say that she knows *before* she sends Vowel

¹⁹ We are giving the reduced form: union closure would however add only \emptyset and \mathcal{X} . It is thus a Boolean channel, as a matrix having two columns labeled *true* and *false*.

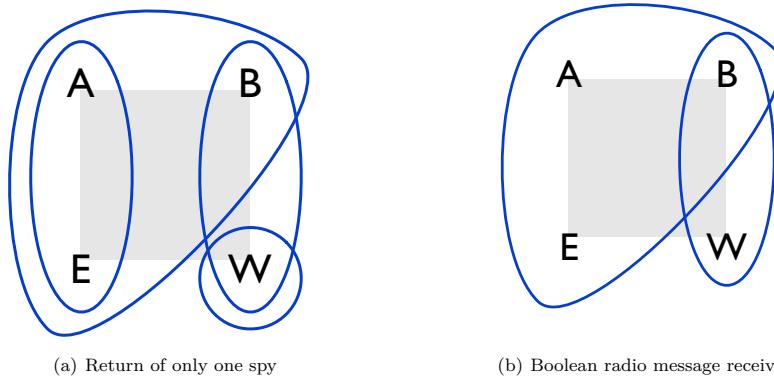


Figure 17.7 Ms. Vowel and Mrs. Early in action

and Early that x cannot be E, expressing that with the test $\pi \subseteq \{ABW\}$.²⁰ Could she ever learn after her spies have returned that not only is x not E (which she does not forget) but actually it is A?[§]

The general “weakest pre-test” question for protecting A is²¹

What security criterion must our x satisfy *before* the spies are sent in order to ensure that M cannot know $x = A$ *after* the spies have reported? (17.7)

Obviously the pre-test $x \neq A$ would be strong enough — if you don’t want A to be leaked, don’t put it in your database. But can we do better?

The effect that M ’s *a priori* knowledge, expressed as a cell μ say, has on her spies’ output cells is simply that each cell π becomes $\pi \cap \mu$ — she learns π from the channel, and she knew μ already. Thus to convert our post-test $\pi \subseteq \{A\} \Rightarrow \pi \subseteq \emptyset$ on π to a pre-test on μ alone, we replace π by $\pi \cap \mu$, to give

$$(\pi \cap \mu) \subseteq \{A\} \Rightarrow (\pi \cap \mu) \subseteq \emptyset . \quad (17.8)$$

We concentrate first on the channel Fig. 17.7(a), instantiating the above for every π it contains — we can do that because we know the channel’s construction and so we know what π ’s it can produce. For example, if we take $\pi = \{W\}$, we get $(\{W\} \cap \mu) \subseteq \{A\} \Rightarrow (\{W\} \cap \mu) \subseteq \emptyset$, which is equivalent to $\mu \subseteq ABE \Rightarrow \mu \subseteq ABE$, that is just *true*. For all four cells of Fig. 17.7(a) it’s

$$\begin{array}{llll} \mu \subseteq ABE & \Rightarrow & \mu \subseteq ABE & \text{true} \quad \text{when } \pi = W \quad (\text{done just above}) \\ \mu \subseteq ABW & \Rightarrow & \mu \subseteq BW & ? \quad \text{when } \pi = AE \\ \mu \subseteq AE & \Rightarrow & \mu \subseteq AE & \text{true} \quad \text{when } \pi = BW \\ \mu \subseteq AW & \Rightarrow & \mu \subseteq W & ? \quad \text{when } \pi = ABE , \end{array} \quad (17.9)$$

where “?” means that it depends on μ (and *true* means that it does not).

In each case we get a test again, but of “pre-cell” μ rather than “post-cell” π , because for any cell χ we can write $\pi \cap \mu \subseteq \chi$ as $\mu \subseteq \chi \cup (\chi - \pi)$. Thus our overall pre-test for

²⁰ More precisely that is the test $\pi \subseteq \mathcal{X} \Rightarrow \pi \subseteq \{ABW\}$.

²¹ This is obviously by analogy with the weakest preconditions of Dijkstra.

Fig. 17.7(a) and the post-test $\pi \subseteq \{A\} \Rightarrow \pi \subseteq \emptyset$ is the second and fourth cases above, that is the

$$\mu \subseteq ABW \Rightarrow \mu \subseteq BW \quad \text{and} \quad \mu \subseteq AW \Rightarrow \mu \subseteq W \quad (17.10)$$

that we get by discarding the *true*'s from (17.9).

Thus a single post-test can generate a *conjunction* of pre-tests, which *conjunctions* we take therefore as our general form of test. If we do it again, but taking now those conjunctions to be of post-tests, we will just generate conjunctions of conjunctions for the new pre-tests — and so we do not have to expand our expressiveness any further. Furthermore, every member of $U\mathcal{X}$ is characterized uniquely by a conjunction of such tests: every conjunction of tests is union closed; and for every union-closed set there is a conjunction of tests that only it satisfies. (See Exercise 17.9.) In the case above however the first conjunct of (17.10) implies the second, and so we end up with the first one alone.²² To cast it back into everyday language, we rewrite (17.10) equivalently as $E \notin \mu \Rightarrow \mu \subseteq BW$, and it becomes our answer to the question posed at (17.7) above: that is, that M 's prior knowledge must be that

- ‡ If M knows that x is not E , it must be because she knows it's B or W .
(Equivalently, if she knows it's not E , she must know it's not A either.)

Under those conditions, her one-spy-returns attack will never reveal that $x = A$.

Now for the “radio spies” Fig. 17.7(b) we get only the second conjunct (because the case $\pi = AE$ of (17.9) is missing), which as we have just seen is weaker than the first and so we can withstand “ M 's knowing more beforehand”. That is, in Fig. 17.7(b) we are secure against M 's knowing beforehand that $x \neq B$ as at (\$) above; but in Fig. 17.7(a) we are not. That's not surprising, since Fig. 17.7(a) ⊑ Fig. 17.7(b) and therefore we expect to be less at risk from the radio spies.

For source-level reasoning we could e.g. write channels as primitive statements `PRINT c st Φ(c,x)` where Φ is a formula in state variables x and bound variable c , which latter is the emitted value: it means that in state x the channel can emit any value c satisfying $\Phi(c,x)$ for that value x of x . (The denotation of `PRINT`'s argument in the general case is therefore a set-valued function of \mathcal{X} , just as for probabilistic channels the general case is a distribution-valued function of \mathcal{X} .)

As a special case we'd write `PRINT Exp(x)` for the deterministic case, i.e. when Φ is $c = Exp(x)$ for some expression Exp in x . With a modality $K\Psi$ we could express that the current cell π satisfied $\pi \subseteq \{x: \mathcal{X} | \Psi(x)\}$, and our tests would then be of the form $(K\Psi_1 \Rightarrow K\Omega_1) \wedge \dots \wedge (K\Psi_N \Rightarrow K\Omega_N)$, or more generally $(\forall c \bullet K\Psi(x,c) \Rightarrow K\Omega(x,c))$ to express weakest-pre-test-generated conjunctions.

With all that, expressing our “weakest pre-test”, i.e. the `wt` approach, at the source level (and making reference to variables implicit) would give in general

$$\begin{aligned} & \text{wt}(\text{PRINT } c \text{ st } \Phi(c), K\Psi \Rightarrow K\Omega) \\ &= (\forall c \bullet K(\Phi \Rightarrow \Psi) \Rightarrow K(\Phi \Rightarrow \Omega)) \quad , \end{aligned}$$

and for the deterministic case $(\forall c \bullet K(Exp=c \Rightarrow \Psi) \Rightarrow K(Exp=c \Rightarrow \Omega))$.

The pre-test $E \notin \mu \Rightarrow A \notin \mu$ that we discovered at (‡) above, to constrain M 's prior knowledge, would be therefore be rendered at the source level as

$$K(x \neq E) \Rightarrow K(x \neq A) \quad \begin{array}{l} \text{If } M \text{ knows } x \text{ is not } E, \\ \text{then she also knows it's not } A. \end{array}$$

²² In (17.10) here the (α', β') on the right is weaker than (α, β) on the left because we have $\alpha' \subseteq \alpha$ and $\alpha' \cap \beta \subseteq \beta'$. Compare Footnote 15 on p. 338 above.

Looking further ahead, we remark that for *updates* to the hidden state x the weakest pre-test is particularly simple, because updates leak nothing: if for example statement S is some assignment $x := \text{Exp}(x)$, then the weakest pre-test is given by

$$\text{wt}(S, K\Psi \Rightarrow K\Omega) = K(\text{wp}(S, \Psi)) \Rightarrow K(\text{wp}(S, \Omega)), \quad ^{23} \quad (17.11)$$

where wp is the conventional Dijkstra-style weakest precondition; and that applies even when S is a demonic assignment (like a choice from a set). Non-leaking statements generate no pre-conjunctions. Conventional pre- and postconditions are embedded as “half tests” $K(\text{true}) \Rightarrow K\Omega$, equivalently just $K\Omega$, and are respected by (17.11).

17.10 Exercises

Exercise 17.1 Recall the “leak everything” and “leak nothing” demonic channels \mathbb{O} and $\mathbb{1}$ respectively. How are they represented as matrices? As reduced sets of cells? As union-closed sets of cells? \square

Exercise 17.2 The matrix- and the cell-based definitions of structural refinement have a slight mismatch: whereas Def. 17.2 requires every row of R to have at least one 1 (since it is the same type as a demonic channel), Def. 17.3 does not require every cell of S to participate in some union contributing to cell of I . Put informally, that means that some cells of S can be simply “discarded” which, to accomplish with a matrix, would require a row of all zeroes in the row corresponding to that cell (as a column of S). Although throwing an S -cell away is consistent with the general idea that in demonic semantics it is a refinement to discard an outcome, mathematically this is an issue that should be tied down.

Why does the difference not matter mathematically? \square

Exercise 17.3 Show that the deterministic meet of two (deterministic) channels is their parallel composition. \square

Exercise 17.4 Recall §17.4. Construct the channel matrix for the internal choice between C^1 and C^2 where Ms. Vowel radios “true” for vowel and Mrs. Early radios “true” for early. Verify that it matches the cell diagram of Fig. 17.7(b), and conclude that indeed internal demonic choice is not the demonic meet.

Determine the internal choice between Ms. Vowel and Mrs. Late and verify that it is different from the above, even though in information-theoretic terms Mrs. Early and Mrs. Late are equivalent. What do you conclude about compositionality? \square

Exercise 17.5 Recall Def. 17.11. Show that the reveal-everything channel \mathbb{O} fails every non-trivial test. \square

Exercise 17.6 Recall Def. 17.17. Show that the set of cells $P^1 \| P^2$ constructed there is union closed. \square

Exercise 17.7 Recall the necessity argument in Thm. 17.18. Is the channel C there deterministic or (properly) demonic? \square

Exercise 17.8 Show that for any subsets α, β of \mathcal{X} the set of x ’s that satisfy the condition $x \subseteq \alpha \Rightarrow x \subseteq \beta$ is union closed. \square

²³ We assume here that S is everywhere terminating.

Exercise 17.9 Assume that \mathcal{X} is finite. Show that for any union-closed set U of subsets of \mathcal{X} there is a finite set of pairs $(\alpha_1, \beta_1) \cdots (\alpha_N, \beta_N)$ such that $\chi \in U$ just when $\chi \subseteq \alpha_n \Rightarrow \chi \subseteq \beta_n$ for all $1 \leq n \leq N$. \square

Exercise 17.10 The channel in Fig. 17.7(a) is complement closed, yet it is not a partition. Does that contradict Lem. 17.9? \square

Exercise 17.11 Suppose we have two sets of cells S, K such that $S \not\subseteq K$ according to Def. 17.3, i.e. that there is some cell in K which is not the union of any subset of cells from S . That is, by analogy with (\sqsubseteq_0) from §9.2.2, a structural definition, one in which probabilistic refinement is effectively the “weighted summation of specification inners to make implementation inners”. (That weighted summation is precisely what the refinement matrix R does.)

Now the demonic S, K above can be regarded as abstractions of properly probabilistic S', K' say, where for each probabilistic inner we retain only its support (a cell) and for the probabilistic outer we retain only its support (thus a set of cells rather than a distribution of cells). In the other direction, one could say that S', K' are probabilistic “realizations” of S, K , obtained by assigning probabilities to the elements in the cells and then to the cells themselves — and of course there could be many hypers that realize a single set of cells.

With all that in mind, explain why the demonic $S \not\subseteq K$, with respect to the “union of cells” definition of refinement, implies that for *any* probabilistic realizations S', K' of them the probabilistic refinement must also fail, that is that $S' \not\subseteq K'$ with respect to the “weighted sum” definition of probabilistic refinement *no matter what* probabilities have been used in S', K' to realize S, K . \square

Exercise 17.12 Define the function of type $\mathbb{D}^2\mathcal{X} \rightarrow \mathbb{P}^2\mathcal{X}$ that takes a hyper-distribution to the set of cells that is its abstraction in the demonic model. \square

Exercise 17.13 Given the situation described in Exercise 17.11, show how to construct from

- a qualitative test (α, β) that is a witness for $S \not\subseteq K$, and
- the actual probabilities introduced by some S', K' that realize S, K (the latter, abstractions of S', K' , have no probabilities: they are only sets of cells),

a prior π and loss function ℓ such that $U_\ell[\pi \triangleright S'] > U_\ell[\pi \triangleright K']$, which —by the Coriaceous Theorem 9.12— (and Exercise 17.11) we know *must exist* and confirms that also $S' \not\subseteq K'$.

(Note that the abstraction function taking S', K' back to S, K resp. is exactly the one you were asked to define in Exercise 17.12.) \square

17.11 Chapter notes

We have located a demonic model of information flow “in between” Landauer and Redmond’s deterministic model [6], their “lattice of information”, and the more recent probabilistic model, based on hyper-distributions, that is the principal subject of this text. Originally presented *ab initio* as “The Shadow Model” for demonic noninterference [11, 12], it is now more clearly structured; and as a result its properties can be divided into those inherent in demonic choice, and those shared with other models of information flow.

Looking backwards, we see that the deterministic model is a restriction (not an abstraction) of the demonic model: they give the same level of detail, but the latter describes more situations than the former. For example, collaboration of the Spies (§17.4) cannot be expressed at all in the deterministic model. Looking forwards, we see that the demonic model is an abstraction (not a restriction) of the probabilistic: they can describe the same systems, but the latter gives a more detailed (i.e. quantitative rather than only qualitative) description. For the Spies, we are abstracting from the probabilities that one or the other might return, and the prior probability on the secret letter A, B, E, W.

All three systems have the same structural definition of refinement, which is particularly evident when we use the matrix formulation: one channel I is a refinement of another channel S just when I can be obtained via post-multiplication by a so-called refinement matrix. This is in fact channel cascade, if the refinement matrix is for that purpose considered to be a channel from S -observables to I -observables.

The deterministic and the demonic systems are lattices wrt. the refinement order; but the probabilistic system is not, as §12.4 showed [10]: it is however a partial order if properly quotiented.

All three systems have a complementary testing-based definition of refinement, one that provides a witness to any refinement failure. All three systems can justify their refinement order by general principles, safety and necessity (§17.8) whereby the refinement relation is reduced to a more primitive form (\preccurlyeq) that is accepted “by the layman”. (In the probabilistic case, the reduction was to the more primitive *Bayes vulnerability*, the probability of guessing the secret in one try [7, 2].)

Finally, we mention that those systems show how the notion of security has become more sophisticated over the decades. Originally a system was said to be secure or insecure, an absolute black-or-white judgment, based on whether it suffered from “strong dependency” or “interference” [3, 5]. Later it was realized that this criterion is too strong, since almost no useful system can be wholly interference-free: even a password-based login system releases information when a login attempt fails.

That led to the idea of comparing two programs’ information flow, particularly comparing a specification with an implementation: in the deterministic case, the implementation cannot leak except when the specification would too; in the demonic case, the implementation cannot leak except when the specification *might*. In the probabilistic case, the comparison is even more sophisticated: both specification and implementation can leak, but the implementation must leak *no more than* the specification does, where the “no more” is measured by gain- or loss functions.

Our long-term aim is to enable this kind of refinement-based reasoning at the source-code level, based on “information-flow aware” assertions like those proposed in §17.9. From those it should be possible to construct an algebra of program transformations that preserve security- and functional characteristics during the program-development process in which specifications are manipulated to become implementations. Indeed the material earlier in this Part IV illustrates some of our efforts in that direction.

Finally, we would like eventually to add a further layer above those mentioned here, allowing probability, demonic choice and secrecy to be handled all at once. Indeed, just as deterministic programs and (purely) probabilistic programs become (non-trivial) meet semi-lattices only when we include demonic choice [1, 9], we suspect that the addition of demonic choice to our hyper-distribution model might recover at least the lower semi-lattice property. We have already made preliminary investigations in that direction [8].

Finally, it is clear that the foundational ideas of Wirth and Dijkstra have contributed to our aims here [13, 4].

Bibliography

- [1] Back, R.-J., von Wright, J.: Refinement Calculus: A Systematic Introduction. Springer, Berlin (1998)
- [2] Bordenabe, N.E., Smith, G.: Correlated secrets in quantitative information flow. In: Proceedings of the 2016 IEEE 29th Computer Security Foundations Symposium (CSF 2016), pp. 93–104. IEEE, Los Alamitos (2016)
- [3] Cohen, E.: Information transmission in computational systems. In: Proceedings of the 6th ACM Symposium on Operating Systems Principles (SOSP 1977), pp. 133–139. ACM, New York (1977)
- [4] Dijkstra, E.W.: A Discipline of Programming. Prentice Hall (1976)
- [5] Goguen, J.A., Meseguer, J.: Unwinding and inference control. In: Proceedings of the 1984 IEEE Symposium on Security and Privacy, pp. 75–86. IEEE, Los Alamitos (1984)
- [6] Landauer, J., Redmond, T.: A lattice of information. In: Proceedings of the 1993 IEEE Computer Security Foundations Workshop (CSFW 1993), pp. 65–70. IEEE, Los Alamitos (1993)
- [7] McIver, A., Meinicke, L., Morgan, C.: Compositional closure for Bayes risk in probabilistic noninterference. In: S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide, P.G. Spirakis (eds.) Proceedings of the 37th International Colloquium on Automata, Languages, and Programming (ICALP 2010), *Lecture Notes in Computer Science*, vol. 6199, pp. 223–235. Springer, Berlin (2010)
- [8] McIver, A., Meinicke, L., Morgan, C.: A Kantorovich-monadic powerdomain for information hiding, with probability and nondeterminism. In: Proceedings of the 2012 27th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2012), pp. 461–470. IEEE, Los Alamitos (2012)
- [9] McIver, A., Morgan, C.: Abstraction, Refinement and Proof for Probabilistic Systems. Monographs in Computer Science. Springer, Berlin (2005)
- [10] McIver, A., Morgan, C., Meinicke, L., Smith, G., Espinoza, B.: Abstract channels, gain functions and the information order. In: 2013 Workshop on Foundations of Computer Security (FCS 2013), p. 17 (2013)
- [11] Morgan, C.: *The Shadow Knows*: Refinement of ignorance in sequential programs. In: T. Uustalu (ed.) Proceedings of the International Conference on Mathematics of Program Construction (MPC 2006), *Lecture Notes in Computer Science*, vol. 4014, pp. 359–378. Springer, Berlin (2006)

Bibliography

- [12] Morgan, C.: *The Shadow Knows*: Refinement of ignorance in sequential programs. *Science of Computer Programming* **74**(8), 629–653 (2009)
- [13] Wirth, N.: Program development by stepwise refinement. *Communications of the ACM* **14**(4), 221–227 (1971)

Part V

Applications



Chapter 18

The Crowds protocol

In this chapter we apply our theory of quantitative information flow to the analysis of the *Crowds* anonymous-communication protocol. The protocol is simple enough to allow an analytic derivation of all relevant quantities, yet rich enough to illustrate a variety of the conceptual tools we developed in Parts II and III. In particular, the analysis will involve channels, hyper-distributions, Bayes- and g -leakage (for a properly constructed gain function), capacity, refinement, and channel composition.

18.1 Introduction to Crowds, and its purpose

Crowds is a simple protocol for anonymous web surfing. The context is that a user, called the *initiator*, wants to contact a *web server* but does not want to disclose his identity to the server; and he achieves that by collaborating with a group of other users, called the *crowd*. The protocol is essentially a simple probabilistic routing protocol:

- In the first step, the initiator selects a user uniformly (including possibly himself) and sends his server-request, his *message*, to that user. The user who receives the message is now the (first) *forwarder*.
- Upon receiving a message, a forwarder –whether first or subsequent– flips a (biased) probabilistic coin: with probability φ he forwards the message to a new user (again chosen uniformly, including himself), but with remaining probability $1-\varphi$ he instead delivers the message directly to the web server. Note that it is a two-stage process: first decide probabilistically *whether* to forward to another user; and then decide probabilistically *to whom*. Those subsequent steps are called *hops*, distinguished from the first step because the first step cannot send to the server.

The protocol is illustrated in Fig. 18.1. If $\varphi < 1$ then the message will with probability 1 eventually arrive at the web server, with the expected number of hops being $1/(1-\varphi)$.

Concerning anonymity, from the point of view of the web server all users seem equally likely to have been the initiator (assuming a uniform prior), due to the first step's always being a (pure) forwarding step. The analysis becomes more interesting however if we assume that some users in the crowd are corrupt, reporting to an adversary who is trying to discover messages' initiators.¹

¹ This is a relatively weak adversarial model; a model in which the adversary can view any communication in the network would be too strong for *Crowds*.

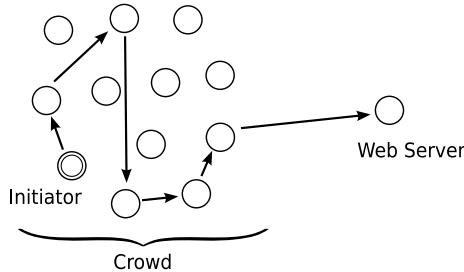


Figure 18.1 The Crowds protocol

If User a forwards a message to a corrupted user, then the corrupted user will report that to the adversary, and we say that a was *detected*. Since a 's being detected is more likely when he is the initiator than when he is not, the protocol does indeed leak information: detecting a creates a posterior where a has a greater chance of being the initiator than he had *a priori*. Still, if $\varphi > 0$ then the posterior is nevertheless *not* a point distribution: User a can “plead not guilty” by claiming that he was merely forwarding for someone else. The greater φ is, the more plausible that claim becomes: the forwarding probability φ can be seen as trading anonymity for utility (expected number of hops).

In their original work on Crowds, Reiter and Rubin introduced the property of “probable innocence”: informally speaking, that property requires that each user appear more likely *not to be* the initiator of the message, than *to be* the initiator (although he might still appear more likely to be the initiator than any other user). That property can be formalized as follows.²

Definition 18.1 (Probable innocence) —

A protocol satisfies *probable innocence* iff, assuming a uniform prior over initiators, the posterior probability of any particular user's being the initiator of a message, given any observation by the adversary, is at most $1/2$.

18.2 Modeling the Crowds protocol

To perform an analysis of the anonymity guarantees of Crowds using our theory of quantitative information flow, we start by modeling the protocol as an information-theoretic channel (Chap. 4). If we let n, c, m be the number of honest, corrupted, and total users respectively (i.e. $n+c = m$), it is “Who is the initiator?” that is the secret, thus

$$\mathcal{X} := \{x_1, \dots, x_n\} ,$$

with x_a denoting that it was (honest) User a . The observations are

$$\mathcal{Y} := \{y_1, \dots, y_n, s\} ,$$

² Note that Reiter and Rubin gave a different formalization of probable innocence, by limiting the probability of specific observable events. We opt for Def. 18.1 instead because, even though it is equivalent in the context of Crowds, it is also meaningful as a generic definition for an arbitrary protocol. (Note that Def. 18.1 does not mention Crowds.)

with y_b denoting that User b was detected (forwarded a message to a corrupted user), and finally s denoting that the message was delivered to the web server without having been forwarded to a corrupted user, i.e. that no user was detected.

To construct the channel matrix $C: \mathcal{X} \rightarrow \mathcal{Y}$, we need to compute the probability of producing any y_b or s given that the secret is x_a . We begin by considering the observation s (no user is detected), and note that it occurs iff the message is forwarded one or more times only to honest users, and then is delivered to the server: a direct calculation summing over the number j of steps gives

$$\begin{aligned} p(s|x_a) &= \sum_{j=0}^{\infty} \frac{n}{m} \left(\varphi \frac{n}{m} \right)^j (1-\varphi) \\ &= \frac{n}{m} (1-\varphi) \left(\frac{1}{1 - \varphi n/m} \right) \\ &= \frac{n - \varphi n}{m - \varphi n} . \end{aligned}$$

Now let D_1 be the event of detecting *some* user on the *first* step, and $D_{\geq 2}$ the event of detecting *some* user after *two or more* steps, i.e. after a hop. Clearly, the two events are disjoint and their union is the complement $\neg s$ of s , i.e. that someone *was* detected, and so we have

$$\begin{aligned} p(D_1 \vee D_{\geq 2} | x_a) &= 1 - p(s|x_a) \\ &= 1 - \frac{n - \varphi n}{m - \varphi n} \\ &= \frac{c}{m - \varphi n} . \end{aligned}$$

Moreover, we have directly that $p(D_1|x_a) = c/m$ and so

$$\begin{aligned} p(D_{\geq 2}|x_a) &= p(D_1 \vee D_{\geq 2} | x_a) - p(D_1|x_a) \\ &= \frac{c(m - (m - \varphi n))}{m(m - \varphi n)} \\ &= \frac{c\varphi n}{m(m - \varphi n)} . \end{aligned}$$

For the probabilities of detecting a *specific* user, we first observe that on the first step we can detect only *the initiator*, that is

$$\begin{aligned} p(D_1 \wedge y_a | x_a) &= c/m , \quad \text{and} \\ p(D_1 \wedge y_b | x_a) &= 0 \quad \text{for } b \neq a . \end{aligned}$$

After the first step, however, by symmetry all users are equally likely to possess the message, hence also equally likely to be detected after two or more steps. That is,

$$p(D_{\geq 2} \wedge y_b | x_a) = \frac{1}{n} p(D_{\geq 2} | x_a) \quad \text{for all } b \text{ (including } a\text{)} ,$$

and so we can bring those together to give

$$\begin{aligned} p(y_a|x_a) &= p(D_1 \wedge y_a | x_a) + p(D_{\geq 2} \wedge y_a | x_a) = \frac{c(m - \varphi(n - 1))}{m(m - \varphi n)} , \quad \text{and} \\ p(y_b|x_a) &= p(D_1 \wedge y_b | x_a) + p(D_{\geq 2} \wedge y_b | x_a) = \frac{c\varphi}{m(m - \varphi n)} \quad \text{for } b \neq a . \end{aligned}$$

Note that due to symmetry the probability of detecting an initiator, or some non-initiator, does not depend on which user the initiator was. In other words, the probabilities $p(y_b|x_a)$ depend only on whether $a=b$ or $a \neq b$, and not on the exact value of a or b . Hence the entries of the channel matrix have only these 3 distinct values:

$$\begin{aligned} C_{x_a,s} &= \alpha := \frac{n - \varphi n}{m - \varphi n}, \\ C_{x_a,y_a} &= \beta := \frac{c(m - \varphi(n-1))}{m(m - \varphi n)}, \\ C_{x_a,y_b} &= \gamma := \frac{c\varphi}{m(m - \varphi n)} \quad \text{for } a \neq b. \end{aligned} \tag{18.1}$$

And so the channel matrix is of the form

$$C = \begin{matrix} & y_1 & \cdots & y_n & s \\ x_1 & \beta & \cdots & \gamma & \alpha \\ \vdots & & \ddots & & \\ x_n & \gamma & \cdots & \beta & \alpha \end{matrix}.$$

Now from that channel matrix C , and assuming a uniform prior θ , we can compute the posterior distributions (inners) as well as the output distribution (outer) produced by C , which together form the hyper-distribution $[\theta \triangleright C]$. Following the calculations of §4.2 we get an output distribution of

$$\rho = (1 - \alpha/n, \dots, 1 - \alpha/n, \alpha),$$

and the corresponding $n+1$ posteriors

$$\begin{aligned} \delta^{y_b} &= (\frac{\varphi}{m}, \dots, \frac{\varphi}{m}, \frac{m - \varphi(n-1)}{m}, \frac{\varphi}{m}, \dots, \frac{\varphi}{m}) \\ \delta^s &= (1/n, \dots, 1/n). \end{aligned}$$

As expected, detecting User b suggests that he is the initiator: the resulting posterior δ^{y_b} assigns higher probability to b than to the remaining users. And that posterior highly depends on the probability of forwarding φ . For example in the extreme case $\varphi=0$ users never forward, so detecting b guarantees that b is the initiator; in other words $\delta^{y_b} = [x_b]$ (the point distribution on x_b) in that case.

Having computed the posterior distributions, we can directly express probable innocence (Def. 18.1) by requiring that $\delta_{x_a}^{y_b} \leq 1/2$ for all a and b — and that holds just when $\delta_{x_b}^{y_b} \leq 1/2$ in particular, giving³

$$\varphi \geq \frac{m}{2(n-1)} \tag{18.2}$$

as the condition for probable innocence.

Intuitively, probable innocence imposes a lower bound on the probability of forwarding, which in turn depends on the ratio of honest to total users.

³ Recall that $\delta_{x_a}^{y_b}$ is the probability assigned by distribution δ^{y_b} to element x_a .

We also need in the $a \neq b$ case that $\varphi/m \leq 1/2$, which is trivial if the total number of users m is at least 2. If it is not, then there is at most one honest user, whose probable innocence is of course *not* guaranteed — no matter what φ might be. Obviously we exclude those degenerate cases.

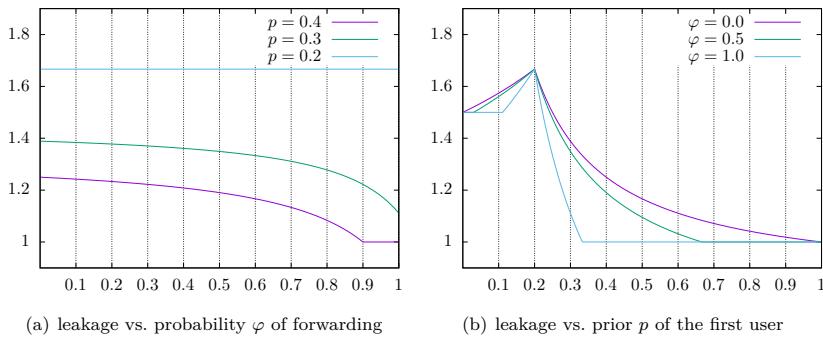


Figure 18.2 Bayes leakage for various priors and forwarding probabilities

18.3 Bayes vulnerability and Bayes leakage

The Bayes vulnerability V_1 is a natural choice for measuring the anonymity guarantees of a protocol. In this case, it measures the adversary's probability of successfully guessing (in one try) the initiator of the message, while the Bayes leakage measures how much that probability increases due to the protocol.

The prior case is simple: we have $V_1(\vartheta) = 1/n$ since all (honest) users have equal probability of being the initiator. In the posterior case, we could compute $V_1[\vartheta \triangleright C]$ from its definition; however, there is an easier way — Thm. 7.2 states that the multiplicative Bayes leakage for a uniform prior is equal to the multiplicative Bayes capacity, which is in turn given by the sum of the column maximums of C . Since $\beta \geq \gamma$, we can calculate that

$$\mathcal{L}_1^{\times}(\vartheta, C) = \mathcal{ML}_1^{\times}(\mathbb{D}, C) = n\beta + \alpha = \frac{n(c+1)}{m} .$$

There is a truly remarkable fact about the above equation: the Bayes leakage (as well as the posterior vulnerability) *does not depend* on the probability φ of forwarding! How can that be possible? If φ is exactly what allows a detected user to plead “not guilty”, how can the case $\varphi=0$ have the same leakage as the case $\varphi=1$?

One way of restoring the dependence of φ would be to consider the max-case Bayes vulnerability, which in this case is equal to the Bayes vulnerability of any of the posteriors δ^{y_b} , that is

$$V_1^{\max}[\vartheta \triangleright C] = V_1(\delta^{y_b}) = \frac{m - \varphi(n-1)}{m} , \quad (18.3)$$

and it clearly does depend on φ . But still, this does not explain why the average-case leakage is independent of φ . Moreover, having to rely on max-case vulnerability to explain the importance of φ would be strange.

Investigating Bayes vulnerability further shows that ϑ is the *only prior* (apart from the degenerate cases of point priors) where this paradoxical behavior appears. Fig. 18.2 shows the multiplicative Bayes leakage of an instance of Crowds with $n=5$ and $c=1$ for various priors. The priors are mapped to a single dimension by taking p in $[0, 1]$ as the probability of the first (honest) user, the other four equally sharing the remaining $1-p$ (so that the uniform prior corresponds to $p=0.2$). On the left-hand side, leakage

is plotted as a function of φ , and each line corresponds to a different prior p . We can see that only the uniform case $p=0.2$ is independent of φ , and indeed in the two non-uniform cases the leakage decreases as φ increases. In fact, for $p=0.4$, the forwarding probability φ can be set so high that the evidence of detecting any user is not sufficient to overcome the *high prior probability* of User 1 — the adversary will always guess User 1, regardless of the output of the protocol, meaning that there is no leakage (i.e. that the multiplicative leakage is 1).

On the right-hand side, the leakage is plotted as a function of the prior p , for different values of φ . We can see the leakage for $\varphi=0$ is always above that of $\varphi=1/2$, which in turn is always above the leakage for $\varphi=1$. The only points where all three graphs meet are the uniform and the point priors.

Thus the paradoxical behavior is restricted to the case of a uniform prior. Still, we haven't explained why that behavior happens. We do that in the following section, by studying a modified version of the protocol.

18.4 Explanation of the paradox

18.4.1 Modified Crowds

Consider a modified version of the protocol in which, when a user forwards a message to a corrupted user, the latter can *somewhat* know whether this forwarding happened as the *first step* of the route or not. Of course this is a flawed protocol: we will only use it as a means for the analysis.

In the model of this modified version the set of secrets remains the same, but the set of observations becomes

$$\mathcal{Y} = \{(y_1, 1), (y_1, 2), \dots, (y_n, 1), (y_n, 2), s\} ,$$

with $(y_a, 1)$ meaning that User a was detected in the first round of the protocol, while $(y_a, 2)$ means that User a was detected in the second or a later round.

Computing the modified matrix $\widehat{\mathbf{C}}$ shows that the real initiator is detected in the first round only if he directly forwards the message to a corrupted user. Hence we have

$$\widehat{C}_{x_a, (y_a, 1)} = \widehat{\beta} = \frac{c}{m} .$$

On the other hand, detecting any user other than the initiator in the first round is impossible: that is because we have

$$\widehat{C}_{x_a, (y_b, 1)} = 0 \quad \text{for } a \neq b .$$

The probability of s remains the same, with $\widehat{C}_{x_a, s} = \alpha$ as before. Finally, after the first round, the message is in the possession of any user with equal probability. Hence, due to symmetry, the probability of detecting any user in the second or a later round is the same, independently of who the initiator was: thus

$$\widehat{C}_{x_a, (y_b, 2)} = \widehat{\gamma} = \frac{1 - \alpha - \beta}{n} .$$

And so the modified channel matrix $\widehat{\mathbf{C}}$ is of the form

$$\widehat{\mathbf{C}} = \begin{bmatrix} (y_1, 1) & \cdots & (y_n, 1) & (y_1, 2) & \cdots & (y_n, 2) & s \\ x_1 & \widehat{\beta} & \cdots & 0 & \widehat{\gamma} & \cdots & \widehat{\gamma} & \alpha \\ \vdots & & \ddots & & & \ddots & & \\ x_n & 0 & \cdots & \widehat{\beta} & \widehat{\gamma} & \cdots & \widehat{\gamma} & \alpha \end{bmatrix} ,$$

and the first immediate observation we can make is that the original C is a refinement of \widehat{C} : we can recover the original Crowds by post-processing with a channel that forgets the extra information:

$$\widehat{C} \sqsubseteq C .$$

Hence the original version never leaks more than the modified version, which is to be expected: it can only be better, never worse.

A more interesting observation is that, seen as an *abstract channel* (§4.4), Matrix \widehat{C} is *independent* of the probability φ of forwarding: we have

$$\widehat{C}^\varphi \sqsubseteq \widehat{C}^{\varphi'} \sqsubseteq \widehat{C}^\varphi \quad \text{for all } \varphi, \varphi', \quad (18.4)$$

where \widehat{C}^φ indicates the dependence of the matrix on φ . We can see that by taking a closer look at the matrix \widehat{C} , where we notice that all columns $(y_a, 2)$ –as well as the s column– are similar: they all provide no information to the adversary since they are produced with the same probability by any initiator. Since $\widehat{\beta}$ is independent of φ , merging those columns together gives us a reduced matrix that is independent of φ . By antisymmetry of (\sqsubseteq) on abstract channels, from (18.4) we have the claimed independence from φ .

In other words, in the original Crowds,

the role of φ is exactly to prevent the adversary from knowing whether the detection happened in the first round or not.

In the modified version, that information is provided explicitly — so φ has no effect, simply moving probability between the similar events $(y_a, 2)$ and s .

18.4.2 Vulnerability of the original protocol

So far we know that the leakage of the modified protocol is independent of φ and is no smaller than the leakage of the original one. Still, this does not explain why the Bayes leakage of the *original* protocol under a uniform prior is independent of φ .

To understand that, we need to look at the formulation of posterior vulnerability as the adversary's expected gain under an optimal strategy S that maps observations to actions (Thm. 5.24):

$$V_g[\pi \triangleright C] = \max_S \mathbf{tr}(G^{\lceil \pi \rfloor} CS) . \quad (18.5)$$

Here as usual \mathbf{tr} is the trace operator, G is the gain function in matrix representation, the matrix $\lceil \pi \rfloor$ has π on the diagonal and 0's elsewhere, and finally S is a channel from \mathcal{Y} to \mathcal{W} .

Let's now look at the *optimal* strategies S and \widehat{S} for Bayes vulnerability with respect to C and \widehat{C} in the case of a uniform prior. In the original protocol S should map y_a to x_a , since detecting User a increases his chances of being the initiator. The s -observable provides no information, so the posterior δ^s is uniform and hence s can be mapped to any initiator.

In the modified protocol, an optimal \widehat{S} should clearly map $(y_a, 1)$ to x_a . The observations $(y_a, 2)$ however are similar to s : they provide no information, so the posterior remains uniform and hence it can be mapped to *any initiator* and not necessarily the *same* for each a . That freedom to choose any initiator is *crucial* to our argument: a possible choice is to map $(y_a, 2)$ to x_a . That means that $(y_a, 1)$ and $(y_a, 2)$ –which are separate events in the modified protocol, but fused in the original one— will be mapped to the *same secret*. Hence we have that

$$CS = \widehat{C}\widehat{S} \quad (18.6)$$

and, since the strategies are optimal, we find that for any φ, φ' we have

$$\begin{aligned} & V_1[\vartheta \triangleright C^\varphi] \\ = & V_1[\vartheta \triangleright \widehat{C}^{\varphi}] && \text{"by (18.5),(18.6)"} \\ = & V_1[\vartheta \triangleright \widehat{C}^{\varphi'}] && \text{"by (18.4)"} \\ = & V_1[\vartheta \triangleright C^{\varphi'}] && \text{"by (18.5),(18.6)"} . \end{aligned}$$

In other words, for Bayes vulnerability and under a uniform prior, there is an optimal strategy that is independent of whether detection happens on the first or a later round, and all φ does is to confuse those two cases. Hence φ does not affect V_1 .

The possibility of mapping $(y_a, 1)$ and $(y_a, 2)$ to the same secret, however, arises only under a uniform prior. In the case of a non-uniform prior π , say, still $(y_a, 1)$ will be mapped to x_a (because this evidence is strong enough to overcome any prior knowledge) but we no longer have that choice for $(y_a, 2)$ — the posterior is the same as π so, in order to be optimal, we need to map it to the user with the higher prior probability! Hence the property (18.6) is broken, and the leakage of the original protocol will be strictly less than that of the modified one. As expected, it will be dependent on φ .

18.5 Why φ matters, even for uniform priors

Comparison with the modified protocol explained why the Bayes vulnerability of the original Crowds is independent of φ under a uniform prior. Although the independence fails for other priors, our intuition was that, *even* in the uniform case, increasing φ offers better anonymity. The original analysis in terms of probable innocence reinforces that intuition: probable innocence is satisfied only if φ is sufficiently large.

We already discussed max-case vulnerability V_1^{\max} as a possible explanation of why φ is important even for uniform priors. But can we have an average-case explanation of this fact? In other words, can we create a gain function g such that $V_g[\vartheta \triangleright C^\varphi]$ depends on φ ?

Going back to our optimal-strategy reasoning, finding such a g is not hard: all we need do is construct actions \mathcal{W} and gain function g in such a way that the optimal actions for $(y_a, 1)$ and $(y_a, 2)$ are necessarily different, even for a uniform prior. Interestingly, a gain function g_{lion} , very similar to g_{tiger} from §3.2.5, behaves in just that way. This gain function corresponds to an adversary who is penalized for making an incorrect guess. (Our favorite penalty is opening a trap door to a pit of lions or tigers.) As for g_{tiger} , one of the options available to such an adversary is the action \perp of not making any guess at all, and her gains are given by

$$g_{\text{lion}}(w, x) := \begin{cases} 1, & \text{if } w = x \\ \frac{1}{2}, & \text{if } w = \perp \\ 0, & \text{otherwise} \end{cases} .$$

Note that g_{lion} and g_{tiger} are conceptually the same, differing only in the actual gains assigned to the action \perp (no guess), and to an incorrect guess.

Going back to the modified Crowds, we see that the “smoking gun” evidence is $(y_a, 1)$ — the adversary can be sure that a is the initiator, so she can safely guess x_a . On the other hand $(y_a, 2)$ provides no evidence at all: the posterior remains uniform, and the best action remains \perp . Hence it is no longer true that the optimal strategy is independent of whether the detection happens on the first or a later round, and so the leakage no longer needs to be independent of φ .

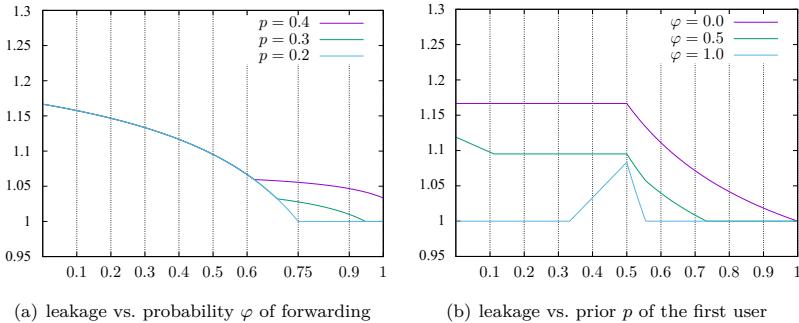


Figure 18.3 Lion leakage for various priors and forwarding probabilities

Indeed, the dependence on φ is confirmed by Fig. 18.3, showing the same graphs as Fig. 18.2 but now for g_{lion} . On the left-hand side we see that, even for the uniform prior given by $p = 0.2$, the leakage decreases as φ increases.

18.5.1 Probable innocence as no lion leakage

More interestingly, it could be argued that the g_{lion} adversary, being penalized for wrong guesses, is exactly the adversary that the original “probable innocence” property was meant to protect from. The idea of plausibly pleading “not guilty” assumes some hypothetical “trial” in which the user is “accused” of being the initiator. A reasonable court of law clearly wants to convict the person with the highest probability of being guilty. However, it doesn’t *have to* convict anyone: a wrongful conviction is considered more harmful than the lack of a correct conviction, which is exactly the g_{lion} scenario.

Under g_{lion} , the expected gain for guessing x after observing y is $1 \times \delta_x^y = \delta_x^y$, where δ^y is the corresponding posterior. On the other hand, the gain of \perp is always $1/2$. As a consequence, as long as $\delta_x^y \leq 1/2$ for all x, y , the adversary (i.e. the court) will always choose \perp , which is also the best guess *a priori*.

The case when the adversary never changes her guess exactly characterizes no leakage. This brings us to the following result, which is a consequence of Thm. 5.31.

Theorem 18.2

A protocol modeled by channel C satisfies probable innocence (Def. 18.1) just when $\mathcal{L}_{g_{\text{lion}}}^X(\vartheta, C) = 1$.

Indeed, from (18.2) we know that Crowds satisfies probable innocence just when $\varphi \geq m/2(n-1) = 0.75$ (for $n = 5, c = 1$). In Fig. 18.3(a) we see that this is exactly the threshold above which leakage becomes 1 in the uniform case.

18.6 Refinement: increasing φ is always safe

In the previous sections, we saw that the Bayes leakage of Crowds, for a uniform prior, is independent of the probability φ of forwarding. Hence, we could decrease φ (to make paths shorter), without affecting anonymity. This behavior, however, is very specific to the combination of g_{id} and ϑ ; we saw that changing the prior to non-uniform, or

the gain function to g_{lion} , both lead to a leakage that might increase when φ decreases. In other words, there are situations in which decreasing φ is *not* safe.

This brings us to the natural and complementary question: is *increasing* φ always safe? Our intuition suggests that this is true; the more we forward, the more we confuse the adversary. Moreover, all the plots in the previous sections show that increasing φ decreases the corresponding leakage measures. But can we be sure that this *always* happens? Could it be the case that for some strange adversary, modeled by a bizarre gain function g , increasing φ causes the leakage to increase as well, leading to a less safe protocol?

To answer this question, we employ the theory of refinement from Chap. 9. Similarly to the previous sections, we denote by C^φ the channel matrix of Crowds for a specific forwarding probability φ . Now consider two such values $\varphi \leq \varphi'$, and let α, β, γ and α', β', γ' be the corresponding elements of C^φ and $C^{\varphi'}$ respectively. From our calculation of C^φ (18.1) we see that $\alpha \geq \alpha'$; intuitively, a higher probability of forwarding makes it less likely that the message will arrive at the server without detection. On the other hand, we have that $\beta \leq \beta'$ and $\gamma \leq \gamma'$; since users forward more often under φ' , they are all more likely to be detected. Moreover, it holds that

$$\beta' - \beta = \gamma' - \gamma = \frac{\alpha - \alpha'}{n} .$$

That is, the increase of probability for each observable y_1, \dots, y_n is the same in all rows of the matrix.

This means that we can convert C^φ to $C^{\varphi'}$ by reducing the probability of s and equally redistributing it to y_1, \dots, y_n . That can be achieved by *post-processing* C^φ with the following channel

$$R := \begin{array}{c} \begin{matrix} & y_1 & \cdots & y_n & s \\ y_1 & \left[\begin{matrix} 1 & \cdots & 0 & 0 \\ \ddots & & & \\ 0 & \cdots & 1 & 0 \\ \kappa & \cdots & \kappa & \lambda \end{matrix} \right] \end{matrix} \\ \text{where} \end{array}$$

$$\kappa := \frac{\alpha - \alpha'}{n\alpha}, \quad \lambda := \frac{\alpha'}{\alpha} .$$

It is easy to verify that R is indeed a valid channel matrix and that $C^\varphi R = C^{\varphi'}$. That means that increasing the probability φ produces a *refinement* of the original protocol: it is indeed a safe operation in the sense that for *any* prior and gain function it can only decrease the protocol's leakage.

The reason the above fails when $\varphi > \varphi'$ is that in that case the matrix R is *not* a valid channel matrix, because $\lambda > 1$. In fact, it can be shown that no channel R such that $C^\varphi R = C^{\varphi'}$ exists. Hence, reducing φ is not always a safe operation, although it might be safe in some special cases –as in the case of Bayes leakage under a uniform prior– but there will always be some adversary for which leakage increases.

The above discussion is summarized in the following result. (See Exercise 18.2.)

Theorem 18.3

For Crowds-protocol channels C^φ and $C^{\varphi'}$ we have $C^\varphi \sqsubseteq C^{\varphi'}$ iff $\varphi \leq \varphi'$.

Proof. The “if” part comes from the construction $C^\varphi R = C^{\varphi'}$ above.

For the “only if”, assume that $\varphi > \varphi'$. We could show that no such channel R exists, but it is in fact easier to use Thm. 9.11 by constructing a counterexample gain function under which $C^{\varphi'}$ leaks strictly more than C^φ .

Let $t = V_1^{\max}[\vartheta \triangleright C^\varphi]$. From (18.3) and $\varphi > \varphi'$, we know that $t < V_1^{\max}[\vartheta \triangleright C^{\varphi'}]$. Then define g_t as g_{id} with an extra action \perp such that $g_t(\perp, x) = t$ for all $x \in \mathcal{X}$. From Thm. 5.31 we get that $V_{g_t}[\pi \triangleright C^\varphi] = t < V_{g_t}[\pi \triangleright C^{\varphi'}]$. \square

18.7 Multiple paths

So far we have studied the anonymity of Crowds when a *single path* is established from the initiator to the server. But what happens if a user establishes *multiple paths* to the same server? On the one hand, using multiple paths could be desirable, for instance for load-balancing purposes. But more importantly, establishing a new path could be *forced by an adversary* controlling either the server or intermediate users, by simply ceasing to respond, thus rendering the original path useless.

Note that we assume that the adversary can always link distinct paths as having originated from the same original user (although she does not know which one). That could be done by inspecting either the content of the request (which might be in plaintext), but also by using metadata such as its size or timing.

The anonymity of the protocol crucially depends on how the new path is created. We consider two cases:

1. The new path is recreated from scratch by the initiator, by restarting the protocol (either voluntarily, or after detecting a failure).
2. The old path is kept until the *last working node*, which detects the failure and tries to “repair” the path by removing the faulty node from its list, randomly selecting a new node and forwarding the request to him.

The two cases are analyzed in detail in the following sections, each requiring different tools from the quantitative-information-flow theory.

18.7.1 Paths recreated by the initiator

When a path is recreated by the initiator, the whole protocol is executed from scratch, producing a new run of the original channel under the same secret. If k paths are created then the protocol can be modeled as $C^{(k)}$: the k independent runs of C , in other words its parallel composition $C^{(k)} = C \parallel \dots \parallel C$ with itself k times (Chap. 8).

That allows us to apply the following result bounding the multiplicative Bayes-capacity of the parallel composition of two arbitrary channels; its proof is the earlier Exercise 8.11.

Theorem 18.4 Given compatible channels C_1 and C_2 , we have

$$\mathcal{ML}_1^{\times}(\mathbb{D}, C_1 \parallel C_2) \leq \mathcal{ML}_1^{\times}(\mathbb{D}, C_1) \times \mathcal{ML}_1^{\times}(\mathbb{D}, C_2) .$$

\square

Applying that to Crowds we get that

$$\mathcal{ML}_1^{\times}(\mathbb{D}, \mathbf{C}^{(k)}) \leq \mathcal{ML}_1^{\times}(\mathbb{D}, \mathbf{C})^k = \left(\frac{n(c+1)}{m} \right)^k.$$

Unless $c=0$ (in which case the protocol has no leakage at all), that bound grows with increasing k . For small values of k , that (together with the “Miracle” Thm. 7.5) provides an easy way of bounding the leakage of the protocol. For large values of k , however, the bound becomes too loose; indeed, it can be shown that $\mathcal{ML}_1^{\times}(\mathbb{D}, \mathbf{C}^{(k)})$ converges asymptotically to the number of distinct rows of \mathbf{C} . For Crowds that is n , meaning that in the long run the system completely exposes the identity of the initiator.

If we are interested in a precise analysis instead of bounds, we can compute directly the desired leakage measure for $\mathbf{C}^{(k)}$. For instance, the max-case Bayes vulnerability will be given by the observation k times of the same User b , that is

$$V_1^{\max}[\vartheta \triangleright \mathbf{C}^{(k)}] = V_1(\delta^{y_b, \dots, b}) = \frac{1}{1 + (\gamma/\beta)^k}.$$

Again, we see that repeated executions increase leakage; as k grows the vulnerability converges to 1. If we are interested in probable innocence we can compute the largest k for which this vulnerability remains below $1/2$.

Based on the analysis above, we conclude that paths should remain static and new ones should be created by the initiator as rarely as possible. In their original design, Reiter and Rubin propose creating new paths only when new users join the Crowds, which should happen in infrequent scheduled events (the purpose being to protect the anonymity of the new users).

18.7.2 Paths repaired by the last working node

An adversary controlling any node in the path can force it to be recreated by simply stopping the forwarding of any traffic. However, recreating the path from scratch causes severe information leakage, as we saw in the previous section. To cope with that, broken paths in Crowds are not recreated by the initiator, but are instead repaired by the last working node, by choosing a new node and forwarding the request to him. New forwarding requests can of course reach corrupted users, so that poses the natural question: does this operation increase information leakage? In their original paper, Reiter and Rubin state:

“Since the [corrupted users] cannot distinguish whether that predecessor is the initiator or not, the random choices made by that predecessor yield no additional information to the [adversary].”

Although the intuition is clear, the argument remains quite informal. Having seen that repeated executions highly affect leakage, we would like to have a *robust formal* argument that repairing paths in this way is safe under any circumstances.

Let $\hat{\mathbf{C}}$ be the channel modeling the protocol when the adversary forces the path to be recreated once. Its secrets \mathcal{X} are the same as for \mathbf{C} , but its observations are now $\mathcal{Y} \times \mathcal{Y}$; the observation (y_a, y_b) means that User a was detected when the path was first created, while User b was detected when the path was repaired. The crucial observation is that we can obtain $\hat{\mathbf{C}}$ by post-processing \mathbf{C} with a channel R from \mathcal{Y} to $\mathcal{Y} \times \mathcal{Y}$ that, given an observation y_a of \mathbf{C} , restarts the protocol (after removing the faulty node) from User a , obtains a new observation y , and then outputs both y_a, y_b .

More precisely, let C' be the channel modeling Crowds with $c-1$ corrupted users (since the non-responding corrupted node was removed), and let R be defined as

$$\begin{aligned} R_{y_a, (y_a, y_b)} &= C'_{x_a, y_b} && \text{for } x_a \text{ in } \mathcal{X} \text{ and } y_b \text{ in } \mathcal{Y}, \\ R_{s, (s, s)} &= 1 , \end{aligned}$$

and 0 elsewhere. It is then easy to see that $\widehat{C} = CR$. Moreover, \widehat{C} can be trivially post-processed back to C (by ignoring the second observation), hence $C \sqsubseteq \widehat{C} \sqsubseteq C$. In other words C and \widehat{C} are different representations of the same abstract channel, which means that their leakage is exactly the same. Using the same argument we can conclude that any number of repaired paths causes no further leakage, in sharp contrast with the analysis of the previous section.

18.7.3 Multiple detections and deviating from the protocol

In our analysis of Crowds we assumed that, in each execution, at most one user is detected by a corrupted node (which is equivalent to assuming that corrupted nodes always deliver the message to the server, as well as communicating with the adversary). However, if instead a corrupted node forwards the message to another user, it is possible to detect further users in the extended path if they too choose to forward to a corrupted node. Do those extra observations increase information leakage? Moreover, a corrupted node might not be obliged to execute the protocol faithfully; it could for instance forward the message to a specific user of its choice. Could such deviations help the adversary in any way?

We can answer those questions in a way similar to the analysis of the previous section. If corrupted nodes cannot affect the actions of honest nodes, then a channel modeling deviations from the protocol can be obtained by post-processing the original C with a channel R , just as above. And that will always lead to a refinement of C that (therefore) cannot leak more information than C itself under any circumstances.

18.8 Exercises

Exercise 18.1 Show that removing a *corrupted* user (while keeping the number of honest users fixed) is safe, i.e. is always a refinement. \square

Exercise 18.2 In §18.6 it was shown rigorously that increasing the forwarding probability φ results in a refinement of the protocol, i.e. that for any prior and gain function the effect of increasing φ cannot be to increase the adversary's gain — increasing φ can never do any harm.

But from that it is elementary that *decreasing* φ cannot *decrease* the adversary's gain (because then increasing φ back to its original value would contradict the above). Thus decreasing φ can never do any good.

If that reasoning is so elementary, why do we bother to prove the “only if” for Thm. 18.3? \square

18.9 Chapter notes

The Crowds protocol was proposed by Reiter and Rubin [5]. The analysis of its multiplicative Bayes leakage is from Espinoza and Smith [4].

Theorem 18.4 was proved by Barthe and Köpf [1]. Repeated independent runs are studied by Boreale, Pampaloni, and Paolini [2]; they show that the multiplicative Bayes capacity of $C^{(n)}$ converges asymptotically to the number of distinct rows in C .

In §18.6, we showed that increasing φ cannot increase leakage. But from the standpoint of efficiency, it is actually desirable to *decrease* φ , since that lowers the protocol's latency. It is therefore valuable to determine the maximum amount by which leakage can increase if φ is decreased. This question is studied by Chatzikokolakis and Smith [3].

Syverson [6] gives the name “entropist conception” to security analysis in terms of some Rényi entropy on a set of possible senders, and he argues that the entropist conception is fundamentally inadequate for analyzing the actual security threats to real-world anonymous communication systems like onion routing and mix networks. An interesting question is whether the richer g -vulnerability framework could accurately model such threats.

Bibliography

- [1] Barthe, G., Köpf, B.: Information-theoretic bounds for differentially private mechanisms. In: Proceedings of the 2011 IEEE 24th Computer Security Foundations Symposium (CSF 2011), pp. 191–204. IEEE, Los Alamitos (2011)
- [2] Boreale, M., Pampaloni, F., Paolini, M.: Asymptotic information leakage under one-try attacks. In: M. Hofmann (ed.) Proceedings of the 14th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS 2011), *Lecture Notes in Computer Science*, vol. 6604, pp. 396–410. Springer, Berlin (2011)
- [3] Chatzikokolakis, K., Smith, G.: Refinement metrics for quantitative information flow. In: M.S. Alvim, K. Chatzikokolakis, C. Olarte, F. Valencia (eds.) The Art of Modelling Computational Systems: A Journey from Logic and Concurrency to Security and Privacy. Essays Dedicated to Catuscia Palamidessi on the Occasion of Her 60th Birthday, *Lecture Notes in Computer Science*, vol. 11760, pp. 1–20. Springer, Berlin (2019)
- [4] Espinoza, B., Smith, G.: Min-entropy as a resource. *Information and Computation* **226**, 57–75 (2013)
- [5] Reiter, M.K., Rubin, A.D.: Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security* **1**(1), 66–92 (1998)
- [6] Syverson, P.: Why I'm not an entropist. In: Proc. 2nd Hot Topics in Privacy Enhancing Technologies (HotPETS 2009) (2009)



Chapter 19

Timing attacks on blinded and bucketed cryptography

In this chapter, we apply the theory of quantitative information flow to the important practical problem of *timing attacks on cryptography*. In such attacks an adversary observes the amount of time taken by a large number of cryptographic operations, in an effort to discover the secret key being used. We show here that if the cryptosystem is implemented with the countermeasures known as *blinding* and *bucketing*, then the maximum leakage can be bounded tightly.

19.1 Cryptographic background

Starting with Kocher's seminal work, it has been known that an adversary able to observe the *times* taken by a large number of decryption operations may be able to recover the secret key, even if the decryption operations are being done remotely, making the timing measurements imprecise. Our interest here is not in understanding *how* key recovery can be done, but instead in modeling the timing information –obtained by the adversary– as a *channel* whose leakage we can analyze.

Consider an implementation of decryption whose execution time is given by a function $t: \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{T}$ such that $t(K, M)$ gives the time required to decrypt message M using secret key K . Here \mathcal{K} , \mathcal{M} , and \mathcal{T} are finite sets of possible secret keys, messages, and decryption times, respectively. (Note that a deterministic model like this is not correct for hardware that uses caching, but it could be made valid by forcing the cache into a fixed initial state before each decryption.)

In a timing attack, the adversary might be able to *choose* a large number of messages M_1, M_2, \dots, M_n and observe the time to decrypt each one using the same secret key K . *Blinding* is a countermeasure that de-correlates the messages from the decryption times; it works by *randomizing* each message M before the decryption, and *derandomizing* afterwards so that the correct decryption is obtained.

To see how blinding can be done in the particular case of *RSA*, note that the *RSA* decryption of M using secret key (d, N) is $M^d \bmod N$. If the corresponding public key is (e, N) , then M can be randomized by choosing a uniformly distributed r that is relatively prime to N , and setting $M' = M \cdot r^e \bmod N$, which makes M' uniformly distributed. Note that M' decrypts to $(M \cdot r^e)^d = M^d \cdot r \bmod N$, so the decryption can be derandomized by multiplying it by $r^{-1} \bmod N$.

The result of blinding is that the adversary observes not $t(K, M)$ for her chosen M , but instead $t(K, M')$ for a *randomly chosen* M' . It follows that a single timing observation is then modeled by a *probabilistic* channel C from \mathcal{K} to \mathcal{T} , whose input is a secret key K and whose output is the time to decrypt a randomly chosen message M' using K . And an n -observation timing attack is then modeled by a *repeated-independent-runs channel* $C^{(n)}$, which is an n -fold version of the parallel composition discussed in §8.1.1.¹

Bucketing is an additional countermeasure that decreases the size of \mathcal{T} by forcing each decryption to take one of b possible times, for some small number b . This is done by choosing a set of b “bucket” times, and then delaying each decryption result until the next bucket time. Experimentally, it has been shown that on 1024-bit RSA, the performance overhead from using $b = 5$ is less than 0.7% with a careful choice of bucket times. Even using $b = 2$ gives an overhead of under 3%. But using $b = 1$, a constant-time implementation, gives an overhead of over 36%.

Bucketing thus allows us to trade off between security and performance: choosing $b = 1$ eliminates timing attacks but substantially lowers performance, while choosing $b = 5$ gives better performance but allows stronger timing attacks.

To conclude: an n -observation timing attack on a blinded cryptosystem with b buckets can be modeled as a repeated-independent-runs channel $C^{(n)}$ where channel matrix C has b columns. In that setting, we typically expect that b is “small” (say, at most 25) and n is “big” (say, a billion or larger).

19.2 A first leakage bound

Motivated by the previous section, we aim to find tight bounds on the maximum leakage of the repeated-independent-runs channel $C^{(n)}$ when C has b columns, where $n \geq 0$ and $b \geq 1$. The leakage measure that we use is *multiplicative Bayes capacity*, as discussed in Chap. 7, for the following three reasons:

- $\mathcal{ML}_1^{\times}(\mathbb{D}, C)$ is *easy to compute* from a channel matrix C : it is simply the *sum of C's column maximums* (Thm. 7.2).
- $\mathcal{ML}_1^{\times}(\mathbb{D}, C)$ is *operationally significant*: it is the maximum factor (over all priors) by which C increases an adversary's probability of guessing the secret input X correctly in one try (§5.3).
- $\mathcal{ML}_1^{\times}(\mathbb{D}, C)$ is *robust*: it is an upper bound on multiplicative g -leakage, for any non-negative gain function g and any prior π (Thm. 7.5).

Recall from §8.1.1 that if C is a channel matrix from \mathcal{X} to \mathcal{Y} then $C^{(n)}$ is the channel matrix from \mathcal{X} to \mathcal{Y}^n given by

$$C_{x,(y_1,\dots,y_n)}^{(n)} := \prod_{i=1}^n C_{x,y_i} . \quad (19.1)$$

For example, with $\mathcal{X} = \{x_1, x_2, x_3\}$ and $\mathcal{Y} = \{y_1, y_2\}$, if C is

C	y_1	y_2
x_1	0	1
x_2	$1/3$	$2/3$
x_3	$1/2$	$1/2$

¹ Equivalently, it is the n -fold sequential composition in the programming model, as explained earlier in §14.6.3.

then the corresponding $C^{(3)}$ is

$C^{(3)}$	$y_1 y_1 y_1$	$y_1 y_1 y_2$	$y_1 y_2 y_1$	$y_1 y_2 y_2$	$y_2 y_1 y_1$	$y_2 y_1 y_2$	$y_2 y_2 y_1$	$y_2 y_2 y_2$
x_1	0	0	0	0	0	0	0	1
x_2	$\frac{1}{27}$	$\frac{2}{27}$	$\frac{2}{27}$	$\frac{4}{27}$	$\frac{2}{27}$	$\frac{4}{27}$	$\frac{4}{27}$	$\frac{8}{27}$
x_3	$\frac{1}{8}$							

Looking now at multiplicative Bayes capacity, for C and $C^{(3)}$ we have

$$\mathcal{ML}_1^{\times}(\mathbb{D}, C) = \frac{1}{2} + 1 = \frac{3}{2}$$

and

$$\mathcal{ML}_1^{\times}(\mathbb{D}, C^{(3)}) = \frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{4}{27} + \frac{1}{8} + \frac{4}{27} + \frac{4}{27} + 1 = \frac{35}{18}.$$

We now prove a first leakage bound, showing that for fixed b the multiplicative Bayes capacity of $C^{(n)}$ grows only polynomially in n .

Theorem 19.1 If C has $b \geq 1$ columns and $n \geq 0$, then $\mathcal{ML}_1^{\times}(\mathbb{D}, C^{(n)}) \leq \binom{n+b-1}{n}$.

Proof. The proof uses the information-theoretic *method of types*. The *type* of an output sequence is the proportion of each of the b possible outputs that it contains. For example, if $\mathcal{Y} = \{y_1, y_2, y_3, y_4\}$ and $n = 10$, then the output sequence $(y_3, y_2, y_2, y_4, y_2, y_3, y_2, y_2, y_2, y_2)$ from a run of $C^{(10)}$ has type $(0, 7/10, 2/10, 1/10)$, meaning “no y_1 ’s and 7 y_2 ’s etc.” Here the key observation is that if two output sequences have the same type, then their columns in $C^{(n)}$ are identical, because the probabilities in (19.1) are invariant under permutations of the output sequence. Therefore all columns with the same type are similar, and can be *merged* without affecting the abstract channel, and hence without affecting the multiplicative Bayes capacity. For example, here is the merged matrix of $C^{(3)}$ above, where now the columns are labeled with types:²

merged $C^{(3)}$	$(1, 0)$	$(2/3, 1/3)$	$(1/3, 2/3)$	$(0, 1)$
x_1	0	0	0	1
x_2	$\frac{1}{27}$	$\frac{2}{9}$	$\frac{4}{9}$	$\frac{8}{27}$
x_3	$\frac{1}{8}$	$\frac{3}{8}$	$\frac{3}{8}$	$\frac{1}{8}$

We recalled above that the multiplicative Bayes capacity of a channel matrix is the sum of its column maximums. Since each column maximum is at most 1, the capacity of the merged $C^{(n)}$ (and hence of $C^{(n)}$ as well) is at most the number of possible types.

Now, the number of types with n observations and b buckets is precisely the number of ways that n indistinguishable “balls” can be placed into b “bins”. Each such placement can be represented as a string of n stars (representing the balls) with $b-1$ bars inserted (representing the boundaries between the bins). For example, with $n = 5$ and $b = 4$, the string

$\ast\ast | \ast | | \ast\ast$

represents the case when we put 2 balls in the first bin, 1 ball in the second bin, 0 balls in the third bin, and 2 balls in the fourth bin. Since the number of ways of choosing n positions for the stars from the $n+b-1$ positions in the string is $\binom{n+b-1}{n}$, the theorem follows. \square

Note that in the particular cases when b is 1, 2, 3, or 4, the bounds given by Thm. 19.1 are 1, $n+1$, $(n+1)(n+2)/2$, and $(n+1)(n+2)(n+3)/6$, respectively.

² Note that this merged matrix may actually have more columns than the *reduced matrix* $(C^{(3)})^r$ of Def. 4.9, because (depending on C) some columns with *different* types might also be similar.

19.3 A better leakage bound

The obvious weakness of Thm. 19.1 is that it uses the trivial upper bound 1 for all the column maximums; we might sense that this is quite pessimistic. For a more careful analysis, we need to determine the maximum probability with which $C^{(n)}$ outputs a sequence of each type t . For example, suppose that $b = 2$ and $n = 1000$ and consider the type $t = (417/1000, 583/1000)$. When would a row of C have the greatest probability of generating an output sequence of type t ? Intuitively, it would seem best for a row of C to give exactly the same distribution on outputs as in t . This intuition turns out to be correct. Indeed, using the information-theoretic *method of types* it can be shown³ that if a row of C gives distribution q on \mathcal{Y} , then the probability that the corresponding row of $C^{(n)}$ outputs a particular sequence of type t is precisely

$$2^{-n(H(t) + D_{KL}(t||q))} ,$$

where $H(t)$ is the *Shannon entropy* of t and $D_{KL}(t||q)$ is the *Kullback-Leibler distance*

$$D_{KL}(\pi||\omega) := \sum_x \pi_x \log_2 \frac{\pi_x}{\omega_x}$$

between t and q . Now, *Gibbs' inequality* says that $D_{KL}(t||q) \geq 0$, with equality iff $t = q$. Hence the above probability is maximized (to $2^{-nH(t)}$) when $t = q$.

Turning our attention now to the merged $C^{(n)}$, we see that to get the maximum possible entry in the column for type t we need to multiply the probability above (of getting a particular sequence of type t) by the number of sequences of type t . But it seems easier to calculate this maximum probability directly. Any type t is of the form

$$\left(\frac{x_1}{n}, \frac{x_2}{n}, \dots, \frac{x_b}{n} \right) ,$$

where x_1, x_2, \dots, x_b are non-negative integers that sum to n . If a row of C matches t , then the corresponding row of the merged $C^{(n)}$ gives output t with probability

$$\frac{n!}{x_1!x_2!\dots x_b!} \left(\frac{x_1}{n} \right)^{x_1} \left(\frac{x_2}{n} \right)^{x_2} \dots \left(\frac{x_b}{n} \right)^{x_b} .$$

To see that, note that the first factor gives the number of sequences of type t , while the second factor gives the probability of each such sequence.

To get the maximum sum of the column maximums of the merged $C^{(n)}$, we just sum the above term over all possible choices of x_1, x_2, \dots, x_b . With some reorganization, that leads to a definition of the following function.

Definition 19.2

For integers $b \geq 1$ and $n \geq 0$, define

$$cap_b(n) := \frac{n!}{n^n} \sum_{\substack{x_1, x_2, \dots, x_b \in \mathbb{N} \\ x_1+x_2+\dots+x_b=n}} \frac{x_1^{x_1} x_2^{x_2} \dots x_b^{x_b}}{x_1!x_2!\dots x_b!} .$$

We have thus proved the following stronger bound on $\mathcal{ML}_1^\times(\mathbb{D}, C^{(n)})$.

³ This is Theorem 11.1.2 in the textbook of Cover and Thomas.

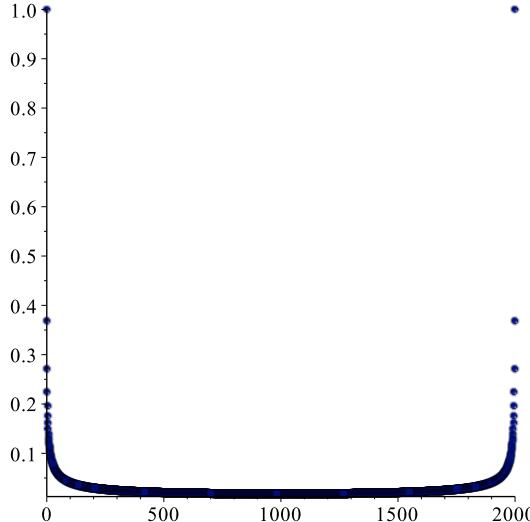


Figure 19.1 Plot of the terms of $cap_2(2000)$ for $0 \leq i \leq 2000$

Theorem 19.3 —

If C has $b \geq 1$ columns and $n \geq 0$, then $\mathcal{ML}_1^X(\mathbb{D}, C^{(n)}) \leq cap_b(n)$.

Note that the bound in Thm. 19.3 is *tight*, since for any n it is achieved by any C which has a row matching each possible type t . Moreover, it is plausible that a blinded timing channel might satisfy that condition, at least approximately, since such a channel has a huge number of rows (one for each possible secret key) but only polynomially many (in n) types.

To get a sense of the significance of Thm. 19.3, consider the case when $b = 2$. In that case, Def. 19.2 simplifies to

$$cap_2(n) = \frac{n!}{n^n} \sum_{i=0}^n \frac{i^i (n-i)^{n-i}}{i!(n-i)!} . \quad (19.2)$$

Now recall that the upper bound in Thm. 19.1 uses the trivial upper bound 1 for each column maximum. In fact the true values are far smaller, as shown in Fig. 19.1, which plots the values of the terms of $cap_2(2000)$ for $0 \leq i \leq 2000$. It shows that the term is 1 when i is 0 or 2000, and that its value falls sharply as i moves away from those endpoints. In particular, it is less than 0.2 for i between 4 and 1996, and less than 0.041 for i between 100 and 1900. The minimum value, about 0.0178, occurs when $i = 1000$.

We find that $cap_2(2000)$ is about 56.72, showing that $\mathcal{ML}_1^X(\mathbb{D}, C^{(2000)}) \leq 56.72$, which is a far better bound than the one given by Thm. 19.1, which is 2001.

Figure 19.2 shows the value of $cap_2(n)$ for $0 \leq n \leq 2000$. As can be seen, its growth appears to be much slower than linear. In fact, as will be seen in §19.4 below, $cap_2(n)$ is approximately $\sqrt{\pi n}/2$.

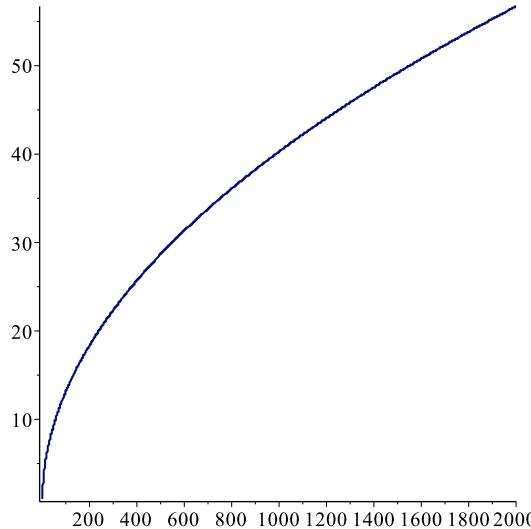


Figure 19.2 Plot of $\text{cap}_2(n)$ for $0 \leq n \leq 2000$

Computational challenges

The improved leakage bound in Thm. 19.3 is valuable, but unfortunately it is difficult to apply it in security analysis. The problem is that the formula in Def. 19.2 is very expensive to compute directly, since it involves $b - 1$ nested sums. In the case of a 100-observation timing attack with 6 buckets, for instance, we are interested in the value of $\text{cap}_6(100)$; but direct computation of this in Maple using exact rational arithmetic already takes about 20 minutes.⁴ And we would actually be interested in computing $\text{cap}_6(n)$ for a far larger number of timing observations, such as $n = 10^9$ or even $n = 10^{12}$. Note that when $b = 6$, the number of terms in Def. 19.2 is given by the binomial coefficient $\binom{n+5}{n}$, which for large n is about $n^5/120$. Hence increasing n from 10^2 to 10^9 increases the number of terms by a factor of 10^{35} . This implies that, even ignoring the fact that the individual terms become more expensive to compute as n increases, computing $\text{cap}_6(10^9)$ would take over 10^{30} years.

It is therefore clear that a better way of computing $\text{cap}_b(n)$ is needed before we can make serious use of Thm. 19.3.

19.4 Analytic results about $\text{cap}_b(n)$

Figure 19.3 gives a table of some of the values of $\text{cap}_b(n)$, where the row gives the value of b and the column gives the value of n . The first row suggests that we have $\text{cap}_1(n) = 1$ for all n , and this is indeed straightforward to verify from Def. 19.2. But the behavior of $\text{cap}_b(n)$ for larger values of b is not obvious. In the rest of this section, we present a number of analytic results that clarify the behavior of $\text{cap}_b(n)$ and make it possible to compute it efficiently.

⁴ The computation was done on an iMac with a 3.2 GHz Intel Core i5 and 16 GB of memory.

$\text{cap}_b(n)$	0	1	2	3	4	5	...
1	1	1	1	1	1	1	...
2	1	2	$\frac{5}{2}$	$\frac{26}{9}$	$\frac{103}{32}$	$\frac{2194}{625}$...
3	1	3	$\frac{9}{2}$	$\frac{53}{9}$	$\frac{231}{32}$	$\frac{5319}{625}$...
4	1	4	7	$\frac{92}{9}$	$\frac{437}{32}$	$\frac{10804}{625}$...
5	1	5	10	$\frac{145}{9}$	$\frac{745}{32}$	$\frac{19669}{625}$...
6	1	6	$\frac{27}{2}$	$\frac{214}{9}$	$\frac{591}{16}$	$\frac{33174}{625}$...
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	...

Figure 19.3 Some values of $\text{cap}_b(n)$, with row = b , column = n **A recurrence satisfied by $\text{cap}_b(n)$**

Through analytic and empirical study, it can be found that $\text{cap}_b(n)$ is governed by a remarkable recurrence.

Theorem 19.4

For all $b \geq 3$ and all $n \geq 0$,

$$\text{cap}_b(n) = \text{cap}_{b-1}(n) + \frac{n}{b-2} \text{cap}_{b-2}(n) .$$

To illustrate, we consider $b = 5$ and $n = 4$, where we have

$$\text{cap}_5(4) = \text{cap}_{5-1}(4) + \frac{4}{5-2} \text{cap}_{5-2}(4) = \frac{437}{32} + \frac{4}{3} \cdot \frac{231}{32} = \frac{745}{32} .$$

Since the proof of Thm. 19.4 is quite challenging, involving specialized techniques of analytic combinatorics, we defer it until §19.5.

Using the Theorem 19.4 recurrence to compute $\text{cap}_b(n)$

Theorem 19.4 applied repeatedly enables us to express $\text{cap}_b(n)$, for any b , in terms of $\text{cap}_2(n)$ and $\text{cap}_1(n)$. For instance we get

$$\begin{aligned} \text{cap}_4(n) &= \text{cap}_{4-1}(n) + \frac{n}{4-2} \text{cap}_{4-2}(n) \\ &= \text{cap}_{3-1}(n) + \frac{n}{3-2} \text{cap}_{3-2}(n) + \frac{n}{2} \text{cap}_2(n) \\ &= \left(\frac{n}{2}+1\right) \text{cap}_2(n) + n \text{cap}_1(n) . \end{aligned}$$

But since $\text{cap}_1(n) = 1$ for all n , this means that every $\text{cap}_b(n)$ can be expressed just in terms of $\text{cap}_2(n)$, through formulae like the following:

$$\begin{aligned}\text{cap}_3(n) &= \text{cap}_2(n) + n , \\ \text{cap}_4(n) &= \left(\frac{n}{2}+1\right)\text{cap}_2(n) + n , \\ \text{cap}_5(n) &= \left(\frac{5n}{6}+1\right)\text{cap}_2(n) + \frac{n^2}{3} + n , \\ \text{cap}_6(n) &= \left(\frac{n^2}{8}+\frac{13n}{12}+1\right)\text{cap}_2(n) + \frac{7n^2}{12} + n , \\ &\vdots\end{aligned}$$

Note that computing $\text{cap}_b(n)$ using our earlier Def. 19.2 would require summing a total of $\binom{n+b-1}{n}$ terms. In particular, calculating $\text{cap}_6(100)$ requires summing almost 100 million terms. But, using the equation above, $\text{cap}_6(100)$ can be computed by summing just 101 terms for $\text{cap}_2(100)$ and evaluating some simple polynomials. We find experimentally that the time to compute $\text{cap}_6(100)$ in Maple is reduced from 20 minutes to about 5 milliseconds, a speed-up by a factor of more than 200,000.

In spite of those positive results, we are still faced with the problem that computing $\text{cap}_2(n)$ from (19.2) becomes expensive as n becomes large. For instance, direct computation of $\text{cap}_2(50000)$ in Maple using exact rational arithmetic takes almost 2 hours.

For that reason, we next give results allowing $\text{cap}_2(n)$ to be approximated very accurately and efficiently.

Results about $\text{cap}_2(n)$

It turns out that $\text{cap}_2(n)$ has a remarkable relationship to Ramanujan's celebrated Q -function, which is defined for $n \geq 1$ by

$$\begin{aligned}Q(n) &= 1 + \frac{n-1}{n} + \frac{n-1}{n} \cdot \frac{n-2}{n} + \frac{n-1}{n} \cdot \frac{n-2}{n} \cdot \frac{n-3}{n} + \dots \\ &= \sum_{k=1}^n \frac{n!}{n^k(n-k)!} .\end{aligned}$$

We have the following theorem.

Theorem 19.5 For all $n \geq 1$ we have $\text{cap}_2(n) = Q(n) + 1$.

□

Again we defer the proof to §19.5.

An amazing consequence of Thm. 19.5 is that $\text{cap}_2(n)$ has a significance unrelated to information leakage: it is also the *expected number of people needed in order to find two having the same birthday, when n is the number of days in a year*. (It is well known for example that $\text{cap}_2(365) \approx 24.616586$.)

Of more practical significance, asymptotic approximations for $Q(n)$ are known, and those (by adding 1 to them) work for $\text{cap}_2(n)$ as well. For instance, from Eqn. (2.6) of Knuth and Pittel (see Chapter Notes) we have the following theorem.

Theorem 19.6

$$\text{cap}_2(n) = \sqrt{\frac{\pi n}{2}} + \frac{2}{3} + \frac{1}{12}\sqrt{\frac{\pi}{2n}} - \frac{4}{135n} + O(n^{-3/2}) .$$

□

This theorem allows $\text{cap}_2(n)$ –and, using Thm. 19.4, every $\text{cap}_b(n)$ – to be approximated very efficiently, with a relative error that tends to 0 as n goes to infinity.

For security analysis, however, an *approximation* to $\text{cap}_2(n)$ like Thm. 19.6 is not quite satisfactory, since for any particular value of n we do not know how big the $O(n^{-3/2})$ term might be. Hence an *upper bound* would be preferable.

As a final major result, whose proof is again deferred to §19.5, an assertion due to Ramanujan can be applied to prove such an upper bound.

Theorem 19.7

For all $n \geq 1$,

$$\text{cap}_2(n) < \sqrt{\frac{\pi n}{2}} + \frac{2}{3} + \frac{1}{12}\sqrt{\frac{\pi}{2n}} .$$

Note that Thm. 19.7 gives an upper bound that is also asymptotically correct, since it is the result of truncating the expansion in Thm. 19.6 after its third term.

Finally, we mention that techniques similar to those used in proving Thm. 19.7 allow us also to prove an asymptotically correct *lower bound*.

Theorem 19.8 For all $n \geq 1$,

$$\text{cap}_2(n) > \sqrt{\frac{\pi n}{2}} + \frac{2}{3} + \frac{1}{12}\sqrt{\frac{\pi}{2n}} - \frac{4}{135n} .$$

□

This lower bound tells us that the upper bound given by Thm. 19.7 on $n = 10^6$, namely

$$\text{cap}_2(10^6) < 1253.98090843 ,$$

is correct to 11 significant digits.

Applying Theorem 19.3 in security analysis

The analytic results about $\text{cap}_b(n)$ now enable us to assess the extent to which Thm. 19.3 improves on Thm. 19.1 in security analysis.

First, we can compare the leakage bounds *asymptotically*. If we regard b as a constant, then the old Thm. 19.1 bound of $\binom{n+b-1}{n}$ is asymptotically $O(n^{b-1})$, while (as can be seen inductively from Thm. 19.4) the new Thm. 19.3 bound on $\text{cap}_b(n)$ is asymptotically $O(n^{b-1/2})$. Thus Thm. 19.3 tells us that the maximum leakage of $C^{(n)}$ is asymptotically only the *square root* of the previous Thm. 19.1 bound.

More concretely, consider an n -observation timing attack against a blinded crypto-system implemented with bucketing. Suppose that, based on the key length we are using, we decide that we can tolerate multiplicative Bayes leakage of at most 10^{26} .

If we want to be secure against $n = 10^9$ timing observations, then the bound from Thm. 19.1 suggests that we cannot use more than 3 buckets, because when $b = 4$ it gives maximum leakage somewhat over our limit:

$$\binom{10^9+4-1}{10^9} > 1.666666 \times 10^{26} .$$

In contrast, Thm. 19.3 tells us that $b = 4$ is actually fine:

$$cap_4(10^9) < 1.981797 \times 10^{13} .$$

This also illustrates that the new bound is indeed roughly the *square root* of the previous bound.

The improved leakage bound from Thm. 19.3 gives us a number of ways that we can modify b and n , while staying below the leakage limit of 10^{26} . If we just want to increase the performance of the cryptosystem, we can keep $n = 10^9$ and increase the number of buckets to $b = 7$:

$$cap_7(10^9) < 6.667823 \times 10^{25} .$$

Or we can increase the number of buckets to $b = 5$ and also increase the number of timing observations to $n = 10^{13}$, allowing us to choose fresh keys much less frequently:

$$cap_5(10^{13}) < 3.333337 \times 10^{25} .$$

Or we can use $b = 6$ and $n = 10^{10}$:

$$cap_6(10^{10}) < 1.566710 \times 10^{24} .$$

Finally, we note that computing all of those $cap_b(n)$ values using Thm. 19.4 and Thm. 19.7 takes an insignificant amount of time in Maple (just a few milliseconds).

19.5 Analytic proofs

In this section we give the proofs deferred from §19.4.

The recurrence of Thm. 19.4 does not seem amenable to proof by elementary techniques. Some progress can be made, however, by focusing on the *columns* of the table in Fig. 19.3, rather than the *rows*. That is, we fix n and view $cap_b(n)$ as a *function of b* . Elementary calculations then let us determine these functions for small values of n :

$$\begin{aligned} cap_b(0) &= 1 , \\ cap_b(1) &= b , \\ cap_b(2) &= \frac{b^2 + 3b}{4} , \\ cap_b(3) &= \frac{b^3 + 9b^2 + 17b}{27} , \\ cap_b(4) &= \frac{b^4 + 18b^3 + 95b^2 + 142b}{256} , \\ cap_b(5) &= \frac{b^5 + 30b^4 + 305b^3 + 1220b^2 + 1569b}{3125} , \\ cap_b(6) &= \frac{b^6 + 45b^5 + 745b^4 + 5595b^3 + 18694b^2 + 21576b}{46656} , \\ &\vdots \end{aligned}$$

One benefit of those polynomials is that they let us verify straightforwardly the Thm. 19.4 recurrence for $0 \leq n \leq 6$ and all $b \geq 3$.

More significantly, we can observe a pattern: for fixed n , we find that $\text{cap}_b(n)$ can be written as n^{-n} times a degree- n polynomial in b with integer coefficients. Some searching for those coefficients in the *On-Line Encyclopedia of Integer Sequences* leads us to sequence A060281 and the *tree polynomials* $t_n(b)$ studied by Knuth and Pittel. Remarkably, the first few tree polynomials are

$$\begin{aligned} t_0(b) &= 1 , \\ t_1(b) &= b , \\ t_2(b) &= b^2 + 3b , \\ t_3(b) &= b^3 + 9b^2 + 17b , \\ t_4(b) &= b^4 + 18b^3 + 95b^2 + 142b . \end{aligned}$$

This coincidence suggests that perhaps

$$\text{cap}_b(n) = n^{-n} t_n(b) \quad (19.3)$$

holds for all $n \geq 0$. It also suggests that it could be fruitful to investigate $\text{cap}_b(n)$ using techniques of *analytic combinatorics*. We do this next.

The tree polynomials

Here we review the theory of tree polynomials as developed by Knuth and Pittel. The starting point is *Cayley's tree function* $T(z)$, which is the formal power series⁵

$$T(z) = \sum_{n=1}^{\infty} n^{n-1} \frac{z^n}{n!} . \quad (19.4)$$

In fact $T(z)$ satisfies the following equation due originally to Eisenstein:

$$T(z) = z e^{T(z)} . \quad (19.5)$$

Hence, if we take derivatives with respect to z , we get

$$T'(z) = e^{T(z)}.1 + z \cdot e^{T(z)}.T'(z) ,$$

which implies

$$T'(z)(1 - z e^{T(z)}) = e^{T(z)} .$$

Now, using (19.5) again, we get

$$T'(z) = \frac{T(z)}{z(1 - T(z))} , \quad (19.6)$$

so showing that $T'(z)$ can be expressed in terms of $T(z)$.

Next, the *tree polynomials* $t_n(b)$ are defined implicitly from $T(z)$ by the equation

$$\left(\frac{1}{1 - T(z)} \right)^b = \sum_{n=0}^{\infty} t_n(b) \frac{z^n}{n!} . \quad (19.7)$$

⁵ A *formal power series* $\sum_{n=0}^{\infty} c_n z^n$ can be understood simply as a notation for an infinite sequence of coefficients $[c_0, c_1, c_2, \dots]$, where two such series are equal iff their corresponding coefficients are equal. Formal power series are *not* to be evaluated with a numerical value for the formal symbol z , so questions of convergence are irrelevant. But, as shown by Niven [15], formal power series can in fact be manipulated validly as if they were power series, using the usual operations (addition, subtraction, multiplication, division, differentiation, exponentiation, and functional composition).

If we let $[z^n]f(z)$ denote the coefficient of z^n in the Taylor expansion of $f(z)$, then (19.7) is equivalent to

$$[z^n] \left(\frac{1}{1 - T(z)} \right)^b = \frac{t_n(b)}{n!} . \quad (19.8)$$

In fact, we can show that the tree polynomials satisfy a recurrence exactly analogous to Thm. 19.4. The argument starts, for $b \geq 3$, by replacing b with $b-2$ in (19.7) to get

$$\sum_{n=0}^{\infty} t_n(b-2) \frac{z^n}{n!} = \frac{1}{(1 - T(z))^{b-2}} .$$

Taking derivatives on both sides with respect to z , we get

$$\sum_{n=0}^{\infty} n t_n(b-2) \frac{z^{n-1}}{n!} = \frac{d}{dz} \frac{1}{(1 - T(z))^{b-2}} .$$

Hence, multiplying both sides by z , we have

$$\begin{aligned} \sum_{n=0}^{\infty} n t_n(b-2) \frac{z^n}{n!} &= z \frac{d}{dz} (1 - T(z))^{2-b} \\ &= z(b-2)(1 - T(z))^{1-b} T'(z) \\ &= z(b-2)(1 - T(z))^{1-b} \frac{T(z)}{z(1 - T(z))} \\ &= \frac{(b-2)T(z)}{(1 - T(z))^b} . \end{aligned}$$

Hence we get

$$\begin{aligned} \sum_{n=0}^{\infty} \frac{n}{b-2} t_n(b-2) \frac{z^n}{n!} &= \frac{T(z)}{(1 - T(z))^b} \\ &= \frac{1 - (1 - T(z))}{(1 - T(z))^b} \\ &= \frac{1}{(1 - T(z))^b} - \frac{1}{(1 - T(z))^{b-1}} . \end{aligned}$$

Finally, by equating the corresponding coefficients of the formal power series, we get the desired recurrence for the tree polynomials:

$$t_n(b) = t_n(b-1) + \frac{n}{b-2} t_n(b-2) . \quad (19.9)$$

Relating $\text{cap}_b(n)$ and $t_n(b)$

Given (19.9), we can prove Thm. 19.4 by proving (19.3). The crucial idea is to use an expansion of $1/(1-T(z))$ as follows:

$$\begin{aligned} \frac{1}{1 - T(z)} &= 1 + \frac{T(z)}{1 - T(z)} \\ &= 1 + z T'(z) \quad (\text{using (19.6)}) \\ &= 1 + z \sum_{n=1}^{\infty} n^n \frac{z^{n-1}}{n!} \quad (\text{using (19.4)}) \\ &= \sum_{n=0}^{\infty} n^n \frac{z^n}{n!} . \end{aligned}$$

Now, starting with (19.8), we can reason as follows:

$$\begin{aligned}
 t_n(b) &= n![z^n] \left(\frac{1}{1-T(z)} \right)^b \\
 &= n![z^n] \left(\sum_{n=0}^{\infty} n^n \frac{z^n}{n!} \right)^b \\
 &= n! \sum_{\substack{x_1, x_2, \dots, x_b \in \mathbb{N} \\ x_1+x_2+\dots+x_b=n}} \frac{x_1^{x_1} x_2^{x_2} \cdots x_b^{x_b}}{x_1! x_2! \cdots x_b!} \\
 &= n^n \text{cap}_b(n) .
 \end{aligned}$$

The second-to-last step follows because the z^n term in the product results from selecting terms from each of the b power series and multiplying them; the sum over x_1, x_2, \dots, x_b corresponds to all the possible ways of selecting those terms.

Hence we can rewrite (19.9) to

$$n^n \text{cap}_b(n) = n^n \text{cap}_{b-1}(n) + \frac{n}{b-2} n^n \text{cap}_{b-2}(n) ,$$

which (dividing both sides by n^n) finally completes the proof of Thm. 19.4.

Relating $\text{cap}_2(n)$ and $Q(n)$

In this case the *On-Line Encyclopedia of Integer Sequences* for the sequence $n^n \text{cap}_2(n)$ leads to sequence A063170 and the discovery that $\text{cap}_2(n) = Q(n) + 1$.

One proof of this relationship starts from Equation (2.8) of Knuth and Pittel [10] (which is also Equation (2.3) of Flajolet et al. [8]):

$$\sum_{n=1}^{\infty} n^{n-1} Q(n) \frac{z^n}{n!} = \ln \frac{1}{1-T(z)} .$$

Differentiating both sides with respect to z , we get

$$\begin{aligned}
 \sum_{n=1}^{\infty} n^n Q(n) \frac{z^{n-1}}{n!} &= (1-T(z)) (-1) (1-T(z))^{-2} (-1) T'(z) \\
 &= \frac{T''(z)}{1-T(z)} \\
 &= \frac{T(z)}{z(1-T(z))^2} ,
 \end{aligned}$$

where the last step uses (19.6). Hence, multiplying both sides by z we have

$$\begin{aligned}
 \sum_{n=1}^{\infty} n^n Q(n) \frac{z^n}{n!} &= \frac{T(z)}{(1-T(z))^2} \\
 &= \frac{1 - (1-T(z))}{(1-T(z))^2} \\
 &= \frac{1}{(1-T(z))^2} - \frac{1}{1-T(z)} .
 \end{aligned}$$

Using (19.7) and equating the corresponding coefficients of the formal power series, we have for $n \geq 1$ that

$$n^n Q(n) = t_n(2) - t_n(1) = n^n cap_2(n) - n^n cap_1(n) ,$$

which, because $cap_1(n) = 1$, implies that

$$cap_2(n) = Q(n) + 1 ,$$

so proving Thm. 19.5. (A second proof of that theorem, using complex analysis, is given in §19.6.)

Using an assertion by Ramanujan to bound $cap_2(n)$

The fact that $cap_2(n) = Q(n) + 1$ lets us obtain asymptotic approximations for $cap_2(n)$ from known asymptotic approximations for $Q(n)$. In this section, we use an assertion by Ramanujan to establish an *upper bound* on $cap_2(n)$.

In Ramanujan's first letter to Hardy, dated 16 January 1913, he asserted that

$$\frac{1}{2}e^n = 1 + \frac{n}{1!} + \frac{n^2}{2!} + \cdots + \frac{n^n}{n!}\theta(n) ,$$

where for all $n \geq 0$ we have

$$\theta(n) = \frac{1}{3} + \frac{4}{135(n+k(n))}, \quad \text{where } \frac{2}{21} \leq k(n) \leq \frac{8}{45} .$$

A complete proof of Ramanujan's assertion was finally given in 1995, by Flajolet, Grabner, Kirschenhofer, and Prodinger [8, Thm. 7]. Following that paper, let

$$R(n) = 1 + \frac{n}{n+1} + \frac{n^2}{(n+1)(n+2)} + \cdots .$$

Since we have

$$\frac{n!}{n^n} e^n = \sum_{k=0}^{\infty} \frac{n!}{k! n^{n-k}} ,$$

it is straightforward to verify that

$$Q(n) + R(n) = \frac{n! e^n}{n^n}$$

and

$$\theta(n) = \frac{1}{2}(R(n) - Q(n)) .$$

Hence we get

$$\begin{aligned} Q(n) &= \frac{n! e^n}{n^n} - R(n) \\ &= \frac{n! e^n}{n^n} - (2\theta(n) + Q(n)) \\ &= \frac{n! e^n}{n^n} - \frac{2}{3} - \frac{8}{135(n+k(n))} - Q(n) , \end{aligned}$$

which gives

$$Q(n) = \frac{1}{2} \frac{n! e^n}{n^n} - \frac{1}{3} - \frac{4}{135(n+k(n))}$$

and, using Thm. 19.5,

$$\text{cap}_2(n) = \frac{1}{2} \frac{n!e^n}{n^n} + \frac{2}{3} - \frac{4}{135(n+k(n))} .$$

Now Ramanujan's remarkable assertion

$$\frac{2}{21} \leq k(n) \leq \frac{8}{45}$$

enables us to establish upper bounds (and also lower bounds) on $\text{cap}_2(n)$. For it gives

$$\begin{aligned} \text{cap}_2(n) &\leq \frac{1}{2} \frac{n!e^n}{n^n} + \frac{2}{3} - \frac{4}{135(n+\frac{8}{45})} \\ &< \frac{1}{2} \frac{n!e^n}{n^n} + \frac{2}{3} - \frac{4}{135n} \left(1 - \frac{8}{45n}\right) \\ &= \frac{1}{2} \frac{n!e^n}{n^n} + \frac{2}{3} - \frac{4}{135n} + \frac{32}{6075n^2} . \end{aligned}$$

In 1955 Robbins strengthened Stirling's approximation, for $n \geq 1$, to

$$n! < \sqrt{2\pi} n^{n+1/2} e^{-n} e^{1/12n} ,$$

which gives

$$\text{cap}_2(n) < \sqrt{\frac{\pi n}{2}} e^{1/12n} + \frac{2}{3} - \frac{4}{135n} + \frac{32}{6075n^2} .$$

Next we obtain bounds for $e^{1/12n}$. For $n \geq 1$ we have

$$\begin{aligned} e^{1/12n} &= \sum_{j=0}^{\infty} \frac{1}{12^j n^j j!} \\ &= 1 + \frac{1}{12n} + \frac{1}{n^2} \sum_{j=2}^{\infty} \frac{1}{12^j n^{j-2} j!} \\ &\leq 1 + \frac{1}{12n} + \frac{1}{n^2} \sum_{j=2}^{\infty} \frac{1}{12^j j!} \\ &= 1 + \frac{1}{12n} + \frac{1}{n^2} \left(e^{1/12} - 1 - \frac{1}{12} \right) , \end{aligned}$$

which gives us

$$\text{cap}_2(n) < \sqrt{\frac{\pi n}{2}} + \frac{2}{3} + \frac{1}{12} \sqrt{\frac{\pi}{2n}} - \frac{4}{135n} + \left(e^{1/12} - 1 - \frac{1}{12} \right) \sqrt{\frac{\pi}{2n^3}} + \frac{32}{6075n^2} .$$

Because for $n \geq 1$ the sum of the last three terms is negative, we get

$$\text{cap}_2(n) < \sqrt{\frac{\pi n}{2}} + \frac{2}{3} + \frac{1}{12} \sqrt{\frac{\pi}{2n}} ,$$

proving Thm. 19.7. As already mentioned, that upper bound is also asymptotically correct, since it is the result of truncating the expansion in Thm. 19.6 after its third term.

Finally, we can similarly use Ramanujan's assertion that $k(n) \geq \frac{2}{21}$ to get the asymptotically correct lower bound

$$\text{cap}_2(n) > \sqrt{\frac{\pi n}{2}} + \frac{2}{3} + \frac{1}{12} \sqrt{\frac{\pi}{2n}} - \frac{4}{135n} ,$$

proving Thm. 19.8. (We omit the details.) Thus $\text{cap}_2(n)$ is tightly bounded both above and below.

19.6 Another proof of Theorem 19.5

In this section, we use complex analysis to give another proof of Thm. 19.5. The main job is to compute the coefficients of the Taylor expansion of $1/(1 - T(z))^b$. Let

$$f(z) := \left(\frac{1}{1 - T(z)} \right)^b .$$

By Cauchy's differential formula, around a circle γ containing a we have

$$f^{(n)}(a) = \frac{n!}{2\pi i} \oint_{\gamma} \frac{f(z)}{(z - a)^{n+1}} dz .$$

Since $f^{(n)}(0) = n![z^n]f(z)$, we have, abbreviating $T(z)$ to T ,

$$[z^n] \left(\frac{1}{1 - T} \right)^b = \frac{1}{2\pi i} \oint_{\gamma} \frac{dz}{z^{n+1}} \left(\frac{1}{1 - T} \right)^b .$$

By (19.5) we have

$$z = Te^{-T} ,$$

which implies that

$$\frac{dz}{dT} = e^{-T} - Te^{-T} = e^{-T}(1 - T)$$

and thus

$$dz = e^{-T}(1 - T) dT .$$

Therefore we have

$$\begin{aligned} [z^n] \left(\frac{1}{1 - T} \right)^b &= \frac{1}{2\pi i} \oint_{\gamma} \frac{e^{-T}(1 - T)dT}{T^{n+1}e^{-(n+1)T}} \left(\frac{1}{1 - T} \right)^b \\ &= \frac{1}{2\pi i} \oint_{\gamma} dT \frac{e^{nT}}{T^{n+1}} \left(\frac{1}{1 - T} \right)^{b-1} \\ &= \frac{1}{2\pi i} \oint_{\gamma} dT \frac{e^{nT}}{T^{n+1}} \sum_{k=0}^{\infty} \binom{b+k-2}{k} T^k \\ &= \frac{1}{2\pi i} \oint_{\gamma} dT e^{nT} \sum_{k=0}^{\infty} \binom{b+k-2}{k} \frac{1}{T^{n-k+1}} , \end{aligned}$$

where the next-to-last step uses the binomial expansion formula for $b > 1$. Now because

$$e^{nT} = \sum_{j=0}^{\infty} \frac{n^j T^j}{j!} ,$$

we get

$$[z^n] \left(\frac{1}{1 - T} \right)^b = \frac{1}{2\pi i} \oint_{\gamma} dT \sum_{j=0}^{\infty} \sum_{k=0}^{\infty} \binom{b+k-2}{k} \frac{n^j}{j! T^{n-k-j+1}} . \quad (19.10)$$

Now we apply a result of complex analysis, which says that for integer n ,

$$\oint_{\gamma} \frac{1}{z^n} dz = \begin{cases} 0 & \text{if } n \neq 1 \\ 2\pi i & \text{if } n = 1 \end{cases} .$$

Hence the only terms in the previous expression that contribute are those where $j = n-k$. And, since $j \geq 0$, we also have $k \leq n$. Hence we can continue (19.10) with

$$\begin{aligned}[z^n] \left(\frac{1}{1-T} \right)^b &= \frac{1}{2\pi i} \oint_{\gamma} \frac{1}{T} dT \sum_{k=0}^n \binom{b+k-2}{k} \frac{n^{n-k}}{(n-k)!} \\ &= \frac{1}{2\pi i} (2\pi i) \sum_{k=0}^n \binom{b+k-2}{k} \frac{n^{n-k}}{(n-k)!} \\ &= \sum_{k=0}^n \binom{b+k-2}{k} \frac{n^{n-k}}{(n-k)!}. \end{aligned}$$

Having done that, we recall that

$$cap_b(n) = n^{-n} t_n(b) = \frac{n!}{n^n} [z^n] \left(\frac{1}{1-T} \right)^b,$$

thereby obtaining a new expression for $cap_b(n)$ when $b > 1$: it is

$$cap_b(n) = \sum_{k=0}^n \binom{b+k-2}{k} \frac{n!}{n^k (n-k)!}. \quad (19.11)$$

Specializing to the case when $b = 2$, we again obtain Thm. 19.5:

$$cap_2(n) = \sum_{k=0}^n \frac{n!}{n^k (n-k)!} = 1 + Q(n).$$

Note finally that (19.11) lets *every* $cap_b(n)$ be computed via a sum of $n+1$ terms, which is useful when b is big and n is small. In fact, Maple recognizes (19.11) as the *generalized hypergeometric function* ${}_2F_0(b-1, -n; ; -\frac{1}{n})$ and can compute it efficiently for quite large n .

19.7 Chapter notes

This chapter is based on work by Smith and Smith [18].

Kocher's seminal work [11] demonstrated the effectiveness of timing attacks on RSA; there he also proposed the countermeasure of blinding. Brumley and Boneh [3] later showed that Kocher's attack remains effective over a network, where timing measurements are imprecise. The additional countermeasure of bucketing was proposed by Köpf and Dürmuth [12]; they also give experimental results about the overhead on 1024-bit RSA for different numbers of buckets.

The information-theoretic *method of types* is described in Chapter 11 of Cover and Thomas [5]. Theorem 19.1 is from Köpf and Smith [13] and Espinoza and Smith [7], and Thm. 19.3 was also proved independently by Goran Doychev and Boris Köpf (personal communication). The special case of Thm. 19.4 when $b = 3$, namely $cap_3(n) = cap_2(n) + n$, was discovered by Lacasse [14, p. 90] in his 2010 dissertation about machine learning but left as a conjecture. Since then, at least four proofs of this special case, using a variety of techniques, have been published by Younsi [20], Chen, Peng, and Yang [4], Sun [19], and Prodinger [16].

The theory of the tree polynomials is developed by Knuth and Pittel [10], who prove that they satisfy recurrence (19.9). Prodinger [16] proves Thm. 19.5, using the proof in §19.6, and proves (essentially) that $t_n(b) = n^n \text{cap}_b(n)$. Ramanujan's Q -function is studied by Flajolet, Grabner, Kirschenhofer, and Prodinger [8].

Analytic combinatorics is treated comprehensively by Flajolet and Sedgewick [9], and they discuss Cayley's tree function in Section II.5.1. In the expansion

$$T(z) = 1 \frac{z^1}{1!} + 2 \frac{z^2}{2!} + 9 \frac{z^3}{3!} + 64 \frac{z^4}{4!} + 625 \frac{z^5}{5!} \dots ,$$

the coefficients 1, 2, 9, 64, and 625 give the number of labeled, rooted, unordered trees of size 1, 2, 3, 4, and 5, respectively. Moreover, the Eisenstein equation

$$T(z) = ze^{T(z)} = z \left(\frac{T(z)^0}{0!} + \frac{T(z)^1}{1!} + \frac{T(z)^2}{2!} + \frac{T(z)^3}{3!} + \dots \right)$$

has a remarkable combinatorial interpretation, in which the initial z factor corresponds to the root, and the terms $\frac{T(z)^k}{k!}$ correspond to the number of ways of having k unordered subtrees.

The well-known Stirling approximation $(n/e)^n \sqrt{2\pi n}$ of n factorial dates from the 1730's, but was improved relatively recently by Robbins: in a three-page article [17], he added the multiplicative term $e^{1/12n}$ to give an upper bound rather than merely an approximation. (He also showed that adding $e^{1/(12n+1)}$ gives a lower bound, which we do not need but still is no less remarkable to have been found after more than 200 years.) The formula has been improved still further by others since.

Repeated-independent-runs channels have also been studied by Boreale, Pampaloni, and Paolini [1]. They prove that $\mathcal{ML}_1^X(\mathbb{D}, C^{(n)})$ converges asymptotically to the number of distinct rows of C , and they use the information-theoretic method of types to prove that the rate of convergence is exponential. The same authors [2] prove stronger rate bounds based on the minimum Chernoff information between the rows of C .

Espinosa and Smith [7] establish bounds on the multiplicative Bayes capacity of a number of different channel compositions. In addition to proving Thm. 19.1 for repeated independent runs, they also consider the case where n *different* channels are run on the same input X , producing a tuple of outputs. They show that the leakage in this case can be far greater than with repeated independent runs — indeed n channels, each having 2 columns, can be constructed such that their composition has multiplicative Bayes capacity of 2^n .

In the context of timing attacks on cryptosystems, Doychev and Köpf [6] establish leakage bounds for *unpredictability entropy*, which allows them to consider resource-bounded adversaries, in contrast with the information-theoretic adversaries considered here. An interesting question is whether the Thm. 19.3 leakage bound, proved here for information-theoretic adversaries, can be carried over to computationally bounded adversaries.

The online entries for sequences A060281 and A063170 were found at <https://oeis.org/A060281> and <https://oeis.org/A063170> respectively.

Bibliography

- [1] Boreale, M., Pampaloni, F., Paolini, M.: Asymptotic information leakage under one-try attacks. In: M. Hofmann (ed.) Proceedings of the 14th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS 2011), *Lecture Notes in Computer Science*, vol. 6604, pp. 396–410. Springer, Berlin (2011)
- [2] Boreale, M., Pampaloni, F., Paolini, M.: Quantitative information flow, with a view. In: V. Atluri, C. Diaz (eds.) Proceedings of the 16th European Symposium on Research in Computer Security (ESORICS 2011), *Lecture Notes in Computer Science*, vol. 6879, pp. 588–606. Springer, Berlin (2011)
- [3] Brumley, D., Boneh, D.: Remote timing attacks are practical. *Computer Networks* **48**(5), 701–716 (2005)
- [4] Chen, W.Y.C., Peng, J.F.F., Yang, H.R.L.: Decomposition of triply rooted trees. *The Electronic Journal of Combinatorics* **20**(2) (2013)
- [5] Cover, T.M., Thomas, J.A.: Elements of Information Theory. Wiley Series in Telecommunications and Signal Processing. Wiley-Interscience (2006)
- [6] Doychev, G., Köpf, B.: Rational protection against timing attacks. In: Proceedings of the 2015 IEEE 28th Computer Security Foundations Symposium (CSF 2015), pp. 526–536. IEEE, Los Alamitos (2015)
- [7] Espinoza, B., Smith, G.: Min-entropy as a resource. *Information and Computation* **226**, 57–75 (2013)
- [8] Flajolet, P., Grabner, P.J., Kirschenhofer, P., Prodinger, H.: On Ramanujan's Q -function. *Journal of Computational and Applied Mathematics* **58**(1), 103–116 (1995)
- [9] Flajolet, P., Sedgewick, R.: Analytic Combinatorics. Cambridge University Press, New York (2009)
- [10] Knuth, D.E., Pittel, B.: A recurrence related to trees. *Proceedings of the American Mathematical Society* **105**(2), 335–349 (1989)
- [11] Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: N. Koblitz (ed.) Proceedings of Advances in Cryptology (CRYPTO 1996), *Lecture Notes in Computer Science*, vol. 1109, pp. 104–113. Springer, Berlin (1996)

Bibliography

- [12] Köpf, B., Dürmuth, M.: A provably secure and efficient countermeasure against timing attacks. In: Proceedings of the 2009 IEEE 22nd Computer Security Foundations Symposium (CSF 2009), pp. 324–335. IEEE, Los Alamitos (2009)
- [13] Köpf, B., Smith, G.: Vulnerability bounds and leakage resilience of blinded cryptography under timing attacks. In: Proceedings of the 2010 IEEE 23rd Computer Security Foundations Symposium (CSF 2010), pp. 44–56. IEEE, Los Alamitos (2010)
- [14] Lacasse, A.: Bornes PAC-Bayes et algorithmes d'apprentissage. Ph.D. thesis, Université Laval, Québec (2010)
- [15] Niven, I.: Formal power series. *The American Mathematical Monthly* **76**(8), 871–889 (1969)
- [16] Prodinger, H.: An identity conjectured by Lacasse via the tree function. *The Electronic Journal of Combinatorics* **20**(3) (2013)
- [17] Robbins, H.: A remark on Stirling's formula. *The American Mathematical Monthly* **62**(1), 26–29 (1955)
- [18] Smith, D.M., Smith, G.: Tight bounds on information leakage from repeated independent runs. In: Proceedings of the 2017 IEEE 30th Computer Security Foundations Symposium (CSF 2017), pp. 318–327. IEEE, Los Alamitos (2017)
- [19] Sun, Y.: A simple proof of an identity of Lacasse. *The Electronic Journal of Combinatorics* **20**(2) (2013)
- [20] Younsi, M.: Proof of a combinatorial conjecture coming from the PAC-Bayesian machine learning theory (2012). Available at <https://arxiv.org/abs/1209.0824>



Chapter 20

Defense against side channels

This chapter, like Chap. 19, considers timing attacks against *RSA* decryption — but from a different perspective. Recall that in Chap. 19 decryption is treated as a *black box* protected only by the “external” defenses of blinding and bucketing, and under a weak observational model that allows the adversary to see only the total amount of time required by a decryption, as might be appropriate if the decryption were done remotely. Moreover, bucketing allows that time to be observed only coarsely (within b buckets), while blinding “de-correlates” multiple timing observations by using a randomly chosen ciphertext in each decryption, so that multiple timing observations can be modeled as a repeated-independent-runs channel.

In this chapter, in contrast, we consider the actual decryption program and we assume the stronger observational model from Chap. 14, which in particular allows the adversary to observe the sequence of branches taken by the program as it executes.¹ We consider “internal” defenses, and we compute the information leakage caused by a single run, so that the selection of program parameters can be evaluated for their effectiveness.

20.1 Evaluating a defense against side channels

Several important factors come into play when considering implementations of cryptography. If a decryption (or encryption) algorithm takes too long then it is considered to be too impractical to use; on the other hand, if a fast algorithm has the effect of leaking so much information that the claimed security properties are significantly weakened, then the purpose of the cryptography is defeated.

Exponentiation, used in many cryptographic implementations, is one such example. A number of side-channel attacks have been shown to be effective, especially when the cryptography is implemented on smart cards, where for example power- or timing analysis can be used to determine some proportion of password bits, making a subsequent brute-force attack effective in determining the entire password. A typical example is *RSA* encryption, a form of public-key cryptography, where the plaintext (treated as a number) is raised to a known exponent, the public key, in order to make the ciphertext; but the arithmetic is arranged so that raising the ciphertext subsequently to a secret exponent, the private key, gives the plaintext back again. Number-theoretic arguments support the view that it is computationally difficult to perform the decryption (by any means) even if both the ciphertext and the public key are known.

¹ While such an observation model might appear pessimistic, in computer security such pessimism has often turned out to be warranted.

Initial values B for base, the ciphertext; E for exponent, the private key.

Final value p for power, the decrypted plaintext.

Precondition is B,E $\geq 0,0$.

All variables are hidden.²

```

VAR p,B,E           - Global variables, none of them observable.
{ VAR b,e:= B,E      - Local variables.
  p:= 1
  WHILE e $\neq 0$  {
    → IF (e mod 2) $\neq 0$  THEN p:= p*b
    b,e:= b2,e $\div 2$       - Invariant is p $\times (b^e) = B^E$ 
    }                      - Is e odd?
}                      - Assume no side channel here.
Now p=BE — but what has an adversary learned about the initial E ?

```

Although our hidden global state comprises p,B,E we concentrate only on the secrecy of E.

Figure 20.1 All secrets leaked by fast exponentiation

One attack vector on *RSA* is for the adversary to observe the decryption process and, from that, determine the private key — or perhaps learn enough about it that other formerly infeasible attacks become feasible. In this chapter we concentrate on that example.

The well-known fast exponentiation algorithm is shown in Fig. 20.1. The base B could be the ciphertext in *RSA* and the exponent E could be the private key, in which case the algorithm would be performing decryption $p := B^E$ so that p is the plaintext. The adversary tries to learn about E by observing the algorithm in operation.

We investigate the information that potentially leaks through the ability of timing analysis to distinguish “long-integer squaring” from “multiplication” operations. That means that if it becomes known which of the operations took place, it is possible to deduce a great deal about the values of the inputs. As usual we assume that the program code is public and that the program’s control flow through the code is observable. All variables however are private, not observable by the adversary. Thus it is only the timing analysis that provides information to the adversary about the unobservable values; in particular, she learns which of the two conditional branches in Fig. 20.1 is taken (as well as being able to observe each test for (re-)entry into the loop body). The former, given the code, is equivalent to determining whether the current value of e is divisible by 2 on each iteration. Since e is successively all bitwise prefixes of the original E, the result is that she can discover all of E, one bit at a time (from least- to most significant).

Although we have already given program-algebraic reasons for conditionals’ leaking, in this specific example we note that it can also be seen as a cautious response to the problem here, and more generally to the well-known issue that the two branches might take different times to execute and that that could be detectable via a side channel. The issue is sometimes referred to as “Don’t branch on high.” — and earlier we called it “super-implicit flow”. Even if that problem is reduced (e.g. by balancing

² It does not matter for our analysis whether B is observable: our results would be the same. In fact in *RSA*, as the ciphertext, it normally *would* be observable.

```

VAR p,B,E           - Global variables, none of them observable.
{ VAR b,e:= B,E      - Local variables.
  p:= 1
  WHILE e≠0 {
    VAR d∈ 2⊕3⊕5      - Invariant is p×(be) = BE.
    VAR r:= e mod d      - Uniform choice from {2,3,5}.
    side channel → IF r≠0 THEN p:= p*br      - Does d divide e?
    b,e:= bd,e÷d      - Assume no side channel here.
  }
}
    
```

What has an adversary learned about the initial E this time?

This is the same as Fig. 20.1 except that it uses a randomly selected divisor d. It therefore takes longer on average; but it reduces the risk from the side channel.

Figure 20.2 Defense against side-channel analysis in exponentiation

the branches), who is to say that another method might not be found in the future to detect which branch was taken?

A more durable defense against this problem, which we consider here, can balance the needs of performance and secrecy: acknowledge that the conditional's being leaked is inevitable, if not now then at some time in the future, and therefore reduce drastically what an attacker would be able to deduce from that. Thus rather than using a bucketing approach, as described in Chap. 19, randomization can be applied to obfuscate the information leaking through the side channel; instead of using 2 as the base for the division, on every iteration independently a (hidden) random choice is made between some small number of potential divisors. Thus although the side channel still leaks information, it is harder for the attacker to link that information to the secret E.

Fig. 20.2 sets out an algorithm that randomizes the choice of divisor each time round the loop; it follows the same basic pattern as Fig. 20.1 except that it makes a random choice of a divisor d instead of (effectively) using d=2 every time. That makes it harder for the adversary to determine what the current value of e is divisible by.

20.2 QIF exploration of the fast-exponentiation algorithm

Using the QIF-semantics described in Chap. 13, to interpret the program code, we could in principle investigate the difference between the programs of Figs. 20.1 and 20.2 in terms of the hyper-distributions they produce for given prior distributions. Here however we will look at some small, concrete examples to get a general sense of the effect of the probabilistic obfuscation that the use of d introduces.

Using a prototype implementation of our small QIF-aware language, we therefore can investigate the effect of running the above small programs on various inputs. In the two runs below we choose the private key E uniformly from {0..15} — that is, E is a 4-bit exponent. In the first case —always divide by 2, as in Fig. 20.1— we show in Fig. 20.3 the output hyper on E alone. The first column (all 1/16) shows that there are sixteen possible inners distributed by the outer just as the hidden input E was, that is uniformly. The second and third columns show the inners, first the value of E

outer probabilities	inner distributions
$E =$	
1/16 →	↓ ↓ with probability
1/16 →	$\begin{array}{c c} 0 & 1 \\ \hline 1 & 1 \end{array}$
1/16 →	$\begin{array}{c c} 2 & 1 \\ \hline 3 & 1 \end{array}$
1/16 →	$\begin{array}{c c} 4 & 1 \\ \hline 5 & 1 \end{array}$
1/16 →	$\begin{array}{c c} 6 & 1 \\ \hline 7 & 1 \end{array}$
1/16 →	$\begin{array}{c c} 8 & 1 \\ \hline 9 & 1 \end{array}$
1/16 →	$\begin{array}{c c} 10 & 1 \\ \hline 11 & 1 \end{array}$
1/16 →	$\begin{array}{c c} 12 & 1 \\ \hline 13 & 1 \end{array}$
1/16 →	$\begin{array}{c c} 14 & 1 \\ \hline 15 & 1 \end{array}$

Each of these 16 inners is a point distribution centered on a different value of E. The outer distribution on those inners is uniform, induced by the uniform prior.

The hyper above describes the adversary's knowing certainly what the value of E is, after execution; the outer probabilities reflect that since she does not know the prior, she does not know "which value of E she will know." But, whatever value E has, she is certain (inner probability 1) to learn it.

Figure 20.3 Output hyper on 4-bit uniformly distributed E when divisor is 2 every time

and then the probability the inner associates with it. In this case all the inners are point distributions: in each case the adversary will know for certain (probability 1) what the private key E was. That is, with probability $1/16$ she will know for certain (i.e. with probability 1) that it was 0, with probability $1/16$ that it was 1, with $1/16$ it was 2 etc. If the prior distribution were different, then the outer of the hyper would be correspondingly different: but in each case the second column, the inners, would still be "probability 1" throughout. In our semantics, in fact the program is equivalent to `PRINT E` — and you can't get leakier than that.

In the second case, we show in Fig. 20.4 a portion of the output hyper when the divisor is chosen uniformly from $\{2, 3, 5\}$ (as in Fig. 20.2) and again the prior on E is uniform. We now look at it in detail.

† The adversary knows that E must be in $\{0..15\}$, that the divisors must be in $\{2, 3, 5\}$, and the rubric in Fig. 20.4 identifies the case in which she has observed a pattern of IF-conditional resolutions, during the run of the program, that are not possible unless in fact $1 \leq E \leq 4$. The hyper abstracts from *what* pattern of branching she saw, and *how* she used it to come to that conclusion — but it is certain that she could have.

outer probabilities	inner distributions
$\frac{1}{16} \rightarrow$	$E =$ $\downarrow \quad \downarrow \text{ with probability}$
$\frac{7}{48} \rightarrow$	$\begin{array}{c c} 0 & 1 \\ \hline \end{array}$
$\frac{5}{36} \rightarrow$	$\begin{array}{c c} 1 & \frac{3}{7} \\ 2 & \frac{2}{7} \\ 3 & \frac{1}{7} \\ 4 & \frac{1}{7} \\ \hline \end{array}$
$\frac{41}{144} \rightarrow$	$\begin{array}{c c} 2 & \frac{3}{20} \\ 3 & \frac{3}{20} \\ 4 & \frac{1}{10} \\ 5 & \frac{3}{20} \\ 6 & \frac{3}{20} \\ 8 & \frac{1}{20} \\ 9 & \frac{1}{20} \\ 10 & \frac{1}{10} \\ 12 & \frac{1}{20} \\ 15 & \frac{1}{20} \\ \hline \end{array}$
$\frac{1}{432} \rightarrow$	$\begin{array}{c c} 3 & \frac{3}{41} \\ 4 & \frac{3}{41} \\ 5 & \frac{5}{41} \\ 6 & \frac{3}{41} \\ 7 & \frac{6}{41} \\ 8 & \frac{5}{41} \\ 9 & \frac{4}{41} \\ 10 & \frac{1}{41} \\ 11 & \frac{3}{41} \\ 12 & \frac{2}{41} \\ 13 & \frac{3}{41} \\ 14 & \frac{3}{41} \\ \hline \end{array}$
\dots (8 other inners) \dots	
$\frac{1}{432} \rightarrow$	$\begin{array}{c c} 12 & 1 \\ \hline 13 & 1 \\ \hline \end{array}$
$\frac{1}{432} \rightarrow$	$\begin{array}{c c} 14 & 1 \\ \hline 15 & 1 \\ \hline \end{array}$

This inner is discussed at
† in the main text.

Values obtained by running Fig. 20.2's code through an interpreter for the QIF-aware programming language *Kuifje*. (See Chapter Notes §13.8, and those for this chapter.)

Figure 20.4 Output hyper (portion only) on 4-bit uniformly distributed E when divisor varies uniformly over 2,3,5

Taking as usual the most cautious approach, we assume therefore that she has. So even with purely “possibilistic” reasoning, the adversary learns that $1 \leq E \leq 4$.³ But now, quantitatively, she knows even more.

The adversary knows that in this case the input E was in fact uniformly distributed over $\{0..15\}$ (from the context) and that the divisors were uniformly distributed over $\{2, 3, 5\}$ (from the source code). She can then carry on with more sophisticated calculations to determine the actual probabilities of the inner — and they show that she can exploit her knowledge more effectively than by simply limiting her guesses to $1 \leq E \leq 4$. Using her knowledge of the inner, she can conclude that her best guess is then that $E=1$, because of the four possibilities in $\{1, 2, 3, 4\}$ it is the most likely to have produced this branching pattern in those circumstances. Moreover, the outer probability $7/48$ tells her that she should expect to see that in about 1 case out of 7.

In other circumstances —e.g. a different prior distribution on E , or a different distribution on the choice of d — she would still be able to make the qualitative deductions of the previous paragraph (i.e. that $1 \leq E \leq 4$). But the outer probability might change from $7/48$ to something else; and the associated inner might change too. Thus she might observe that branching pattern with a different probability; and, having observed it, she might find that there is different value in $\{1, 2, 3, 4\}$ that E is most likely to have. But she will still know that $1 \leq E \leq 4$.

And at the end of Fig. 20.4 we see —surprisingly— that it still might be possible for the adversary to know E with certainty even when the program is obfuscated this way — for example, with probability $1/432$ she will know for sure that $E=12$ (and similarly 13, 14, or 15). But in this example, the *a priori* probability of $E=12$ is (still) $1/16$, and so that undesirable situation occurs with probability only $(1/432)/(1/16) = 1/27$. That is, with probability $1 - 1/27 \approx 96\%$ it will not be possible to conclude that $E=12$; and that is because there are other inners that also have 12 in their support (for example the third and fourth shown). This is part of the value of *quantitative* analysis — without that calculation (of 96%) there would be no way of knowing whether the obfuscation of choosing a varying divisor was worth the performance penalty.

20.2.1 Cost/benefit analysis

A further benefit of quantitative reasoning is that a cost/benefit analysis is possible, from which one can decide perhaps that the system is secure because it is not worth attacking from the adversary’s point of view — because she has better targets. For that, we look at the adversary in more abstract terms.

It is not the private key E that the adversary wants, but rather the money she can get by using it. Since the posterior Bayes vulnerability over the whole hyper of Fig. 20.4, i.e. including the inners not shown, turns out to be $161/1296 \approx 1/8$, if the adversary’s profit for guessing correctly were say \$1, then her *expected* profit from an attack is about $\$1/8$, i.e. about 12 cents. But now —even more abstractly— we imagine that if she guesses incorrectly, she is caught in the act and is punished: the extra abstraction is then that we assign a notional cost to her of say \$5 for having to suffer that punishment. In that setting, then, one might imagine that she would never bother to try to guess the key: as we saw, her probability of guessing correctly is only about $1/8$, and thus of guessing incorrectly is $7/8$, giving an expected profit of $1/8 \times \$1 - 7/8 \times \5 , i.e. a *loss* of about \$4.25. If that *were* so then, since she is rational, indeed she would choose the action of not guessing at all: there is no point, for she will lose on average \$4.25 every time she does.

³ This qualitative reasoning was possible already (see Chap. 17, whose semantic domain was sets of sets rather than the present distributions of distributions).

But that reasoning is incorrect. Here is why.

Recall for example that with probability $1/432$ she will learn that E was 12 (and similarly for 13,14,15)⁴ — and in those cases, she *will* guess. With a bit of arithmetic, we capture the true scenario of gaining \$1 if the guess is correct and losing \$5 if the guess is incorrect by using a gain function similar to g_{tiger} from §3.2.5; it is

$$g(w, e) := \begin{cases} (+1 \text{ if } w=e \text{ else } -5) & \text{if } w \neq \perp \\ 0 & \text{if } w = \perp \end{cases}$$

with w in $\{0..15\}$ standing for the action “guess w ”, plus an extra value \perp standing for the action “don’t guess”. (Note that here although the gain function g can be negative, the vulnerability V_g cannot — the least it can be is zero.)

Now we find that V_g applied to the (completed) hyper of Fig. 20.4 is $^{31}/432$, giving the adversary a profit of 7 cents on average. Operationally, that is explained by noting that she will guess only in the cases where the inner is a singleton (total outer probability of $1/16 + 4 \times 1/432 \approx 0.07$); in the other cases she will not guess. If as suggested above we disallowed $E=0$ “for security reasons”, that expected profit for the cautious adversary would lose the $1/16$ term, dropping to about 1 cent. It is still not zero — but quite close.

Thus in practical terms we can conclude that the original exponentiation algorithm, with uniformly distributed prior E over $0..15$, offers a rational, cautious adversary of the kind described above an expected profit of \$1 for each attack — and she will attack every time because —knowing E for sure— she runs no risk of being penalized \$5. But in the obfuscated algorithm, with $E=0$ excluded, her expected profit is only 1 cent, not 1 dollar: a two orders of magnitude improvement in security, given the assumptions made in our example. If there are other public-key implementations where that same adversary enjoys a higher offer of say 5 cents for a correct guess —still quite low— we can nevertheless conclude that *this* implementation is fairly safe.

Such conclusions are part of what the “quantitative” in *QIF* brings us, the ability to *compare* security, to decide whether it’s worth it, and to choose between competing designs.⁵ Unless the adversary is irrational of course.

20.3 Chapter notes

The algorithm in Fig. 20.2 was proposed by Walter [3] as a way to balance the needs of performance of cryptographic computations and side-channel analysis. Walter describes several types of side channel vulnerabilities, including power analysis which averages over a number of exponentiation power traces, but also attacks that take advantage of repeated use of the same pre-computed multipliers during an individual exponentiation. Walter also provides an analysis of the performance of his algorithm and notes that a random choice over $\{2, 3, 5\}$ in the algorithm gives a good balance between security and performance.

The *QIF*-aware language used in this chapter has the semantics developed in Chap. 14. It was first implemented by Schrijvers based on an IFIP WG2.1 presentation of this *QIF* programming model (Morgan, Bennington Vt. in Dec. 2014); and it has been expanded and prepared for publication by Gibbons, McIver, Morgan and

⁴ There is also the case where $E=0$, in which case she learns that for sure — and guesses 0. In practical circumstances that choice of E would be forbidden; but to keep things simple in the presentation, we have left it in.

⁵ Recall the discussion in §3.5 about absolute vs. relative security, explaining why a home-owner locks his patio door even when it is simply a large glass pane, easily broken: if the burglar is rational, she will prefer an unlocked patio door.

Schrijvers [2] using Haskell’s monadic libraries for probability [1]. It was later made into a Haskell package by Marton Bognar, and improved in efficiency. (Gibbons gave it the name “*Kuifje*”, punning with the Dutch/Flemish name of the character Tintin and his quiff.)

It was the embedding of *Kuifje* into Haskell that calculated the hyper-distributions shown in Fig. 20.4. Although at the time of writing *Kuifje* is not able to deal with the large inputs required for realistic cryptographic computations, it is a useful experimental tool to explore the underlying risks associated with side channels in implementations of cryptography, and what can be done about them.

Bibliography

- [1] Erwig, M., Kollmansberger, S.: Probabilistic functional programming in Haskell. *Journal of Functional Programming* **16**, 21–34 (2006)
- [2] Gibbons, J., McIver, A., Morgan, C., Schrijvers, T.: Quantitative information flow with monads in Haskell. In: G. Barthe, J.P. Katoen, A. Silva (eds.) *Foundations of Probabilistic Programming*. Cambridge University Press, New York (2020)
- [3] Walter, C.D.: MIST: An efficient, randomized exponentiation algorithm for resisting power analysis. In: B. Preneel (ed.) *Topics in Cryptology - CT-RSA 2002, Proceedings of The Cryptographer’s Track at the RSA Conference, Lecture Notes in Computer Science*, vol. 2271, pp. 53–66. Springer, Berlin (2002)



Chapter 21

Multi-party computation: The Three Judges protocol

Multi-party computation is the solution to the situation in which two (or more) agents say A, B, \dots have private variables a, b, \dots and wish to calculate a public result $x = f(a, b, \dots)$ *without* revealing to any one of them more than is strictly necessary — in this case for example A must learn no more than what it can deduce from its variable a (which it knows anyway) and x (which is in the end known by everyone). One of the original (toy) examples was the two millionaires who wanted to find out which one of them was the richer, but without either one's finding out how much richer or poorer the other one was.

A simpler example however is the Lovers' protocol which we described in §15.6, where the two participants discover whether they are both in love, but —if they are not— the one *not* in love does not discover whether the other *is* in love. Unavoidable however is that if they are both in love, each discovers the other's love; and if they are not both in love but one of them is, it learns that the other is not.¹

The distinction between unavoidable information flow (the result) and avoidable flow (leaks) is an important characteristic of this paradigm.

The motivation for this chapter, the case study “The Three Judges protocol”, is however more than simply to showcase multi-party computation. It is to show how layers of abstraction can structure the development of what turns out to be a reasonably complicated protocol: here we will use the *specifications* of Oblivious Transfer (§15.5) and of The Lovers' protocol (§15.6) from Chap. 15. That is, the *implementation* we give here is in terms of the *specifications* of lower-level components: and that is enough. The monotonicity of our security-aware semantics with respect to refinement is the guarantee that because the information-flow properties of those lower-level specifications are refined by their *own* implementations, we can safely replace those specifications by their implementations without further reasoning.

¹ Recall from §15.3.3 that with multiple agents we sometimes use neuter gender for them all.

21.1 Introduction to The Three Judges

Three judges A, B, C are to give a majority decision, innocent or guilty, by exchanging messages but concealing their individual votes a, b, c respectively.²

The specification is as follows, using variables in $\{0, 1\}$ or isomorphically $\{\text{false}, \text{true}\}$ as convenient. It evaluates $a+b+c \geq 2$ atomically, that is, revealing nothing about a, b, c directly while, nevertheless, revealing the value of the overall judgment to everyone:

$\text{VIS}_A \ a; \text{VIS}_B \ b; \text{VIS}_C \ c$ $\text{PRINT } a+b+c \geq 2$	- Global variables, - of mixed visibility.
--	---

(21.1)

From the point of view of Judge A , for example, the above program becomes

$\text{VIS } a; \text{ HID } b, c$ $\text{PRINT } a+b+c \geq 2$	
--	--

(21.2)

because Judge A knows its own vote, but not the vote of B or C . Similar specializations apply to B and C . And to an onlooker in the gallery, say Agent X , the program would be

$\text{HID } a, b, c$ $\text{PRINT } a+b+c \geq 2$	
---	--

(21.3)

expressing that X can see the overall verdict, but none of its constituents.

Note that this specification *unavoidably* leaks some information to Judge A (and, by symmetry, to the other two), and it leaks to Agent X as well: for example when A judges “not guilty” and yet the defendant is found guilty by majority, Judge A has learned that both B, C must have judged “guilty” — and that is a release of information from them to it. Similarly on a guilty verdict overall, even Agent X knows that at least two judges voted “guilty” — but it does not know which. That is thus a release of information about all three judges to the general public. (Recall that in the introduction we mentioned that some leakage is unavoidable, if a mechanism is to serve any purpose at all.)

The above behavior, *by definition of refinement*, allows a similar behavior in the implementation, strictly speaking a declassification: but we need no special measures to deal with it. All of that is carried by the simple specification above: it does not need “fine print”³ to make its meaning precise — though of course extra explanation might make the specifier’s intention easier to understand. The specification determines both functional correctness —that the verdict is by majority— and *QIF* correctness: that only the overall verdict, and any information that can be deduced from that, is released.

Note also that our usual “open-source” convention that the code is known is here instantiated in the openness of the legal procedure being applied.

We now recall from Chap. 15 that the *location* of a variable has no direct impact on semantics (in our treatment here); but it does affect our judgment of what is directly executable and what is not. In particular, an expression is said to be *localized* just when all its variables are located at the same agent, and only localized expressions can be directly executed (by that agent, thus). Thus the expression $a+b+c \geq 2$ is not localized, in spite of its being meaningful in the sense of having a well defined value;

² Though this is similar to the (generalized) Dining Cryptographers (Exercise 21.4), it is more difficult: we do not reveal anonymously the total number of guilty votes as is done there; rather we reveal only a Boolean — whether the total is a majority.

³ “The manufacturer offers no guarantee that information will not be released if that information release is essential to the operation of the protocol.”

and it is precisely *because* it is not localized that we must develop the specification further. Assignment statements $a := \text{Exp}$, where a is in Judge A , say, and Exp is localized in Judge B , are implemented by B 's calculating Exp and then sending its value in a message to be received in Judge A 's variable a . Whether a special protocol is necessary for that transmission depends on the HID/VIS status of variable a .

21.2 Developing an implementation of the Three Judges

21.2.1 First attempt

We begin with an implementation attempt that fails, and we do that because our flawed approach –and failure– will illustrate two things. The first is that our model prevents incorrect developments, that is, it stops us from constructing implementations less secure than their specifications: arguably this “negative” aspect of a method is its most important property, since it would be trivial to describe a method that allowed secure refinements... and all others as well. The key to a successful development method is what it does *not* allow.

The second thing illustrated here is further, though still informal “*gedanken*” evidence that a conditional IF $\text{Exp} \dots$ should be considered to reveal its condition Exp implicitly. (Recall §13.4.3 on p. 241.) This (super-)*implicit flow* is a property forced upon us by our advocacy of program algebra and our use of compositionality: since the THEN- and the ELSE branch of a conditional can be developed differently *after* the conditional has been introduced, we must expect that those differences might later reveal to an attacker which branch is being executed (hence revealing the condition implicitly). We will see an example of that in this section, as part of the failure.

We start this first attempt with some arithmetic, that for Boolean a, b, c we have

$$a+b+c \geq 2 \quad \equiv \quad (b \vee c) \text{ if } a \text{ else } (b \wedge c) \quad ,^4$$

which suggests this first development step (in which we omit global declarations):

$$\begin{array}{lll} \text{PRINT } a+b+c \geq 2 & & \text{“our specification”} \\ \stackrel{?}{=} \text{IF } a \text{ THEN PRINT } b \vee c & & \text{“arithmetic above”} \\ & \text{ELSE PRINT } b \wedge c & . \end{array} \quad (21.4)$$

For the ELSE-part we must implement PRINT $b \wedge c$ as a protocol between B, C that does not unnecessarily reveal b to A, C or c to A, B ; and furthermore the result should be sent (securely) to A without either of B, C knowing unnecessarily what that result is. For that we adapt the Lovers' protocol from Prog. (15.24) to run between B and C , altering its PRINT-statements so that instead of broadcasting their values instead they send them securely to A . For the THEN part we can do the same, after rewriting $b \vee c$ as $\neg(\neg b \wedge \neg c)$ via De Morgan. (See Exercise 21.1.) Then we adjust the ELSE part to share as much code as possible with the THEN part (for which see Exercise 21.2), and factor out the common suffix $.$ All that gives finally

⁴ The if-expression here is ad-hoc syntax for mathematics; it's not what we use in the programming language.

```

{ VISA aB,aC; VISB b0,b1; VISC c0
  IF a THEN (b0⊻ b1)::= ~b; aB::= b1      - Message B→A .5
    ELSE (b0⊻ b1)::= b; aB::= b0      - Message B→A .
    c0::= bc                                - Oblivious Transfer B→C.   (21.5)
    aC::= c0                                - Message C→A .
    PRINT aB⊻ aC                         - Judge A announces verdict.
}

```

Note that the code contains Oblivious Transfer $B \rightarrow C$ as a sub-protocol.

But now we strike a problem: in order to go further with the development, we must allow Judge A somehow to arrange that B carries out either $(b_0 \nabla b_1) ::= \neg b$ or $(b_0 \nabla b_1) ::= b$, with that arrangement's depending on the value of A 's private variable a . Since B 's two potential computations are different, there is no way that can occur without B 's learning the value of a in the process: the code is already incorrect.

Thus we must abandon this attempt, and admit that the questionable step at (21.4) above was indeed wrong: and in hindsight it was obviously wrong even then. How can A ask B to do different computations, depending on a , without revealing a to B ? In order to allow us to develop distributed implementations, we make the (reasonable) assumption that each agent knows the code it is instructed to execute, even with those instructions coming possibly from another agent. In this case Agent B must execute two different code fragments depending on the value of a , which variable is supposed to be hidden from B .

That is one of the reasons our semantics recognizes super-implicit flow: the transformation of *Prog* into *IF Exp THEN Prog ELSE Prog* is not usually allowed, because it potentially leaks *Exp* even if the two branches are the same.

21.2.2 Second development attempt (sketch)

An “obvious” remedy for our first attempt, that in §21.2.1 Judge B is aware of which procedure it must follow and hence learns Judge A 's variable a , is to make B follow *both* procedures, speculatively: it does not know which one A will actually use.

The difficulty now moves to Judge A , who learns both $b \wedge c$ and $b \vee c$ from B . Although those two values do not (always) determine b and c themselves, the problem is that they do provide strictly more information to A than its knowing only a and $a+b+c \geq 2$ would have provided on their own.⁶ This is indeed a subtlety that so easily could be overlooked; but careful attention to refinement ensures that this “quick fix” fails also — as it should.

Our attention is therefore drawn to arranging for B and C together to do both of the two-party calculations, but then for A to get the results of only one of them. That leads naturally to the successful approach of the next section, a combination of two two-party computations (letting Judges B, C do both calculations) and two (more) Oblivious Transfers (letting Judge A learn about only one of them).⁷

⁵ Recall from §15.6 that $(b_0 \nabla b_1) ::= \dots$ chooses the two variables on the left uniformly so that their exclusive-or equals the expression on the right.

⁶ If a and $a+b+c \geq 2$ are both *false*, then Judge A concludes $\neg(b \wedge c)$, for which there are the three possibilities *false/false*, *true/false*, *false/true*. Judge A 's additionally knowing $b \vee c$ would eliminate at least one of those three.

⁷ The “more” refers to the fact that the two-party computations contain Oblivious Transfers.

21.2.3 Successful development

To repair the problem we encountered above we must arrange that Agents B, C as far as possible carry out procedures independent of A 's variable a , in particular so that calculations relating to $b \vee c$ and to $b \wedge c$ both occur, irrespective of which result A actually needs.

To achieve this we need a slightly more general form of two-party computation. We start from our specification (21.1) again, and introduce the specification of such a two-party conjunction but with its result variables made local so that the introduced code is equivalent to `SKIP` (as shown further below):

```
{ VISB b0; VISC c0           - Two-party conjunction,
  (b0∇c0):∈ b∧c          - but result is local, so discarded.
}
PRINT a+b+c≥2 ...
```

From Judge A 's point of view, the introduced statement is trivially equivalent to `SKIP`: all assignments are to local variables that A cannot see. From Judge B, C 's points of view, it is equivalent to `SKIP` because it is an instance of the Encryption Lemma: each of those two agents can see only one of the two variables assigned to, and so learns nothing about the expression $b \wedge c$.⁸

We should note here the surprising power of being able to reason at this abstract level: we can discuss the information leakage of a statement like $(b_0 \nabla c_0) : \in b \wedge c$ independently of knowing how, or indeed whether it can be implemented, given that the variables concerned, b and c and b_0 and c_0 , belong to different agents and are physically separated.

The statement $(b_0 \nabla c_0) : \in b \wedge c$ we have introduced in fact can be implemented: it is merely a more general form of two-party conjunction than the `PRINT b\wedge c` we illustrated earlier in §15.6 — that is because the conjunction is not actually revealed, not yet; instead it is split into two “shares”, one belonging to each judge B, C . Since each judge has only one share, the conjunction is not revealed at all at this stage. But those shares can be used as inputs to further two-party computations, while preserving security, and the contribution of the conjunction to a larger computation is revealed at a later point.

The extra generality introduced by the shares does not cause us extra work here, since we are using only the specification for our reasoning and that (we will see) suffices. When we come to *implement* the general two-party conjunction in more primitive terms, however, we then have further work to do. But it is work that we have done before, in §15.6.

⁸ The following bogus counter-argument is an example of what having a careful definition of equality and refinement helps us to avoid.

Judge B might know that its own b is *false*, and then perhaps receive *false* also into its b_0 . It can then conclude that c_0 is *false* — which is a leak since c_0 is supposed to be private to C , invisible to B .

In fact this is not a leak, because to consider it so we must refer to the *specification* of this fragment. But that is simply `SKIP` and there is no c_0 declared there: the revealed variable is local to the implementation only.

That is, publishing the value of a hidden variable declared only in the implementation might *look* like a leak in the conventional interpretation — consider { `HID c; ...; PRINT c` } for example — but it is actually a leak only if that variable c has come to contain information (via assignments say in the “...” portion) about other, more global hidden variables that were present in the specification, originally. Our semantics checks for that automatically.

With exactly the same reasoning as above we can introduce two-party disjunction and, with both conjunction and disjunction present, perform some reorganization:

```

... = { VISB b0; VISC c0; (b0∇c0) ∈ b ∧ c }           "disjunction as well"
{ VISB b1; VISC c1; (b1∇c1) ∈ b ∨ c }
PRINT a+b+c ≥ 2

= { VISB b0,b1; VISC c0,c1           "Reorganize declarations and scoping"
  (b0∇c0) ∈ b ∧ c
  (b1∇c1) ∈ b ∨ c
  PRINT a+b+c ≥ 2
}

= { VISB b0,b1; VISC c0,c1           "arithmetic"
  (b0∇c0) ∈ b ∧ c
  (b1∇c1) ∈ b ∨ c
  PRINT ba∇ca                         ← Only this changes here.
} ...
,

```

where the statement `PRINT ba∇ca` is an abbreviation for the deterministic internal choice `PRINT (IF a THEN b1∇c1 ELSE b0∇c0)`.

Now since $b_a \nabla c_a$ is revealed to everyone, and thus to A in particular, it does no harm first to capture that value in variables of A , and then to have Agent A reveal those instead:

```

... = { VISA aB,aC           "Introduce local private variables of A"
      VISB b0,b1
      VISC c0,c1

      (b0∇c0) ∈ b ∧ c
      (b1∇c1) ∈ b ∨ c
      (aB∇aC) ∈ ba∇ca           ← Two-party exclusive-or: still to do.
      PRINT aB∇aC
}
.

```

(21.6)

The point of using two variables a_B and a_C rather than one is to be able to split the transmission of information $B,C \rightarrow A$ into two separate Oblivious Transfers $B \rightarrow A$ and $C \rightarrow A$.

Thus the protocol boils down to three two-party computations: a conjunction $b \wedge c$, a disjunction $b \wedge c$, and an exclusive-or $a_B \nabla a_C$. The *rhs* of the last is actually within an internal choice on a , that is IF a THEN $b_1 \nabla c_1$ ELSE $b_0 \nabla c_0$ — the internal choice has moved from the `PRINT` to an assignment.

Only the exclusive-or is yet to do: it will become two more Oblivious Transfers, one from B to A and one from C to A , making four Oblivious Transfers in total.

21.2.4 Two-party exclusive-or

Our final step is to split the two-party exclusive-or into two separate assignments. This is achieved by introducing a local shared variable h that is visible to Judges B, C only, i.e. not to A , and encrypting both hidden variables with it. Variable h will be only temporary, however. Concentrating on the last statement at (21.6) just above, we proceed

```
( $a_B \nabla a_C$ ) :=  $b_a \nabla c_a$ 
= { VISBC  $h \in \text{true}_{1/2} \oplus \text{false}$ 
     $a_B := b_a \nabla h$ 
     $a_C := c_a \nabla h$ 
} ...
```

which is justified trivially for B, C since the only assignments of non-constants are to variables visible only to A . For A the justification comes from the use of classical equality reasoning: the effect of the two fragments above on the pair a_B, a_C is identical, and there are no overwritten visible values.

We will now show that in fact the extra variable h is not necessary: by absorbing it into earlier statements, and with some rearrangement of scopes we can rewrite our code at (21.6) as

```
... = { VISA  $a_B, a_C$                                 "Introduce local private variables of A"
        VISB  $b_0, b_1$ 
        VISC  $c_0, c_1$ 
        VISBC  $h \in \text{true}_{1/2} \oplus \text{false}$ 
( $b_0 \nabla c_0$ ) :=  $b \wedge c$ 
( $b_1 \nabla c_1$ ) :=  $b \vee c$ 
 $a_B := b_a \nabla h$ 
 $a_C := c_a \nabla h$ 
PRINT  $a_B \nabla a_C$ 
} ...
```

where we have simply moved the declaration and initialization of h right to the beginning. But with its being there, we now absorb it into the earlier two-party computations: we introduce temporary variables $b'_{\{0,1\}}$ and $c'_{\{0,1\}}$, which correspond to their unprimed versions except that they, too, are encrypted with h . That gives

```
... = { VISA  $a_B, a_C$ 
        VISB  $b_0, b_1, b'_0, b'_1$ 
        VISC  $c_0, c_1, c'_0, c'_1$ 
        VISBC  $h \in \text{true}_{1/2} \oplus \text{false}$ 
( $b_0 \nabla c_0$ ) :=  $b \wedge c$ ;  $b'_0, c'_0 := b_0 \nabla h, c_0 \nabla h$ 
( $b_1 \nabla c_1$ ) :=  $b \vee c$ ;  $b'_1, c'_1 := b_1 \nabla h, c_1 \nabla h$ 
 $a_B := b'_a$ 
 $a_C := c'_a$ 
PRINT  $a_B \nabla a_C$ 
} ...
```

where we have replaced the $b_a \nabla h$ and $c_a \nabla h$ at the end of the code with their simpler, primed versions where the encryption is built-in. Now we can rearrange the statements

```

... =      { VISA aB,aC; VISB b0,b1; VISC c0,c1           "consolidation"
          (b0∇c0) ∈ b ∧ c           ← Two-party conjunction. *
          (b1∇c1) ∈ b ∨ c           ← Two-party disjunction. *
          aB := ba                  ← Oblivious Transfer.
          aC := ca                  ← Oblivious Transfer.

          PRINT aB∇aC           ← The Verdict, pronounced by Judge A.
} .

```

* — Contains Oblivious Transfer.

Figure 21.1 High-level design for the Three Judges protocol

using h so that not only h but also the unprimed $b_{\{0,1\}}$ and $c_{\{0,1\}}$ become auxiliary; that is, we have this equality for the conjunction

$$= \quad (b'_0 \nabla c'_0) \in b \wedge c; \quad b'_0, c'_0 := b_0 \nabla h, c_0 \nabla h$$

and similarly for the disjunction. Removing the auxiliaries h , $b_{\{0,1\}}$ and $c_{\{0,1\}}$, and then applying a trivial renaming to get rid of the primes, we end up with the code of Fig. 21.1, which is precisely what we sought.

21.2.5 Summary

We have now successfully implemented the Three Judges protocol. Its specification is at (21.1). Its high-level design is given in Fig. 21.1: two two-party computations, then two Oblivious Transfers. (Each two-party computation itself contains a further Oblivious Transfer.)

In more detail: one of the judges, here A , acts as the coordinator. The other two judges B, C use two-party computations to calculate separate shares of the conjunction and the disjunction of their own verdicts b, c : but they do not learn what either of those 'junctions are, because they do not combine the shares: that will be done by A .

Judge A however does not get *all* the shares: instead it uses two more Oblivious Transfers, one from each of B, C , to collect the shares only for the conjunction or for the disjunction, depending on its own verdict a . It does not get the other 'junction, and it does not need it: what it does get is enough for it to announce the overall verdict.

In Fig. 21.2 we give the code for all that, with the (two) two-party computations instantiated but the (now four) Oblivious Transfers still left as specifications. Then in Fig. 21.3 we instantiate one of the (four) Oblivious Transfers.

- The Three Judges protocol.

```

{ VISA aB,aC; VISC c0,c1
  VISB b0,b1 ∈ true1/2⊕false, true1/2⊕false
    c0:= (IF c THEN b0∇ b0 ELSE b0)
    c1:= (IF c THEN ¬b1 ELSE b0∇ b1)
    aB:= (IF a THEN b1 ELSE b0)
    aC:= (IF a THEN c1 ELSE c0) } Four Oblivious Transfers.
    PRINT aB∇aC - Guilty, or not guilty?
}

```

We replace the two two-Party 'junctions by their implementations as Oblivious Transfers: each becomes two statements instead of one. The random initializations $b_{\{0,1\}}$ are then collected at the start.

The preservation of correctness is guaranteed by the monotonicity of the security semantics.

Figure 21.2 The Three Judges protocol, assuming Oblivious Transfers as primitives

Starting from Fig. 21.2, we replace the specification of the first of its four Oblivious Transfers $c_0 := (\text{IF } c \text{ THEN } b \nabla b_0 \text{ ELSE } b_0)$ by its implementation in more elementary terms (§15.5):

```

{ VISA aB,aC; VISB b0,b1; VISC c0,c1
  b0,b1 ∈ true1/2⊕false, true1/2⊕false

  { VISB m'0,m'1; VISC c',m'
    c' ∈ true1/2⊕false
    m'0,m'1 ∈ true1/2⊕false, true1/2⊕false
    m' := m'c'           - Done in advance by trusted third party.      †

    VISABC x,y0,y1   - Note these are visible to all three judges.
    x := c∇c'
    y0 := b0∇m'x
    y1 := b∇b0∇m'¬x
    c0 := yc∇m'   - Although yc is public, only Judge C knows m'.      ‡
  }

  c1 := (IF c THEN ¬b1 ELSE b∇b1)
  aB := (IF a THEN b1 ELSE b0)
  aC := (IF a THEN c1 ELSE c0)           } Three more Oblivious Transfers,
                                                each one to be
                                                expanded as above.

  PRINT aB∇aC
}
.
```

Each of the other three transfers would expand to a similar block of code, making about 40 lines of code in all. But it is not necessary to reason about those 40 lines at all, *not at all* — the reasoning finished with just the four (significant) lines of Fig. 21.1. The subsequent expansion leading to the above is just bookkeeping, whose soundness is guaranteed by the compositionality of the security semantics.

Note that aside from the statement marked † (and its three other instances within the three other, unexpanded Oblivious Transfers), all messages are wholly public because of their declarations; that is, all the privacy needed is provided already by the exclusive-or’ing with hidden Booleans, as ‡ shows.

The only private communications ($\dagger \times 4$) are done with the aid of a trusted third party. This party’s involvement occurs only *before* the protocol begins, and it is trusted not to observe any data exchanged subsequently between the agents; alternatively, the subsequent transfers can themselves be encrypted without affecting the protocol’s correctness. (A trusted third party without those limitations could implement trivially any protocol of this kind, simply by collecting the secret data, processing it, and then distributing the result.)

Figure 21.3 Three Judges protocol in elementary terms

21.3 Exercises

Exercise 21.1 (Lovers in action) Explain *exactly* how the Lovers' protocol of (15.24) can be instantiated to give the THEN branch of (21.4). (Use De Morgan, as suggested there.) \square

Exercise 21.2 Continue Exercise 21.1 by doing the steps from (21.4) to (21.5) in detail, with particular attention to the factoring out of the IF-branches' common suffix. \square

Exercise 21.3 Work out in detail the steps leading from Fig. 21.1 to Fig. 21.2. \square

Exercise 21.4 (The Dining Cryptographers) The Dining Cryptographers' protocol concerns three cryptographers sitting at lunch. When they ask for the bill, they are told it has already been paid — but by whom? They engage in the following protocol to find out whether one of them paid, or whether the *NSA* paid. (The generalized version of this problem has many cryptographers, and reveals how many paid.)

Each pair flips a fair coin that the remaining cryptographer cannot see; and they then announce whether they paid, each telling the truth however or lying depending on whether the two flipped coins they can see show the same or different faces respectively.

If an odd number say “I paid” then they know that one of them did; otherwise, they know it was the *NSA*.

Formalize a specification of this, and then develop an implementation in the style of this chapter. Discuss these issues with respect to your implementation:

- (a) Does your implementation work if more than one cryptographer paid?
- (b) Doesn't your implementation reveal to a cryptographer who did not pay whether one of the other two did pay, an information flow. Isn't it therefore insecure?

\square

21.4 Chapter notes

Multi-party protocols provide a way for independent agents to participate in a joint computation without revealing their inputs. Andrew Yao first demonstrated the idea on the Millionaires' problem [9, 2], and such protocols have become classic information-flow problems in that one must prove that just enough information is leaked, but no more information than necessary. In this chapter we have shown how our refinement relation is especially useful for that task. The specification is typically very simple to describe, because it is summarized by the publication of the final value; the bulk of the proof is then to use algebraic identities that preserve the information-sensitive refinement relation. This technique can be used on a number of well-known multi-party protocols, including Chaum's Dining Cryptographers [1, 4, 3]. (See Exercise 21.4.)

An important feature of the technique is the ability to use specifications within larger proofs. The Three Judges protocol (as does the Millionaires' problem) makes use of Rivest's implementation [8] of Rabin's Oblivious Transfer protocol [7]. This has the effect of reducing the proof to a collection of smaller information-flow problems, which all fit together through the assurance of compositionality. In fact this algebraic technique would not be possible at all without this compositional feature, required for example, by the implicit flow in IF *Exp* THEN *Prog* ELSE *Prog*.

In our related work for noninterference with demonic choice and without probability [4, 5, 6], we give further arguments for this point of view, explained directly in terms of program algebra. (See also Chap. 17.)

Much of the work in this chapter was done with Larissa Meinicke. The algebraic identities used have been proved (by hand); but a machine-verified proof of their correctness would be an essential part of using them in practice.

Bibliography

- [1] Chaum, D.: The Dining Cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology* **1**(1), 65–75 (1988)
- [2] McIver, A., Morgan, C.: *Sums and Lovers*: Case studies in security, compositionality and refinement. In: A. Cavalcanti, D.R. Dams (eds.) *Proceedings of the 16th International Symposium on Formal Methods (FM 2009)*, *Lecture Notes in Computer Science*, vol. 5850, pp. 289–304. Springer, Berlin (2009)
- [3] McIver, A., Morgan, C.: The thousand-and-one cryptographers. In: A.W. Roscoe, C.B. Jones, K.R. Wood (eds.) *Reflections on the Work of C. A. R. Hoare*. Springer, London (2010)
- [4] Morgan, C.: *The Shadow Knows*: Refinement of ignorance in sequential programs. In: T. Uustalu (ed.) *Proceedings of the International Conference on Mathematics of Program Construction (MPC 2006)*, *Lecture Notes in Computer Science*, vol. 4014, pp. 359–378. Springer, Berlin (2006)
- [5] Morgan, C.: *The Shadow Knows*: Refinement of ignorance in sequential programs. *Science of Computer Programming* **74**(8), 629–653 (2009)
- [6] Morgan, C.: Compositional noninterference from first principles. *Formal Aspects of Computing* **24**(1), 3–26 (2012)
- [7] Rabin, M.O.: How to exchange secrets by oblivious transfer. Tech. Rep. TR-81, Harvard University (1981). Available at <https://eprint.iacr.org/2005/187>
- [8] Rivest, R.L.: Unconditionally secure commitment and oblivious transfer schemes using private channels and a trusted initializer. Tech. rep., MIT (1999). Available at <http://theory.lcs.mit.edu/~rivest/Rivest-commitment.pdf>
- [9] Yao, A.C.: Protocols for secure computations. In: *Proceedings of the 1982 IEEE 23rd Annual Symposium on Foundations of Computer Science (FOCS 1982)*, pp. 160–164. IEEE Computer Society, Washington, DC (1982)



Chapter 22

Voting systems

Peaceful transfer of power is a core principle of modern democracies, and electoral commissions in particular must navigate the election process in a scenario where participants don't trust each other completely (or at all). What democracies aspire to is “free and fair elections”; what election commissions can deliver is an adherence to “integrity” and “secrecy” for the voting population. *Integrity* means that votes are counted “as cast” so that the result respects the electors’ choices. *Secrecy* means that electors can vote for whom they like, without suffering coercion (beforehand) or embarrassment (afterwards); indeed, true secrecy means that electors can lie plausibly about how they voted.

The tension between secrecy and integrity cannot be completely resolved however: the mere act of announcing the election results inevitably reveals something about electors’ wishes, assuming that the announcement is faithful to their wishes as expressed on the ballots. (Recall the unavoidable information flow discussed for the Lovers’ protocol in §15.6 and for the Three Judges protocol in Chap. 21.) That tension is something that all electoral commissions deal with, and they try to find the best trade-off between maintaining some degree of privacy while implementing integrity.

In this chapter we take an information-flow perspective on the relation between privacy for electors, on the one hand, and policies for transparency in reporting aggregated data, on the other. For example, we define gain functions for modeling an adversary who tries to determine which electors cast which ballots. How much privacy is lost through reporting? Can reporting be done differently to ensure the same transparency while better protecting privacy — or vice versa?

So how should reporting strike the best balance between protecting the right of electors to cast their votes in secret, and ensuring that it is not possible for the election result to be manipulated by taking advantage of provisions made for secrecy? In particular, when is it appropriate to consolidate reports for privacy reasons?

All that is what we discuss in this chapter, and we take both a comparative and a quantitative approach.

22.1 Elections and privacy risks

An election can compromise the privacy of the electors who participate in it. The amount and type of information released –when statistics about vote counting are published– varies widely by jurisdiction; and it accords with electoral culture and practice. Typically, electoral commissions publish intermediate results — such as tallies

for the candidates. They might also publish other information, such as the number of incorrectly filled-in ballots and even the identities of all the electors who voted. Such information contributes to assuring integrity and building trust — it enables public scrutiny, and also facilitates analysis by commentators, political scientists and political parties.

But what exactly are the privacy risks of releasing such information?

A well-known risk of that kind occurs in extreme outcomes, when the tallies reveal whether everyone voted the same way, or whether there were no votes for particular candidates. That is a higher risk than it might seem at first: for it is leaking not only “how many” but also “who” since, for example, if most votes in an electoral district are for a particular candidate (or political party, which mathematically is just a set of candidates), then an adversary can with high confidence guess the vote of any individual elector in that electorate. Even short of such extremes, the risks are still not well understood in general. Instead there is usually an informal assumption that large-scale elections have sufficiently many electors to make such scenarios unlikely.

Like many obvious truths, that assumption can sometimes be false. Beyond the obvious “are in the same electorate” correlations between electors, there could be other factors: family or friends, social-media posts or phone-location tracking. Moreover, the assumption that an elector can hide his choice within a large electorate is not valid for example in *preferential* elections, where electors are asked to rank the candidates in order from most to least preferred: there the electors’ privacy might be vulnerable to a “signature attack” (which we discuss later), even if the electorate is enormous.

22.2 An illustrative and simplified Q/F model for elections

We will split our modeling of elections into two parts: the tallying of the ballots, and the casting of the ballots.

22.2.1 The tallying

Here we identify the principal features that we will use for modeling tallying, the counting process by which votes cast on ballots are turned into the determination of winning candidates.

candidate A *candidate* in set \mathcal{C} is a person hoping to be elected.

ballot A ballot in set \mathcal{B} is the record of a vote, abstracting from its form: paper, or electronic or otherwise (e.g. verbal). In the simplest case, where every ballot selects a single candidate, we would have simply $\mathcal{B} := \mathcal{C}$. In preferential elections however, where ballots must rank the candidates, we would have $\mathcal{B} := \mathbb{S}\mathcal{C}$, the non-repeating sequences of candidates.

ballot box A ballot *box* is a multiset in $\mathcal{X} := \mathbb{M}\mathcal{B}$ of ballots cast, where a multiset is like a set but allows its elements to appear zero-or-more times rather than at most once. We write $b \# x$ for the number of times ballot b appears in ballot box x . We are therefore making the assumption, in this abstraction, that a ballot contains no explicit indication of who cast it: it contains no elector’s name or other explicit identifying information.

outcome The set \mathcal{Y} is the possible outcomes of a tally. For tallies with a single winner, we might have $\mathcal{Y} := \mathcal{C}$, so that a tally outcome is simply the candidate who won the election; for multiple winners we might have $\mathcal{Y} := \mathbb{P}\mathcal{C}$, the set of winners; and for tallies reporting actual counts, we might have $\mathcal{Y} := \mathcal{C} \rightarrow \mathbb{N}$, giving the number of votes counted for each candidate.¹

tally A *tally*, as a process, is a channel $T: \mathcal{X} \rightarrow \mathcal{Y}$ from ballot box to outcome. Usually, but not always, it is a deterministic channel.

In choosing the names \mathcal{X} and \mathcal{Y} for the type $\mathcal{X} \rightarrow \mathcal{Y}$ of the tally channel, we are following the naming conventions we have established earlier: that \mathcal{X} is the secret, whose prior distribution is known (but not the actual value), and that \mathcal{Y} is the observations the channel permits, which might reveal information about the secret.

Here is a simple example of such modeling. In a simple-majority tally for a single winner, with no counts given, we would have $\mathcal{B} = \mathcal{C}$, i.e. that a ballot simply names a candidate; and so $\mathcal{X} = \mathbb{M}\mathcal{C}$ and $\mathcal{Y} = \mathcal{C}$; the tally channel T would indeed be deterministic so that $T_{x,c} = 1$ just when c is such that $c\#x$ is largest among all elements of x , and $T_{x,c} = 0$ otherwise.² If however the winner's count were given, we could have instead $\mathcal{Y} = (\mathcal{C}, \mathbb{N})$, i.e. the winner and how many votes he received; and finally, if all candidates' counts were to be given, we could choose \mathcal{Y} to be \mathcal{X} itself, since for example the more explicit “counting function” of type $\mathcal{C} \rightarrow \mathbb{N}$ is in fact isomorphic to $\mathbb{M}\mathcal{C}$ (as mentioned above). Thus in the last case the tally channel T is actually the leak-everything channel O — reveal all ballots completely.

As a target for an attack, even in the O case the tally itself does not offer much: our secret is \mathcal{X} , the composition of the ballot box; and the most an adversary can learn is something (or perhaps everything) about the ballots — and that does not *per se* reveal anything about the *electors* or which votes they cast. Indeed, the electors were not mentioned at all in the above list of modeling artifacts. For that, we must look at how ballots are cast.

22.2.2 The casting

The principal risk to electors' privacy is that somehow a ballot can be traced back to the elector who cast it; and there are a number of ways that could happen. One way is that the electoral process requires electors to identify themselves on the ballot; but we will not discuss that approach further here. Another way however is the signature attack, mentioned above, which is applicable to preferential elections where the information released even in anonymized ballots can be used to re-identify particular electors with high confidence. It is often called the “Italian” attack³ because it is said to have been used in Italy as a method to demand that electors vote as they are told rather than as they wish. A coercer forces her victim to “sign” his ballot by his including in it a unique pattern of preferences, but only among the candidates ranked so low that their local order hardly matters. After the election, if the ballots' contents are released so that each elector can check that his ballot is there, the coercer can use the

¹ The number of votes counted for a candidate is not always the same as the number of votes cast: in preferential elections, for example, the count for a candidate can be the outcome of a complex, even probabilistic calculation.

² We are ignoring tied outcomes in this simple example.

³ “... Raimondo Maira, candidate at the regional elections of 1991, in return for a payment of 25 million lira, had obtained from the local Mafia family control and protection for its electoral office and leafleting. Even if no violence was deployed, it was understood that those who did not respect the vote suggestion of the family could suffer consequences.” D. della Porta and A. Vannucci *Corrupt Exchanges: Actors, Resources, and Mechanisms of Political Corruption*, New York, Aldine De Gruyter (1999).

low-ranking preferences to find the vote that her victim cast, and then check that its high-ranking preferences are as she demanded. But we do not address the signature attack here, either.

A third risk to privacy, perhaps the most common and the one we do address, is that an overall election is divided into electorates, and each electorate is limited to voting for certain candidates: in that way, what seems to be a large election is actually many small ones, run independently in parallel. And in that case, tally outcomes for certain candidates can release information about electors in the electorates allowed to vote for those candidates. For example in countries with remote elector communities, an electorate can be very small indeed: even if the tally is aggregated nationally, if the number of votes for candidate c say is equal to the number of electors in the (only) electorate allowed to vote for c then every elector in that electorate has his vote revealed.

We model those aspects of an election as follows:

candidate Recall from §22.2.1 that a *candidate* in set \mathcal{C} is a person hoping to be elected.

ballot Similarly, recall from above that a ballot in set \mathcal{B} records a vote for a candidate.

elector An *elector* in set \mathcal{E} is a person who is allowed to cast a vote in the election.

voting pattern A *voting pattern* $\mathcal{Z} := \mathcal{E} \rightarrow \mathcal{B}$ is a function giving for each elector the ballot he cast.⁴

casting A *casting*, as a process, is a channel $\mathbf{A}: \mathcal{Z} \rightarrow \mathcal{X}$ from voting pattern to ballot box.

Although we have already seen above that there are many possibilities for the tallying channel \mathbf{C} , in fact there is only one realistic possibility for \mathbf{A} (for “anonymize”) unless we model ballots’ being lost or misread: we have that $\mathbf{A}_{z,x}$ is 1 just when for any ballot b we have that $b \# x$ is the number of electors e such that $z(e) = b$. The anonymizing aspect of \mathbf{A} on a ballot (e, b) is that it “erases the e ” — but it leaves the “how many” of the b .

22.2.3 The Dalenius perspective: casting then tallying

If we concentrate on security vulnerabilities introduced by the tallying process alone, we cannot at first sight explain how an attacker would get from there to conclusions about voting patterns: as remarked above, the electors do not even appear in the model of tallying set out in §22.2.1.

Fortunately, in Chap. 10 we have already discussed how a channel of type $\mathcal{X} \rightarrow \mathcal{Y}$ —here, the tallying process— that leaks information about its input X —the contents of the ballot box— could incidentally leak information about some *other* secret Z —the voting pattern of the electors— using a correlation unknown (or irrelevant) to the designers or operators of that channel. For example the tallying process, say a computer program, could be used in many different elections if they share the same voting legislation. The designers of the legislation would not have been aware of the elections or electorates to which their legislation would ultimately be applied — possibly decades or even centuries into the future. For example they would not have been likely to understand in those early days of democracy what a “smartphone” is and how a ballot could be

⁴ For simplicity we will assume that every elector must vote. To model optional voting, we could add a special ballot \perp which is not counted.

cast with it. It's for those reasons we separate the election process into two parts, concentrating on tallying: it is fixed by legislation, implemented by paper-handling procedures or computer programs, and does not change without explicit legislative action. On the other hand, casting procedures can change unpredictably due to fashion and market forces.

We deal with that separation using the approach of §10.1, where the Dalenius channel –there called J (for “joint”) but here called A (for “anonymize”)– is cascaded with the tallying channel T ; and we analyze the leakage caused by that composition, in fact the matrix multiplication $E = AT$. Channel A models the ballot-casting process, which might change from time to time and place to place; Channel T models the tallying process, which is comparatively stable; and Matrix E captures the (here much simplified) election as a whole.

22.3 Election by simple majority: first past the post

In a simple-majority election of one candidate, the winner is the candidate who receives the most votes.⁵ The system is often called “first past the post”.

22.3.1 QIF channels for simple-majority elections: two examples

As suggested in the examples of §22.2.1, for single-candidate single-vote tallying we define $B := C$, and so $\mathcal{X} = MC$ and $\mathcal{Z} = E \rightarrow C$. We will consider two kinds of tallying outcomes: announcing simply the winner will be $\mathcal{Y}_W := C$, and announcing tallies for all candidates will be $\mathcal{Y}_C := C \rightarrow N$. (Recall however our earlier remark that $C \rightarrow N$ and MC are isomorphic, so that in the latter case in fact $T = O$. We write $C \rightarrow N$ here only because it is more intuitive.)

We will start by assuming that there are only two candidates $c_{1,2}$ and three electors $e_{1,2,3}$. In Fig. 22.1 we give two matrices for the corresponding overall election channels E , that is in which the casting channel A has been included. Then, rather than use a prior over ballot boxes \mathcal{X} , we take the approach of §22.2.3 and address the leakage with respect to the prior over voting patterns \mathcal{Z} — that is, we compose the tallying process with the casting process. Election channel W is thus of type $\mathcal{Z} \rightarrow \mathcal{Y}_W$, releasing only the name of the winning candidate, and C is of type $\mathcal{Z} \rightarrow \mathcal{Y}_C$, giving the tallies. The rows are labeled by the possible voting patterns in \mathcal{Z} : for example the ad-hoc syntax $\langle c_1 c_2 c_2 \rangle$ is the pattern in which e_1 voted for c_1 and $e_{2,3}$ voted for c_2 . The columns in the matrices correspond to the different outcomes: in W there are only two possible announcements: “ c_1 won” or “ c_2 won”. But in C there are four: either “ c_1 won (3,0)” or “ c_1 won (2,1)” or “ c_2 won (1,2)” or “ c_2 won (0,3)”. Thus C certainly releases more information than W , and for transparency it would be desirable to use C , but our interest is in the impact that would have on privacy.

Let $\zeta: \mathbb{D}\mathcal{Z}$ thus be the prior over all possible voting patterns for the small electorate (only three electors) of this example. That prior would ordinarily be determined empirically via pre-election polls in that electorate, or its historical data. When the winner is announced via channel W , we would observe c_1 as the winner with probability $\sum_{z: \mathcal{Z}_{c_1}} \zeta_z$, where \mathcal{Z}_{c_1} is the set of voting patterns in which c_1 was the most often selected, and similarly for c_2 .

⁵ As we remarked earlier, we are for simplicity ignoring tied outcomes. It's worth remarking though that if an exact tie is resolved by a coin flip, then we already have the apparatus to model that: the tally channel T becomes properly probabilistic. Another example of probabilistic outcomes in tallying occurs in preferential elections for several candidates, where the reallocation of surplus votes is explicitly random.

		\mathcal{Y}_W		\mathcal{Y}_C			
		c_1	c_2	(3, 0)	(2, 1)	(1, 2)	(0, 3)
$\mathcal{Z} \rightarrow$	$\langle c_1 c_1 c_1 \rangle$	1	0	$\langle c_1 c_1 c_1 \rangle$	1	0	0
	$\langle c_1 c_1 c_2 \rangle$	1	0	$\langle c_1 c_1 c_2 \rangle$	0	1	0
	$\langle c_1 c_2 c_1 \rangle$	1	0	$\langle c_1 c_2 c_1 \rangle$	0	1	0
	$\langle c_2 c_1 c_1 \rangle$	1	0	$\langle c_2 c_1 c_1 \rangle$	0	1	0
	$\langle c_2 c_1 c_2 \rangle$	0	1	$\langle c_2 c_1 c_2 \rangle$	0	0	1
	$\langle c_1 c_2 c_2 \rangle$	0	1	$\langle c_1 c_2 c_2 \rangle$	0	0	1
	$\langle c_2 c_2 c_1 \rangle$	0	1	$\langle c_2 c_2 c_1 \rangle$	0	0	1
	$\langle c_2 c_2 c_2 \rangle$	0	1	$\langle c_2 c_2 c_2 \rangle$	0	0	1

The row labels are of type $\mathcal{Z} = \mathcal{E} \rightarrow \mathcal{C}$, indicating how the electors $e_{1,2,3}$ cast their ballots: thus $\langle c_2 c_1 c_1 \rangle$ means e_1 chose c_2 and $e_{2,3}$ chose c_1 . Channel W simply announces the winning candidate, but channel C gives the vote counts: thus (2, 1) labeling a column in C means that candidate c_1 received 2 votes and candidate c_2 only 1 vote.

Figure 22.1 Two reporting channels with three electors and two candidates

The information flow in C is different, of course: by announcing the tallies there, we now have four possible observations, which in particular might reveal whether or not one of the candidates received no votes. If for example the voting pattern z is revealed to be definitely either $\langle c_1 c_1 c_1 \rangle$ or $\langle c_2 c_2 c_2 \rangle$ then the losing candidate knows for certain that no elector voted for him, and indeed that the winner was chosen unanimously.

We can now observe that $C \sqsubseteq W$, using structural refinement, because the first (last) two columns of C can be combined to give the first (last) column of W ; and by the soundness of structural refinement, we know that C might leak more information than W in some scenarios defined by gain functions, but never less. And we will look at gain functions in §22.5 below.

Recalling Thm. 7.2, we see that the multiplicative Bayes capacity of W is 2 whereas that of C is 4, so that the maximum leakage in C is twice as much as it is for W in some scenarios.

22.4 Election by preferences: instant run-off

A “run off” in a single-winner election is the process where two casting-and-tallying rounds are held: often the first round selects the two candidates (from many) who are the most favored; and then a second round is a *run off* between just those two, where electors cast ballots afresh. Thus it is “cast, tally, cast, tally, done”. An *instant run-off* election is a variation of that, where ballots rank candidates (in some cases *all* candidates) in order of preference. If no candidate receives more than half the first-preference votes in the first round, then the candidate receiving the fewest first-preference votes is eliminated, and his name is removed from all ballots (with ballots on which he was the first preference instead awarded to the candidate who was the second preference on that ballot). Then the tallying is done a second time; and the process is repeated if still there is no candidate with more than half the votes: thus here we have “cast, tally, …, tally, done”. That is, there is only one vote-casting process in *instant run off*: electors do not change their ballots between tallying rounds.

22.4.1 Q/F channels for instant-run-off elections: two examples

For instant run-off we define two channels P (“partial”) and F (“full”). Whereas F publishes the number of times each full preference ranking occurs in all the ballots, channel P publishes only the number of times each candidate is the first-choice preference. That means that P does not provide enough information to verify the eventual winner(s) and losers, except in the case when some candidate wins a majority based on first preferences only. It is however the case that the candidate who has fewest first preferences is never elected. Thus although F might be preferred for transparency reasons, in fact there is a known privacy risk involved in releasing full preferences, and many electorates publish first preferences only, so that P provides a realistic model for reporting in practice. We do know that $F \sqsubseteq P$ since P ’s outputs can be computed from F ’s, and therefore (full) channel F ’s leakages are all upper bounds for the corresponding leakages in (partial) channel P .

Definition 22.1 (Ballot) Define now $\mathcal{B} := \mathbb{S}\mathcal{C}$, so that a ballot is a non-repeating sequence of candidates, and so that $\mathcal{X} = \mathbb{M}\mathcal{B}$ becomes $\mathcal{X} := \mathbb{M}\mathbb{S}\mathcal{C}$ and $\mathcal{Z} = \mathcal{E} \rightarrow \mathcal{B}$ becomes $\mathcal{Z} := \mathcal{E} \rightarrow \mathbb{S}\mathcal{C}$. Define F in $\mathcal{Z} \rightarrow (\mathcal{B} \rightarrow \mathbb{N})$ to be the reporting channel that announces tallies for each possible ranking in \mathcal{B} . (As with C in §22.3, it is the composition of the anonymizing channel A and the reveal-everything channel O .) Thus $F = A \circ O$, so that

$$F_{z,y} := \begin{cases} 1 & \text{if } (\forall b: \mathcal{B} \cdot y(b) = (\#e \mid z(e) = b)) \\ 0 & \text{else} \end{cases},$$

where $y: \mathcal{Y}_F$ with $\mathcal{Y}_F = \mathbb{M}\mathcal{B} = \mathbb{M}\mathbb{S}\mathcal{C}$, and the quantifier $(\# \dots)$ gives the number of values of the bound variable for which the predicate holds. Similarly let P in $\mathcal{Z} \rightarrow (\mathcal{C} \rightarrow \mathbb{N})$ model the reporting channel that publishes tallies for first preferences only: here we have $y: \mathcal{Y}_P$ with $\mathcal{Y}_P = \mathcal{C} \rightarrow \mathbb{N}$, and

$$P_{z,y} := \begin{cases} 1 & \text{if } (\forall c: \mathcal{C} \cdot y(c) = (\#e \mid c = \text{fst } z(e))) \\ 0 & \text{else} \end{cases},$$

where $\text{fst } z(e)$ is the first element of $z(e)$ and so y is a function from candidate to number of electors that selected that candidate as first preference. (More abstractly, that is the type $\mathbb{M}\mathcal{C}$). \square

22.5 Gain functions for privacy of elections: a first example

As suggested above, with the channel framework we have set up we can look at gain functions that might be used to evaluate the privacy of the above election types. Recall that a gain function is intended to estimate how much the adversary gains by taking certain actions in a set \mathcal{W} ; and we (the defenders) use it to estimate how best to defend ourselves against her. Here –strictly speaking– we will be using Dalenius vulnerability as in Def. 10.1; but since we have composed the Dalenius correlation A already, we continue with the usual notation (i.e. without using the D -for-Dalenius superscript as in Def. 10.1). We consider three possibilities, concentrating first on the simple-majority election.

Suppose that the adversary will benefit from guessing correctly how some elector voted, but her benefit does not depend on which elector she chooses or who he voted for: any victim, and any vote will be a gain for her.⁶ We define for that a gain function g_1 which allows an adversary to choose from action set $\mathcal{W} = \mathcal{C} \times \mathcal{E}$, where (c, e) is her action of guessing that candidate c was selected by elector e .

⁶ Recall the discussion in §3.2.4 that compares guessing any user’s password with guessing a specific user’s password.

Definition 22.2 (Identify a single candidate/elector pair) We define the gain function $g_1: \mathcal{W} \times \mathcal{Z} \rightarrow [0, 1]$ by

$$g_1((c, e), z) := \begin{cases} 1 & \text{if } c = z(e) \\ 0 & \text{else} \end{cases},$$

where $z(e)$ is the candidate voted for by elector e in voting pattern z . \square

We now make some further assumptions. As in §22.3.1, let $\zeta: \mathbb{D}\mathcal{Z}$ be the prior over all possible voting patterns \mathcal{Z} in a three-elector electorate; but do not assume that the adversary has any knowledge about the voting pattern before the election results are announced. So she takes ζ to be the uniform distribution ϑ , i.e. that each of the three electors is equally likely to choose either of the two candidates. Then we have $V_{g_1}(\vartheta) = 1/2$, because whichever pair (c, e) the adversary picks, only half of the set of possible election results has candidate c selected by elector e . After the results are announced however, both \mathbf{W} and \mathbf{C} result in the same multiplicative leakage: we have $\mathcal{L}_{g_1}^{\times}(\vartheta, \mathbf{W}) = \mathcal{L}_{g_1}^{\times}(\vartheta, \mathbf{C}) = 3/2$ because, *in both cases*, the adversary improves her guess given the information released, using the following reasoning.

If candidate c_1 wins then the adversary *always* guesses (c_1, e) for any elector e , and if c_2 wins then the adversary always guesses (c_2, e) . That is because in the post-hoc situation the winner is the candidate that is most likely to have been selected by most electors. Surprisingly, although the maximal possible leakage for \mathbf{C} is 4, the actual leakage in this particular case is $\mathcal{L}_{g_1}^{\times}(\vartheta, \mathbf{C}) = 3/2$, which means that much of the extra information given by releasing tallies is not useful to an adversary who is *only* interested in trying to guess how some elector voted. And indeed that particular attack cannot be mitigated in any way, since at the very least the winner must be announced: as we have seen $\mathcal{L}_{g_1}^{\times}(\vartheta, \mathbf{W})$ is also $3/2$. We discuss that phenomenon further in §22.7.1.

Now we set out two more gain functions that, together with the one above, capture three different aspects of privacy. We will use them in §22.7 below.

Definition 22.3 (Average correct guesses) Here we let \mathcal{W} be \mathcal{Z} itself, and the adversary tries to guess as many electors' votes as possible: we define

$$g_{\#}(z', z) := \#\{e \mid z'(e) = z(e)\},$$

where we recall that the quantifier $(\# \cdots)$ gives the number of values of the bound variable for which the predicate holds. \square

Definition 22.4 (Identify a single elector/vote-not-cast pair) This gain function gives 1 to the adversary if she can find an elector/candidate pair such that the elector did *not* vote for that candidate.

$$g_0((c, e), z) := \begin{cases} 1 & \text{if } c \neq z(e) \\ 0 & \text{else} \end{cases}.$$

It is the complement of Def. 22.2, with the parameters defined in the same way. \square

We can use those gain functions to express privacy for simple-majority single-candidate elections by analyzing the behavior of the derived leakages. For example the leakages corresponding to the gain functions of Def. 22.2 and Def. 22.4 together express a strong form of privacy: if an adversary can *neither* guess an elector's selection *nor* whom they did not select, then any elector later linked to that ballot is able to lie with confidence.

22.6 The effect of small electorates in general

A particular focus of our analysis is to explore how privacy is affected by reporting results for small samples of electors. When the overall election channel E is a good preserver of privacy with respect to a gain function g then $\mathcal{L}_g^{\times}(\zeta, \mathsf{E})$ will be close to 1 — but the more that privacy could be compromised, for some elector, the larger (than 1) that leakage will be. If the leakage is comparable to the maximum possible leakage then it means that the adversary is able to use much of the information leaked to find out how electors voted.

Elections generally involve large numbers of electors compared to candidates, and principles for ensuring privacy often have that assumption at their heart. But, as argued above, that assumption might not be valid for various reasons — including the location of polling places, the type of election, and the need for transparency. We can, however, try to understand in terms of information leaks the pressures those exigencies put on electors' privacy.

The underlying assumption in elections is that when the number of electors is huge, privacy for individual electors is protected in reporting, because each possible choice is *bound to have been made by several electors*. That is usually the case in simple voting systems such as first past the post (§22.3), even with few candidates, but is not the case in preferential elections when electors are asked to rank all candidates. We will use our model for information flow to study the degree to which privacy is lost when the number of electors is small, even if we assume that the reporting channel preserves privacy when the number of electors \mathcal{E} is large.

Let \mathcal{S} (for “small”) be a subset of all electors \mathcal{E} , that is $\mathcal{S} \subset \mathcal{E}$, and write $\vartheta_{\mathcal{S}}$ and $\vartheta_{\mathcal{E}}$ for the uniform distributions on $\mathcal{S} \rightarrow \mathcal{B}$ and $\mathcal{E} \rightarrow \mathcal{B}$ respectively. There are a number of ways to try to compare the extent to which privacy is affected between scenarios when \mathcal{S} is small and \mathcal{E} is large. In general if $\mathcal{L}_g^{\times}(\vartheta_{\mathcal{S}}, \mathsf{E}) \geq \mathcal{L}_g^{\times}(\vartheta_{\mathcal{E}}, \mathsf{E})$, then the greater the difference between $\mathcal{L}_g^{\times}(\vartheta_{\mathcal{S}}, \mathsf{E})$ and $\mathcal{L}_g^{\times}(\vartheta_{\mathcal{E}}, \mathsf{E})$, the greater will be the impact on privacy to electors in the sample \mathcal{S} . However the difference $\mathcal{L}_g^{\times}(\vartheta_{\mathcal{S}}, \mathsf{E}) - \mathcal{L}_g^{\times}(\vartheta_{\mathcal{E}}, \mathsf{E})$ abstracts from the severity of the scenario. For example if $\mathcal{L}_g^{\times}(\vartheta_{\mathcal{S}}, \mathsf{E}) - \mathcal{L}_g^{\times}(\vartheta_{\mathcal{E}}, \mathsf{E})$ is small, the impact on privacy could still be considered a problem if that difference is large compared to the actual leakage $\mathcal{L}_g^{\times}(\vartheta_{\mathcal{S}}, \mathsf{E})$. The next definition therefore gives the proportional loss of privacy relative to the leakage scenario $\mathcal{L}_g^{\times}(\vartheta_{\mathcal{S}}, \mathsf{E})$.

Definition 22.5 Let E be an overall election channel, and g a gain function as defined above. Let \mathcal{S} be a subset of all electors \mathcal{E} . The proportional loss of privacy to electors in \mathcal{S} relative to the election as a whole, i.e. consisting of electors in \mathcal{E} , is

$$\frac{\mathcal{L}_g^{\times}(\vartheta_{\mathcal{S}}, \mathsf{E}) - \mathcal{L}_g^{\times}(\vartheta_{\mathcal{E}}, \mathsf{E})}{\mathcal{L}_g^{\times}(\vartheta_{\mathcal{S}}, \mathsf{E})} = 1 - \frac{\mathcal{L}_g^{\times}(\vartheta_{\mathcal{E}}, \mathsf{E})}{\mathcal{L}_g^{\times}(\vartheta_{\mathcal{S}}, \mathsf{E})}. \quad (22.1)$$

□

When the size of the electorate is so large that the reporting channel provides good privacy overall, that is $\mathcal{L}_g^\times(\vartheta_{\mathcal{E}}, \mathsf{E}) \approx 1$, we can approximate (22.1) by $1 - 1/\mathcal{L}_g^\times(\vartheta_{\mathcal{S}, \mathsf{E}})$. In any case, that provides an upper bound on the proportional loss of privacy to electors in sample \mathcal{S} caused by the contribution of the information flow in the channel restricted to them alone.

22.7 Case studies of small-electorate impact

In this section we investigate more thoroughly the privacy properties described above with respect to two types of elections: first past the post, and instant run-off. We take primarily a comparative approach, recognizing that no reporting channel can be absolutely risk-free, but that some kinds of reporting leak more information than others, and that some of the information released could potentially impact privacy.

In all our analyses we assume that the adversary knows nothing about the likely voting preferences, and throughout we use the uniform distribution ϑ over \mathcal{Z} to model the adversary's prior knowledge. We note however that the general method of *QIF* does not require that assumption.

22.7.1 First past the post, in small electorates

Recall that first-past-the-post elections are used to elect a single candidate by simple majority. Electors in \mathcal{S} pick exactly one candidate from \mathcal{C} , and the candidate who obtains the most votes is declared the winner. The next definition sets out a generalization of Fig. 22.1's methods of reporting results (for arbitrary numbers of candidates and electors).

Definition 22.6 Let $\mathcal{Z} := \mathcal{S} \rightarrow \mathcal{C}$ be the voting pattern, and define $\mathsf{W}: \mathcal{Z} \rightarrow \mathcal{C}$ to be the reporting channel that announces the winner only:

$$\mathsf{W}_{z,c} := \begin{cases} 1 & \text{if } \text{maj}(z) = c \\ 0 & \text{else} \end{cases}, \quad (22.2)$$

where $\text{maj}(z)$ is the candidate with the maximum number of votes determined by pattern z .⁷

Define $\mathsf{C}: \mathcal{Z} \rightarrow (\mathcal{C} \rightarrow \mathbb{N})$ to be the reporting channel that announces the set of tallies for each candidate:

$$\mathsf{C}_{z,x} := \begin{cases} 1 & \text{if } (\forall c: \mathcal{C} : \#_c(z) = x(c)) \\ 0 & \text{else} \end{cases}, \quad (22.3)$$

where $\#_c(z) := (\#\{e: \mathcal{S} \mid z(e) = c\})$ is the number of votes that candidate c received in the voting pattern z . \square

As noted earlier, we have the refinement $\mathsf{C} \sqsubseteq \mathsf{W}$ because C releases much more information than does W .

More interesting however is whether releasing the tallies provides information that an adversary could use. The next theorem says that it is in fact not useful if all the adversary wants to do is to guess electors' votes: the extra information in the announcements of tallies can be used by the adversary neither to guess how an elector voted (g_1) nor to increase her expected number of correct guesses ($g_\#$).

⁷ Again we ignore ties though, as remarked earlier, they can be handled probabilistically. The important property to consider is that the tie-breaking mechanism should be independent of the set of ballots, otherwise it might reveal more information than expected.

Theorem 22.7 Let \mathbf{W} and \mathbf{C} be the reporting channels defined at Def. 22.6 and let $g_1, g_\#$ be the gain functions defined in §22.5. Let ϑ_S in $\mathbb{D}\mathcal{Z}$ be the uniform prior on voting pattern $\mathcal{Z} = \mathcal{S} \rightarrow \mathcal{B}$. Then the following equalities hold:

$$\mathcal{L}_{g_1}^\times(\vartheta_S, \mathbf{W}) = \mathcal{L}_{g_1}^\times(\vartheta_S, \mathbf{C}) \quad \text{and} \quad \mathcal{L}_{g_\#}^\times(\vartheta_S, \mathbf{W}) = \mathcal{L}_{g_\#}^\times(\vartheta_S, \mathbf{C}) .$$

Proof. (Informal sketch.) The prior gain $V_{g_1}(\vartheta_S)$ is the same for both leakage calculations, therefore we only need show that $V_{g_1}[\vartheta_{S^D}\mathbf{W}] = V_{g_1}[\vartheta_{S^D}\mathbf{C}]$. Note that for any announcement of tallies for an election whose voting pattern is z , it is still the case that most electors in \mathcal{S} voted for the candidate with the majority, and so the adversary's optimum guessing strategy is to pick $(maj(z), e)$, which is exactly the same guessing strategy if only the winner is announced. Hence since the optimal guessing strategies are the same for both \mathbf{W} and \mathbf{C} , the leakage wrt. g_1 must also be the same.

For $g_\#$, a similar argument shows that the adversary's optimal strategy is to guess that all electors voted for the candidate who won.

□

A second type of attack is when the adversary tries to guess whether an elector *did not* vote for a candidate; and here we get a different result.

Theorem 22.8 Let \mathbf{C} and \mathbf{W} be the reporting channels defined at Def. 22.6 and let g_0 be the gain function defined in Def. 22.4. If $|\mathcal{C}| \geq 3$, i.e. there are at least three candidates, then $\mathcal{L}_{g_0}^\times(\vartheta_S, \mathbf{W}) < \mathcal{L}_{g_0}^\times(\vartheta_S, \mathbf{C})$.

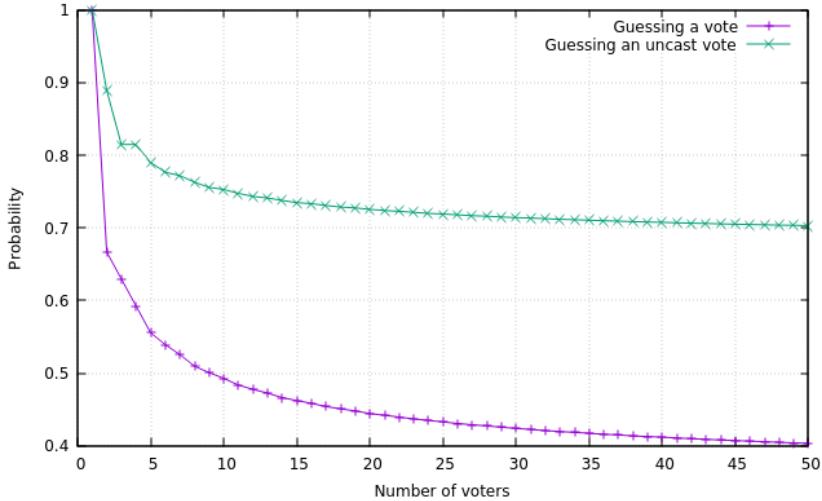
Proof. (Informal sketch.) When tallies are released, the adversary can increase her gain by guessing that the candidate who received the least number of votes is most likely not someone the elector selected. That information is not available in \mathbf{W} . □

The qualitative results from Thm. 22.7 and Thm. 22.8 are perhaps not very surprising, although they do highlight that privacy is not determined by a single scenario. More interesting however is to compare the actual relative differences in the cases where the assumption that the aggregate electorate exceeds the number of ways of voting is false. As mentioned above, that is a frequent problem in cases where results from remote polling places are publicly announced.

To that end, it is instructive to compute the actual leakages for first-past-the-post channels \mathbf{W} and \mathbf{C} . In these experiments the number of electors in a “reporting batch” given by the size of the small electorate \mathcal{S} is varied, with the assumption that there are three possible candidates to choose from, with an initial uniform prior over possibilities. This set up is akin to an electoral commission's releasing e.g. polling place tallies where only a few electors cast their votes, and no prior knowledge over elector preference.

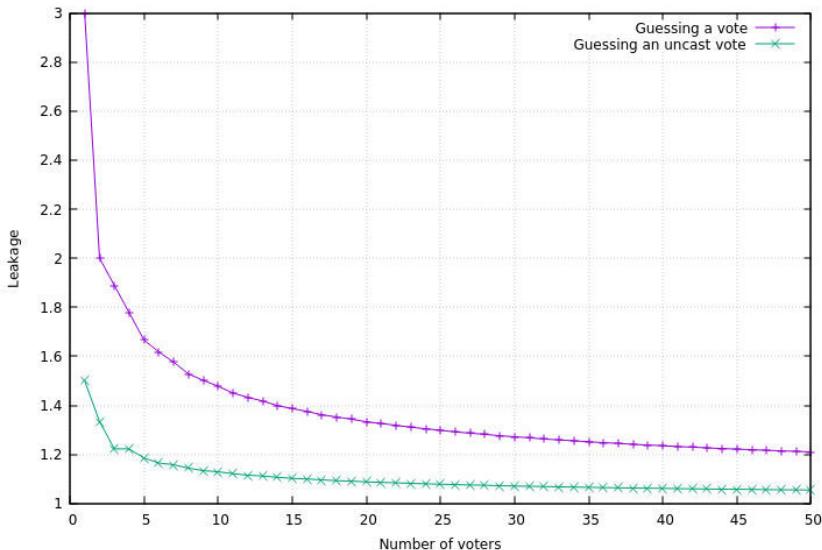
The graphs in Fig. 22.2 illustrate the average vulnerabilities for the privacy questions: “Can the adversary guess which candidate an elector selected?” and “Can the adversary guess which candidate an elector did not select?” The former corresponds to $V_{g_1}[\vartheta_{S^D}\mathbf{W}]$ and the latter to $V_{g_0}[\vartheta_{S^D}\mathbf{W}]$, where the channel \mathbf{W} releases only the winning candidate. Recall from Chap. 5 that posterior vulnerabilities give the expected posterior vulnerability for a given privacy question — here this translates to computing the probabilities that an adversary can guess correctly the answer to the respective question.

Since $\mathbf{C} \sqsubseteq \mathbf{W}$, these are lower bounds on the more revealing reporting channel \mathbf{C} which releases the tallies as well. For $V_{g_1}[\vartheta_{S^D}\mathbf{W}]$, the vulnerability of guessing the



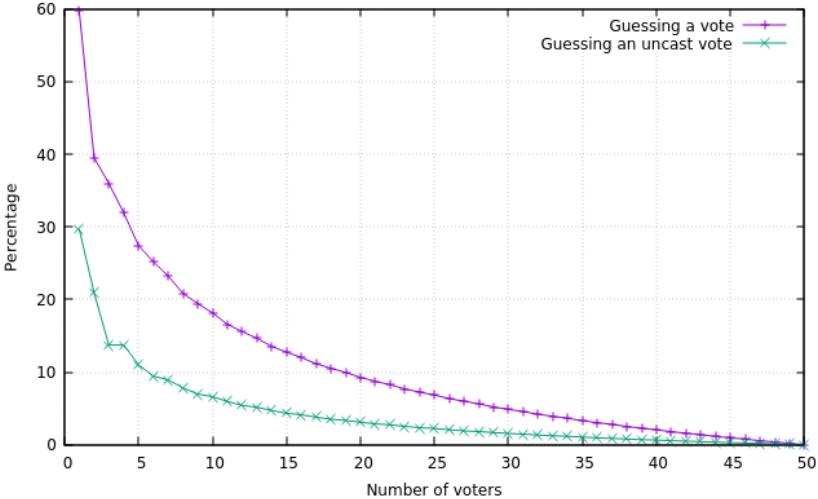
Vulnerabilities for (fixed) $|\mathcal{C}| = 3$ candidates, varying electorate size $|\mathcal{S}|$ (horizontal axis).

Figure 22.2 Posterior vulnerabilities $V_{g1}[\vartheta_S \triangleright W]$ (lower) and $V_{g0}[\vartheta_S \triangleright W]$ (upper)



Leakages for (fixed) $|\mathcal{C}| = 3$ candidates, varying electorate size $|\mathcal{S}|$ (horizontal axis).

Figure 22.3 Leakages $\mathcal{L}_{g1}^\times(\vartheta_S, W)$ (upper) and $\mathcal{L}_{g0}^\times(\vartheta_S, W)$ (lower)



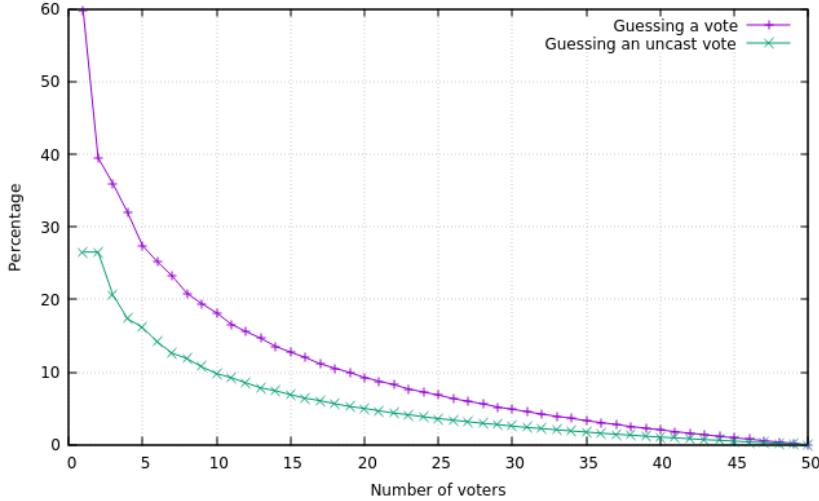
Analysis for (fixed) $|\mathcal{C}| = 3$ candidates, varying electorate size $|\mathcal{S}|$ (horizontal axis).

Figure 22.4 Proportional loss in privacy (Def. 22.5) for \mathbf{W} wrt. g_1 (upper) and g_0 (lower)

elector preference (i.e. the probability of correctly guessing the selected candidate on a given elector's ballot) rapidly approaches the prior vulnerability (of 0.33) (lower curve in Fig. 22.2), showing that privacy for this question behaves well in large reporting batches. However for the dual question $V_{g_0}[\vartheta_{\mathcal{S}^D}\mathbf{W}]$ the story is quite different: when the electorate consists of 50 electors there is still a 70% chance that an adversary can correctly guess who an elector did not vote for (upper curve in Fig. 22.2).

The graph in Fig. 22.3 displays the multiplicative leakages for the two privacy questions above, illustrating the contribution of the channel \mathbf{W} to the posterior vulnerabilities. Here we see that reporting the winner leaks more information about guessing who the elector voted for than who the elector did not vote for. When we compare that with Fig. 22.2, we see that the probability that the adversary succeeds in guessing who an elector *did not* vote for is already relatively high — but since the corresponding leakage calculations are low, as shown in Fig. 22.3, it means that this vulnerability is already manifest in the prior. It seems that it really is easier to deduce who someone did *not* vote for, simply because there are many more electors than candidates.

Fig. 22.4 and Fig. 22.5 illustrate the proportional loss in privacy (Def. 22.5) respectively for \mathbf{W} and \mathbf{C} . In both graphs the upper curves correspond to the percentage loss in privacy for the question “Can the adversary guess which candidate an elector selected?” and in the lower curves to the question “Can the adversary guess which candidate an elector did not select?” All curves are relative to the large sample size of 50 so that for the question “Guess which candidate was selected.” for both \mathbf{W} and \mathbf{C} , there is a greater than 30% loss in privacy in reporting batches consisting of fewer than 5 electors. By Thm. 22.7 the percentages in \mathbf{W} and \mathbf{C} are the same for this question.



Analysis for (fixed) $|\mathcal{C}| = 3$ candidates, varying electorate size $|\mathcal{S}|$ (horizontal axis).

Figure 22.5 Proportional loss in privacy (Def. 22.5) for \mathcal{C} wrt. g_1 (upper) and g_0 (lower)

However for the dual question ‘‘Guess which candidate was not selected.’’ –as Thm. 22.8 suggests– there is a difference in the contribution to posterior vulnerability between the two channels \mathcal{W} and \mathcal{C} . For \mathcal{W} (Fig. 22.4, lower curve) the contribution of \mathcal{W} is approximately 10% for a reporting batch of electors of size $|\mathcal{S}| = 5$ (horizontal axis), whereas for \mathcal{C} (Fig. 22.5, lower curve) it is more than 15%. This increase means that the actual information contained in the tallies can be used effectively by the adversary to improve her ability to determine who electors did not vote for. In some election environments that could be problematic.

22.7.2 Instant run-off in small electorates

Recall that in instant-run-off elections, the electors are asked to rank candidates in order of preference; we defined channels for that in Def. 22.1.

As in all of the reporting where the anonymous ballots are released, provided the size of the electorate \mathcal{E} is large compared to the number of observations, anonymity is protected fairly well. However as we saw with first past the post, the risks of privacy are increased when that assumption is invalid, i.e. in a small electorate \mathcal{S} , in the sense that the contents of individual ballots can be inferred. In ranking-based voting systems, there are potentially $|\mathcal{C}|!$ possible observations if the more discerning channel \mathcal{F} is used.

In Fig. 22.6 and Fig. 22.7 we illustrate what can happen in the case of \mathcal{F} for various degrees of possible rankings. Figure 22.6 displays the multiplicative leakages associated with gain function g_1 that models the adversary’s trying to guess how an elector ranked all the candidates, and so it indicates the scale for which channel \mathcal{F} ’s information leak helps the adversary improve her guess. The lowest curve corresponds

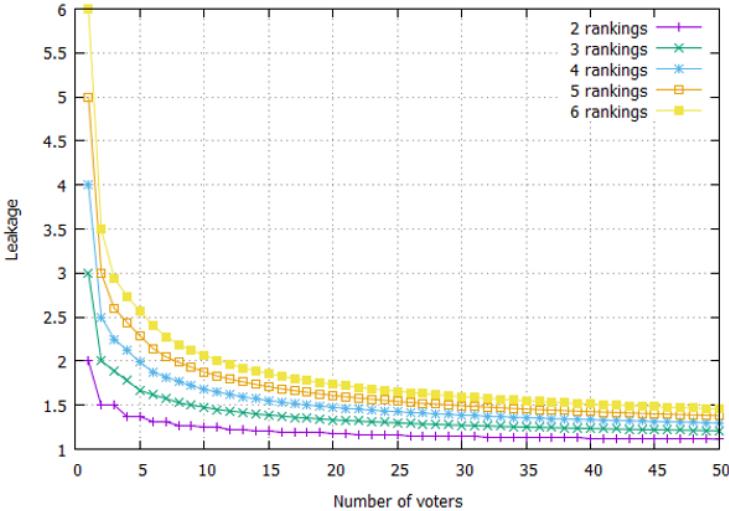


Figure 22.6 Graph of $\mathcal{L}_{g_1}^X(\vartheta, F)$ with 2 to 6 possible rankings (that is 2 to 3 candidates)

to 2 candidates (therefore only two rankings) and the highest curve corresponds to 3 candidates (therefore 6 possible rankings), with the curves in between illustrating leakages corresponding to numbers of observations between 2 and 6, which simulates a situation where fewer results are reported. As for first past the post we see that the leakages decrease as the number of electors increases: when there are 50 electors the leakages for all curves are below 1.5. However for about 5 electors the leakage is three times as much as in the case when there are 3 candidates (leakage 2.5) compared to 2 candidates (0.8). That means that potentially the adversary can identify $2.5/5 = 1/2$ of ballots with electors in a small sample of 5 electors.

A different perspective is given in Fig. 22.7, where we examine the relationship between numbers of electors versus numbers of candidates. The horizontal axis corresponds to the numbers of observations (related to the number of rankings $|C|!$) and the vertical axis corresponds to the leakage. The highest line is the leakage when there is 1 elector (thus his vote is revealed entirely by F) and the lowest line corresponds to 7 electors. The other lines correspond to numbers of electors in between 1 and 7. They show that for small samples of electors and relatively small numbers of candidates, a great deal of privacy is potentially at risk: with 7 electors and 4 candidates the leakage is more than three times that for 50 electors and 3 candidates.

Finally in Fig. 22.8 we illustrate similarly the leakages associated with gain function g_0 , where the adversary tries to guess a ranking that an elector did not select. This time the lowest curve corresponds to the leakage when there are 3 candidates and the highest to when there are only 2, with the in-between curves illustrating leakages corresponding to numbers of observations between 2 and 6. Dual to Fig. 22.6, the leakage (and therefore the contribution of F) decreases with increasing numbers of candidates. In Fig. 22.9 each curve (as for Fig. 22.7) corresponds to a fixed number of electors, ranging from 1 to 7, and the horizontal axis corresponds to numbers of observations: the contribution of F to this question is much less than for g_1 .

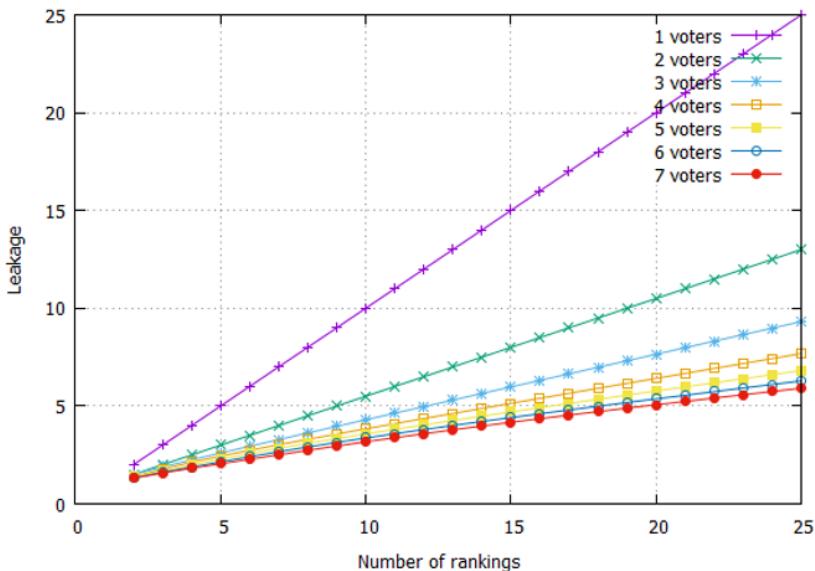


Figure 22.7 Graph of $\mathcal{L}_{g_1}^{\times}(\vartheta_S, F)$ with 1 to 7 electors (voters) and number of candidates ranging from 1 to 4 (that is 1 to 24 possible rankings)

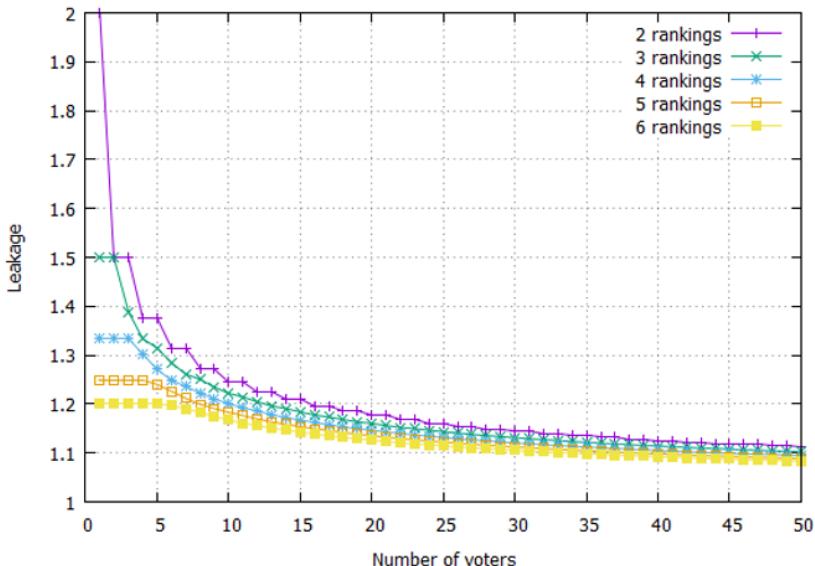


Figure 22.8 Graph of $\mathcal{L}_{g_0}^{\times}(\vartheta, F)$ with 2 to 6 possible rankings (that is 2 to 3 candidates)

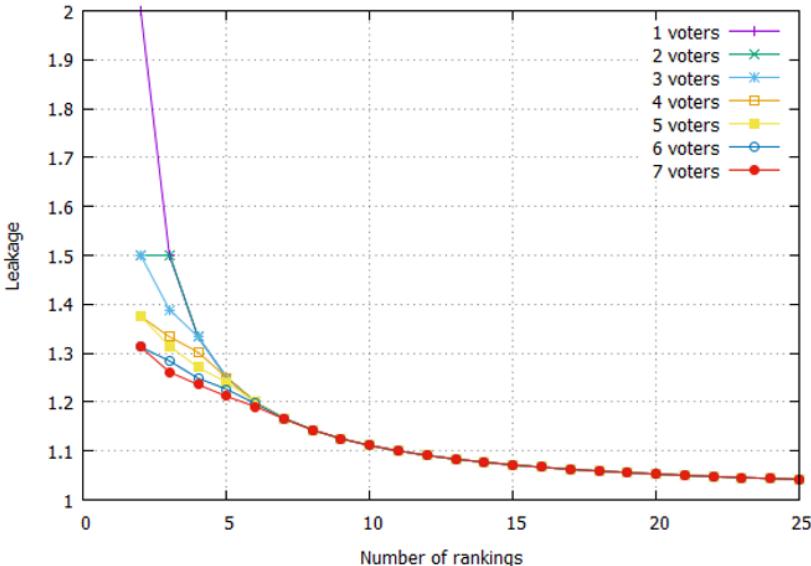


Figure 22.9 Graph of $\mathcal{L}_{g_0}^X(\vartheta, F)$ with 1 to 7 electors (voters) and number of candidates ranging from 1 to 4 (that is 1 to 24 possible rankings)

22.8 Chapter notes

It is well known that reporting methods for elections incur some risk to privacy, and indeed the relation to coercion-type attacks has been described by Smith [5]. The feasibility of a successful “Italian attack” in modern preferential elections has been studied by Naish and by Wen et al. [3, 7, 6]. In Australia it is common for small batches of votes to be revealed because results are usually reported at a fine-grained level for transparency reasons [7]. The privacy risks from that are unclear.

An example is the 2015 results for the New South Wales Legislative Assembly,⁸ which used instant-run-off voting and included reports for first preferences by polling place and voting method.

Computing the vulnerabilities and leakages depicted in Figs. 22.4–22.9, directly through Thm. 5.7, is computationally very expensive because that definition relies on summing over the set of all possible ballot boxes \mathcal{X} — which has $|\mathcal{C}|^{|\mathcal{E}|}$ elements. To compute leakages for reasonable electorate sizes, it was necessary to transform those expensive sums into manageable ones — and it turns out that the resulting expressions correspond to quantitative measures used in seemingly unrelated areas such as hash-code analysis [2, 4] and load balancing [1].

We thank Tahirya Rabejaja and Roland Wen for their contributions to this chapter.

⁸ <http://pastvtr.elections.nsw.gov.au/SGE2015/la-home.htm>

Bibliography

- [1] Czumaj, A., Stemann, V.: Randomized allocation processes. In: Proceedings of the 1997 IEEE 38th Annual Symposium on Foundations of Computer Science (FOCS 1997). IEEE Computer Society, Washington, DC (1997)
- [2] Gonnet, G.H.: Expected length of the longest probe sequence in hash code searching. *Journal of the ACM* **28**(2), 289–304 (1981)
- [3] Naish, L.: Partial disclosure of votes in STV elections. In: *Voting Matters*, 30, pp. 9–13. MacDougall Trust (2013)
- [4] Raab, M., Steger, A.: “Balls into bins” — A simple and tight analysis. In: M. Luby, J.D.P. Rolim, M. Serna (eds.) *Proceedings of the Second International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM 1998)*, *Lecture Notes in Computer Science*, vol. 1518, pp. 159–170. Springer, Berlin (1998)
- [5] Smith, R.: Internet voting and voter interference (2013). (A report prepared for the New South Wales Electoral Commission)
- [6] Wen, R.: Online elections in Terra Australis. Ph.D. thesis, University of New South Wales (2010)
- [7] Wen, R., McIver, A., Morgan, C.: Towards a formal analysis of information leakage for signature attacks in preferential elections. In: C. Jones, P. Pihlajaasaari, J. Sun (eds.) *Proceedings of the 19th International Symposium on Formal Methods (FM 2014)*, *Lecture Notes in Computer Science*, vol. 8442, pp. 595–610. Springer, Berlin (2014)



Chapter 23

Differential privacy

Statistical databases store the data of a large number of individuals, and data analysts are allowed to pose statistical queries about those data. Typical queries include average values, total counting, or the percentage of the entries that satisfy a given property. Such statistical databases are of crucial importance in many areas: for instance, medical databases can guide pharmacological research, and census databases can help authorities decide how to budget for coming years. There is a problem however that general queries like the above can sometimes be processed in a way that reveals detailed information about individuals. The field of *statistical disclosure control* addresses that issue: how do you reveal accurate statistics about a set of individuals while preserving their privacy?

In principle we would like to consider aggregate information as public, and specific information about any individual as private. However, since the two kinds of information are intrinsically linked, it is not easy to make available the former without revealing the latter. Consider, for example, a database that stores the values of the salaries of a set of individuals, and consider the general queries “What is the average salary of the people in the database?” and “How many people are in the database?” Both queries are about aggregate information, but by posing them immediately before and after the addition of a new individual to the database, the data analyst can infer exactly the salary of that individual.

Another important issue in privacy protection is the presence of *side information*, which is any information about individuals coming from sources external to the database itself (e.g. from prior beliefs, public sources, newspapers, or other databases); we discussed that earlier, in Chap. 10, as the Dalenius perspective. The combination of statistical queries and suitable side information can pose serious threats to the privacy of the individuals in the database, and even of those not present in the database (as indeed we saw earlier in Chap. 10, and then again in Chap. 22 where general information about vote tallies, together with external information about which voters were in which electorates, could reveal a voter’s choice). For example, if one knows that a certain person’s salary is exactly the same as the average salary of the people in the database, then the average-salary query will reveal the salary of this person, independently of whether they are in the database or not.

To address the problem of statistical disclosure control, Dwork and her colleagues proposed the notion of *differential privacy*, which has received great attention in the privacy community. Essentially, differential privacy ensures that the presence or absence of any individual in a database, or changing the data of any individual, does not significantly affect the probability of obtaining any specific answer for a certain

query. Intuitively, that implies that it is “safe” for an individual to opt in to (or out of) a database, since their choice will not significantly affect the information the data analyst will obtain.

As we will see, differential privacy includes a parameter ϵ that indicates how strong the privacy guarantee is. One of the main reasons for the success of differential privacy is that it is compositional with respect to that parameter: from the parameters ϵ_1, ϵ_2 of two differentially private components, one can determine an ϵ with respect to which their composition is differentially private. A second advantage is that the definition of differential privacy is independent of side information, which means that the design of the mechanism can abstract from the side information of the adversary.

Differential privacy relies however on the randomization of the query answer, and thus imposes a trade-off in fulfilling two opposing goals. On one hand, *privacy* demands the minimization of the amount of information about the database (and, in particular, about individuals) revealed through the randomized answers. On the other hand, the mechanism should provide good *utility*, i.e. some adequate closeness between true- and randomized answers.

There are several approaches in the literature to designing differentially private mechanisms that offer a good trade-off between privacy and utility. For numeric queries, one typical method consists of adding *Laplace* noise to the true answer to the query. For non-numeric queries, McSherry and Talwar have proposed the *exponential mechanism*.

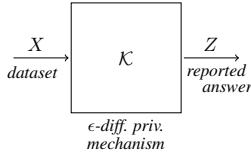
Differential privacy can be seen as having essentially the same goal as quantitative information flow, namely to control the leakage of sensitive information. It is therefore desirable to understand the relationship between those two approaches. In this chapter, we present a number of results showing that *QIF* can shed light on certain aspects of differential privacy. In particular, we show that mechanisms that interrogate statistical databases can be seen as information-theoretic channels, and we find that differential privacy induces a bound on multiplicative Bayes capacity (and hence on multiplicative g -leakage for non-negative gain functions). Moreover, we prove that the bound is tight, i.e. that there always exists a differentially private mechanism that attains that bound. As a result, the bound gives insight into the appropriate choice of the parameter ϵ that is part of the definition of differential privacy.

23.1 Notation and definition

Let \mathcal{X} be the set of all possible databases. Two databases x, x' in \mathcal{X} are *adjacent* (or *neighbors*), written $x \sim x'$, if they differ in the presence of, or in the value associated with, exactly one individual. We call (\sim) the *adjacency relation* on databases. A *mechanism* is a probabilistic function \mathcal{K} from \mathcal{X} to some set of possible answers \mathcal{Z} .¹

Given some $\epsilon \geq 0$, we say that a mechanism is ϵ -differentially private if the ratio between the probabilities of two adjacent databases to give a certain answer is bounded by e^ϵ (where e is the base of natural logarithms).

¹ As will be explained below, the reason for using \mathcal{Z} as the set of possible answers (rather than our usual \mathcal{Y}) is that we will use \mathcal{Y} for the set of *true answers* to the query; those true answers will then be “obscured” to values in \mathcal{Z} by adding noise to them.


 Figure 23.1 Mechanism \mathcal{K}

Definition 23.1 (ϵ -differential privacy) A mechanism \mathcal{K} from \mathcal{X} to \mathcal{Z} satisfies ϵ -differential privacy, for some $\epsilon \geq 0$, if for all pairs x, x' in \mathcal{X} with $x \sim x'$, and all $S \subseteq \mathcal{Z}$, we have

$$\Pr[\mathcal{K}(x) \in S] \leq e^\epsilon \cdot \Pr[\mathcal{K}(x') \in S].$$

Note that since (\sim) is symmetric, the above inequality “applies both ways”. \square

Since in this work we consider finite \mathcal{X}, \mathcal{Z} , all probability distributions are discrete, and so in the above definition it is in fact sufficient to consider probabilities of the form $\Pr[\mathcal{K}(x) = z]$.

The parameter ϵ represents the level of privacy, and clearly the smaller it is, the more indistinguishable the two adjacent databases are. Intuitively, ϵ -differential privacy of \mathcal{K} for a small ϵ means that the presence or absence of an isolated individual has a negligible influence on the query results that \mathcal{K} can return from the database.²

23.2 Mechanisms as information-theoretic channels

Let $\mathcal{I} = \{0, 1, \dots, u-1\}$ be a finite set of cardinality u representing the individuals who might participate in the database, and let $\mathcal{V} = \{val_0, val_1, \dots, val_{v-1}\}$ represent the v different possible values for some sensitive attribute of each individual (e.g. disease name, in a medical database). The case of multiple sensitive attributes can be modeled simply by considering \mathcal{V} as a set of tuples. The absence of an individual from the database is modeled by a special “null” value in \mathcal{V} . A database $x = x_0 \dots x_{u-1}$ is a u -tuple where each x_i in \mathcal{V} is the value associated with the individual i . The set of all databases is therefore $\mathcal{X} = \mathcal{V}^u$. In fact there is a natural graph structure associated with the set of databases, the *Hamming graph*: the vertices of the graph are the databases themselves, and the adjacency relation is defined by $x \sim x'$ if and only if x and x' differ in the value (or presence) of exactly one individual.

A *query* on the database is a (deterministic) function $f: \mathcal{X} \rightarrow \mathcal{Y}$, where \mathcal{Y} is the domain of the true answers $y = f(x)$ to the query. On the other hand, a (noisy) *mechanism* or channel on the database is a probabilistic mapping from \mathcal{X} to \mathcal{Z} , where \mathcal{Z} represents the domain of the answers reported by the mechanism — and it does not necessarily coincide with \mathcal{Y} . We model such a mechanism as a channel matrix $C: \mathcal{X} \rightarrow \mathcal{Z}$.³ The definition of differential privacy can then be directly expressed as a property of the channel: a channel matrix C satisfies ϵ -differential privacy if and only if

$$C_{x,z} \leq e^\epsilon \cdot C_{x',z} \quad \text{for all } x, x' \text{ in } \mathcal{X} \text{ such that } x \sim x', \text{ and all } z \text{ in } \mathcal{Z}.$$

² When we say that a mechanism “is differentially private”, without mentioning ϵ , we mean that there is such an ϵ (and that it is finite).

³ We use (\rightarrow) for deterministic functions and (\rightarrow) for probabilistic functions (which are isomorphic to probabilistic channels). Note that the deterministic case is a special case of the probabilistic one.

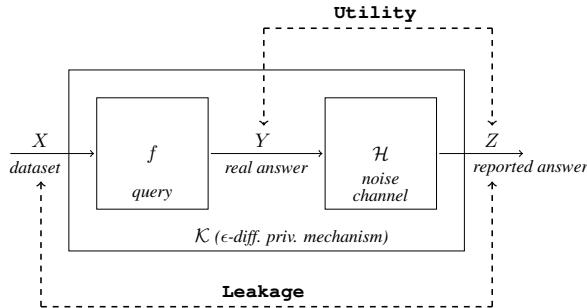


Figure 23.2 Leakage and utility for oblivious mechanisms

For the sake of clarity, sometimes we abuse notation slightly and identify a differentially private mechanism \mathcal{K} with its associated matrix C , thus writing $\mathcal{K}_{x,z}$ for the conditional probability $C_{x,z}$ assigned by the channel matrix.

Given a distribution on \mathcal{X} , the function f determines a distribution on \mathcal{Y} , and the mechanism \mathcal{K} determines a distribution on \mathcal{Z} . We will use X , Y , and Z to represent the random variables corresponding to \mathcal{X} , \mathcal{Y} , and \mathcal{Z} and their distributions, respectively. Thus, \mathcal{K} can be seen as a transformation from X to Z (as shown in Fig. 23.1), just as f can be seen as a transformation from X to Y . We say that \mathcal{K} is *oblivious* if the reported answer Z depends only on the true answer Y and not at all on the database X . In that case, the (oblivious) channel \mathcal{K} can be decomposed into a deterministic channel from X to Y modeling the query f and a noisy, probabilistic channel \mathcal{H} from Y to Z modeling the addition of noise to the true answer. Those two channels are *in cascade*, since the output of the first one is the input for the second one, as in Fig. 23.2.

The *leakage* of the channel associated with the mechanism is a measure of the information about the database in \mathcal{X} that the adversary can obtain by observing the reported answer in \mathcal{Z} , and hence it represents a relation between X and Z . On the other hand, the *utility* of the mechanism is a measure of how much one can learn about the true answer in \mathcal{Y} from one reported in \mathcal{Z} , hence it represents a relation between Y and Z . Note that in an oblivious mechanism the utility depends only on the noise channel \mathcal{H} .

23.3 The relation between differential privacy and multiplicative g -leakage

Both quantitative information flow and differential privacy measure the protection of sensitive information provided by a mechanism, and in both cases this measure is linked to the probabilistic knowledge that an adversary gains about the secret from the outcome of the mechanism. It is therefore natural to ask how they are related.

In this section we answer this question positively by showing that differential privacy induces a bound on the multiplicative Bayes capacity, and therefore on the multiplicative g -leakage for non-negative gain function g . Furthermore, we prove that the bound is tight, i.e. that there always exists a differentially private mechanism that attains the bound. On the other hand, the opposite inference does *not* hold — the multiplicative Bayes capacity of a mechanism does not allow us to establish a finite bound (that is $\epsilon < \infty$) on the level of differential privacy.

We recall that we see a mechanism for answering database queries as a concrete channel $C: \mathcal{X} \rightarrow \mathcal{Z}$ where $\mathcal{X} = \mathcal{V}^u$ represents the set of the databases of u individuals, and \mathcal{V} is a set of cardinality v representing the possible values of the individuals. We have the following bound.

Theorem 23.2 If C satisfies ϵ -differential privacy, then its multiplicative Bayes capacity is bounded from above by

$$\mathcal{ML}_1^\times(\mathbb{D}, C) \leq \left(\frac{v e^\epsilon}{v - 1 + e^\epsilon} \right)^u .$$

□

We omit the proof here; see the Chapter Notes for a discussion.

And from the Miracle theorem (Thm. 7.5) we then derive immediately that the above is a bound for g -leakage as well.

Corollary 23.3 If C satisfies ϵ -differential privacy, then for any prior π and non-negative gain function $g: \mathbb{G}^+ \times \mathcal{X}$ we have

$$\mathcal{L}_g^\times(\pi, C) \leq \left(\frac{v e^\epsilon}{v - 1 + e^\epsilon} \right)^u .$$

□

The bound $Bnd(u, v, \epsilon) = (v e^\epsilon / v - 1 + e^\epsilon)^u$, on the right-hand side just above, is a continuous function of ϵ . As expected, the bound is 1 when $\epsilon = 0$, since in that case all rows of the mechanism are identical, leading to no leakage.⁴ Also, the bound converges to v^u as ϵ approaches infinity. Intuitively, that accords with the observation that the ratio between two elements in the same column of the matrix could increase without bound as ϵ approaches infinity. In that case, the outputs could uniquely indicate which input caused them, and the probability of success for the adversary, i.e. the posterior Bayes vulnerability would be 1 in this extreme case. That would make the multiplicative leakage of the mechanism equal to the reciprocal of the prior Bayes vulnerability; and since the prior vulnerability is least when the distribution on all possible databases is uniform, the worst-case leakage is bounded by v^u .

The next theorem shows that the bound obtained above is tight.

Theorem 23.4 For every u , v , and ϵ it is possible to define a mechanism C that satisfies ϵ -differential privacy and whose multiplicative Bayes leakage on the uniform prior (i.e. its multiplicative Bayes capacity) is $\mathcal{ML}_1^\times(\mathbb{D}, C) = Bnd(u, v, \epsilon)$.

Proof. Let $\mathcal{Z} = \mathcal{X}$ and define channel matrix C as follows:

$$C_{x,z} := \frac{Bnd(u, v, \epsilon)}{v^u (e^\epsilon)^{d(x, z)}} \quad \text{for every input } x \text{ and output } z.$$

Here $d(x, z)$ denotes the distance in the Hamming Graph between x and z . It is easy to see that C satisfies ϵ -differential privacy and that its multiplicative Bayes capacity is $Bnd(u, v, \epsilon)$. □

⁴ Applying Def. 23.1 gives directly only that (\sim) -adjacent rows are equal; but since any x and x' can be related by a finite number of (\sim) -steps in between, the equality applies in general by transitivity.

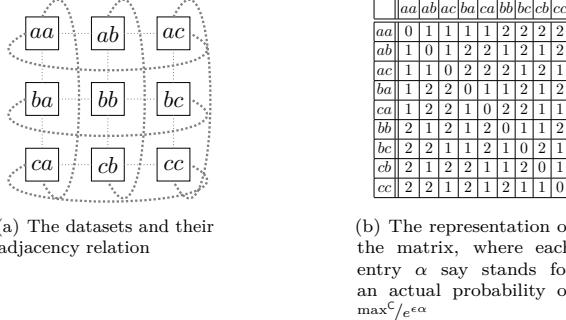


Figure 23.3 All possible databases and the matrix with highest multiplicative Bayes leakage that gives ϵ -differential privacy for Example 23.5

Deciding what makes a particular parameter ϵ appropriate for differential privacy is not trivial, and in general one must consider several factors, depending on the particular threats that are of concern; note that they might be modeled with different gain functions g . But if we focus on Bayes vulnerability, then the relationship we have just shown above gives important insight into the implications of the choice of a particular ϵ . For instance, if $u=100$ and $v=2$, then with $\epsilon=5$ we find that the multiplicative Bayes capacity can be as high as $2^{99.03}$, giving a posterior Bayes vulnerability (for the uniform prior) of over $1/2$, which intuitively means that this combination of parameters is not safe.

We now give an example of the use of $Bnd(u, v, \epsilon)$ as a bound for the Bayes leakage.

Example 23.5 Assume we are interested in the eye color of a certain population $\mathcal{I} = \{Alice, Bob\}$. Let $\mathcal{V} = \{a, b, c\}$, where a stands for *absent* (the null value) and b stands for *blue* and c stands for *coal black*. Each dataset is a tuple $x_0x_1 \in \mathcal{V}^2$, where x_0 represents the eye color of *Alice* (cases $x_0 = b$ and $x_0 = c$) or that *Alice* is not in the dataset (case $x_0 = a$), whereas x_1 provides the same information for *Bob*. Note that $v=3$. Figure 23.3(a) represents the graph structure (\mathcal{X}, \sim) of the database domain, i.e. the set \mathcal{X} of all possible datasets and its adjacency relation. Figure 23.3(b) represents the matrix with input \mathcal{X} that provides ϵ -differential privacy and has the highest multiplicative Bayes leakage. In the figure, an entry α say stands for an actual probability of $\max^C / e^{\epsilon \alpha}$ where \max^C is the highest value in the matrix, namely

$$\max^C = \left(\frac{ve^\epsilon}{v - 1 + e^\epsilon} \right)^u \frac{1}{v^u} = \frac{e^{2\epsilon}}{(2 + e^\epsilon)^2} .$$

□

23.3.1 Bounds on leakage do not imply differential privacy

We noted above that the converse of Thm. 23.2 does not hold, i.e. a bound on the leakage does not necessarily allow us to derive a bound on the level of differential privacy of that channel (i.e. on the parameter ϵ). One reason is that the vulnerability is defined as an expected value, i.e. it is the result of averaging the contribution of all the columns to the leakage, while differential privacy represents the worst case. Hence there could be a column that breaks differential privacy entirely, for instance

π	C	z_1	z_2	\dots	z_m
α	x_1	β	$\frac{1-\beta}{m-1}$	\dots	$\frac{1-\beta}{m-1}$
$\frac{1-\alpha}{n-1}$	x_2	$\frac{1}{m}$	$\frac{1}{m}$	\dots	$\frac{1}{m}$
\vdots	\vdots	\vdots	\vdots	\ddots	\vdots
$\frac{1-\alpha}{n-1}$	x_n	$\frac{1}{m}$	$\frac{1}{m}$	\dots	$\frac{1}{m}$

Figure 23.4 Prior distribution and channel matrix for Example 23.6

in the case where the column contains both zero- and nonzero elements, and yet whose leakage does not contribute very much to the average (typically because the corresponding output has very low probability). In that case, the Bayes leakage could be very small, and still ϵ -differential privacy would not hold for any ϵ .

Another difference is that leakage is sensitive to the prior distribution, whereas differential privacy is not. The following example illustrates that.

Example 23.6 Let $C: \mathcal{X} \rightarrow \mathcal{Z}$ be a channel such that $|\mathcal{X}| = n$ and $|\mathcal{Z}| = m$. Assume that the prior π is given by $\pi_{x_1} = \alpha$ and $\pi_{x_i} = \frac{1-\alpha}{n-1}$ for $2 \leq i \leq n$, and let the channel matrix C be given by $C_{x_1, z_1} = \beta$ and $C_{x_1, z_j} = \frac{1-\beta}{m-1}$ for $2 \leq j \leq m$ and $C_{x_i, z_j} = \frac{1}{m}$ otherwise. That channel is represented in Figure 23.4, and simple calculations show that the multiplicative Bayes leakage of the channel approaches 0 as α approaches 0, independently of the value of β .

The differential privacy of the channel, however, depends only on the value of β — more precisely, assuming that the adjacency relation is such that $x_1 \sim x_2$ we find that the parameter ϵ of differential privacy is

$$\max \left\{ \ln \frac{1}{m\beta}, \ln m\beta, \ln \frac{m-1}{m(1-\beta)}, \ln \frac{m(1-\beta)}{m-1} \right\} ,$$

which increases without bound as β approaches 0.

Thus we have that the multiplicative Bayes leakage is affected by α , a parameter of the prior, and is independent of β ; yet the differential privacy is affected only by β , and is independent of α . \square

23.4 Exercises

Exercise 23.1 We have a number u of individuals and a database x that contains records representing the ages of those individuals, expressed as a natural number. We assume that all individuals are present in the database, and that they all have an age between 1 and 100.

Now consider the query “What is the average age of the individuals present in the database, rounded to the closest natural number?” For each of the three following mechanisms for answering that query, say whether or not it is differentially private, and compute the multiplicative Bayes capacity of the corresponding channel. If your answer is “Yes, it is differentially private,” give the smallest privacy parameter ϵ that applies.

- (a) $\mathcal{K}_1(x) = 25$.
- (b) $\mathcal{K}_2(x)$: Flip a fair coin. If it comes up heads, then give as the result of the query the average age of the people in x (rounded to the closest natural number). Otherwise, give the result 25.
- (c) $\mathcal{K}_3(x)$: Flip a fair coin. As just above, if it comes up heads give as the result the average age of the people in x (rounded to the closest natural number). But otherwise, this time, give a random result: a number chosen randomly (i.e. with uniform probability) between 1 and 100.⁵

Would any of the above answers change if we removed the rounding to the closest natural number?

□

Exercise 23.2 (Compositionality) Given two mechanisms $\mathcal{K}_1: \mathcal{X} \rightarrow \mathcal{Z}_1$ and $\mathcal{K}_2: \mathcal{X} \rightarrow \mathcal{Z}_2$, we define their composition $\mathcal{K}_1 \times \mathcal{K}_2$ as a mechanism of type $\mathcal{X} \rightarrow \mathcal{Z}_1 \times \mathcal{Z}_2$ that generates the answer from input x by applying \mathcal{K}_1 and \mathcal{K}_2 independently to x to obtain the first and second elements of the output pair, respectively. Prove that differential privacy is compositional in the following sense: if \mathcal{K}_1 and \mathcal{K}_2 are differentially private with parameters ϵ_1 and ϵ_2 respectively, then $\mathcal{K}_1 \times \mathcal{K}_2$ is $(\epsilon_1 + \epsilon_2)$ -differentially private.

□

Exercise 23.3 (Preservation of differential privacy under post-processing)

Prove that post-processing (composition in cascade) preserves differential privacy. That is, given an ϵ -differentially private mechanism $\mathcal{K}: \mathcal{X} \rightarrow \mathcal{Y}$ and a channel matrix $\mathcal{H}: \mathcal{Y} \rightarrow \mathcal{Z}$, prove that their cascade \mathcal{KH} (Def. 4.18) is also ϵ -differentially private.

□

Exercise 23.4 (Counting queries and geometric mechanisms) Let u be the number of individuals in the database, and let \mathcal{Y} be the set $\{0, 1, \dots, u\}$. A *counting query* is a function $f: \mathcal{X} \rightarrow \mathcal{Y}$ such that $f(x)$ is the number of individuals in x who satisfy a certain property (for instance, the number of individuals who have a degree in computer science). Given a number α in $(0, 1]$, the (*truncated*) *geometric mechanism* with parameter α is an oblivious mechanism (§23.2) whose noise channel \mathcal{H} is a probabilistic function from \mathcal{Y} to \mathcal{Y} defined as follows:

$$\mathcal{H}_{m,n} = c_n \alpha^{|n-m|} \quad \text{where} \quad c_n = \begin{cases} \frac{1-\alpha}{1+\alpha} & \text{if } 0 < n < u \\ \frac{1}{1+\alpha} & \text{if } n = 0 \text{ or } n = u \end{cases}.$$

Let \mathbf{C} be the matrix obtained by composing f and \mathcal{H} in cascade, i.e. so that for all x we have $\mathbf{C}_{x,n} = \mathcal{H}_{f(x),n}$.

- (a) Prove that \mathbf{C} is always a channel matrix, i.e. is stochastic: that for all databases x we have $\sum_{n=0}^u \mathbf{C}_{x,n} = 1$.
- (b) Prove that \mathbf{C} is ϵ -differentially private for some ϵ , and establish the smallest such ϵ for which that is so.
- (c) Compute the multiplicative Bayes capacity of \mathbf{C} .

□

⁵ This is a variation on Warner's protocol, also mentioned in the Notes for Chap. 8.

23.5 Chapter notes

Differential privacy originated in the works of Dwork and her collaborators [9, 6, 8, 7]. Mechanisms for achieving differential privacy include Laplace noise [9] for numeric queries, the exponential mechanism [13] for non-numeric queries, and more sophisticated mechanisms that correlate noise between queries, enabling more information to be extracted from the database while still preserving differential privacy [3, 14, 11].

Barthe and Köpf [2] were the first to investigate the connection between differential privacy and multiplicative Bayes leakage over the set of all possible databases. They consider “end-to-end” differentially private mechanisms, which correspond to what in this chapter we call the mechanism \mathcal{K} , and interpret those mechanisms as information-theoretic channels. They provide a bound for leakage, but point out that it is not tight in general.

The tight bound in Thm. 23.2 was first shown by Alvim et al. [1], using a quite intricate proof. Later, ElSalamouny, Chatzikokolakis, and Palamidessi [10] gave a simpler proof of the same bound.

The relation between differential privacy and information theory has been investigated also in other works, mainly in the context of Shannon information theory. McGregor et al. [12] gave an upper bound on mutual information for differential privacy in a two-party differential-privacy setting. Later this upper bound was used by De [5] to prove that $I(X^u; Y) \leq 3\epsilon u$, where X^n is a database with u entries and Y is the output of a differentially private mechanism. Wang et al. [15] introduced the concept of “mutual-information privacy” metric. Cuff and Yu [4] observed that, in order to properly characterize differential privacy as an information-theoretic concept, it is necessary to model the implicit assumption of differential privacy on the knowledge of the adversary, namely the assumption that an adversary trying to exploit the difference between two adjacent databases x and x' knows the value of all the records in common between x and x' . They proposed the notion of “mutual-information differential privacy” (MI-DP), where the adversarial knowledge is modeled using the concept of conditional mutual information, and showed that MI-DP is sandwiched between ϵ -DP and a more relaxed version of DP called (ϵ, δ) -DP.

Bibliography

- [1] Alvim, M.S., Andrés, M.E., Chatzikokolakis, K., Degano, P., Palamidessi, C.: On the information leakage of differentially-private mechanisms. *Journal of Computer Security* **23**(4), 427–469 (2015)
- [2] Barthe, G., Köpf, B.: Information-theoretic bounds for differentially private mechanisms. In: Proceedings of the 2011 IEEE 24th Computer Security Foundations Symposium (CSF 2011), pp. 191–204. IEEE, Los Alamitos (2011)
- [3] Blum, A., Ligett, K., Roth, A.: A learning theory approach to non-interactive database privacy. In: Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC 2008), pp. 609–618. ACM, New York (2008)
- [4] Cuff, P., Yu, L.: Differential privacy as a mutual information constraint. In: Proceedings of the 23rd ACM Conference on Computer and Communications Security (CCS 2016), pp. 43–54. ACM, New York (2016)
- [5] De, A.: Lower bounds in differential privacy. In: R. Cramer (ed.) *Proceedings of the 9th Theory of Cryptography Conference (TCC 2012)*, *Lecture Notes in Computer Science*, vol. 7194, pp. 321–338. Springer, Berlin (2012)
- [6] Dwork, C.: Differential privacy. In: M. Bugliesi, B. Preneel, V. Sassone, I. Wegener (eds.) *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP 2006)*, *Lecture Notes in Computer Science*, vol. 4052, pp. 1–12. Springer, Berlin (2006)
- [7] Dwork, C.: A firm foundation for private data analysis. *Communications of the ACM* **54**(1), 86–95 (2011)
- [8] Dwork, C., Lei, J.: Differential privacy and robust statistics. In: M. Mitzenmacher (ed.) *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC 2009)*, pp. 371–380. ACM, New York (2009)
- [9] Dwork, C., McSherry, F., Nissim, K., Smith, A.: Calibrating noise to sensitivity in private data analysis. In: S. Halevi, T. Rabin (eds.) *Proceedings of the 3rd Theory of Cryptography Conference (TCC 2006)*, *Lecture Notes in Computer Science*, vol. 3876, pp. 265–284. Springer, Berlin (2006)
- [10] ElSalamouny, E., Chatzikokolakis, K., Palamidessi, C.: Generalized differential privacy: Regions of priors that admit robust optimal mechanisms. In: F. van Breugel, E. Kashefi, C. Palamidessi, J. Rutten (eds.) *Horizons of the Mind. A Tribute to Prakash Panangaden*, *Lecture Notes in Computer Science*, vol. 8464, pp. 292–318. Springer, Berlin (2014)

Bibliography

- [11] Hardt, M., Rothblum, G.N.: A multiplicative weights mechanism for privacy-preserving data analysis. In: Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science (FOCS 2010), pp. 61–70. IEEE Computer Society, Washington, DC (2010)
- [12] McGregor, A., Mironov, I., Pitassi, T., Reingold, O., Talwar, K., Vadhan, S.: The limits of two-party differential privacy. In: Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science (FOCS 2010), pp. 81–90. IEEE Computer Society, Washington, DC (2010)
- [13] McSherry, F., Talwar, K.: Mechanism design via differential privacy. In: Proceedings of the 2007 IEEE 48th Annual Symposium on Foundations of Computer Science (FOCS 2007), pp. 94–103. IEEE Computer Society, Washington, DC (2007)
- [14] Roth, A., Roughgarden, T.: Interactive privacy via the median mechanism. In: Proceedings of the 42nd Annual ACM Symposium on Theory of Computing (STOC 2010), pp. 765–774. ACM, New York (2010)
- [15] Wang, W., Ying, L., Zhang, J.: On the relation between identifiability, differential privacy, and mutual-information privacy. In: Proceedings of the 2014 52nd Annual Allerton Conference on Communication, Control, and Computing (CCCC 2014), pp. 1086–1092. IEEE, Los Alamitos (2014)

Glossary and Index

Symbols are listed in order of occurrence, with “*see*” intended to give an immediate indication of the meaning; more however can be found at that main entry, including a page number.

The reference *qv* cross-refers to the index entry of the second significant word; thus “structural refinement *qv*” cross-refers to main entry “refinement”.

A down-arrow ↓ means the item is in a footnote; a **bold** page-number indicates a definition (either explicit and numbered, or possibly run-in — and there might be several). A § refers to a whole (sub-)section.

References to people are indexed if the reference is in the main text (i.e. not within Chapter Notes) or is cited in the Preface or in a chapter’s bibliography. In the last case, its index entry is within cite-brackets and refers to the beginning of the section in which it appears.

Glossary

— Chapter 1 —

- ϑ *see* distribution, uniform
 π_x *see* distribution, probability of x
 C *see* concrete channel
 $\mathcal{L}_1^+(\pi, C)$ *see* additive (Bayes)
leakage
 $\mathcal{L}_1^\times(\pi, C)$ *see* multiplicative (Bayes)
leakage
 (\sqsubseteq) *see* refinement order

— Chapter 2 —

- $[\delta]$ *see* support, of distribution
 $\mathbb{D}\mathcal{X}$ *see* distribution type over \mathcal{X}
 $(1/2, 1/6, 0, 1/3)$ *see* distribution
written as tuple of probabilities
 $H(\pi)$ *see* Shannon entropy
 $[x]$ *see* point distribution
 $:=$ *see* definitions, used for
 \coloneqq *see* assignment statement,
probabilistic
 $\pi^{\text{coin}}, \pi^{\text{dice}}, \pi^{\text{mother}}$ *see* prior
distribution, examples
 $H_\infty(\pi)$ *see* Rényi min-entropy
 $V_1(\pi), V_3(\pi)$ *see* Bayes n -guess
vulnerability

— Chapter 3 —

- $V_g(\pi)$ *see* vulnerability, of π according
to gain function g
 G *see* gain function, represented
as matrix
 g -function *see* gain function
 $g(w, x)$ *see* gain function, reward
 $U_\ell(\pi)$ *see* uncertainty, of π according
to loss function ℓ
 g_{id} *see* gain-function catalog,
identity
 $g_{\sigma^{-1}}, g_{\pi^{-1}}$ *see* gain-function catalog,
distribution reciprocal
 g^c *see* gain-function catalog,
complemented
 g_{Gates} *see* Bill Gates
 g_{tiger} *see* gain function, pit of tigers
 ℓ_G *see* guessing entropy
 ℓ_H *see* Shannon entropy
 $\mathbb{G}\mathcal{X}$ *see* gain-function class

$\mathbb{G}^{\text{fin}}\mathcal{X}$ *see* gain-function classes, finite-action
 $\mathbb{G}^+\mathcal{X}$ *see* gain-function classes, non-negative
 $\mathbb{L}\mathcal{X}$ *see* loss-function class
 $\mathbb{G}^\dagger\mathcal{X}$ *see* gain-function classes, one-bounded
 $g_{\times k}$ *see* gain function, algebra, scaling
 $g_{+\kappa}$ *see* gain function, algebra, shifting

 — Chapter 4 —
 $C_{-,y}$ *see* channel, matrix, column
 $C_{x,-}$ *see* channel, matrix, row
 $C_{x,y}$ *see* channel, matrix, element
 C *see* channel, matrix, written as
 $p(y|x)$ *see* conditional probability
 $\mathcal{X} \rightarrow \mathcal{Y}$ *see* matrix, typed
 p_X *see* marginal distribution
 p_{XY} *see* joint distribution
 $p(X|y)$ *see* posterior distribution
 $p(x|y)$ *see* conditional probability
 $[\pi \triangleright C]$ *see* hyper distribution, made from π and C
 Δ *see* hyper-distribution, usually written
 Δ_δ *see* hyper-distribution, outer probability
 $\pi \triangleright C$ *see* joint distribution (matrix)
 $\mathbb{D}^2\mathcal{X}$ *see* hyper-distribution, has type
 $[\![C]\!]$ *see* semantic brackets
 J *see* pushing prior through channel
 (\circ) *see* functional composition
 μ *see* expected value, multiply of monad
 id *see* identity function
 $[\pi]$ *see* point hyper-distribution
 $\mathbf{1}$ leaks nothing *see* unit channel
 $\mathbf{0}$ leaks everything *see* zero channel
 $[\delta^i]$ *see* sub-point hyper-distribution

 — Chapter 5 —
 $\mathcal{E}_\alpha F$ *see* expected value of function F over distribution α
 $\mathcal{L}_g^+(\pi, C)$ *see* additive g -leakage
 $\mathcal{L}_g^\times(\pi, C)$ *see* multiplicative g -leakage

$I(X; Y)$ *see* Shannon entropy (mutual information)
 tr *see* trace of matrix
 $[\![\pi]\!]$ *see* matrix with π on diagonal

 — Chapter 6 —
 — Chapter 7 —
 $\mathcal{ML}_1^\times(\mathbb{D}, C)$ *see* multiplicative Bayes capacity
 (\curvearrowright) *see* concave functions
 $\mathcal{ML}_1^+(\mathbb{D}, C)$ *see* additive Bayes capacity
 $\mathcal{ML}_g^+(\mathbb{D}, C)$ *see* additive capacity
 $\mathcal{ML}_g^\times(\mathbb{D}, C)$ *see* multiplicative capacity
 g_H *see* Shannon-like gain function

 — Chapter 8 —
 (\parallel) *see* parallel composition
 \uplus *see* disjoint union
 $C_{\mathcal{X}}$ *see* compatible channels on \mathcal{X}
 $(_P \boxplus)$ *see* external probabilistic choice
 $(\triangleleft \mathcal{A} \triangleright)$ *see* external conditional choice
 $(_P \boxplus)$ *see* external general choice
 $(_P \oplus)$ *see* internal probabilistic choice
 (\equiv) *see* semantic equivalence
 \heartsuit, \diamondsuit *see* compositionality
 $(\parallel\parallel)$ *see* parallel composition, concrete
 $(_P \oplus)$ *see* internal general choice
 $(\ll \mathcal{A} \gg)$ *see* internal conditional choice

 — Chapter 9 —
 (\sqsubseteq_o) *see* structural refinement
 (\sqsubseteq_G) *see* testing refinement
 (\preccurlyeq) *see* strong Bayes-vulnerability order
 C *see* context of deployment
 $(\sqsubseteq_?)$ *see* potential refinement order
 K *see* kludge
 $V_n(\pi)$ *see* Bayes n -guess vulnerability
 C *see* context of deployment (set of)

 — Chapter 10 —
 V_g^D *see* Dalenius g -vulnerability
 V_1^D *see* Dalenius Bayes vulnerability

$\mathcal{DL}_g^{+/\times}$	<i>see</i> Dalenius g -leakage	
\sqsubseteq^D	<i>see</i> (strong) Dalenius g -vulnerability order	
— Chapter 11 —		— Chapter 17 —
$\hat{\mathbb{V}}$	<i>see</i> generic posterior vulnerability	$\mathbb{E}\mathcal{X}$ <i>see</i> equivalence relations over \mathcal{X}
\mathbb{V}	<i>see</i> generic prior vulnerability	$\mathbb{P}\mathcal{X}$ <i>see</i> powerset type over \mathcal{X}
\mathbb{V}^λ	<i>see</i> non-continuous vulnerability function	$(\cdot), (\times)$ <i>see</i> matrix multiplication
\max_S	<i>see</i> max	P^\cup <i>see</i> union closure
$\hat{H}_\alpha(\Delta)$	<i>see</i> Rényi entropy, posterior	\mathbb{U} <i>see</i> union closed, subsets
— Chapter 12 —		\emptyset <i>see</i> union closure
— Chapter 13 —		(\preccurlyeq) <i>see</i> refinement, primitive
{...}	<i>see</i> local scope	
[b]	<i>see</i> square brackets, Iverson	
— Chapter 14 —		— Chapter 18 —
HMM	<i>see</i> Hidden Markov Model	D_1 <i>see</i> Crowds protocol, probability of detecting some user on the first step
$[\delta]$	<i>see</i> normalization of subdistribution δ	$D_{\geq 2}$ <i>see</i> Crowds protocol, probability of detecting some user on the second or subsequent step
$\langle \delta \rangle$	<i>see</i> weight of subdistribution δ	s <i>see</i> Crowds protocol, no user detected
\mathbb{D}	<i>see</i> subdistribution	g_{lion} <i>see</i> gain-function catalog, pit of lions
(C;M)	<i>see</i> Hidden Markov Model, concrete	— Chapter 19 —
•	<i>see</i> unit type	D_{KL} <i>see</i> Kullback-Leibler distance
[J]	<i>see</i> hyper-distribution, made from joint distribution J	$cap_b(n)$ <i>see</i> blinding- and bucketing capacity
[[H]]	<i>see</i> hidden Markov matrix, denotation of	$[z^n]f(z)$ <i>see</i> Taylor series
\mathbb{N}	<i>see</i> natural numbers	— Chapter 20 —
$(p \oplus)$ vs. $(p \boxplus)$ vs. $(p +)$	<i>see</i> probabilistic choice	— Chapter 21 —
$C^{(n)}, P^{(n)}$	<i>see</i> repeated independent runs	— Chapter 22 —
— Chapter 15 —		(#) <i>see</i> multiset, membership of
(∇)	<i>see</i> exclusive-or	\mathbb{M} <i>see</i> multiset, constructor
(\sqcap)	<i>see</i> demonic choice	\mathbb{S} <i>see</i> non-repeating-sequences constructor
$\mathbb{Exp} Y[x \setminus \mathbb{Exp} X]$	<i>see</i> substitution	\mathcal{B} <i>see</i> see election ballot
$(x \nabla y) : \in \mathbb{Exp}$	<i>see</i> assignment statement, such that	\mathcal{C} <i>see</i> see election candidate
— Chapter 16 —		\mathcal{X} <i>see</i> election ballot box
\perp	<i>see</i> nontermination	\mathcal{T} <i>see</i> election tally, channel
$\mathbb{DD}\mathcal{X}$	<i>see</i> sub-hyper-distribution	\mathcal{Y} <i>see</i> see election, outcome
(\sqsubseteq_P)	<i>see</i> partial refinement	\mathcal{E} <i>see</i> election elector
(\sqsubseteq_T)	<i>see</i> total refinement	\mathcal{A} <i>see</i> election casting, channel
(\sqsubseteq_\perp)	<i>see</i> termination refinement	\mathcal{Z} <i>see</i> election voting pattern
$\Delta \sim \Delta'$	<i>see</i> sub-hyper distributions, weights are equal	\mathcal{C} <i>see</i> all-tallies, channel
		\mathcal{E} <i>see</i> election channel
		\mathcal{W} <i>see</i> winning candidate, channel
		\mathcal{Y}_C <i>see</i> all tallies, observables
		\mathcal{Y}_W <i>see</i> winning candidate, observables
		\mathcal{F} <i>see</i> full preferences, channel
		\mathcal{P} <i>see</i> partial preferences, channel

Glossary and Index

\mathcal{Y}_F	<i>see</i> full preferences, observables	— Chapter 23 —
\mathcal{Y}_P	<i>see</i> partial preferences, observables	(\sim) <i>see</i> database, adjacency relation ϵ <i>see</i> ϵ -differential privacy
$g_1, g_\#, g_0$	<i>see</i> gain functions for elections	$Bnd(u, v, \epsilon)$ <i>see</i> bound for g -leakage in differential privacy

Index

A

- A** *see* election, casting channel
- ABORT**
 - is the everywhere nonterminating program [308, 309](#)
 - is a right-zero for sequential composition [312](#)
 - is the (\sqsubseteq) -least program [311, 313](#)
 - leaks potentially everything [312](#)
 - stands for all program failures [311](#)
- [J.-R. Abrial] [247](#)
- abstract channel [56, 57, 57](#)§
 - deterministic [60](#)
 - is semantics [159, 238](#)
 - origin of [67](#)
 - see also* concrete channel is syntax, refinement order
- abstract vs. concrete channels [173](#)↓
- abstraction
 - induced by partitions [328](#)
 - layers [399](#)
- Access Denied** [vii](#)
- actions \mathcal{W}
 - g -vulnerability, ℓ -uncertainty are envelopes [37, 209](#)
 - as subsets of \mathcal{X} *see* gain-function catalog, binary infinite set of actions [37, 209](#)
 - usually restricted to be finite [40](#)
- [Onur Aciçmez] [12](#)
- ad-hoc privacy goal [179](#)
- ad-omnia privacy goal [179](#)
- adaptive adversary *qv*
- additive Bayes capacity $\mathcal{ML}_1^+(\mathbb{D}, C)$ of channel C [111](#)
 - not necessarily realized on uniform prior [111](#)
 - realized on sub-uniform prior [112, 119, 166](#)
- additive capacity
 - fixed π — $\mathcal{ML}_{\mathbb{G}}^+(\pi, C)$ [120](#)§
 - fixed g — $\mathcal{ML}_g^+(\mathbb{D}, C)$ [119](#)§
 - one-bounded — $\mathcal{ML}_{\mathbb{G}^1}^+(\mathbb{D}, C)$ [123](#)§, [127](#)
- additive capacity $\mathcal{ML}_g^+(\mathcal{D}, C)$ of channel C over \mathcal{D} [116](#)

- additive capacity $\mathcal{ML}_{\mathbb{G}}^+(\pi, C)$ of channel C over π [120](#)
- additive leakage [9](#)§
 - g -leakage [80](#)
 - invariant under shifting [81](#)
 - of prior π and channel C wrt. gain function g is $\mathcal{L}_g^+(\pi, C)$ [80](#)
 - Bayes additive leakage $\mathcal{L}_1^+(\pi, \mathbb{C})$ of prior π through channel \mathbb{C} [9](#)
 - limits to [107](#)
- additivity
 - of functions on uncertainty [199](#) ff
- adjacency relation (\sim) *see* differential privacy
- adversary
 - actions bring rewards [25](#)
 - adaptive [97, 200](#)
 - assumed to be information theoretic [21, 54](#)↓, [161](#)↓
 - benevolent [35, 72](#)
 - can read the source code [viii, 51](#)↓, [225, 239, 242, 246](#)
 - debugger (single-step) [242](#)
 - has perfect recall *see* power of adversary
 - is feminine [xi](#)
 - knows how the system works [51](#)
 - knows secret's distribution only, not its value [5, 17](#)
 - model of *see* power of adversary
 - multiple ($\text{HID}_A, \text{VIS}_A$) [17, 285, 287](#)
 - optimal strategy [8, 28, 34, 39, 44, 73, 76, 82, 87–90, 120, 153, 155, 175](#)↓, [175, 359, 360, 423](#)
 - possible actions might include “do nothing” [395](#)
 - sometimes called “agent” [285](#)
 - strategy *qv*
 - thinks about what we are thinking [101](#)
 - updates knowledge from prior p_X to posterior $p_{X|y}$ [71](#)
- AES** *see* side-channel attacks
- Agent X [400](#)
- Agents A, B, \dots [287](#)

- agents are adversaries 285
 all-tallies
 channel C 417
 observables \mathcal{Y}_C 417
 alleles 139
 almost-sure termination qv
 The Alphabet Spies 334 §
 [Arthur Américo] xii, 143
 [Miguel E. Andrés] xii, 66, 199, 441
 anonymity
 protocols x
 see also Crowds protocol
 anonymizing channel *see* election,
 casting channel A
 antisymmetry
 of refinement (\sqsubseteq) 168
 for probabilistic channels 158
 of refinement (\sqsubseteq) for demonic
 channels 332
 of structural refinement (\sqsubseteq_\circ) qv
 “any fool can see” qv
 [Suguru Arimoto] 127
 assignment statement
 conventional $x := \text{Exp}$ 265
 probabilistic $X \in \text{Exp}$
 defines random variable X
 according to distribution Exp
 21
 probabilistic $x \in \text{Exp}$
 assigns to program-variable x
 according to distribution Exp
 226, 234, 258, 264
 such that $(x \nabla y) \in \text{Exp}$ 297,
 402↓
 Australia, states of, example of
 refinement 148
 authors and readers
 are plural xi
 average
 of posterior distributions 53, 228
 see also expected value, multiply
 μ
 averaging
 generic axiom AVG *see* axioms,
 generic
 AVG *see* axioms, generic
 axioms
 for gain- and loss functions 183 §
 for leakage 183
 axioms, generic
 expressed monadically 277
 of averaging (AVG) 190, 277
 of bounds (BNDS) 195
 of continuity (CNTY) 185
 of convexity (CVX) 185, 216, 277
 of data-processing inequality
 (DPI) 188
 of maximum (MAX) 192
 of monotonicity (MONO) 189
 of noninterference (NI) 188, 277
 of quasiconvexity (Q-CVX) 186

B

- \mathcal{B} *see* election ballot
 B method (of Abrial) for program
 development via refinement
 247
 [Ralph-Johan Back] 247, 346
 [Michael Backes] 97, 127
 balance between safety and
 performance 395
 barbecue set used underwater 161↓
 [Gilles Barthe] 366, 441
 barycentric representation 213–216
 of distributions 28, 34, 205
 of entropies 209
 of gain/loss functions 28, 205
 of hypers 208
 of inners 53
 utility of interpolation 208
 base-rate fallacy 83
 [David A. Basin] 66, 97
 [Leonard E. Baum] 279
 Bayes n -guess vulnerability
 $V_n(\pi)$ of distribution π 164
 when $n=3$ 22, 32, 102
 Bayes (1 guess) vulnerability 5 §, 22
 $V_1(\pi)$ of distribution π 5, 20
 (not) maximum over all inners 8
 as a game
 posterior 82
 prior 21
 as partition gain function 32
 can decrease in the dynamic view
 71
 Dalenius 172
 generalized by g -vulnerability as
 $V_{\text{gid}}(-)$ 30
 induced by discrete metric 31

- induces primitive refinement (\preccurlyeq)
 340
- is insufficiently general 21, 25
- max case 357
- maximum of sums vs. sum of
 maximums 83, 149
- of Crowds protocol 357§
- of deterministic channel on
 uniform prior 8
- posterior 8§, 82§, 157
- as a game 82
- static vs. dynamic 8
- sum of column maximums 83,
 149, 157
- prior 8
- as a game 21, 82
- strong order (\preccurlyeq) qv
- three guesses (V_3) 22, 32, 102
- weighted average over all inners
 8
- see also* examples
- Bayes leakage
- another name for min-entropy
 leakage 97
- is a testing, not a structural
 technique 149
- multiplicative
- on uniform prior 84, 357
- of Crowds protocol 357§
- soundness of structural refinement
 (\sqsubseteq_\circ) for 149
- Bayes multiplicative capacity
 $(\mathcal{ML}_1^\times(\mathbb{D}, -))$
- compositionally bounded for
 parallel composition 143
- Bayes vulnerability *see* Bayes (1
 guess) vulnerability
- Bayes' Theorem 51
- beliefs vs. knowledge of secrets 22
- [Daniel J. Bernstein] 12
- Bertrand's Boxes 272
- frogs vs. toads does not affect
 information flow 274
- best practice in software development
 (claim refuted by completeness of
 structural refinement) 154
- [Mohit Bhargava] 66
- birthday paradox *see* Ramanujan
- [David Harold Blackwell] 168, 331↓
- Blahut-Arimoto algorithm 119, 126
- blinding 369, 385, 389
- blinding- and bucketing capacity
 $cap_b(n)$ 372
- analytic results 374
- [Avrim Blum] 441
- bound $Bnd(u, v, \epsilon)$ for g -leakage in
 differential privacy 437,
 438
- BNDS *see* axioms, generic
- Marton Bognar 396
- [Dan Boneh] 12, 385
- [Joseph Bonneau] 103
- [Nicolás E. Bordenabe] xii, 179, 346
- [Michele Boreale] 199, 366, 385
- Boris and Natasha (spies) 63, 137,
 335↓
- the bottle
- letting an adversary out of 242
- bounds
- generic axiom BNDS *see* axioms,
 generic
- Nicolas Bourbaki 247↓
- brackets, square qv
- [Christelle Braun] 127
- [Franck van Breugel] 279
- Are you a Bromide?* *see* Gelett
 Burgess
- [Stephen Brookes] 247
- [Billy Bob Brumley] 12
- [David Brumley] 12, 385
- bucketing 369, 370, 385, 389, 391
- [Gelett Burgess] 164↓, 168
- C**
- C** *see* context of deployment
- C** *see* election candidate
- C** *see* context of deployment (set of)
- C** *see* all-tallies channel
- cache fault as side channel 3
- Caesar's Cipher vii
- camembert, soft 58, 67
- $cap_b(n)$ *see* blinding- and bucketing
 capacity
- capacity
- bounds for Bayes- 144
- capacity of channel 10, 103, 107§
- additive qv
- additive Bayes- qv
- can be independent of prior and
 gain function 166
- feedback, and memory 127
- multiplicative qv

- multiplicative Bayes- *qv*
related to maximum transmission
rate **117**
Shannon- *qv*
used to define refinement
(counterexample) **166** §
cascade **188, 189, 192, 196, 440**
defined at abstract level **65**,
140↓, **172**↓, **269** ff
defined for concrete channels
137 §
in oblivious differentially private
mechanism **436**
induces structural refinement (\sqsubseteq_\circ)
150, 347
information leakage **65**
multiplicative Bayes capacity
induced by **178–180**
of channels **64, 143**
cannot be defined for abstract
channels **65, 140**
is not compositional **143**
is product of their matrices
150
of Dalenius channel **172, 178,**
417
preserves differential privacy **440**
casting (of ballots) *see* election
cavalry charges **95**↓
Cayley's tree function **379, 386**
cell *see* equivalence relation
census data
United States **22**
used by Dalenius **179**
center of mass
of hyper-distribution **61**
of posteriors induced by a channel
53
see also geometric presentations
channel ↓
abstract *qv*
Bayes vulnerability can *decrease*
in dynamic view **71**
capacity *qv*
cascade *qv*
see also refinement order
comparison *see* refinement order
compatible $C_{\mathcal{X}}$ **132**
composed with other channels
63
composed with programs **63**,
225 ff
concrete *qv*
concrete is typed
 $\mathcal{X} \rightarrow \mathbb{D}\mathcal{Y}$ or equivalently $\mathcal{X} \rightarrow \mathcal{Y}$
131
demonic **166, 331**
vs. deterministic **50**
deterministic **6** §, **167, 325–331,**
415
induces partition **148, 326**
is function from input to
observables **49, 148**
structural refinement (\sqsubseteq_\circ)
148 §, **149**
structural refinement (\sqsubseteq_\circ) is
sound **149**
die rolling **5**
external choice
conditional *qv*
general **135**
probabilistic *qv*
extreme, leaks nothing ($\mathbb{1}$) or
everything (\mathbb{O}) **60**
identity ($\mathbb{1}$)
for parallel composition *qv*
leaks nothing **7, 120, 338**
implementation of **131**
in election
all-tallies C *see* all-tallies
channel
anonymizing *see* election,
casting channel
casting A *see* election, casting
channel
election E *see* election channel
full preferences F *see*
full-preferences channel
partial preferences in election P
see partial-preferences
channel
tally T *see* election, tally
channel
winning-candidate W *see*
winning-candidate channel
initial and final states are equal
(if terminating) **58, 239**
internal choice
conditional ($\ll \mathcal{A} \gg$) **137**
general **137**
probabilistic *qv*

- matrix **49**
 canonical representation **59**
 column y of matrix C is $C_{-,y}$ **49**
 contains extraneous structure
 for information flow **59**
 element $C_{x,y}$ at row x and
 column y **49**
 essentially deterministic **60**
 for Crowds protocol **356**
 is not delivered to customer
161
 is syntax **159**
 large **50**
 normal form **59, 159**↓
 reduced **59, 151**
 row x of C is $C_{x,-}$ **49**
 similar columns **59, 227**↓
 stochastic **50, 151**
 typed $\mathcal{X} \rightarrow \mathcal{Y}$ with input rows \mathcal{X}
 and observable columns \mathcal{Y}
148, 332
 written as program fragment
50
 written sans-serif C **50**
 noisy (in differential privacy) *qv*
 noninterfering **126**
 nonterminating, can update the
 state **310**
 number of distinct outputs *see*
 multiplicative Bayes capacity
 parallel composition of *qv*
 probabilistic
 structural refinement (\sqsubseteq_\circ)
150 §
 probabilistic vs. deterministic **50**
 programs
 are primitive **225**
 leak but don't update (if
 terminating) **226–228, 258**
 refinement order on *qv*
 sequential-program fragment **58**
 similar columns **59, 227**↓
 specification of **131**
 too large for analysis **131**
 typed $\mathcal{X} \rightarrow \mathcal{Y}$ for deterministic
 channels **150**
 vowels and early letters **325**
 zero (O)
 leaks everything **338**
- channel** ↑
- [David Lee Chaum] **409**
 [Chris Chen] **xii**
 [Han Chen] **199**
 [William Y.C. Chen] **385**
 Chernoff information **386**
 ciphertext **389**
 [David J. Clark] **97, 168, 199, 279**
 [Michael R. Clarkson] **xii, 22**
 CNTY *see* axioms, generic
 coarser-than order *see* partition
 cockatiel *see* Yoshi
 [Ellis S. Cohen] **66, 247, 346**
 collateral leakage **103**
see also Dalenius leakage
 colored balls in boxes *see* Bertrand's
 Boxes
 column y of channel C is written $C_{-,y}$
50
 command not directly executable
287, 297, 400
 Communicating Sequential Processes
247
 comparing leakages vs. comparing
 vulnerabilities **162**↓
 comparison of channels
 refinement order (\sqsubseteq) *qv*
 compatible channels $C_{\mathcal{X}}$ **132**
 complement-closed subsets of $\mathbb{U}\mathcal{X}$
 deterministic special case **334**
 completeness
 expressive, of gain/loss functions
 qv
 of structural refinement (\sqsubseteq)
 over deterministic channels
330
 over probabilistic channels *see*
 Coriaceous theorem
 of structural refinement (\sqsubseteq_\circ) *qv*
 over demonic channels **337** §
 over deterministic channels
149
 compositional closure **26**↓, **165**,
164–166, 168↓, **168, 175, 176**,
339 §
 closes “is not refined by” **165**↓
 compositionality **138, 138** §, **143**,
159 §
 as algebraic property of
 system-component
 interactions **144**

- corresponding operators (\heartsuit) and
 (\diamondsuit) 138
 essential for program construction
 viii
 eye color is not 138
 failure 249
 in reasoning about leakage x
see also semantics
 computational resource bounds
 not considered here 21
 concave functions (\smile) 37, 39, 42, 81,
 111, 164, 197, 208, 209, 216,
 300
 loss functions specifically 39, 40,
 186, 300↓
 concrete channel
 is described by a matrix 159
 is function from input to
 distribution of observables
 49
 is syntax 159
 is written C in sans-serif 57,
 108↓
 see also abstract channel is
 semantics, refinement order
 conditional choice IF-THEN-ELSE 268
 if external, leaks its condition
 241, 270, 390, 392, 401
 if internal, does not leak its
 condition 137
 program constructor 241 §
 special case of probabilistic choice
 244
 see also power of adversary
 conditional distribution *see* factoring
 conditional probability
 $p(x|y)$ induced by joint
 distribution p_{XY} 52
 $p(y|x)$ induced by joint
 distribution p_{XY} 52
 $p(y|x)$ of output y given input x
 50, 50
 conditional uncertainty 214
 congruence 160↓
 conic hull 218
 context \mathcal{C} of deployment 101, 159,
 160, 161↓, 166
 is more complex than simply
 “some other channel” 161
 its importance 161
 set \mathbb{C} of 165
 weird or unlikely 101, 161↓, 168,
 177
 see also examples
 continuity
 example \mathbb{V}^λ of non-continuous
 vulnerability 185
 generic axiom CNTY *see* axioms,
 generic
 motivation 185
 of functions on uncertainty 199 ff
 of gain/loss functions 39, 40,
 186 §, 300↓
 surprising non-continuous
 behavior 185↓
 convergence of distributions 187
 convex and continuous function on
 distributions 122, 187
 convex combination 41, 63 §, 78, 88,
 185 ff, 194
 of channels 131, 133
 of distributions 112
 convex functions (\smile) 37, 41, 41, 44,
 78, 81, 112, 164, 185–188,
 208, 300
 their properties 97
 convex hull
 of hyper’s support contains all
 refinements of it 213
 convex set 41
 convex subset of Euclidean space 155,
 316
 convexity
 generic axiom CVX *see* axioms,
 generic
 motivation 185
 of gain functions 39, 40, 186 §,
 300↓
 convexity implies continuity 186
 cookie cutter
 for *meet* in deterministic lattice
 326
 Coriaceous theorem 155, 168, 173,
 177
 completeness of structural
 refinement (\sqsubseteq_\circ) 155
 extended to possibly
 nonterminating programs
 316
 meaning of “coriaceous” 155↓

correlation between secrets
 between county and age of beneficiaries 171
 between initial and final states 233
 between initial state and external variables 233
see also Dalenius leakage
 corrupted users in Crowds protocol *qv*
 cost/benefit analysis of security 394 §
 [Thomas M. Cover] 44, 66, 199, 385
 Cretan who always says “no” 241
 Crowds protocol 353 §
s for “no user detected” 355
 anonymity 353, 354
 apparent paradox 357, 358 §
 corrupted users in the crowd 353
 detected user 354
 forwarder 353
 hop 353, 355
 initiator 353
 modified 358 §
 posterior vulnerability
 independent of forwarding probability 357
 probability D_1 of detecting some user on the first step 355
 probability $D_{\geq 2}$ of detecting some user on the second or subsequent step 355
 probable innocence 354, 356, 361 §
 more general significance 354
 variations 363–365
 web server 353
 (La) Cryptographie Militaire 51↓, 66
 [Imre Csizsár] 199
 postulates for functions on uncertainty 199
 [Paul Cuff] 441
 customer
 is always right (about refinement) *qv*
 litigious 153, 154, 161
 refinement for 147 §
 uses testing refinement ($\sqsubseteq_{\mathcal{C}}$) 152
 CVX *see* axioms, generic
 cyclic property of trace *qv*
 [Artur Czumaj] 429

D

\mathbb{D} type constructor for discrete distributions 17, 275
 d -Lipschitz *qv*
 [Markus Dürmuth] 385
 [Geir Dahl] 168, 331↓
 [Tore Dalenius] 179
 Dalenius (Desideratum) 179
 Bayes vulnerability 172
 calling it “Desideratum” is unfair to Dalenius 180
 knowledge 172
 leakage 11, 161, 165, 433
 g -leakage $\mathcal{DL}_g^{+/\times}$ 177
 expressed abstractly 269 ff
 can be bounded 177
 caused by correlation between secrets 172, 233, 269
 could also be called “collateral leakage” 172↓
 in elections 416
 is caused by correlation between secrets 103
 occurs in programming generally 234
see also examples
 necessity 26↓, 175
 safety 175
 side information 433
 strong g -vulnerability order 173
 vulnerability V_g^D 172, 173
 ordering (strong) 173
see also examples
 data refinement 301
 data-processing inequality 64 §, 65, 150↓, 153↓, 153, 178, 188↓, 188, 220, 363
 generic axiom DPI *see* axioms, generic
 database
 adjacency relation (\sim) 434
 for the Hamming graph *see* differential privacy
 query 435
 statistical *see* differential privacy
 [Anindya De] 441
 [Augustus De Morgan] 147↓
 debugger as adversary *qv*

- Decision Theory
 Proper Scoring Rule 200
 related to gain- and loss functions 44, 200
 declassification 298
 allowed by the specification 400
 decomposition of hyper-distribution *qv*
 defender is masculine xi
 definitions
 use := rather than = 18
 [Pierpaolo Degano] xii, 441
 [Morris H. DeGroot] 44
 demonic channel *qv*
 demonic choice (\sqcap) 231, 287, 301, 311
 added to hyper-distribution model 347
 external 336
 in sequential programs 283
 internal 336
 not included in treatment of non-termination 308
 deniability, plausible *qv*
 [Dorothy E. Denning] 66, 199, 247
 [Peter J. Denning] 66, 247
 deployment *see* context
 desideratum, Dalenius *qv*
 detected user *see* Crowds protocol
 deterministic channel 303
 as Boolean matrix 50, 328
 as program fragment 266
 soundness of structural refinement (\sqsubseteq_\circ) *qv*
 see also channel
 developer
 refinement for 147 §
 uses structural refinement (\sqsubseteq_\circ) 148
 diabetes test 54
 die rolls, coin flips, mother's maiden name
 Bayes vulnerability of 20
 see also examples
 differential privacy 143, 199, 433
 ϵ -differential 435
 adjacency relation (\sim) 434
 compositionality of 440
 exponential mechanism 434
 eye color 438
 Hamming graph 435
 Laplace noise 434
 multiplicative g -leakage bound 434
 mutual-information differential privacy (MI-DP) 441
 noisy channel 435
 preservation under post-processing 440
 statistical database x, 433
 statistical disclosure control 200, 433
 Warner's protocol 440
 [Edsger W. Dijkstra] 247, 264↓, 264, 279, 287, 304, 321, 343↓, 346
 Dining Cryptographers 298, 409
 generalized 409
 disclosure *see* information leakage
 discrepancy
 between probability distributions 127
 discrete metric on \mathcal{X} 31, 278
 disjoint union (\sqcup) 132, 133
 disk (geometric analogy)
 indicates outer probability of an inner 211
 distance between secret values 30
 distance measure between distributions 121
 distribution
 discrete, is a point in N -space 205, 208
 full support 54
 inner *qv*
 see also hyper-distribution
 isomorphism 77↓
 marginal 51
 of distributions *see* hyper-distribution
 of hypers 168
 outer *qv*
 see also hyper-distribution
 posterior *qv*
 prior *qv*
 probability π_x of x in distribution π 5
 sub-uniform *qv*
 type $\mathbb{D}\mathcal{X}$ over (finite) set \mathcal{X} 17
 uniform ϑ 5
 realizes multiplicative Bayes capacity 108
 written as tuple $(1/2, 1/6, 0, 1/3)$ of probabilities 17

-
- distribution-reciprocal gain function
 $g_{\pi^{-1}}$ 127
complement $g_{\pi^{-1}}^c$ 122, 123
for prior π 118
- “do nothing” action of adversary 395
- domain *see* Jones & Plotkin
- “Don’t branch on high.” 241, 390
- [Goran Doychev] 97, 385
- [Sir Arthur Conan Doyle] 247
- DPI *see* axioms, generic
- DSA, DSS *see* side channel
- duality between gain- and loss functions 183↓
- [Cynthia Dwork] 179, 199, 234, 441
- dyadic distribution 19, 39, 39, 85
- dynamic view of leakage qv
- E**
- \mathcal{E} *see* election elector
- \mathbb{E} *see* election channel
- $\mathbb{E}\mathcal{X}$ is equivalence relations over \mathcal{X} 326
- $\mathcal{E}_\alpha F$ is the expected value of F over distribution α 72, 190
- early letters and vowels
spies examples qv
- Eisenstein equation 386
- election
all-tallies observables \mathcal{Y}_C 417
ballot \mathcal{B} 414, 416
ballot box \mathcal{X} 414
candidate \mathcal{C} 414, 416
casting (of ballots) 415§
casting channel \mathcal{A} 416, 417
channel (entire) \mathbb{E} 417
elector \mathcal{E} 416
electorates
election divided into 416
small 416, 421
first past the post (simple majority) 417, 422§
full-preferences, observables \mathcal{Y}_F 419
- instant run-off
in small elections 426§
with preferences 418
- outcome \mathcal{Y} 415
- partial-preferences, channel \mathbb{P} 419
- partial-preferences, observables \mathcal{Y}_P 419
- preferential 414
- tally channel T 415
- tallying program 3§, 414§
see also programs
- voting pattern \mathcal{Z} 416
- winning-candidate observables \mathcal{Y}_W 417
- [Ehab ElSalamouny] 441
- ELSE *see* conditional choice
- Encryption Lemma 286–288, 293, 297, 301, 302, 403
- double- 297, 303
- [Kai Engelhardt] xii
- entropy
guessing- qv
Shannon- qv
unpredictability 386
see also loss function, (ℓ)-uncertainty
- envelope *see* strategy
- ϵ -differential privacy 435
 (ϵ, δ) - 441
criteria for choice of ϵ 438
see also differential privacy 435
- equality
between programs 272§
proof of 275
- equivalence of parallel and sequential composition of pure channels 237, 259, 290, 370↓
- equivalence relation
cell of 148, 326
lattice (and partial order) over $\mathbb{E}\mathcal{X}$ 326
type $\mathbb{E}\mathcal{X}$ on \mathcal{X} 326
- [Martin Erwig] 395
- [Barbara Espinoza] xii, 44, 66, 97, 143, 168, 179, 346, 366, 385
- Euclidean metric on distributions equivalent to Kantorovich, Manhattan 40↓, 185↓
- [David Evans] 12
- examples and exercises** ↓
 g -vulnerability of lottery 44
RER-B train distance 31
RSA decryption 162
adjacency relation (\sim) 438
adversary, multiple 287
antisymmetry of refinement (\sqsubseteq) 158
- Australia, states of 148

- barycentric representation 213,
216
- base-rate fallacy 83
- Bayes capacity 439
- Bayes leakage
- of Crowds protocol 357 §
- Bayes vulnerability 20
- max case 357
 - of Crowds protocol 357 §
- Bertrand's Boxes 272
- bound $Bnd(u, v, \epsilon)$ for g -leakage
- in differential privacy 438
- cascading
- of channels is not compositional
143
- channel
- deterministic 415
 - die rolling 5
 - vowels and early letters 325
- channel can *decrease* Bayes
- vulnerability in dynamic view
71
- channel matrix
- reduced qv
- colored balls in boxes *see*
Bertrand's Boxes
- compositionality 143
- of differential privacy 440
- conditional choice IF-THEN-ELSE,
- if external, leaks its condition
390
- context \mathcal{C} of deployment 166
- continuity (motivation) 185
- convexity (motivation) 185
- correlation between county and
- age of beneficiaries 171
- cost/benefit analysis of security
394 §
- Dalenius
- g -leakage $\mathcal{DL}_g^{+/\times}$ in elections
416
 - g -vulnerability V_g^D 173
 - leakage 171
- deterministic channel 303
- as Boolean matrix 50
- diabetes test 54
- die rolls, coin flips, mother's
- maiden name
- Bayes vulnerability of 20
- differential privacy
- eye color 438
- Dining Cryptographers 409
- Encryption Lemma 301, 302,
403
- double- 303
- external choice
- conditional ($\triangleleft \mathcal{A} \triangleright$) 134
 - general ($P \boxplus$) 135
 - probabilistic ($p \boxplus$) 134
- fast exponentiation algorithm
390
- gain function for elections
- $g_1, g_\#, g_0$ 419–421
- Bill Gates' password 33, 34, 92,
93
- goldfish and piraña 274
- guessing up to k times 32
- imperfect disease test 83, 96,
118, 123
- internal choice
- conditional ($\ll \mathcal{A} \gg$) 143
 - general ($P \oplus$) 143
 - probabilistic ($p \oplus$) is not
compositional 136, 143
- Jack, John and Jill Dalenius
- context 177
- justifying refinement's definition
- with a Dalenius context
176 §
- lifting of a function with \mathbb{D} 261
- lion leakage 360, 361
- marginal, joint and conditional
- distributions 52
- Miracle theorem requires
- non-negative g 109
- Monty Hall problem 95
- mother's maiden name 18
- multiplicative Bayes capacity
- $\mathcal{ML}_1^\times(\mathbb{D}, C)$ of channel C
418
- multiply (μ) of monad 262
- mutual information 85
- non-healthy $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$ 61
- Oblivious Transfer 291
- parallel composition ($\|$) of
- channels 133
- partition induced by dice-sum 6
- password checker 66, 317
- password guessing 33, 91
- password resetting 246
- PIN checking 96

- pit-of-lions gain function g_{lion}
 360
 pit-of-tigers gain function g_{tiger}
 34, 102
 posterior vulnerability $V_g(-)$
 102
 comparison 426
 graphed in 2 dimensions 216
 graphed in 3 dimensions 216
 is convex for $(_p \oplus)$ 143
 is linear for $(_p \boxplus)$ 143
 preservation of differential privacy
 under post-processing 440
 prior vulnerability 425
 probabilistic channel matrix 50
 program derivation 286, 287 §,
 291 §
 proof of compositionality 142
 push forward 261
 reduced channel matrix 59
 refinement order (\sqsubseteq) 361 §
 repeated independent runs 275,
 301, 363, 370
 reuse of random bits (illegitimate)
 302
 Shannon leakage 126
 should capacity be used to define
 refinement 166
 Sorcerer’s Apprentice 292 ↓
 specific channels and their leakage
 91–93
 spies
 Boris and Natasha 137
 Ms. Vowel and Mrs. Early 334
 strong Bayes-vulnerability order
 (\preccurlyeq) is weaker than refinement
 (\sqsubseteq) 163
 structural refinement (\sqsubseteq_\circ) 418
 its soundness for (\sqsubseteq_G) 418
 sum of die rolls 18
 The Lovers’ protocol 296, 409
 The Three Judges protocol 400
 three coin flips 18
 Three Prisoners problem 96
 tiger leakage 125
 trace **tr** of matrix 359
 trusted third party 408
 union closure 334
 unit channel (1)
 leaks nothing 143
 vulnerability
 non-negative 395
 whether average-age query is
 differentially private 439
 zero channel (0) 415, 417
 leaks everything 143
examples and exercises ↑
 exclusive-or (∇) 286
 execution time as side channel 3
 exercises *see examples*
Exp means “any expression” 290
 expansibility 199 ff
 expected value
 $\mathcal{E}_\alpha f$ can be written $(\mu \odot \mathbb{D}f)(\alpha)$
 261
 is weighted average 59
 of function F over distribution α ,
 written $\mathcal{E}_\alpha F$ 72, 122, 190
 of hyper is prior 53, 61
 exponential mechanism *see*
 differential privacy
 exponential-time algorithm
 additive Bayes capacity 113
 exponentiation algorithm, fast 390
 expressive completeness
 of gain/loss functions 164, 208,
 209
 extended state space 269
 external choice
 conditional ($\triangleleft \mathcal{A} \triangleright$) 134
 abstract 140
 concrete 134, 134 §
 demonic (\sqcap) 143, 336
 general ($_P \boxplus$) 135
 abstract 140
 concrete 135 §
 is compositional 142
 leaks the choice 270, 275
 probabilistic $(_p \boxplus)$ 63, 64, 131,
 134, 264, 266
 abstract 139
 between programs 244
 concrete 133, 133 §
 uniform ($\boxplus \cdots \boxplus$) 312
 external vs. internal choice 136, 270,
 297 ↓
 eye color
 example in differential privacy
 438
 is not compositional for making
 children 138

F

F *see* full-preferences channel
 factoring a joint distribution into
 marginal and conditional
 172
 [Dominik Feld] 97
 [Natasha Fernandes] xii
 first past the post (simple majority)
 see election
 [Philippe Flajolet] 385
 fleas 147↓
 [Robert W. Floyd] 247
 assertions attached to flowcharts
 283
 fool, any... can see that refinement
 does not hold 164, 340↓
 formal power series 379
 forwarder *see* Crowds protocol
 frequentist vs. Bayesian interpretation
 of distributions 22
 frogs and toads in boxes *see*
 Bertrand's Boxes
 full-preferences
 channel F 419
 observables \mathcal{Y}_F 419
 function-application parentheses
 omitted if the argument has
 brackets 72
 functional behavior of programs 228
 functional composition (\circ) 59
 functional programming
 use of monads 277↓
 functor 276

G

$g_1, g_\#, g_0$ *see* gain functions for
 elections
 g -function *see* gain function
 g -vulnerability *see* vulnerability
 gain function viii
 algebra 42
 scaling $g_{\times \kappa}$ 30, 42, 117 ff
 shifting $g_{+\kappa}$ 42, 117 ff
 also called g -function
 chosen according to circumstances
 71
 complemented g^c 32
 cannot be negative 32
 definition 10, 25

for elections $g_1, g_\#, g_0$ 419,
 419–421
 induced from metric on \mathcal{X} 30,
 117
 induced vulnerability is
 continuous and convex 27,
 40
 is dual of loss function 232
 measures utility for the adversary
 231
 represented as matrix 26
 reward $g(w, x)$ for taking action
 w when secret is x 25
 unsuitable for programs (use loss
 functions instead) 231
 see also loss function
 gain functions $g_1, g_\#, g_0$ for elections
 420
 gain-function catalog: 29–39
 k guesses 32
 complemented identity g_{id}^c 32
 binary gain function, returning 0
 or 1 only 31
 distribution reciprocal 30
 for Bill Gates g_{Gates} qv
 for elections $g_1, g_\#, g_0$ 419,
 419–421
 for guessing entropy *see*
 loss-function catalog
 for guessing passwords 33
 for medical diagnosis 35
 for Shannon entropy *see*
 loss-function catalog
 general binary gain functions 32
 identity gain function g_{id} 29
 see also Bayes vulnerability
 induced from metric on \mathcal{X} 30
 partition gain functions 32
 pit of lions, with no-guess option
 360
 pit of tigers, for penalizing wrong
 guesses 34
 two-block gain functions 32
 gain-function classes: 39–41
 finite-valued, non-negative
 vulnerabilities $\mathbb{G}\mathcal{X}$ 27, 39
 finitely many actions $\mathbb{G}^{\text{fin}}\mathcal{X}$ 40 §
 non-negative $\mathbb{G}^+\mathcal{X}$ 40, 117 ff
 one-bounded gain $\mathbb{G}^\dagger\mathcal{X}$ 41, 91,
 117 ff, 120
 one-spanning $\mathbb{G}^1\mathcal{X}$ 120↓, 128

- [David Gale] 199
 Gale-Klee-Rockafellar theorem 187, 199
 game formulation
 posterior Bayes vulnerability 82
 prior Bayes vulnerability 21
 [M.R. Garey] 127
 Bill Gates
 gain function g_{Gates} 33, 33–34, 92, 92–93
 other users 33, 92, 93
 password 33–34, 93
 see also examples
GCL *see* guarded-command language of Dijkstra
gedanken experiment 235, 242 §, 311, 321, 401
 gender *see* adversary, agents, authors & readers, defender
 assignment of xi
 generative interpretation of distributions *see* frequentist
 generic axioms *see* axioms, generic
 generic posterior vulnerability ($\hat{\mathbb{V}}$)
 axioms
 averaging (AVG) 190
 bounds (BNDS) 195
 data-processing inequality (DPI) 188
 maximum (MAX) 192
 monotonicity (MONO) 189
 noninterference (NI) 188
 generic prior vulnerability (\mathbb{V}) axioms
 continuity (CNTY) 185
 convexity (CVX) 185
 quasiconvexity (Q-CVX) 186
 geometric mechanisms
 differential privacy of 440
 geometric presentations: 205 §
 antisymmetry
 of refinement for reduced matrices 218
 center of mass of hyper 53
 gain functions 27, 28
 link logic and algebra 220
 posterior vulnerability
 in 2 dimensions 216
 in 3 dimensions 216
 priors and posteriors 53
 Shannon entropy 209
 vulnerability as curve 205
 vulnerability as surface 205
 g_H is a Shannon-like gain function 117, 119, 120↓, 126
 [Jeremy Gibbons] xii, 247, 395
 Gibbs' inequality 38, 372
 [Jean-Yves Girard] 66
 [Michèle Giry] 199, 275, 279
 Giry monad 201, 279
 goat, in Monty Hall problem 95
 [Joseph A. Goguen] 58↓, 66, 247, 346
 goldfish and piraña 274
 [Gaston H. Gonnet] 429
 [Philipp Grabher] 12
 [Peter J. Grabner] 385
 [John Graham-Cumming] 247
 greatest lower bound of anti-refinement chain 275
 Gregor Mendel 139↓
 [Daniel Grund] 12
 guarded-command language *GCL* of Dijkstra 264↓, 264, 321
 guessing an “innocent” value 123
 guessing entropy 167
 given by loss function ℓ_G 36
 guessing up to k times 32
- H**
- $H(\pi)$ is Shannon entropy of distribution π 18, 84
 $H_\infty(\pi)$ is Rényi min-entropy of distribution π 22
 Hamming graph *see* differential privacy
 [Moritz Hardt] 441
 Godfrey Harold Hardy (and Ramanujan) 382
 Haskell 249
 see also QIF-aware language
 [Ian J. Hayes] 247, 304
 [Jifeng He] 304
 “he” xi
 he, she, it, we, they 300↓
 see also adversary, agents, authors & readers, defender
 healthiness conditions 62, 200
 [Eric C.R. Hehner] 147↓, 168
 [Holger Hermanns] 279
 hexagonal hyper-distribution qv

- [Michael Hicks] xii, 97
- HID *see* variable, hidden
- HID_A *see* adversary, multiple hidden Markov matrix
- denotation ([H]) of 260
 - for block entry 269
 - for block exit 269
- Hidden Markov Model (*HMM*) x, 11, 58, 226, 255 §
- abstract 260
 - abstract vs. concrete 255
 - concrete (C;M) 255, 258
 - initial and final state space might differ 258, 268, 269
 - its origins 279
 - sequential composition
 - of abstract *HMM*'s 262
 - of concrete *HMM*'s 258
- high and low users 67
- high security
- program variable *qv*
 - see also* low security
- HMM* *see* Hidden Markov Model
- [C.A.R. Hoare] 44, 247, 304
- Hoare logic 44, 283
- probabilistic 230
- Hoare triple 231, 247, 249
- probabilistic not compositional
 - with demonic choice 231, 249
- Sherlock Holmes 241↓
- hook-up theorem 143
- hop *see* Crowds protocol
- [Roger Howe] 199
- Huffman code and tree 19, 38
- [Sebastian Hunt] 97, 168, 199, 279
- hyper *see* hyper-distribution
- hyper-distribution 7, 56
- not* a lattice under refinement (\sqsubseteq) 216 §
 - called “hyper” for short 56
 - centered 218
 - decomposition into prior and concrete channel 62
 - distribution on 168
 - expected value of inners equals prior 61
 - has finitely many inners (usually) 56↓, 61
 - has type $\mathbb{D}^2\mathcal{X}$ 56
 - hexagonal 216
- informal justification of 54–56
- inner distributions of *qv*
- is distribution of distributions 56
- Kantorovich metric for 278
- made from π through C is $[\pi \triangleright C]$ 56
- made from joint distribution *J* is $[J]$ 237, 260
- obtained by “forgetting” observation values 55
- origin of 67
- outer distribution of 56
- outer probability Δ_δ that hyper Δ assigns to inner δ 56
- point hyper $[\pi]$ on prior π 60, 63, 227, 228, 235, 237
- primitive 213 §
- sub-point 309
- usually written capital Greek, e.g. Δ 56
- with countably infinitely many inners 56↓
- hypergeometric function 385
- hyperplane, separating *qv*
- I
- $I(X; Y)$ *see* Shannon entropy, mutual information
- “I don’t know much about...” art 168
- security 164↓
- identity function id 59
- IF statement *see* conditional choice
- imperfect disease test 83, 96, 118, 123
- implementation
- of channel 131
 - refines specification 228 §, 283
- implicit flow
- super- *see* power of adversary
 - independence of channel and markov components of an *HMM* 256, 259, 260
- information
- lattice of *qv*
 - “information always hurts” property 199↓
 - “information can’t hurt” property 199
 - information flow 3–4
 - non-quantitative, qualitative, possibilistic 67

-
- probabilistic eqv. qualitative 50,
 284, 287, 325 §, 325
 small vs. large 4
 unavoidable vii, ix, 3, 4, 66, 399,
 400, 413
 information leakage
 called “disclosure” by Dalenius
 179
 estimation of 128
 of cascade 65
 information preservation 200
 information radius 127
 initiator *see* Crowds protocol
 inner distributions 72
 finitely vs. countably infinitely
 many 56↓
 of a hyper-distribution 7
 input, to a channel 49
 instant run-off *see* election
 integer exponentiation in logarithmic
 time 161
 internal choice 244
 cannot be defined for abstract
 channels 64, 140
 conditional ($\ll \mathcal{A} \gg$)
 concrete 137 §, 143
 demonic (\sqcap) 143, 336
 demonic is not compositional
 345
 deterministic 404
 general ($P \oplus$)
 concrete 137 §, 143
 probabilistic ($P \oplus$) 64, 136 §, 264
 concrete 136, 136 §
 is not compositional 136, 143
 uniform ($\oplus \cdots \oplus$) 312
 vs. external- *qv*
 intersection-closed subsets of $\mathbb{U}\mathcal{X}$ 334
 [Cynthia Irvine] 66
 isomorphism
 between distributions 77↓
 “it” xi
 Italian attack, in elections *see*
 signature attack
 iteration (i.e. loops) 243
 Iverson brackets [b] convert Boolean b
 to 0, 1 244
- J**
 J is usually $\pi \triangleright C$ for nearby π and C 57
 Jack, John and Jill: an example of
 Dalenius context 177
 Jeremy Jacob 248↓
 [David Jansen] 279
 Jensen’s inequality 111
 [Dale M. Johnson] 66
 [David S. Johnson] 127
 join *see* lattice
 joint distribution
 p_{XY} on input type X and
 observation type Y 51
 expresses correlation 171
 matrix of π through C is $\pi \triangleright C$ 56,
 57
 [Claire Jones] 308, 321
 [Cliff B. Jones] 247
 Jones & Plotkin probabilistic power
 domain 308
 [Rajeev Joshi] 247
 judge or jury
 investigating refinement *qv*
 [Daniel Jurafsky] 279
- K**
 K *see* kludge
 [Boris Köpf] xii, 66, 97, 127, 366,
 385, 441
 [Benjamin Lucien Kaminski] 321
 Kantorovich metric on distributions
 278
 equivalent to Euclidean,
 Manhattan 40↓, 185↓, 278
 for hypers 278
 Karp’s “21 NP-complete problems”
 see set-packing
 Richard Karp 128
 [Joost-Pieter Katoen] 321
 [Yusuke Kawamoto] xii
 [Auguste Kerckhoff] 66
 Kerckhoff’s Principle 6, 51, 186↓,
 225↓, 400
 kernel of function 148↓, 326
 [Aleksandr Yakovlevich Khinchin]
 199
 [Daniel Kifer] 199
 [Peter Kirschenhofer] 385
 [Victor Klee] 199

Kleisli composition 139↓, 235–240,
246, 248, 261 §, 262, 263, 268,
270, 276, 277, 310
multiplies matrices 276
kludge, a possibly faulty
implementation 164↓, 330↓
[Donald E. Knuth] 385
[Paul C. Kocher] 12, 385
[Steve Kollmansberger] 395
[Dexter Kozen] 44, 247, 304, 321
logic of probabilistic programs
284, 308
Kozen triple 231
Kuifje *see* QIF-aware language
Kullback-Leibler distance 38, 372

L

La Cryptographie Militaire 51↓, 66
label (in)dependence
related to compositionality 141
label equivalence 140
label independence
of operators 141
of properties 141
[Alexandre Lacasse] 385
[Yves Lafont] 66
[Imre Lakatos] 164↓, 164, 168
[Jaisook Landauer] 149, 168, 346
Laplace noise 441
see also differential privacy
large channel matrices *qv*
lattice 216
demonic channels are, under
refinement (\sqsubseteq) 216
deterministic channels are, under
refinement (\sqsubseteq) 216
hyper-distributions are *not*, under
refinement (\sqsubseteq) 216, 331,
347
join (lub) 216, 326
meet (glb) 216, 326
over $\mathbb{E}\mathcal{X}$ 326
over $\mathbb{U}\mathcal{X}$ 333
see also lattice of information
lattice of information 167, 325–350
demonic cells form a lattice 333
deterministic partitions form a
lattice 326
Landauer and Redmond 334,
346

probabilistic hypers do *not* form a
lattice 216 §, 331, 347
[F. William Lawvere] 275, 279
leak statement *see* PRINT
leak-everything channel (O) *see* zero
channel
leak-nothing channel (1) *see* unit
channel
leakage 5§
additive *qv*
associated with differentially
private mechanism 436
bounded below by monotonicity
axiom MONO 189
Dalenius *qv*
dynamic view of 71, 77, 189↓,
257
is *not* necessarily convex or
concave 81
lion 360, 361 §, 362
multiplicative *qv*
Shannon viii, 126
static view of 72, 97, 189↓, 257
tiger 125
[Jing Lei] 441
[K. Rustan M. Leino] 247
lifting
of a function with \mathbb{D} 261
[Katrina Ligett] 441
[Bing-Rong Lin] 199
linear combination of hypers 139
linear interpolation
of vertices in barycentric
representation 208
linear optimization problem 90
linear programming 119, 152, 157,
163
lion
escaping from, with running shoes
is *relative* security 43↓
lion gain function *see* gain function
catalog, pit
lion leakage 360, 361 §, 362
Lipschitz conditions
q-Lipschitz 121
d-Lipschitz 121, 121
1-Lipschitz 121
Lithuanian woman and Alan Turing
180, 234
litigious customer *qv*

- local scope of variables indicated by
 {...} 243, 286
- local variables 258, 268 §
- localized expression 400
- location of variables *qv*
- locking a door is only *relative* security
 43↓
- logarithms
- used to define capacity 127
 - used to define leakage 97
- loss function viii
- class \mathbb{L} is dual of gain functions
 232
 - measures utility for the defender
 231
- non-negative-
- uncertainty-inducing loss
 functions $\mathbb{L}\mathcal{X}$ 40
- replaces gain function for
 programs 231 §, 313, 316
- see also* gain function, entropy
- loss-function catalog:
- for guessing entropy 35
 - for Shannon entropy 37
- lottery example of *g*-vulnerability 44
- Lovers' protocol 304, 401
- low security
- program variable *qv*
- see also* high security
- M**
- \mathbb{M} *see* constructor, multiset
- M *see* spies
- [Stephen Magill] 97
- $maj(z)$ is the candidate with the
 maximum number of votes in
 z 422
- majorization 168, 331
- [Pasquale Malacaria] 97, 168, 199,
 279
- Manhattan metric on distributions
 185, 278
- equivalent to Euclidean,
 Kantorovich 40↓, 185↓
- [Heiko Mantel] 247
- marble (analogy)
- indicates outer probability of an
 inner 211
- [Piotr Mardziel] xii, 97
- marginal distribution *qv*
- p_X on prior X induced by joint
 distribution p_{XY} 51
- p_Y on observables Y induced by
 joint distribution p_{XY} 51
- see also* factoring
- marginal, joint and conditional
 distributions 52
- markov
- component of *HMM* 255
- Markov chain 153↓, 199, 279
- Markov matrix 227
- markov programs
- are primitive 225
 - update but don't leak (if
 terminating) 226, 228, 258
- markov vs. Markov 225↓
- markov-atomic block 278
- [James H. Martin] 279
- [James L. Massey] 44
- matrix
- channel *qv*
 - refinement 150
 - stochastic 49, 157, 227
 - trace **tr** *qv*
 - typed $\mathcal{X} \rightarrow \mathcal{Y}$ 50, 332
 - with π on diagonal
 is $\lceil \pi \rfloor$ 89, 118 ff, 172
- matrix multiplication *qv*
- [Laurent Mauborgne] 97
- $\max_S f$ is maximum of f over S 193↓
- MAX** *see* axioms, generic
- max-case Bayes vulnerability 357,
 360
- maximum
- generic axiom **MAX** *see* axioms,
 generic
- [Stephen McCamant] 97
- [Daryl McCullough] 66, 143
- [Andrew McGregor] 441
- [John McLean] 66
- [Frank McSherry] 434, 441
- (proper) measures *see* refinement
 order
- meet *see* cookie cutter, lattice
- [Larissa A. Meinicke] xii, 44, 66, 143,
 168, 179, 247, 279, 321
- [Ziyuan Meng] xii
- merging of partition cells *qv*
- [José Meseguer] 66, 247, 346
- message-passing over insecure network
 288

- method of types qv
- metric
discrete on \mathcal{X} 31, 40, 185↓, 278
privacy 441
- metric on distributions
Euclidean 40↓, 185↓, 278
Kantorovich 40↓, 185↓, 278
Manhattan 40↓, 185, 278
- MI-DP *see* mutual-information differential privacy
- [Jonathan K. Millen] 127
- The Millionaires' problem 409
- Robin Milner 264↓
- (The) Miracle theorem 118, 166, 178, 364, 437
for additive leakage 120, 124§, 124, 126
requires non-negative g 109
see also multiplicative Bayes capacity
- [Ilya Mironov] 441
- [Sanjoy Mitter] 127
- [Eugenio Moggi] 247
- monad 248, 275
probabilistic 275
see also Giry
- monadic functors 261↓, 262
use in functional programming 277
- MONO *see* axioms, generic
- monotonicity 159§, 304, 399
concrete 160
generic axiom MONO *see* axioms, generic
of refinement 131, 298
is actually monotonicity of contexts with respect to refinement 165↓
- monsters 164, 168
see also Lakatos
- Monty Hall problem 95
see also examples
- [Joseph M. Morris] 247
- [Ira S. Moskowitz] 199
- mother's maiden name 18
- Ms. Vowel and Mrs. Early (spies) 64↓, 137, 334
- μ *see* average, expected value, multiply
- multi-party computation 399
shares 299, 403, 406
- multiplication of matrices: sometimes (\cdot) , sometimes (\times) , sometimes juxtaposition 328↓, 329
- multiplicative g -leakage 80
bounded by differential privacy 434, 436§
- invariant under scaling 81
- of prior π and channel C wrt. gain function g is $\mathcal{L}_g^x(\pi, C)$ 80
- multiplicative Bayes capacity
 $\mathcal{ML}_1^x(\mathbb{D}, C)$ of channel C 96, 107, 107§, 127, 418
due to Dalenius cascade 179, 180
- is number of distinct outputs for deterministic channel 108
- is upper bound for multiplicative g -leakage (The Miracle theorem) 109
realized on uniform prior ϑ 108
- multiplicative Bayes capacity bounds qv
- multiplicative capacity
fixed π — $\mathcal{ML}_{\mathbb{G}^+}^x(\pi, C)$ 118§
fixed g — $\mathcal{ML}_g^x(\mathbb{D}, C)$ 117§
efficient algorithm unknown 118
- \mathcal{D} -multiplicative capacity $\mathcal{ML}_{\mathcal{G}}^x(\mathcal{D}, C)$ of channel C 116, 321
- \mathbb{D} -multiplicative capacity $\mathcal{ML}_{\mathbb{G}^+}^x(\mathbb{D}, C)$ of channel C 119§
- multiplicative leakage 9§
Bayes multiplicative leakage
 $\mathcal{L}_1^x(\pi, C)$ of prior π through channel C 9
limits to 107
- multiply (μ) of monad 262
applies weighted average to hyper's inners 216
- can be used at any level 262↓
takes weighted average of hyper's inners 261
see also average
- multiset
constructor \mathbb{M} 414
- membership ($b \# x$) is how many times b occurs in x 414

mutual information 85
 can be misleading 97
 for differential privacy (MI-DP)
 441
 privacy metric 441
 see also Shannon entropy
 [Andrew C. Myers] 22

N

[Lee Naish] 429
 name of secret *qv*
 Natasha *see* Boris
 natural numbers N 261
 natural transformation
 multiply μ 275 §
 unit η 275 §
 necessity
 of a refinement relation ($\sqsubseteq_?$)
 165
 see also refinement order
 neuter gender 287, 399↓
 [Richard E. Newman] 199
 [James Newsome] 97
 NI *see* axioms, generic
 [Kobi Nissim] 441
 [Ivan Niven] 385
 “No security through obscurity”
 (Kerckhoffs’ Principle) 6,
 51, 400
 [Karsten Nohl] 12
 noisy channel, in differential privacy
 436
 non-continuous vulnerability *qv*
 non-healthy $\mathbb{D}\mathcal{X} \rightarrow \mathbb{D}^2\mathcal{X}$ 61
 non-repeating—sequences constructor
 \\$ 414
 noninterference 61, 67, 109, 188, 242,
 347
 generic axiom NI *see* axioms,
 generic
 its origins 247
 nontermination 307
 see also termination
 nontermination (\perp)
 for channels 310
 for markovs 308
 is “catastrophic” 307, 312
 normal form
 of channel matrix *qv*
 normalization $[-]$ (of subdistribution)
 257

NP-complete problem
 whether additive capacity
 $\mathcal{ML}_1^+(\mathbb{D}, C)$ is at least
 threshold t 113, 119
 “null” value indicates not present in
 database 435, 438

O

Oblivious Transfer 291 §, 291, 402
 inlined 299
 protocol 304
 observables
 distinguished from output (or not)
 49
 importance of choosing what is
 modeled 49
 induce partition 6
 “one” channel *see* unit channel
 one-time pad 286, 287 §, 293
 operation
 of making a child 138
 thoughtless *qv*
 [Dag Arne Osvik] 12
 outer distribution 72
 of a hyper-distribution 7
 output
 distinguished from observables (or
 not) 49

P

\mathbb{P} type constructor for powersets
 277, 325
 \mathbb{P} *see* election, partial-preferences
 channel
 [Francesca Pampaloni] 199, 366, 385
 The Panama Canal is a cascade 150↓
 [Prakash Panangaden] 22, 66, 199
 Pandora’s Box of refinement criteria
 163
 [Michela Paolini] 366, 385
 parallel composition (\parallel) of channels
 133
 $\mathbb{1}$ is identity 60
 abstract 139, 139 §, 142
 concrete 132, 132 §, 142
 written (\parallel) 142, 159
 demonic 340
 probabilistic
 abstract 159
 concrete written (\parallel) 142, 159

- partial correctness 247
 partial order
 over equivalence relations $\mathbb{E}\mathcal{X}$ *qv*
 partial refinement (\sqsubseteq_P) 320
 between possibly nonterminating
 programs 314, 314
 partial-preferences
 channel P 419
 observables \mathcal{Y}_P 419
 partition
 coarser than another partition 148
 induced by deterministic channel 148, 326
 induced by observables of
 dice-sum 6
 merging of cells 148, 326
 Blaise Pascal 30
 password
 checker 66, 72, 96, 317
 checking
 program *see* programs
 current vs. previous 233
 guessing 33, 91
 inadvisability of frequent updates 233↓, 233, 246
 [Janet F.F. Peng] 385
 [Colin Percival] 12
 perfect recall 248
 [Ted Petrie] 279
 $pGCL$, probabilistic Guarded
 Command Language 26↓,
 44, 168, 264, 304
 see also guarded-command
 language
 phishing 3
 piecewise linearity
 of gain/loss functions over finite
 action-set \mathcal{W} 209
 pit-of-lions gain function g_{lion} 360
 pit-of-tigers gain function g_{tiger} 34,
 102
 [Tonianne Pitassi] 441
 [Boris Pittel] 385
 [Henryk Plötz] 12
 plaintext 389
 plausible deniability 136, 143
 see also probable innocence
 pleading “not guilty” *see* Crowds
 protocol, probable innocence
 [John O. Pliam] 44
- [Gordon Plotkin] 308, 321
 point distribution 8, 18, 19, 28, 39,
 42↓, 53, 60, 72, 85, 94, 112,
 188↓, 205, 267, 276, 354, 356,
 392
 as outer 227
 barycentric vertices 208, 209
 implicit conversion to 244, 265,
 266
 on x is ηx 276
 on x written $[x]$ 18
 produced by degenerate markov
 227
 produced by non-probabilistic
 program 227
 point hyper 63, 188, 190, 196, 227,
 228, 237, 264, 276, 309
 linear combination of 235
 on π is $\eta\pi$ 276
 on π written $[\pi]$ 60
 polynomial time
 find maximizing gain function 91
 multiplicative Bayes capacity of
 $C^{(n)}$ 371
 reduction of Set Packing to
 additive Bayes capacity 116
 polytope 210↓
 regular 205, 210
 positivity 199 ff
 possibilistic information flow *qv*
 possibilistic programs 279
 post-expectation 26, 231
 posterior
 wrt. gain function 72
 posterior Bayes-vulnerability as a
 game 82
 posterior distribution 5, 7, 17, 239
 $p(X|y)$ on prior X given
 observation y 51, 52
 final 240 §
 if the same for different
 observations 54
 initial 240 §
 itself has a probability of
 occurring 52
 posterior vulnerability $V_g(-)$ 71 §,
 102, 153
 comparison 426
 dynamic 8
 expected-value– vs. max-case–
 94

- generic qv
graphed in 2 dimensions 216
graphed in 3 dimensions 216
in elections 423
independent of forwarding
 probability in Crowds qv
is convex for $(_p \oplus)$ 143
is linear for $(_p \boxplus)$ 143
static 8
wrt. gain function 71 §
 is convex 78
trace (**tr**) formulation 88
postulates for functions on uncertainty
 see Csizsár, Rényi, Shannon
potential refinement order $(\sqsubseteq?)$ 161
power analysis 395
power domain *see* Jones & Plotkin
power of adversary
 can read the source code viii,
 51↓, 225, 239, 242, 246
 implicit flow in conditional
 “ordinary” vs. super 278
 super- 241, 278, 390
 model of 241
 perfect recall 242
powerset monad \mathbb{P} 277
PPDL *see* probabilistic propositional dynamic logic
A Practical Theory of Programming 168
pre-expectation 231
 measures utility for the defender 231
pre-order 151
preferences *see* election, instant run-off
primitive refinement (\preccurlyeq) 320, 340, 347
 induced by Bayes vulnerability 340
 see also refinement order (\sqsubseteq)
PRINT
 leak statement 225, 234, 243,
 258, 264, 265 §
 of constant is equivalent to **SKIP** 228
 simulated by assignment to
 low-security local variable 243
prior Bayes vulnerability as a game 21
prior distribution 5, 238
examples $\pi^{coin}, \pi^{dice}, \pi^{mother}$ etc.
 17
final 240 §
initial 240 §
reflects knowledge rather than outcome 18
prior vulnerability 86, 183, 425
 axiomatization and completeness 201
generic qv
least on uniform prior 437
might be zero 80
privacy goal
 ad hoc, ad omnia 179
probabilistic assignment statement
 qv
probabilistic channel matrix 50, 331
probabilistic choice
 external $(_p \boxplus)$ 63, 64, 131, 264,
 266
 leaks its choice 244, 245, 264,
 270, 273, 313
 external vs. internal 245
 generalizes conditional 230
 internal $(_p \oplus)$ 64, 264, 267
 does not leak its choice 245,
 264
 internal $(_p \oplus)$
 between expressions 243 §
syntax 243
uniform external (\boxplus)
 between programs 244, 312
uniform internal (\oplus)
 between expressions 243 §
varieties $(_p \oplus)$ vs. $(_p \boxplus)$ vs. $(_p +)$ 266
with P an expression 243, 266
probabilistic monad qv
see also Giry
probabilistic propositional dynamic logic *PPDL* of Kozen 321
probability
 conditional $p(y|x)$ qv
probable innocence *see* Crowds
 protocol, plausible deniability
[Helmut Prodinger] 385
program
 standard 230

- program algebra 283 §, 399 §
 for re-ordering statements 290
 its origins 304
 reasonable (or not) 234
- program constructors *see* conditional choice, iteration, probabilistic choice, sequential composition
- program derivation 286, 287 §, 291 §
see also examples
- program variable
 global vs. local, high vs. low 242
 high-security 225, 242
 low-security 225, 242
- programming language
 concrete syntax 243 §
- programming logic 283
- programs
RSA decryption 4, 162
 Agent X 400
 channel qv
 coin flip 231
 comparing them 229
 encryption key 288
 Encryption Lemma 286
 demonic 287
 external vs. internal choice 244, 245
 fast exponentiation 162, 390, 391
 functional behavior 228
 high and low variables 285
 illegitimate re-use of random bits 302
 implicit flow 241
 infinite loop 312
 potential 312
 iteration qv
 leak via assignment to low-security variable 243
 Lovers' protocol
 implementation 299
 specification 296
 markov qv
 Oblivious Transfer
 implementation 295
 specification 291
 One-Time Pad 287, 289
 password checker 66, 72, 317, 318
 318
- password checking 3
 timing side-channel 4, 66
 password resetting 246
 PIN checking 96
 timing side channel 96
- primitive *see* markov programs, channel programs
- super-implicit flow 241, 390
- tallying in elections 3, 414
- Three Judges
 implementation 407, 408
 specification 400
 two-party conjunction 403
- Proofs and Refutations* 168
- Proper Scoring Rule *see* Decision Theory
- protocol
 Crowds qv
 layers 293, 300
see also abstraction layers
- Lovers' qv
 Oblivious Transfer qv
 Three Judges qv
 Warner's qv
- prototype implementation of *QIF*-aware programming language 391 §
- pseudocode 50
- public-key cryptography 389
- push forward
 of a function over a probability distribution 261, 276
- pushing prior π through channel C 57, 189, 191, 192, 195, 198, 257, 276
- often written J 57

Q

- Q -function, Ramanujan's qv
 q -Lipschitz qv
QIF-aware language 44, 275
Kuifje, embedded in Haskell 249, 393, 396
- quadratic-programming problem 120
- qualitative information flow qv
- quantification of secrecy
 as threat, as uncertainty 18
 intuition 18

-
- quasiconcave functions *see*
 quasiconvex functions
- quasiconvex functions **186, 186, 188,**
193–195
- separation **121**
- quasiconvexity
- generic axiom Q-CVX *see* axioms,
 generic
- Q-CVX *see* axioms, generic
- R**
- [Martin Raab] **429**
- [Tahiry Rabehaja] **xii, 168, 279, 429**
- [Michael O. Rabin] **304, 409**
- [Lawrence R. Rabiner] **279**
- Srinivasa Ramanujan **376, 377,**
382–383
- Q*-function **376**
- birthday paradox **376**
- first letter to Hardy **382**
- recursivity **199 ff**
- [Timothy Redmond] **149, 168, 346**
- reduced channel matrix **55, 59, 151**
- demonic **339**
- example **237**
- preserves meaning **59**
- see also* channel
- Referential Transparency **160↓, 168,**
298
- refinement (\sqsubseteq)
- primitive refinement induced by
 Bayes vulnerability **340**
- The Refinement Calculus **247**
- refinement definitions
- equivalent
- for channel matrices, abstract
 channels and hypers **232↓**
- hyper-native definition of *see*
 refinement order
- lifted
- abstract channels **151, 152,**
232
- sequential QIF programs **232**
- sequential probabilistic partial
 programs **314**
- structural
- concrete channels **150, 232**
- demonic channels **332**
- deterministic concrete channels
149, 328
- hyper-distributions **168, 233,**
278
- hyper-distributions (geometric)
213, 232
- taxonomy for programs **319**
- testing
- channels **229, 233**
- sequential demonic channels
336, 337
- sequential demonic programs
228
- refinement order (\sqsubseteq) \downarrow **viii, 319,**
320, 361 §
- $\sqsubseteq \sqsubseteq 1$ **60**
- abstract vs. concrete channels
151
- argued before judge or jury **164,**
165
- arguments reversed **150↓**
- between QIF programs **232**
- between possibly nonterminating
 programs (\sqsubseteq_P) **314**
- cascade induces **150**
- channels and programs together
230
- channels wrt. *all* gain functions
229
- classical refinement is a partial
 order
- and a lattice **228**
- comparison of channels **103**
- concrete channel **151**
- defined in terms of capacity
 (counterexample) **166 §**
- demonic channels **332**
- deterministic channels **325 §**
- see also* structural refinement
- equivalent definitions *see*
 refinement definitions
- for probabilistic programs **279**
- for probabilistic programs **279**
- for the customer (\sqsubseteq_G) *qv*
- for the developer (\sqsubseteq_o) *qv*
- general **157**
- hierarchical **147**
- hyper-native definition of **168,**
278
- is a partial order **10**
- and a lattice in the
 deterministic case **326**
- on abstract channels **157**

- is antisymmetric
on abstract channels 58, 157
- is equivalently structural (\sqsubseteq_{\circ}) or testing ($\sqsubseteq_{\mathbb{G}}$) 156
- is preservation of properties 147
- is the compositional closure of the strong Bayes-vulnerability order (\preccurlyeq) 165
- its definition
compelled by contexts 180
justified by general principles 339, 347
- its origins 247
- its properties on possibly nonterminating programs 314
- matrix 150
- monotonicity 131, 341
- necessity
in Dalenius context 175
in demonic context 340
- not antisymmetric on channel matrices 58
- over proper measures 233, 248
- potential ($\sqsubseteq_{?}$) qv
- pre-order on channel matrices 157
- primitive (\preccurlyeq) 320, 340, 347
induced by Bayes vulnerability 340
- probabilistic channels 328 §
see also structural refinement
- reflexive and transitive 157
- robustness 163
- safety
in Dalenius context 175
in demonic context 340
- stepwise 10, 51, 131, 147, 168
- structural (\sqsubseteq_{\circ}) qv
- taxonomy of orders 319 §
- terms of reference 148
- testing ($\sqsubseteq_{\mathbb{G}}$) qv
- testing, soundness, completeness 336
- the customer is always right 164
- what everyone accepts vs. what everyone needs 164
- see also* partial, primitive, termination, total
- refinement order** \uparrow
- The Refinement Paradox 248
- [Jan Reineke] 97
- [Omer Reingold] 441
- [Michael K. Reiter] 366
- relative difference of g -vulnerabilities qv
- relative security 43↓, 395
- Rényi
entropy $\hat{H}_{\alpha}(\Delta)$, alternative definition for posterior 197
- min-entropy $H_{\infty}(\pi)$ of distribution π 22, 197
- postulates for functions on uncertainty 199
- repeated independent runs 133, 275, 301, 363, 370, 386
- $C^{(n)}$ is n parallel compositions of channel C 370
- $P^{(n)}$ is n sequential compositions of program P 275
- RER-B* train distance example 31
- resolute formality 271
- reuse of random bits (illegitimate) 302
- RFID* *see* side channel
- [Ronald L. Rivest] 304, 409
- [Herbert Robbins] 385
- robustness
of leakage analyses 10
- [R. Tyrrell Rockafellar] 44, 97, 199
- Rocky and Bullwinkle 63↓
- [A.W. Roscoe] 247, 304
- [Aaron Roth] 441
- [Guy N. Rothblum] 441
- [Tim Roughgarden] 441
- row x of channel C is written $C_{x,-}$ 50
- RSA*
decryption 4, 162
encryption 389
public-key cryptography 369
see also side channel
- [Aviel D. Rubin] 366
- [Andrey Rybalchenko] 97, 127
- [Alfréd Rényi] 22, 44, 66, 97, 127, 199

S

$\mathbb{S}\mathcal{C}$ *see* non-repeating sequences over set \mathcal{C}
 [Ib Holm Sørensen] 304
 [Andrei Sabelfeld] 247
 safety
 of a refinement relation ($\sqsubseteq?$)
 161, 162, 165
 of refinement qv
 see also Dalenius
 [J.W. Sanders] 247, 304
 [David Sands] 247
 [Andre Scedrov] xii
 [Maurits van der Schee] 12
 [Fred B. Schneider] xii, 22
 [Tom Schrijvers] xii, 247, 395
 science of security 3
 secret 17
 changing over time 11
 is a distribution (not a value) 5
 its name vs. its value 6↓, 148↓,
 225↓, 269↓
 relative to particular adversary
 17, 285
 secrets
 changing over time x
 [Robert Sedgewick] 385
 [Karen Seidel] 44, 321
 semantic brackets
 convert channel matrix C to
 abstract channel $\tilde{C} = [\![C]\!]$
 57, 159
 semantic equivalence, of channel
 matrices (\equiv) 138, 159
 semantics
 compositionality qv
 failure 249
 is crucial 141
 for information flow generally
 248
 for nontermination 307 ff
 of *QIF* programs 58, 232–234,
 255 ff, 283 ff, 391, 392, 395,
 399 ff
 Dalenius aware 234, 248
 implemented in *Kuifje* *see*
 QIF-aware language
 when variables not overwritten
 234
 of abstract channels qv

of nondeterministic (demonic)
 programs 345 ff
 of probabilistic and demonic
 programs 231, 249
 of standard programs 240
 Separating-Hyperplane Lemma 155
 sequential composition 276
 congruence of definitions 262
 heterogeneous 238
 homogeneous 238
 of abstract *HMM*'s 268
 of channel then markov 238, 259
 of concrete *HMM*'s 258
 of markov then channel 240, 260
 of programs in general 235 §
 of two channels is their parallel
 composition 237, 259, 290,
 370↓
 of two markovs 236, 258
 sequential program
 generalization of channel qv
 set-packing problem 114, 116
 Karp's “21 NP-complete
 problems” 128
 The Shadow Model for demonic
 noninterference 304, 322,
 331, 346
 is the qualitative precursor of
 hyper-distributions and
 inners 331
 [Adi Shamir] 12
 [Claude E. Shannon] vii, 22, 199
 postulates for functions on
 uncertainty 199
 source-coding theorem 38, 85
 Shannon capacity 110, 119, 126, 127
 Shannon entropy viii, 18 §, 22, 37,
 84 §, 167, 199, 205, 372 §
 can be misleading for
 confidentiality viii, 20
 expressed with loss function 37
 given by loss function ℓ_H 38,
 117↓, 126↓
 graphed as surface above
 barycentric triangle 209
 is uncertainty rather than
 vulnerability 85
 mutual information $I(X; Y)$ 85,
 153↓, 199
 of distribution π is written $H(\pi)$
 18, 84

- Shannon leakage viii
see also examples
- shares *see* multi-party computation
- “she” xi
- [Robin Sibson] 127
- Sibson’s information radius 127
- side channel ix, 3
 attacks x, 4
 RSA , DSS , remote timing,
 caching (AES , RSA , DSA ,
 Dutch $RFID$) 12
 cache state 4
 power consumption 4
 timing attack 4, 162, 317, 369,
 390
- side information 433
see also Dalenius
- [Pieter Siekerman] 12
- signature (“Italian”) attack 414,
 415↓, 415, 429
- similar columns *see* channel matrix
- simple majority *see* election, first
 past the post
- SKIP program
 identity of sequential composition
 228
- Skolemization 88↓
- smartphone
 used in elections 417
- [Adam D. Smith] 441
- [David M. Smith] xii, 385
- [Rodney Smith] 429
- [Michael B. Smyth] 321
- Smyth power domain 321, 332
- software deployed in (an unexpected)
 context 101, 161↓
- [Dawn Song] 97
- [Isaac M. Sonin] 168
- Sorcerer’s Apprentice example 292↓
- soundness of structural refinement (\sqsubseteq)
 over deterministic channels 330
- soundness of structural refinement
 (\sqsubseteq_\circ) 197
 over demonic channels 337§
- over deterministic channels 149
- over probabilistic channels 153,
 418
- see also* structural refinement
- source-level reasoning 347
 for demonic channels 342§
- specific channels and their leakage
 91–93
- specification
 of channel 131
 refined by implementation 228§,
 283
- spies examples
 and adversary M
 Ms. Vowel and Mrs. Early 334
 Boris and Natasha 137
- square brackets
 $[-]$ in general make a point
 distribution on $(-)$ 60
 $[J]$ make hyper-distribution from
 joint distribution J 57, 309
 $[\pi]$ make sub-point
 hyper-distribution from
 subdistribution π 57↓, 308
 $[\pi \triangleright C]$ make hyper-distribution
 from prior π and channel C
see also hyper-distribution
- Iverson brackets $[b]$ convert
 Boolean b to 0, 1 244
- squashing a hyper back to a
 distribution 261, 262, 276
see also multiply (μ)
- [Mudhakar Srivatsa] 97
- standard program 230
- [Starbug] 12
- state space
 extended 269
 of program
 as Cartesian product 284
- states of Australia qv
- static view of leakage qv
- statistical database ix, x, 179, 433
- statistical disclosure control *see*
 differential privacy
- [Angelika Steger] 429
- [Volker Stemann] 429
- stepwise refinement *see* refinement
 order
- Stirling approximation to $n!$ 386
 improved 386
- stochastic matrix qv
- strategy 155
- g -vulnerability, ℓ -uncertainty are
 envelopes 76
- dominated 75
- of adversary 75, 153
- [Marco Stronati] xii

- strong g -leakage order *see* strong g -vulnerability order
- strong g -vulnerability order **153**
- strong Bayes-vulnerability order (\preccurlyeq)
160, 162, 175, 176
 is weaker than refinement (\sqsubseteq)
163
 its compositional closure is
 refinement (\sqsubseteq) **165**
see also primitive refinement
- strong Dalenius g -vulnerability order
 (\sqsubseteq^D) *qv*
- strong dependency **67, 241, 247, 347**
- strong subadditivity
 of functions on uncertainty **199 ff**
- structural refinement (\sqsubseteq_\circ) **148**§, **149**,
168, 319, 328, 418
 completeness for (\sqsubseteq_G) **155**
see also Coriaceous theorem
 defined by existence of cascade
150
 defined for abstract channels
151
 deterministic channels **148, 149**,
150
 equivalent to testing refinement
 (\sqsubseteq_G) **156**
 for probabilistic channels **150**
 is antisymmetric
 on abstract channels **157**
 its soundness
 for (\sqsubseteq_G) **153, 418**
 for deterministic channels *qv*
 of hypers, geometrically **210**
- sub-hyper-distributions $\mathbb{DH}\mathcal{X}$ **311**
 $\Delta \sim \Delta'$ means that Δ, Δ' have
 equal weight **313**
- sub-point hyper-distribution **309**,
309
 $[\delta^i]$ on δ^i **63**
- sub-protocols **298**
see also abstraction layers
- sub-uniform distribution **7, 112** ff,
116↓, **166**
 realizes additive Bayes capacity
112
- subadditivity
 of functions on uncertainty **199 ff**
- subdistribution \mathbb{D} **257, 308, 321**
- subgradient **187**
- substitution of $\text{Exp}X$ for x in $\text{Exp}Y$
290
- sum of die rolls **18**
- sum-property **199 ff**
- [Yidong Sun] **385**
- super-implicit flow *see* power of
 adversary
- support
 $[\delta]$ of distribution δ **17, 56**
 full- **17**
- symmetry
 of functions on uncertainty **199 ff**
- [Paul F. Syverson] **199, 366**
- T**
- T *see* election, tally channel
 tallying program *see* election
 [Kunal Talwar] **434, 441**
 [Sekhar Tatikonda] **127**
 taxonomy, of refinement *qv*
 [Paul Taylor] **66**
- Taylor series
 $[z^n]f(z)$ is coefficient of z^n in the
 Taylor expansion of $f(z)$
380
- [Tachio Terauchi] **127**
- termination
 almost sure, with probability one
316, 321
see also nontermination
- termination refinement (\sqsubseteq_\perp) **313**,
313–314, 320
- terms of reference *see* refinement
- testing refinement (\sqsubseteq_G) **152, 319**
 equivalent to structural
 refinement (\sqsubseteq_\circ) **156**
- testing-based vs. structural definition
 of refinement **347**
- tetrahedron
 regular **210**↓
 right **205, 208, 213**§
- The Lovers' protocol **296**§, **296, 409**
- The Three Judges protocol **400**
- THEN *see* conditional choice
 “they” **xi**
- [Joy A. Thomas] **44, 66, 199, 385**
- thoughtless operations, their
 importance **95**

- threat
 measured by vulnerability 5, 8,
 18–22
see also vulnerability
- three coin flips 18
- three guesses *see* Bayes vulnerability
- Three Prisoners problem 96
see also examples
- tiger gain function
see examples and exercises
see gain-function catalog, pit
- tiger leakage 125
- timing attack
 as side-channel *qv*
 on RSA 385
- timing side-channel 66
- total correctness 247
- total refinement (\sqsubseteq_T) 320
 between terminating programs
 313, 314
- trace \mathbf{tr} of matrix 359
 cyclic property 88
 is sum of diagonal entries 87
 used to formulate posterior
 g -vulnerability 88, 153
- tree function of Cayley *qv*
- tree polynomials 386
- [Evan Tromer] 12
- trusted third party 291, 408
- [Kathleen Trustrum] 168
- Alan Turing 180, 234
 and Lithuanian woman 180
- two-party computation
 of conjunction 296, 404
 of disjunction 404
 of exclusive-or 405
- type constructors
 \mathbb{D} for discrete distributions 17,
 275
- types
 method of 371, 385, 386
 program- vs. mathematical sets
 268, 269
- U**
- $\mathbb{U}\mathcal{X}$ union-closed sets of subsets of \mathcal{X}
 333
- uncertainty 19–22
- (ℓ)-uncertainty 208
 $U_\ell(\pi)$ of distribution π according
 to loss function ℓ 27
- can be measured in many ways
 44
- conditional 214
- is concave 42
- is envelope of strategies 37
see also strategy
- underline *see* union closure
- uniform(–) make a uniform
 distribution 272, 318
- uniform distribution ϑ 5
 realizes multiplicative Bayes
 capacity 108
- union closed
 subsets $\mathbb{U}\mathcal{X}$ of $\mathbb{P}\mathcal{X}$ 333
- union closure 334
 P^\cup of demonic channel P 333
 added cell indicated by underline
 334
- unit channel (1)
 as a matrix is a single column of
 1's 227↓
- is η 276
 leaks nothing 60, 64, 109, 143,
 178, 227↓, 227, 241, 266, 277,
 328, 345
- unit type (•) 258, 269
- unpredictability entropy 386
- usability of output 200
- utility
 associated with differentially
 private mechanism 436
- V**
- V^λ *see* continuity
- V_1^D *see* Dalenius Bayes vulnerability
- [Salil P. Vadhan] 441
- variable
 hidden (HID) 284
 local scope *qv*
 location 287, 400
see also localized expression
 program- vs. random- 269
 visibility declarations 284 §
 visible (VIS) 284
- variant (probabilistic) 316
- VDM** *see* Vienna Development
 Method
 [Trevor N. Vickers] 247
- Vienna Development Method **VDM**
 247↓
- VIS** *see* variable, visible

-
- VIS_A** *see* adversary, multiple
[Dennis Volpano] 66
- voting
- integrity 413
 - lying about 413
 - protocols x
 - secrecy 413
 - signature (“Italian”) attack 416
 - transparency 413
- vowels and early letters
- the spies Ms. Vowel and Mrs.
 - Early *qv*
 - see also* examples
- vulnerability
- non-negative 395
- g*-vulnerability 10, 208
- V_3 , three guesses *see* Bayes vulnerability
 - V_g overloaded as a function 72
 - $V_g(\pi)$ of π according to gain function g 26
 - absolute difference 80, 101
 - averaged over hyper 72, 183
 - Dalenius *qv*
 - determined by gain function g 26
 - graphed as curve 205
 - graphed as surface 205
 - is convex 41, 78, 185
 - is envelope of strategies *see* strategy
 - matrix based 86 §
 - maximized over hyper 72, 93
 - non-negative 80
 - pit of lions g_{lion} 360
 - pit of tigers g_{tiger} 34, 102
 - posterior- *qv*
 - prior- *qv*
 - relative difference 80, 101
 - see also* Bayes vulnerability
- W**
- W** *see* winning-candidate channel
[Philip Wadler] 247
- [Colin D. Walter] 395
- [Weina Wang] 441
- [Stanley L. Warner] 143
- Warner’s protocol 64↓, 136, 143,
335↓, 440↓
- weakest pre-test wt 344
- weakest precondition wp 247, 343↓,
345
- web server *see* Crowds protocol
- weight $\langle \delta \rangle$ of (sub-)distribution δ 257, 313
- “weird” contexts vs. those that are reasonable or likely 101, 161↓, 177
- [Roland Wen] xii, 429
- [Alfred North Whitehead] 95↓, 97
- winning-candidate
- channel W 417
 - observables \mathcal{Y}_W 417
- [Niklaus Wirth] 168, 247, 346
- witness for non-refinement
- demonic 336, 338
 - deterministic 330
 - probabilistic 150, 154
- witness for refinement
- demonic 332
 - deterministic 329
 - probabilistic 150, 168
- [J. Todd Wittbold] 66
- wlog. means “without loss of generality”
- world, as state of adversary’s knowledge 6, 52, 93
- wp** *see* weakest precondition
- wprob. means “with probability”
- [Joakim von Wright] 247, 346
- wrt. means “with respect to”
- wt** *see* weakest pre-test
- X**
- X**
- hidden state space of a QIF-aware program 225
- set of inputs to channel 49
- see also* election, ballot box
- Y**
- Y**
- set of outputs (observables) from channel 49
- the type of observable leaked by a PRINT statement 265
- see also* election, outcome

- [Harold R.L. Yang] 385
 [Andrew Chi-Chih Yao] 304, 409
 [Hirotoshi Yasuoka] 127
 \mathcal{Y}_C *see* election, all-tallies
 observables
 \mathcal{Y}_F *see* election, full-preferences
 observables
[Lei Ying] 441
Yoshi the cockatiel v
[Malik Younsi] 385
 \mathcal{Y}_P *see* election, partial-preferences,
 observables
[Lanqing Yu] 441
 \mathcal{Y}_W *see* election, winning-candidate
 observables
- Z**
 $[z^n]f(z)$ *see* Taylor series
 \mathcal{Z} *see* election, voting pattern
 Z specification method 247
 Zermelo 247↓
 zero channel (\mathbb{O}) 415, 417
 is $\mathbb{D}\eta$ 276
 is the identity matrix 76, 227↓
 leaks everything 7, 60, 64, 76,
 143, 227↓, 266, 277, 345
 leaks more than any other
 channel 89
 produces an outer isomorphic to
 the prior 77
[Junshan Zhang] 441
[Lijun Zhang] 279