Technical Report

UCAM-CL-TR-785
ISSN 1476-2986

Number 785

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

Anonymity, information, and machine-assisted proof

Aaron R. Coble

July 2010

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

This technical report is based on a dissertation submitted January 2010 by the author for the degree of Doctor of Philosophy to the University of Cambridge, King's College.

Technical reports published by the University of Cambridge Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/techreports/

ISSN 1476-2986

Abstract

This report demonstrates a technique for proving the anonymity guarantees of communication systems, using a mechanised theorem-prover. The approach is based on Shannon's theory of information and can be used to analyse probabilistic programs. The information-theoretic metrics that are used for anonymity provide quantitative results, even in the case of partial anonymity. Many of the developments in this text are applicable to information leakage in general, rather than solely to privacy properties. By developing the framework within a mechanised theorem-prover, all proofs are guaranteed to be logically and mathematically consistent with respect to a given model. Moreover, the specification of a system can be parameterised and desirable properties of the system can quantify over those parameters; as a result, properties can be proved about the system in general, rather than specific instances.

In order to develop the analysis framework described in this text, the underlying theories of information, probability, and measure had to be formalised in the theorem-prover; those formalisation are explained in detail. That foundational work is of general interest and not limited to the applications illustrated here. The meticulous, extensional approach that has been taken ensures that mathematical consistency is maintained.

A series of examples illustrate how formalised information theory can be used to analyse and prove the information leakage of programs modelled in the theorem-prover. Those examples consider a number of different threat models and show how they can be characterised in the framework proposed.

Finally, the tools developed are used to prove the anonymity of the dining cryptographers (DC) protocol, thereby demonstrating the use of the framework and its applicability to proving privacy properties; the DC protocol is a standard benchmark for new methods of analysing anonymity systems. This work includes the first machine-assisted proof of anonymity of the DC protocol for an unbounded number of cryptographers.

Acknowledgements

My wife Alex has lived each of my failures and successes and every moment of elation or despair that has gone into these four years of work. She has withstood all of it with patience, love, and kindness and for that she has my undying gratitude. Without the encouragement and support of my parents and my sister I would not be where I am today. They were there holding my hand through my first steps and have cheered me on through the final sprint. Thank you.

I'd like to thank Larry Paulson for all his guidance during the course of my research and for remaining confident in my ability even when my own confidence faltered. Everyone that attended the HVG/ARG afternoon teas helped to create a wonderful community for social and academic discussion; those meetings were always a welcome break in the workday. Unfortunately, I cannot mention everyone I would like to, but a few names demand mention. Magnus Myreen has been a friend, companion, mentor, and more. Joe Hurd gave me a great deal of his time and patience early on during this work and discussions with him truly gave my research its first footing. Mike Gordon has always been at the ready with some friendly wisdom and Thomas Tuerk has revived the tea meetings, with a smile and delicious cakes, whenever I have fallen behind.

Finally, I would like to thank the Gates Cambridge Trust and the University of Cambridge Computer Laboratory for their financial support. Without their generosity, none of this would have been possible.

Contents

Chapter 1

Introduction

"The only truly secure system is one that is powered off, cast in a block of concrete and sealed in a lead-lined room with armed guards — and even then I have my doubts."
– Gene Spafford

The overall aim of this text is to present a mathematically rigorous framework for analysing the anonymity and information leakage properties of software systems. This chapter motivates the need for such a framework and provides a high-level background on anonymous communications and formal methods for security analysis. Finally, the specific contributions developed in this text are outlined.

1.1 Anonymous communications

Privacy and anonymity are essential to society in both the physical and the electronic domain. Anonymous police tips and witness protection programs are common in the physical realm and provide beneficial services. Similarly, the Internet can provide an electronic medium for free expression, but not if users can be identified and censored by totalitarian governments. There have been many attempts to provide anonymous electronic communication systems [38, 52, 125, 126], but the problem is challenging and remains open. Danezis has recently written a thorough and up-to-date historical review [37] of research on anonymity and privacy-enhancing technologies (PETs). A concise review of important milestones in this area follows.

Early research on anonymity in electronic communication began in 1981 when Chaum proposed the *mix* as a means of preventing an attacker from linking the sender and recipient of a message [24]. His initial design pooled incoming messages, encrypted with layers of public-key cryptography, and then decrypted and reordered those messages before delivering them, thereby obfuscating the correspondence between incoming and outgoing messages. Despite the age of this area of research and the relative ease with which one can obtain some degree of anonymity in the physical world, achieving the same on electronic networks remains a challenging problem.

Current research on anonymity covers a broad spectrum ranging from electronic voting to systems for anonymous web-browsing and email. For example, Novak, et al. used statistical analysis of language style to break the psuedonymity of postings on electronic message boards. Similarly, Rao and Rohatgi determined that Internet users can be identified by their web-browsing patterns [123] and Bissias, et al. demonstrated that such analysis can be performed even on encrypted streams [13]. Chaum, et al. have designed specialised techniques for verifiable electronic elections [23]. Despite sharing common goals, the techniques used in different areas of anonymity research can vary greatly; in this document, focus is placed on applications such as systems for anonymous web-browsing and email, rather than e-voting and similar topics.

Since the conception of the mix, numerous systems have been designed with the aim of providing anonymity to communication partners. In 1988 Chaum proposed a system based on anonymous broadcast, dining cryptographer networks [22], and a number of modern systems have been based on that intuition as well [64]. Many other systems based on mix networks have been proposed, along with a wide range of mixes using different message pooling and delaying strategies and various attack detection and defense

mechanisms. Díaz provides a good overview of mix types [49], a number of attacks on mixes are presented by Serjantov [137], and an attack detection and prevention mechanism for mixes was proposed by Danezis and Sassman [39]. Anonymous communication systems can generally be divided into those which demand low latency, needed for interactive applications like web-browsing, and those with high latency, suitable for applications such as email. Most mix-based specifications have a high latency and thus are applicable for remailers and other systems where immediacy of response is not necessary. Mixmaster [108] and its successor Mixminion [38] are both anonymous remailer systems that have been deployed. Onion Routing [67, 147] and its successor Tor [52] are both examples of circuit-based low-latency systems that have been implemented. With more than 100,000 users, Tor is the most widely deployed low-latency anonymous communication system. Most low-latency systems don't involve any explicit mixing, but instead rely on messages being rerouted through a number of forwarding nodes before reaching their destinations. Some low-latency, peer-to-peer systems have been developed, such as Crowds [125] and MorphMix [126], which attempt to eliminate the need for any trusted servers and provide greater scalability.

The number of attacks devised against anonymous communication systems is even greater than the number of system designs. Though unproven, it is generally believed that an attacker who can observe the entire network over a sufficient period of time can always link the origin and destination of messages with near certainty, *unless the system can provide constant-bandwith traffic.* Constant-bandwidth solutions are prohibitively expensive or inefficient for all but a small class of users, thus attacks and defenses target adversaries whose resources limit them to observing/controlling only some portion of the network. For many widely deployed systems, this seems to be a reasonable assumption. Attacks on mixes themselves are typically mounted by preventing messages from entering the mix or flooding the mix with dummy traffic [137], while attacks on mix networks proceed by trying to infer or observe the path a message has taken through the network by compromising mixes or using statistical analysis of observations. Similarly for other systems, attacks proceed by attempting to direct the routing of a message through specific compromised nodes [149], by linking observed messages through statistical *traffic analysis* [36, 110, 124], or by otherwise compromising or inferring the entire path of the message. In order to maintain efficiency, low-latency systems generally can only defend against a weaker attacker than high-latency systems, but the techniques used in each are often similar. Another important consideration when designing anonymous communications systems is perfect forward secrecy, ensuring that the system cannot be forced to link communication partners after their communication has ended. In the United Kingdom, defending anonymity systems against attacks using legal compulsion has become a topic of increasing interest in light of the the Regulation of Investigatory Powers Act [33].

1.1.1 Quantifying Anonymity

When analysing anonymous communications systems, it is useful to measure the degree of anonymity a system provides and the conditions under which it is ensured. Throughout research on the subject, anonymity has proved to be difficult to define and many definitions have been suggested. Pfitzmann and Köhntopp [120] attempted to standardise definitions by proposing the following:

> "*Anonymity* is the state of being not identifiable within a set of subjects, the *anonymity set*. The *anonymity set* is the set of all possible subjects who might cause an action. ... Anonymity is the stronger, the larger the respective anonymity set is and the more evenly distributed the sending or receiving, respectively, of the subjects within that set is."

Quantifying anonymity using the size of the anonymity set has been a part of this line of research from very early on, but it is easy to think of settings in which it is not a good metric. The difficulty is deciding what it means to be "identifiable" within the anonymity set. For example, if we have a large group of suspects, one of which we suspect with a very high probability while suspecting the rest with a very low probability, then this might be considered less anonymous than a small group in which all are equally suspect. Despite this shortcoming, using the size of the anonymity set as a metric makes sense in historical context. One of the first problems proposed in the field was Chaum's dining cryptographers problem [22]. As this problem was initially presented, all the members of the anonymity set were equally likely to have performed the action, so size of anonymity set is a sensible metric.

Introducing probability into the definition of anonymity allows for a variety of anonymity degrees to be defined. The following four degrees appear regularly throughout the literature [125, 140, 74]:

Definition 1 (beyond suspicion/total anonymity)**.** *Given that the adversary can determine a particular action has occurred, the individual in question is no more likely to have been the actor than anyone else.*

Definition 2 (probable innocence/partial anonymity)**.** *Given that the adversary can determine a particular action has occurred, the individual in question is no more likely to have been the actor than to not have been the actor.*

Definition 3 (possible innocence/weak anonymity)**.** *Given that the adversary can determine a particular action has occurred, there is some possibility that the individual in question was not the actor i.e. the certainty of the individual being the actor is not 100%.*

Definition 4 (α-anonymity)**.** *Given that the adversary can determine a particular action has occurred, the individual in question is suspected with probability less than α, for some $\alpha \leq 1$.*

Definitions 2 and 3 can be generalised using α-anonymity as 0.5-anonymity[1] and 1.0-anonymity respectively. The term *partial* anonymity is used ambiguously throughout the literature, taking many different meanings; in this text, that term will refer to any scenario which is neither the ideal case of an even distribution over all suspects nor the worst case of a single user suspected with probability 1.

What is sufficiently anonymous in one situation may be completely useless in another. In some cases, the degree of anonymity required may depend on the legal system under which the user is placed. If the burden of proof falls to the prosecution and an individual is assumed innocent until proved guilty "beyond a reasonable doubt", then possible innocence may be sufficient — otherwise, it is possible that only total anonymity will protect the user.

The definitions for degrees of anonymity discussed above have done much to clarify the terminology, but do not ease the difficulty of comparing the relative anonymity provided by two different systems or two configurations of the same system. Recently Serjantov and Danezis [136] and independently Díaz, et al. [50] addressed this issue by proposing an information-theoretic metric for anonymity. This metric uses entropy to quantify the extent that actions are "evenly distributed", which you may recall from Pfitzmann and Köhntopp's definition presented earlier. The definition for this metric, as presented by Serjantov and Danezis, is summarised below.

Definition 5 (Entropy-based metric for anonymity)**.** *For a finite set of users Ψ and a finite number of roles $\mathcal{R}$, let $\mathcal{U}$ be an attacker's a posteriori probability distribution for a user being assigned a role with respect to a particular message. Let $\mathcal{U}$ be restricted to $\mathcal{U} : \Psi \times \mathcal{R} \rightarrow [0,1]$ such that for each $r \in \mathcal{R}, \sum_{u \in \Psi} \mathcal{U}(u,r) = 1$. For a given role r, let $p_u = \mathcal{U}(u,r)$. Define the effective size $\mathcal{S}$ of the anonymity set, for a particular role, to be equal to the entropy of the distribution and use this as a metric for the strength of anonymity provided.*

$$\mathcal{S} = -\sum_{u \in \Psi} p_u \log_2(p_u)$$

Recently, a number of information-theoretic metrics based on entropy have been suggested for measuring anonymity. Chatzikokolakis [18] has examined the use of such metrics, including the measure of (conditional) mutual information adopted in this text. Informally, $\mathbb{I}_\mu(\mathcal{O}; \mathcal{A}|\mathcal{L})$, the conditional mutual information of $\mathcal{O}$ and $\mathcal{A}$ given $\mathcal{L}$, measures the number of bits one can learn about $\mathcal{A}$ by observing $\mathcal{O}$ with knowledge of $\mathcal{L}$; in the case of anonymity, $\mathcal{A}$ is a random variable over the possible identities of an actor, $\mathcal{O}$ is a random variable over the visible outputs of a system, and $\mathcal{L}$ is a random variable over the visible inputs to the system. A detailed presentation of this measure will be developed over the course of subsequent chapters.

Though purely probabilistic definitions provide strong metrics for anonymity, Halpern and O'Neill [74] identify a class of situations that they do not effectively capture. It is not always realistic to expect that an attacker's a priori probability for attributing an action to a user is the same for all users. For example, an attacker might attribute a message written in Russian to a Russian with much higher probability than to a Spaniard, but this information is not part of the operation of the system. Though the system cannot change the a priori beliefs of the attacker, the user would like to know that the system does not leak any *additional* information. Halpern and O'Neill address this situation by proposing a definition for conditional anonymity within a runs-and-traces framework. That definition is summarised below:

[1] This is a slight mismatch of the notation, as Definition 2 allows a probability of suspicion ≤ 0.5 to be considered anonymous, while 0.5-anonymity requires a strict $<$.

Definition 6 (Conditional anonymity)**.** *Let $i \in \Psi$ be a user, $j \in \Psi$ be the attacker, a be an action, and $\alpha = P(a$ attributed to $i \mid j$ observes some $x \in \Psi \backslash \{j\}$ has performed $a)$ be j's a priori probability of attributing a to i given he knows someone other than himself has performed a. The user i is* conditionally anonymous *with respect to j and a if, for all runs of the system in which j observed that a was performed by some $y \in \Psi \backslash \{j\}$, j's a posteriori probability of attributing a to i is equal to α i.e. j's beliefs regarding the likelihood of i performing a are not changed by his observation of any possible run of the system in which a occurs.*

The anonymity metrics above are very strong in their ability to represent varying degrees of anonymity, but do not easily capture our intuition of anonymity as an attacker's knowledge (or lack thereof). Such an epistemic definition allows for intuitive specifications of anonymity properties of systems to be analysed. Halpern and O'Neill used a modal logic of knowledge to represent an attacker's knowledge as the sequence of observable actions performed in a run of the system. Syverson and Stubblebine [148] presented one explicitly epistemic approach based on an extension of the basic **S5** axioms characterising knowledge. Both approaches allow statements about the anonymity of a system to be easily and clearly made, but have the major drawback of being able to deal only with total anonymity i.e. they can only determine that a system provides perfect anonymity or does not. This shortcoming is implicit in the view of knowledge as "all-or-nothing".

There are several common points that must be specified when defining anonymity for any of the approaches presented. The anonymity provided by a system must be with respect to a particular attacker who may be *active* (can inject messages/tamper with traffic/etc.) or *passive* (can only observe) and *local* (only has access to some subset of the network/system) or *global* (has access to the entire network/system). The attacker may also control some set of the users of the system in addition to traffic on the links between users. Finally, we must specify whose anonymity is being protecting. Returning to Pfitzmann and Köhntopp's [120] definitions:

Definition 7 (Sender anonymity)**.** *Sender anonymity is the property that no message can be linked to any sender and no sender can be linked to any message*

Definition 8 (Receiver anonymity)**.** *Receiver anonymity is the property that no message can be linked to any receiver and no receiver can be linked to any message.*

Definition 9 (Relationship anonymity)**.** *Relationship anonymity is the property that, though we may be able to link messages to senders and messages to receivers, we cannot link any sender and receiver as communicating with each other. Relationship anonymity is also often referred to as* unlinkability*.*

1.1.2 Analysis Approaches

Realistic anonymity systems are unlikely to provide perfect anonymity, therefore analysis must be probabilistic in order to gain useful information about the partial anonymity a system may provide. Since by-hand analysis of realistic anonymity systems can quickly become intractable or error-ridden, various attempts have been made to automate this analysis. One of the first such attempts was Schneider and Sidiropoulos' [133] use of CSP and the tool FDR to prove a finite instance of the dining cryptographers problem [22]. Morgan has recently used an extension of the Guarded Command Language with "ignorance-preserving refinement" to the same effect [109]. Both approaches capture only total anonymity and did not include any notion of probability. Recently, Deng et al. [40] used PCTL and the probabilistic model-checker PRISM [82] to do a similar analysis of the dining cryptographers problem using a probabilistic definition of anonymity. They also examined a slightly modified version of the problem in which total anonymity is not preserved. Shmatikov [140, 141] also used PCTL and PRISM to analyse the Crowds [125] system and identified an attack. Dingledine et al. used PCTL and PRISM to compare free-route and cascade mix topologies with synchronous batching [53]. A more detailed review of research on techniques for analysing anonymity systems can be found in Section 5.6.

1.2 Formal methods for security analysis

Gene Spafford once said [44],

> "The only truly secure system is one that is powered off, cast in a block of concrete and sealed in a lead-lined room with armed guards – and even then I have my doubts."

While some, if not most, of the security community will agree with this sentiment, it does not negate the desire to develop technologies that increase the security of systems along with methods to analyse and quantify the relative security of those technologies. As privacy-enhancing technologies (PETs) are developed, the need arises for methods to analyse and quantify the relative privacy guarantees of various PETs. It is important that these methods of analysis are quantitative, rather than a boolean result of "secure" or "insecure", because many deployable PETs must make concessions for the sake of efficiency and do not guarantee absolute privacy. Pioneering work done independently by both Serjantov and Danezis [136] and Díaz et al. [50] proposed the use of entropy as a metric for privacy, thereby linking Shannon's information theory [138] to quantitative analysis of privacy. Their work has had a great impact on the field and most quantitative privacy analysis is now rooted in information theory. Subsequent work in this area, such as Chatzikokolakis' use of Bayes risk [18], has offered a variety of related metrics for privacy based on information theory.

In the past, various products of formal methods research, such as theorem-provers and model-checkers, have fruitfully been applied to security analysis. Paulson developed the technique of using a theorem-prover to formalise security protocols and to perform inductive proofs of security for those protocols [119]. Model-checking has been widely used both for finding bugs in security protocols and for verifying finite instances of security protocols; Lowe's use of FDR to find a bug in the Needham-Schroeder protocol [94] is a well-known example. Each of these analysis techniques has its relative advantages and disadvantages. Model-checking requires less human effort because it is fully automatic, once a system and its desired properties are formalised. However, model-checking is typically limited to small, finite instances of a system due to a state explosion that renders larger instances intractable. Clearly, verification of a particular instance of the system does not necessarily provide any guarantees about the system generally. This makes model-checking techniques more applicable for bug-finding in security systems than for proving security guarantees.

Theorem-proving usually requires a greater amount of human effort than model-checking, because proofs are not automatic and require interaction with the theorem-proving system; nevertheless, this additional effort can be worthwhile. The advantage of theorem-proving is that proofs can be quantified on the parameters of the system, allowing for a proof of security for the system generally rather than for a particular finite instance. This advantage of theorem-proving is exemplified by the proof of privacy of the dining cryptographers protocol discussed in Chapter 5, which is valid for an *unbounded number* of protocol participants. While theorem-proving aims for proofs of correctness rather than bug-finding, an unsuccessful proof attempt will provide insight into the reason that the system fails. The further advantage of interactive theorem-proving over pen-and-paper proofs is that proofs are guaranteed to be correct up to the assumptions of the model. Recent work by Blanchet [14] has focused on bridging the gap between the computational-complexity proofs used by cryptographers and the assumption of perfect cryptographic primitives commonly used in formal methods proofs.

I do not claim that a machine-assisted proof is an absolute guarantee of security. Any proof of security is only valid up to the level of abstraction at which the system is modelled, a point widely noted in the literature where security and formal methods research intersect. However, LCF-style theorem-provers, such as the HOL4 system used in this work, do guarantee the logical consistency of all proofs they produce [68]. This is achieved by using a small logical core from which all theorems must be derived using basic inferences rules. A substantial body of mathematical theory has been formalised in HOL4 over the course of its development. Since pen-and-paper proofs are often long, complicated, and error-prone, the guarantee of correctness provided by using a theorem-prover is invaluable, particularly for security applications. Gordon [68] provides a nice historical overview and introduction to the LCF philosophy and HOL theorem-proving.

Historically, formal analysis of security-sensitive systems has been evenly divided between the camps of model-checking and theorem-proving, but the privacy domain seems to be an exception. Model-checking techniques have been widely used to analyse various PETs, for example Shmatikov et al.'s use of PRISM [141] to analyse the Crowds protocol [125]. Yet, the work of Kawabe et al. [86] is the only use of theorem-proving techniques for formal privacy analysis that I am aware of, other than my own work. This text aims to continue filling that void in research on formal methods for privacy analysis, as Kawabe has begun to do. In spirit this work belongs both to the branch of PETs research begun by Serjantov, Danezis, and Díaz and to the branch of formal methods work begun by Paulson. Recent work

by Malacaria et al. proposed an information-theoretic approach to analysing the information leakage in programs [99] and that work serves as inspiration for the formalised analysis technique presented here. Overall, mathematical and logical rigour have trumped ease of implementation in this work, but some tool support and automation have also been developed for ease of use.

1.3 Overview of technique

This section provides a brief look ahead at the framework for analysing anonymity systems that forms the core of this text. We will begin by examining a number of goals for that framework, which have motivated key choices in the developments presented later.

1.3.1 Motivating goals

Systems proposed for anonymous web-browsing or email often rely on subtle details and are deployed in complex dynamic environments, rendering it difficult to analyse or prove exactly how much anonymity they provide. Analysis of anonymity systems has improved through the development of probabilistic metrics for measuring anonymity [50, 136] and the use of automated tools suitable for reasoning about probabilities [140]. With the development of new techniques, further analysis of anonymity systems will increase understanding of how different systems perform in varying settings, identify potential attacks on systems, and provide direction for the development of future systems.

From the outset, one of the primary requirements for the techniques developed in this text was that they be sufficiently *formal* to provide a high assurance of mathematical and logic consistency for any proofs developed. In a security-sensitive setting, this level of assurance is necessary for proofs to be considered beneficial. An additional objective for the framework was the ability to prove *general* properties of systems, parameterised on various settings (e.g. the number of participants in the dining cryptographers protocol) — *not just for small instances of the system.* The first objective suggests the use of a model-checking or theorem-proving approach, while the second objective narrows that choice to the use of an interactive theorem-prover. The historical importance of both model-checking and theorem-proving in security analysis, coupled with the scarcity of research using theorem-provers in the privacy domain, supports the decision to use an interactive theorem-prover.

Since most deployed anonymity systems must compromise between efficiency and perfect security, techniques for analysing those systems must be capable of examining *partial-anonymity.* Moreover, in order to compare different PETs and environments, it is important that they are examined using *quantitative* metrics for anonymity. Finally, many PETs make use of probabilistic behaviour in order to achieve their results, so an appropriate analysis framework must also be able to capture that *probabilistic* behaviour. All of those requirements, together with the recent research on anonymity metrics discussed above, suggest the use of an information-theoretic metric for anonymity, specifically conditional mutual information. Since the analysis framework needed to incorporate probability theory, I decided to use the HOL4 theorem-prover, for which a formalisation of probability theory already existed. Unfortunately, that formalisation ultimately needed to be redeveloped in a more general form, as is explained in Chapter 2. A thorough introduction to the HOL4 system will not be undertaken here, but such a tutorial can be found elsewhere [117] along with complete documentation for the system [114, 115, 116]. In most cases, knowledge of HOL4 syntax should not be needed to understand this text, but a glossary of such notation has been provided in Appendix A for convenience.

Finally, it is essential that the framework allow for easy and intuitive specification of systems and minimise the effort of interactive proofs, whenever possible. The first of these requirements motivated the decision to model systems as HOL functions, directly in the logic of the theorem-prover. That approach makes it much easier to define models of systems and prove basic correctness properties about them. In addition, automation and tool support have been developed to ease the difficulty of proofs using the framework.

1.3.2 Contributions

The developments presented in this thesis contribute to a number of distinct lines of research. The foundational work in Chapter 2 contributes to the theorem-proving community by providing a more general formalisation of measure and probability theories and the first formalisation of Lebesgue integration in

the HOL4 system. Those formalisations provide a basis for a wide range of future developments and some researchers (e.g. Hasan et al.) have already begun to build upon them, but those applications have not yet been completed and published. Building from the formalisations in Chapter 2, the formalisation of information theory developed in Chapter 3 is the first formalisation of information theory to be developed in a theorem-prover. That work also presents countless opportunities for new applications of theorem-proving in a variety of domains (e.g. coding theory).

The framework for analysing information leakage presented in Chapter 4 is the first of its kind. The major innovation it provides over previous theorem-proving approaches is the use of *quantitative* metrics for leakage, based on Shannon's information theory. This allows for reasoning about *partial* information leakage. While model-checkers have been used for quantitative analysis of information leakage, they are generally limited to small instances of a system. In contrast, the theorem-proving approach presented in Chapter 4 can use parameterised specifications and inductive techniques to prove properties for all configurations of a system. The automation and tool support developed in conjunction with that framework is a major contribution to its usability.

The proof of the dining cryptographers protocol in Chapter 5 is novel on two fronts. For one, it is the first machine-checked proof of anonymity of the dining cryptographers protocol, for an unbounded number of participants. Secondly, it is the first use of a theorem-prover in the anonymity domain that allows for specification of probabilistic systems and a quantitative metric for anonymity. That case study exhibits the applicability of the framework in Chapter 4 to proofs of privacy and anonymity, including partial anonymity.

Chapter 2

Probability, Measure, and Integration

"You cannot avoid measure theory"
– David Williams, Probability with Martingales

The primary objective for the work in this thesis was to develop a framework for quantifying the amount of private or identifying information leaked by a piece of software. The method for that analysis (Chapter 4) is based on Shannon's theory of information, which I have formalised in a higher-order logic theorem-prover (Chapter 3). Beginning with the decision to structure this work within a theorem-prover, I resolved that mathematical rigour would trump expediency whenever both could not be achieved; the work in this chapter is a direct result of that decision. Information theory builds directly upon probability theory, which is inextricably connected to measure and integration theories. These three theories must be formalised before information theory can be developed in the theorem-prover.

2.1 Motivation

As David Williams notes in *Probability with Martingales*, "you cannot avoid measure theory" in a mathematically rigourous treatment of probability theory. Measure theory is necessary for two reasons. Most importantly, it ensures mathematical consistency of the formalised definitions for probability theory. If measure theory were not used, inconsistencies based on the Banach-Tarski paradox could be introduced. Banach and Tarski [154] proved that it is possible to define a non-measurable set if the axiom of choice is assumed. A non-measurable set is a set for which no measure can be defined without resulting in a contradiction; non-measurable sets are outside the scope in which probability theory operates. Since the axiom of choice is assumed for most of conventional mathematics, as well as higher-order logic, Banach and Tarski's paradox must be considered. If we were to define a probability measure outright and then apply this definition to a non-measurable set, an inconsistency would arise. Any proofs involving such a definition would be meaningless, because a contradiction could be derived from the definition itself.

In order to avoid the Banach-Tarski paradox and ensure the correctness of proofs that use the formalisations in this text, probability measures must not be applied to non-measurable sets. This can be achieved by defining probability theory as an extension of measure theory; this theory defines the sets that are measurable with respect to a given measure. Constructing new definitions incrementally as extensions of others prevents inconsistencies and maintains a straightforward correspondence between formalised theories and textbook definitions.

Another motivation for using measure theory in this chapter is to provide a mathematical link between definitions involving discrete probability measures and continuous probability measures. This unification of discrete and continuous probability under one general theory was a driving force behind the development of Kolmogorov's measure-theoretic treatment in the first half of the twentieth century [89]. Prior to his work, probability theory was marginalised by many mathematicians as lacking mathematical rigour. Using measure theory, definitions can be formalised that generalise the discrete and continuous cases, providing a concrete connection between the two.

General definitions also have practical implications for formalisation work. Often, they are easier to work with because mechanised proofs involving such definitions are not complicated by unnecessary details. On numerous occasions, I have struggled with a mechanised proof only to find that it could be proved easily in a more general form and then applied to the particular case of interest. By using more general definitions, the formalisation developed in the theorem-prover can be applied to a wider range of future uses.

Let's revisit Williams's quote from *Probability with Martingales*, this time in its entirety:

> "You cannot avoid measure theory: an *event* in probability is a measurable set, a *random variable* is a measurable function on the sample space, the *expectation* of a random variable is its integral with respect to the probability measure; and so on."

Williams's words accurately capture the inseparability of the topics of probability, measure, and integration that form the core of this chapter. Some definitions in a measure-theoretic treatment of probability theory are constructed using (Lebesgue) integration. For example, both expectation and conditional expectation are defined using the Lebesgue integral and are needed to define concepts in information theory. Thus, just as measure theory must be formalised before information theory, so must integration theory.

The work presented here is not the first effort to formalise probability theory in a theorem-prover nor has the importance of measure and integration theories gone unnoticed in previous work. However, the formalisations in this chapter offer significant advantages over previous developments. In the next section, we will examine these contributions in comparison with past work.

2.2 Related work and novel contributions

Hurd's formalisations of measure and probability theories in HOL4 [84] and Richter's formalisation of Lebesgue integration in the Isabelle theorem-prover [128] have served as a guide for the work presented below. Also noteworthy are Białas's and Nędzusiak's [11, 12, 111, 112] formalisations of measure theory and probability theory in the Mizar theorem-prover, Hasan's [79, 78] extensions of Hurd's work to include expected value, Harrison's [77] formalisation of the gauge integral in HOL4, and Lester's work on topology in the PVS system [93].

2.2.1 Hurd's measure and probability theories in HOL4

Hurd [84] developed a formalisation of measure theory in HOL4, upon which he constructed definitions for probability spaces and functions on them. Hurd then used his formalisation to verify the correctness of probabilistic algorithms; his most interesting application was the verification of the Miller-Rabin primality test.

While Hurd's work [84] was a major milestone for machine-verification of probabilistic algorithms, the scope of that work was limited. He formalised many definitions from probability theory and proved important results about independent functions on probability spaces. Despite these contributions, his work did not include some probability-theoretic concepts needed to define information theory. For example, Hurd proved that the probability of the empty event is 0, the probability of the set of all events is 1, and a probability measure is countably additive. However, he did not formalise definitions for random variables, expectation, or conditional expectation.

Hurd's formalisations restrict the measure and probability spaces that can be constructed. A measure or probability space is a triple $(S, \mathbb{S}, \mu)$ consisting of a set called the space (S), a set of subsets of the space known as the measurable-sets or events ($\mathbb{S}$), and a (probability) measure (μ). Hurd's definitions do not explicitly include the space, which he specifies implicitly as the universal set of the appropriate HOL type. The universal set of a HOL type α is defined as

$$\texttt{UNIV} = \{\texttt{x} : \alpha \mid \top\},$$

i.e. all the elements of type α. Hurd's formalisations are restricted to probability and measure spaces of the form $(\texttt{UNIV}, \mathbb{S}, \mu)$. In fact, he formalised measure and probability spaces as pairs $(\mathbb{S}, \mu)$; the space on which they are defined is implied by the HOL type of the pair. This approach does not allow measure and probability spaces where S is not the universal set of a HOL type.

As mentioned above, one of the primary motivations for using measure theory was to generalise the definitions for continuous and discrete probability-measures under a single theory.[1] By formalising probability theory as an extension of measure theory, such general definitions can be used in the theorem-prover. These definitions can then be proved equivalent to simpler definitions for discrete (i.e. countable) and continuous spaces. This approach has been taken below so that simpler definitions can be used where applicable.

Although similar equivalences can be proved in Hurd's formalisation, they cannot be used easily. For example, one might be interested in a probability space $(S, \mathbb{S}, \mu)$, where S is a countable set of elements of HOL type α. If the universal set of α is countable, then a definition involving $(S, \mathbb{S}, \mu)$ can be simplified to its countable form. Although S is countable, the universal set of α might not be.[2] In these cases, Hurd's formalisation makes it difficult to simplify a definition to its countable form. Before such a reduction can be applied, a new HOL type must be defined for the elements of S. This requires a considerable effort, so it is not feasible to undertake for each countable space of interest. For example, in order to define a HOL type for a countable subset of the reals, it is necessary to redefine arithmetic operations on this set and prove properties of these operations. These complications are avoided in the formalisation below, because spaces are explicitly specified.

The formalisations of measure theory and probability theory presented in this chapter are modeled after Hurd's restricted formalisations. Some of Hurd's definitions and proofs could be generalised with only minor modifications; however, many of Hurd's proofs were not valid for the general case and had to be replaced. Despite overlap with Hurd's work, the formalisations presented here broaden the applicability of formal methods to probabilistic algorithms and ease the difficulty of that analysis.

2.2.2 Richter's integration theory in Isabelle/HOL

Recall that Lebesgue integration is needed to define a number of concepts in probability theory. Amongst these are expectation and conditional expectation, which feature heavily in subsequent chapters. Richter [128] formalised Lebesgue integration in the Isabelle/HOL theorem-prover. The formalisation of Lebesgue integration developed below is modelled after his work.

Unfortunately, Richter's formalisations are insufficient for the applications in this text on several accounts. Richter's work was guided by Hurd's and suffers from the same restrictions. In addition, his work allows only integration of functions that are measurable from the real numbers to the real numbers. The applications below require integration of functions from an arbitrary HOL type to the real numbers. Finally, Richter developed his formalisation in the Isabelle/HOL system, while the work in this text uses the HOL4 system; translation between the two systems is nontrivial.

Thus, a formalisation of Lebesgue integration that generalises Richter's had to be developed in HOL4. This approach allows equivalences to be proved between the general definition of the Lebesgue integral and simpler forms for specific classes of spaces. These simplifications are new contributions to the formalisation of Lebesgue integration. Furthermore, the work presented below is the only formalisation of Lebesgue integration that has been developed in the HOL4 system.

2.2.3 Białas and Nędzusiak's probability theories in Mizar

Białas and Nędzusiak developed the first formalisations of measure theory and probability theory, using the Mizar theorem-prover [11, 12, 111, 112]. Their pioneering work marked the introduction of probability theory into the theorem-proving domain and has provided insight and inspiration for subsequent developments in this line of research, such as Hurd's HOL formalisation of probability theory [84].

2.2.4 Hasan's expectation in HOL4

Building upon Hurd's work, Hasan [78, 79] formalised definitions of expected value for discrete and continuous probability distributions. He then used his formalisations to prove properties of various probability distributions such as the Bernoulli distribution. Hasan's definitions of expected value were restricted to the discrete space of the natural numbers (a countable HOL type) and the continuous space

[1] Measure theory even generalises measures that are neither continuous, nor discrete, nor a mixture of the two.
[2] For instance, α might be the real numbers or tuples representing program executions as in Chapter 4.

of the real numbers. His formalisations were not particularly affected by the limitations of Hurd's because he was only considering spaces that are the universal set of a HOL type.

Hasan did not formalise a general definition of expected value using Lebesgue integration, so he could not formalise a mathematical connection between his definitions for the discrete case and the continuous case. The formalisations developed below generalise his by allowing arbitrary spaces (rather than only the naturals or the reals) and by defining expected value using Lebesgue integration; this definition generalises those for the discrete and continuous cases.

Hasan's work devoted to proving properties of well-known probability distributions is tangential to aims of this text; however, his proofs could be reproduced using the more general framework developed in this chapter.

2.2.5 Harrison's gauge integral in HOL4

Harrison's formalisation of the real numbers in HOL4 [77] included a definition of the gauge integral for functions over the reals. His formalisation is not sufficiently general for the work in this chapter, which requires a definition of integration for functions from an arbitrary type to the real numbers. Moreover, Lebesgue integration is the natural choice for developing probability theory from measure theory and is used in most textbooks [54, 100, 155]. Thus, the Lebesgue integral has been selected for the formalisations developed below. Future work could include a proof of equivalence between the formalisation of integration in this chapter and Harrison's, when considering bounded real-valued functions on bounded intervals.[3]

2.2.6 Lester's topology and probability in PVS

Also noteworthy is Lester's work on topology in the PVS theorem-prover [93]. Lester developed formalisations for measure theory and Lebesgue integration in PVS. He then built on that work to formalise some portion of probability theory. At the time that the formalisations in this chapter were completed, I was unaware of Lester's work. There is some overlap between his formalisations and the work presented here. However, a number of important contributions of this chapter do not appear in Lester's work; amongst these are the formalisation of conditional expectation and proofs of equivalence between general definitions and simpler definitions for discrete spaces.

It is difficult to say precisely how much Lester's work overlaps with the formalisations developed below. One difficulty in comparing these two formalisations is that different approaches were taken for each. In this chapter, measures must be finite-valued (i.e. take values from the reals), but measurable functions are real-valued and may take negative values. Lester allowed measures to be infinite (i.e. take values from the reals, extended with positive and negative infinity elements), but required measurable functions to be positive. Below, we will see that Lester's restrictions simplified the formalisation process; however, some generality was lost because functions taking both positive and negative values cannot be integrated. Whether Lester's restriction to positive measurable-functions or my restriction to finite measures is a greater limitation would depend on the specific application. Another difficulty in comparing Lester's work with this chapter is that a different proof-assistant was used for each (PVS and HOL4 respectively).

⋆ ⋆ ⋆

The remainder of this chapter describes the formalisation of measure theory, Lebesgue integration, and probability theory, within the context of the HOL4 theorem-prover. Measure theory will be examined first, followed by Lebesgue integration, and ultimately probability theory. In general, the presentation will progress from the simplest to the most complex definitions for each theory. A number of properties of these formalisations have been proved in the theorem-prover, some of which appear in this chapter. These proofs should reassure the reader that the formalisations behave as expected and appropriately capture the textbook definitions. All of the HOL definitions and theories mentioned in this text can be found in the appendices and the full HOL4 theory files are freely available on the internet [].

Definitions from measure, integration, and probability theories will be reviewed prior to their formalisations in order to facilitate easier reading. However, these definitions will not be discussed in great detail; more detailed presentations can be found in standard textbooks on those subjects. Doob's and

[3] Under these conditions, Lebesgue and gauge integrability are equivalent.

Williams's books are an excellent starting point [54, 155]. Doob focuses more on the development of measure theory with applications to probability theory. Williams emphasises probability theory with a basis in measure theory. In general, the definitions from measure theory, Lebesgue integration, and probability theory found in this chapter are Doob's [54] restated to respect the notational conventions used here.

The presentation of the formalisations in this chapter avoids any unnecessary details about their implementation in the HOL4 theorem-prover. However, implementation details are included where necessary or of particular interest. In such cases, sufficient explanation is provided for someone unfamiliar with the syntax of HOL4. A glossary of HOL4 notation can be found in Appendix A.

2.3 Measure theory formalised in HOL4

This section is devoted to the formalisation of measure theory in higher-order logic, upon which the formalisations of Lebesgue integration (Section 2.4) and probability theory (Section 2.5) are built. Knowledge of this section will aid in understanding subsequent sections. As mentioned above, the formalisations developed here are modeled after and generalise Hurd's [84].

2.3.1 Subset classes and σ-algebras

Our investigation of measure theory in HOL begins with a study of important classes of subsets of a *space*. Before identifying particular classes of subsets, it is necessary to characterise what it means for a set to be a class of subsets of a particular space. To that end, we will review a definition for a *subset class* and then examine the formalisation of that definition in HOL.

Definition 10 (subset class)**.** $\mathbb{S}$ *is a* **subset class** *of a space* S *iff all the elements of* $\mathbb{S}$ *are subsets of* S*.*

The formalisation of Definition 10 in higher-order logic is straightforward as can be seen below.

Formalisation 1 (subset class)**.**

$$\texttt{subset\_class sp sts} \;=\; \forall\, \texttt{x}.\; \texttt{x IN sts} \;\Rightarrow\; \texttt{x SUBSET sp}.$$

We will now review a textbook definition for an important type of subset class known as an *algebra*; afterwards we will examine the formalisation of this definition in higher-order logic.

Definition 11 (algebra)**.** *A class* $\mathbb{S}$ *of subsets of a space* S *define an* **algebra** *iff*

(i) $\mathbb{S}$ *contains the empty set,*
(ii) $\mathbb{S}$ *is closed under complementation (within* S*), and*
(iii) $\mathbb{S}$ *is closed under finite unions.*

Note that conditions (ii) and (iii) together are equivalent to condition (ii) and the condition that $\mathbb{S}$ is closed under finite intersections. Thus, an algebra is necessarily closed under both finite unions and finite intersections. Definition 11 has been formalised in higher-order logic as

Formalisation 2 (algebra)**.**

$$\begin{aligned}\texttt{algebra (sp, sts)} \;=\; &\texttt{subset\_class sp sts} \wedge \\ &\{\}\ \texttt{IN sts} \wedge (\forall\, \texttt{s}.\ \texttt{s IN sts} \Rightarrow \texttt{sp DIFF s IN sts}) \wedge \\ &(\forall\, \texttt{s t}.\ \texttt{s IN sts} \wedge \texttt{t IN sts} \Rightarrow \texttt{s UNION t IN sts}),\end{aligned}$$

where $\{\}$ *represents the empty set and* `DIFF` *and* `UNION` *are the set-difference and -union operations respectively. Note that within a space* S*, the complement of a set* $s \subseteq S$ *is the difference of* S *and* s*.*

A simple line-by-line comparison reveals that Formalisation 2 captures Definition 11. The first line of the formalisation captures the implicit condition in Definition 11 that `sp` and `sts` form a subset class. A straightforward correspondence between textbook definitions and their formalisations has been maintained wherever possible.

Let's digress for a moment to examine a concrete example of Hurd's restriction on spaces in his formalisation of measure theory [84]. As explained in Section 2.2, the differences between Hurd's formalisations

and those in this chapter are subtle, but they have a substantial impact on the respective formalisations. Hurd's formalisation requires that all spaces considered are the universal set of some HOL type (i.e. the set of all elements of that type). For example, Hurd's formalisation of Definition 11 above is

$$\begin{aligned}&\texttt{algebra sts} \;=\; \{\}\ \texttt{IN sts} \;\wedge\\ &\quad(\forall\ \texttt{s.}\ \texttt{s IN sts} \;\Rightarrow\; \texttt{COMPL s IN sts}) \;\wedge\\ &\quad(\forall\ \texttt{s t.}\ \texttt{s IN sts} \;\wedge\; \texttt{t IN sts} \;\Rightarrow\; \texttt{s UNION t IN sts}),\end{aligned}$$

where `COMPL s` is the set-complementation operation equal to `UNIV DIFF s`.[4] Notice that Hurd's formalisation does not require that `sts` is a class of subsets of `UNIV`. That requirement is trivially true in his formalisation because all sets of a given type are subsets of the universal set of that type.

We now continue looking at important classes of subsets by reviewing a definition for a *σ-algebra* followed by its formalisation in higher-order logic.

Definition 12 (σ-algebra)**.** *A subset class* $\mathbb{S}$ *of a space* S *defines a* **σ-algebra** *iff* $(S, \mathbb{S})$ *defines an algebra and* $\mathbb{S}$ *is closed under* countable *(possibly infinite) unions.*

Formalisation 3 (σ-algebra)**.**

$$\begin{aligned}&\texttt{sigma\_algebra (sp, sts)} \;=\; \texttt{algebra (sp, sts)} \;\wedge\\ &\quad(\forall\ \texttt{c.}\ \texttt{countable c} \;\wedge\; \texttt{c SUBSET sts} \;\Rightarrow\; \texttt{BIGUNION c IN sts}),\end{aligned}$$

where `BIGUNION S` *is the union operation applied over the elements of a set of sets* `S` *and is equivalent to the mathematical notation* $\bigcup_{x \in S} x$*.*

For a space S, $(S, \{\emptyset, S\})$ is the smallest σ-algebra and $(S, \mathcal{P}(S))$ the largest, where $\emptyset$ and $\mathcal{P}(S)$ denote the empty set and the powerset of S respectively.

Typically, $\sigma(S, \mathbb{G})$ is used to denote the smallest σ-algebra defined on a space S and containing a generating set of subsets $\mathbb{G}$. The function constructing this smallest σ-algebra can be defined in HOL as follows:

Formalisation 4 ($\sigma(S, \mathbb{G})$)**.**

$$\texttt{sigma (sp, sts)} \;=\; (\texttt{sp, BIGINTER}\ \{\texttt{s} \mid \texttt{sts SUBSET s} \;\wedge\; \texttt{sigma\_algebra (sp, s)}\}),$$

where `BIGINTER s` *is the set of elements that are in all the sets in* `s`*.*

The following reassuring property has been proved about `sigma` in HOL4:

$$\forall\ \texttt{sp sts.}\ \texttt{subset\_class sp sts} \;\Rightarrow\; \texttt{sigma\_algebra (simga (sp, sts))}$$

i.e. `sigma (sp, sts)` defines a σ-algebra assuming `sts` is a class of subsets of `sp`.

2.3.2 Measure spaces

One of the most important developments presented in this section is the HOL definition of a *measure space*. Before the definition of a measure space can be formalised, it is first necessary to formalise the properties of *positivity* and *countable additivity*, which measure spaces must satisfy. We now review definitions for those properties and then examine their formalisations in higher-order logic.

Definition 13 (positivity)**.** *Let* $\mathbb{S}$ *be a class of subsets of space* S *containing the empty set and* λ *a function from* $\mathbb{S}$ *to* $\mathbb{R}$*. Then* λ *is* **positive** *with respect to* $(S, \mathbb{S})$ *iff* $\lambda(s) \geq 0$ *for any* $s \in \mathbb{S}$ *and* $\lambda(\emptyset) = 0$*.*

Formalisation 5 (positivity)**.**

$$\begin{aligned}&\texttt{positive (sp, sts, lambda)} \;=\\ &\quad(\texttt{lambda}\ \{\} = 0) \;\wedge\; (\forall\ \texttt{s.}\ \texttt{s IN sts} \;\Rightarrow\; 0 \;\leq\; \texttt{lambda s}).\end{aligned}$$

[4] i.e. the difference between the universal set of the type of `s` and `s`

Definition 14 (countable additivity)**.** *Let* $\mathbb{S}$ *be a class of subsets of space* S *and let* $\mathbb{S}$ *contain the empty set. Let* λ *be a function from* $\mathbb{S}$ *to* $\mathbb{R}$*. Then* λ *is* **countably additive** *with respect to* $(S, \mathbb{S})$ *iff*

$$\lim_{n\to\infty} \sum_{j=0}^{n} \lambda(s_j) \to \lambda(\bigcup_i s_i),$$

for any sequence $s_0, s_1, \ldots$ *of elements of* $\mathbb{S}$ *such that* $\bigcup_i s_i \in \mathbb{S}$ *and all* s_j *and* s_k *are* disjoint *when* $j \neq k$*.*

Formalisation 6 (countable additivity)**.**

$$\begin{array}{l}\texttt{countably\_additive}\ (\texttt{sp}, \texttt{sts}, \texttt{lambda})\ = \\ \quad \forall\ (\texttt{f} : \texttt{num} \to \texttt{sts}).\ (\forall\ \texttt{m}\ \texttt{n}.\ \texttt{m} \neq \texttt{n}\ \Rightarrow\ \texttt{DISJOINT}\ (\texttt{f}\ \texttt{m})\ (\texttt{f}\ \texttt{n}))\ \wedge \\ \qquad \texttt{BIGUNION}\ (\texttt{IMAGE}\ \texttt{f}\ (\texttt{UNIV} : \texttt{num} \to \texttt{bool}))\ \texttt{IN}\ \texttt{sts}\ \Rightarrow \\ \qquad (\texttt{lambda} \circ \texttt{f})\ \texttt{sums} \\ \qquad (\texttt{lambda}\ (\texttt{BIGUNION}\ (\texttt{IMAGE}\ \texttt{f}\ (\texttt{UNIV} : \texttt{num} \to \texttt{bool})))),\end{array}$$

where `IMAGE f s` *is the set of values taken by* `f` *when applied to the elements of* `s`*,* $\circ$ *is the function composition operation, and* `g sums c` *is equivalent to the mathematical notation* $\lim_{n\to\infty}(\sum_{i=0}^{n} \texttt{g}(i)) \to \texttt{c}$*.*

The correspondence between Definition 13 and Formalisation 5 is apparent. Definition 14 is more complicated, so Formalisation 6 requires careful examination. Note that, a sequence of sets s_i can be characterised by a function f from $\mathbb{N}$ to the elements of s_i, assuming that s_i is countable and its elements are disjoint; this function f maps the natural numbers to unique elements of s_i. This approach has been used to implicitly represent a countable sequence of sets in Formalisation 6.

At last we are ready to examine the HOL formalisation of a *measure space*; we begin by reviewing a textbook definition for measure spaces.

Definition 15 (measure space)**.** *Let* $\mathbb{S}$ *be a class of subsets of* S *such that* $(S, \mathbb{S})$ *defines a* σ*-algebra and let* λ *be a function from* $\mathbb{S}$ *to* $\mathbb{R}$*. Then* $(S, \mathbb{S}, \lambda)$ *defines a* **measure space** *iff* λ *is* positive *and* countably additive *with respect to* $(S, \mathbb{S})$*. Equivalently, one can state that* $(S, \mathbb{S})$ *is* λ-measurable*.*

S is referred to as the space *of the measure space,* $\mathbb{S}$ *as the* measurable sets *of the measure space, and* λ *as the* measure *of the measure space.*

Building on the formalisations developed above, it is straightforward to capture Definition 15 in a HOL formalisation.

Formalisation 7 (measure space)**.**

$$\begin{array}{l}\texttt{measure\_space}\ (\texttt{sp}, \texttt{sts}, \texttt{lambda})\ =\ \texttt{sigma\_algebra}\ (\texttt{sp}, \texttt{sts})\ \wedge \\ \quad \texttt{positive}\ (\texttt{sp}, \texttt{sts}, \texttt{lambda})\ \wedge\ \texttt{countably\_additive}\ (\texttt{sp}, \texttt{sts}, \texttt{lambda}).\end{array}$$

Rather confusingly, the term *measurable space* is used synonymously with the term σ-algebra in most textbooks. This is something of a misnomer as there is not necessarily a measure λ such that $(S, \mathbb{S}, \lambda)$ is a measure space, for an arbitrary σ-algebra $(S, \mathbb{S})$. For example, Vitali's theorem[5] demonstrates the existence of non-measurable sets in the space $(\mathbb{R}, \mathcal{P}(\mathbb{R}))$, assuming the axiom of choice. Since a set and its powerset form a σ-algebra, it follows that there are σ-algebras which lack a well-defined measure. Nonetheless, the term measurable space will be used here in the usual textbook sense, but the reader should remember that this is a necessary (but not sufficient) condition for the existence of a measure on a space.

2.3.3 Measurable functions

We complete our exploration of measure theory in HOL4 by studying definitions for *measurable functions* and *measure-preserving functions*. These properties are important for the definitions of Lebesgue integration (Section 2.4) and random variables (Section 2.5).

[5] Vitali's theorem implies that a measure returning the length of an open interval cannot be defined on $(\mathbb{R}, \mathcal{P}(\mathbb{R}))$, when the axiom of choice is assumed.

Definition 16 (measurable function)**.** *Let* $(S', \mathbb{S}')$ *and* $(S, \mathbb{S})$ *be measurable spaces. Let* $f[s]$ *denote the image of a function* f *on a set* s *and* $f^{-1}[s]$ *denote the inverse image of* f *on* s *i.e.* $\{y \mid \exists x.\ x \in s \wedge f(x) = y\}$ *and* $\{x \mid f(x) \in s\}$ *respectively. A function* f *from* S *into* S' *is* **measurable** *with respect to* $(S, \mathbb{S})$ *and* $(S', \mathbb{S}')$ *iff* $f^{-1}[s'] \in \mathbb{S}$ *for any* $s' \in \mathbb{S}'$*. Often,* f *is referred to as a measurable function from* $(S, \mathbb{S})$ *into* $(S', \mathbb{S}')$*.*

Notice that, as with the standard usage of the term measurable space, a function may be measurable from one space to another without guaranteeing the existence of appropriate measures for those spaces.

Formalisation 8 (measurable function)**.**

```
measurable (sp, sts) (sp', sts') f = sigma_algebra (sp, sts) ∧
  sigma_algebra (sp', sts') ∧ (∀ x. x IN sp ⇒ f x IN sp') ∧
  (∀ s'. s' IN sts' ⇒ ((PREIMAGE f s') INTER sp) IN sts),
```

where `INTER` *is the set-intersection operation and* `PREIMAGE f s` *denotes* $f^{-1}[s]$*.*

Now lets look at the translation of Definition 16 into higher-order logic. Recall from above that a space is measurable if and only if it defines a σ-algebra. Thus, the first two conditions of Formalisation 8 correspond to the requirement in Definition 16 that both of the spaces involved are measurable. The next condition corresponds to the requirement that the function is from the one space into the other. The final condition is the property required of the inverse-image of the function by the definition. In Formalisation 8, the inverse image of function `f` on set `s'` is intersected with space `sp`, but no such intersection occurs in Definition 16. This intersection is needed because HOL4 functions are total and `f` maps every value of the appropriate HOL type (including those outside of `sp`) to a value of the appropriate HOL type (which may or may not be in `sp'`). The third condition of the formalisation ensures that `f` maps values in `sp` into `sp'`, but there may still be values outside of `sp` which `f` maps into `sp'`. In the mathematical definition, a function from S to S' is only defined on S, so the intersection with S is unnecessary; in the formalisation, however, the inverse image of `f` must be intersected with the space `sp` in order to consider only values in `sp`. The inverse image of `f` must be intersected with the space `sp` in Formalisation 9 for the same reason as in Formalisation 8.

Definition 17 (measure-preserving)**.** *Let* $(S, \mathbb{S}, \lambda)$ *and* $(S', \mathbb{S}', \lambda')$ *be measure spaces and* f *a measurable function from* $(S, \mathbb{S})$ *into* $(S', \mathbb{S}')$*. Then* f *is* **measure preserving** *with respect to* $(S, \mathbb{S}, \lambda)$ *and* $(S', \mathbb{S}', \lambda')$ *iff* $\lambda(f^{-1}[s']) = \lambda'(s')$*, for any* $s' \in \mathbb{S}'$*.*

Formalisation 9 (measure-preserving)**.**

```
measure_preserving (sp, sts, lambda) (sp', sts', lambda') f =
  measure_space (sp, sts, lambda) ∧
  measure_space (sp', sts', lambda') ∧
  measurable (sp, sts) (sp', sts') f ∧
  (∀ s'. s' IN sts' ⇒
  (lambda ((PREIMAGE f s') INTER sp) = lambda' (s'))).
```

Finally, lets examine a definition of the *Borel* space and measurability thereupon.

Definition 18 (Borel space)**.** *Let* $\mathbb{B}$ *be the sets of open (or equivalently closed) intervals of the real numbers. The* Borel space *of real numbers is the smallest* σ*-algebra generated by* $\mathbb{B}$*, namely* $\sigma(\mathbb{R}, \mathbb{B})$*.*

Formalisation 10 (Borel space)**.**

```
borel_space = sigma UNIV (IMAGE (λa. {x | x ≤ a}) UNIV).
```

The σ-algebra generated by the Borel sets of real numbers is of particular importance for real-valued measurable functions. Borel-measurability of a set implies its Lebesgue measurability, which is useful when proving integration properties for real-valued functions.

Definition 19 (Borel-measurable)**.** *A function* f *is* Borel-measurable *with respect to a space* $(S, \mathbb{S})$ *iff it is measurable from* $(S, \mathbb{S})$ *onto the Borel space.*

Formalisation 11 (Borel-measurable)**.**

$$\texttt{borel\_measurable}\ (\texttt{sp}, \texttt{sts})\ \texttt{f}\ =\ \texttt{measurable}\ (\texttt{sp}, \texttt{sts})\ \texttt{borel\_space}\ \texttt{f}.$$

This concludes our study of measure theory formalised in higher-order logic. The majority of the effort in formalising the definitions above was devoted to proving that they satisfy appropriate properties. The proofs of those theorems are not included in this text, but the statement of each in HOL4 notation can be examined in Appendices B and C.

2.4 Lebesgue integration formalised in HOL4

We will now focus on Lebesgue integration formalised in higher-order logic. Recall from Section 2.2.2 that this formalisation must be applicable to measurable functions from a space of an arbitrary type to a space that is a subset of the real numbers. Since the formalisation of measure theory in Section 2.3 defined measures to be real-valued (rather than using the extended reals or hyper reals), the integrals in this section are necessarily finite-valued; in the case of integrals diverging to positive or negative infinity, the integral is undefined.[6] This restriction to finite-valued measures was also adopted in the work of Hurd [84] and Richter [128] and still allows for a broad range of applications. In the formalisations below, the Lebesgue integral is defined using *positive simple functions*, as is done in standard textbooks [54, 100, 155] and in Richter's work. Positive simple functions are also frequently referred to as *step functions*.

2.4.1 Indicator functions and positive simple functions

We begin our study of Lebesgue integration in HOL by reviewing definitions for *indicator functions* and *positive simple functions*, which are needed to define the Lebesgue integral.

Definition 20 (indicator function)**.** *The* **indicator function** *of a set A, denoted by 1_A, is a real-valued function defined to be*

$$1_A(a) = \begin{cases} 1 & (a \in A) \\ 0 & (a \notin A) \end{cases}$$

Formalisation 12 (indicator function)**.**

$$\texttt{indicator\_fn A}\ =\ \lambda\texttt{a}.\ \texttt{if a IN A then 1 else 0}.$$

We now move on to review the definition of a *positive simple function* and its integral with respect to a measure space. Afterwards we will look at the formalisation of these definitions in higher-order logic.

Definition 21 (positive simple function)**.** *Let $(S, \mathbb{S}, \lambda)$ be a measure space. A function f is a* **positive simple function** *or* **step function** *with respect to $(S, \mathbb{S}, \lambda)$ iff it can be defined as a linear combination of indicator functions of a finite number of disjoint elements of $\mathbb{S}$ with strictly positive coefficients i.e.*

$$f = \lambda x.\ \sum_{i=0}^{n} c_i(1_{a_i}(x)),$$

for some finite *set of indices $i \in \{0, \ldots, n\}$, a set of coefficients c_i satisfying*

$$\forall i \in \{0, \ldots, n\}.\ 0 < c_i,$$

and a set of measurable sets of $(S, \mathbb{S}, \lambda)$, a_i, satisfying

$$(\forall i \in \{0, \ldots, n\}.\ a_i \in \mathbb{S}) \wedge (\forall i, j \in \{0, \ldots, n\}.\ i \neq j \Rightarrow a_i \cap a_j = \emptyset).$$

Equivalently, the coefficients c_i above can be required to be non-negative, rather than strictly positive, and the measurable sets a_i required to form a partition of S i.e. $\bigcup_{i=0}^{n} a_i = S$.

The integral of a positive simple function f, defined by coefficients c_i and measurable sets a_i, on space S with respect to its measure λ is defined as

$$\int_S f\ d\lambda = \sum_{i=0}^{n} c_i(\lambda(a_i)).$$

[6] Strictly speaking, functions in HOL4 cannot have an undefined value and are instead assigned the value of an arbitrary constant `ARB`.

The HOL definition of a positive simple function below uses the second representation from Definition 21, allowing the coefficients to be non-negative and requiring the measurable sets to form a partition of the measure space. This form simplifies the proofs of some basic properties of positive simple functions; Richter [128] used the same approach.

Formalisation 13 (positive simple function)**.**

$$\begin{aligned}&\texttt{pos\_simple\_fn}\ (\texttt{sp},\texttt{sts},\texttt{lambda})\ \texttt{f}\ \texttt{s}\ \texttt{a}\ \texttt{c}\ =\ (\forall\ \texttt{x}.\ 0 \leq \texttt{f}\ \texttt{x}) \wedge \\ &\quad (\forall\ \texttt{x}.\ \texttt{x}\ \texttt{IN}\ \texttt{sp} \Rightarrow (\texttt{f}\ \texttt{x} = \texttt{SIGMA}\ (\lambda\ \texttt{i}.\ \texttt{c}\ \texttt{i} * (\texttt{indicator\_fn}\ (\texttt{a}\ \texttt{i})\ \texttt{x}))\ \texttt{s})) \wedge \\ &\quad (\forall\ \texttt{i}.\ \texttt{i}\ \texttt{IN}\ \texttt{s}\ \Rightarrow\ \texttt{a}\ \texttt{i}\ \texttt{IN}\ \texttt{sts})\ \wedge\ (\forall\ \texttt{i}.\ 0 \leq \texttt{c}\ \texttt{i}) \wedge \\ &\quad \texttt{FINITE}\ \texttt{s}\ \wedge\ (\forall\ \texttt{i}\ \texttt{j}.\ \texttt{i}\ \texttt{IN}\ \texttt{s}\ \wedge\ \texttt{j}\ \texttt{IN}\ \texttt{s}\ \Rightarrow\ \texttt{DISJOINT}\ (\texttt{a}\ \texttt{i})\ (\texttt{a}\ \texttt{j})) \wedge \\ &\quad (\texttt{BIGUNION}\ (\texttt{IMAGE}\ \texttt{a}\ \texttt{s}) = \texttt{sp}),\end{aligned}$$

where (sp, sts, lambda) *is a measure space,* f *is the function in question,* s *is a set of natural numbers representing the indices,* a *is a function from natural numbers to sets representing the measurable sets* a_i*, and* c *is a function from natural numbers to the real numbers representing the coefficients* c_i*.* SIGMA f s *is equivalent to* $\sum_{x \in s} f(x)$*, where* s *is a finite set.*

The first condition in Formalisation 13 ensures that f takes only non-negative values.[7] This condition is implicit in the positivity of the measure λ and the non-negativity of the coefficients c_i in Definition 21. The second condition requires that the values taken by f on sp are defined by a linear combination of the coefficients and the indicator functions of the measurable sets. The rest of the conditions are respectively: the a i's are in sts, the coefficients are non-negative, the set of indices is *finite*, the a i's are *disjoint* for different indices, and finally the a i's form a partition of sp.[8]

Building on the formalised definition of a positive simple function, the integral thereof has been defined in HOL as follows.

Formalisation 14 (integral of a positive simple function)**.**

$$\begin{aligned}&\texttt{pos\_simple\_fn\_integral}\ (\texttt{sp},\texttt{sts},\texttt{lambda})\ \texttt{s}\ \texttt{a}\ \texttt{c}\ = \\ &\quad \texttt{SIGMA}\ (\lambda\ \texttt{i}.\ \texttt{c}\ \texttt{i}\ *\ \texttt{lambda}\ (\texttt{a}\ \texttt{i}))\ \texttt{s}.\end{aligned}$$

Note that there is no explicit reference to the function that is being integrated; instead it is represented through an index set, coefficients, and measurable sets. The representation of a positive simple function f by a set of coefficients and measurable sets is not necessarily unique i.e. there may be many combinations of coefficients and measurable sets that are equivalent. Moreover, the integral of f is unique regardless of which representation is used to compute it. Thus, the integral of f can be referred to directly, without making explicit reference to sets and coefficients.

Following Richter's [128] lead, the *set of integrals* of a function f has been formalised, allowing the integral of f to be referred to directly. The set of integrals of f contains a single unique element when f is a positive simple function and is empty otherwise. Below we will examine the HOL definitions for the set of representations of a positive simple function and the set of integrals of a particular positive simple function.

Formalisation 15 (set of representations of a positive simple function)**.**

$$\begin{aligned}&\texttt{psfs}\ (\texttt{sp},\texttt{sts},\texttt{lambda})\ \texttt{f}\ = \\ &\quad \{(\texttt{s},\texttt{a},\texttt{c})\ |\ \texttt{pos\_simple\_fn}\ (\texttt{sp},\texttt{sts},\texttt{lambda})\ \texttt{f}\ \texttt{s}\ \texttt{a}\ \texttt{c}\}.\end{aligned}$$

Formalisation 16 (set of integrals of a positive simple function)**.**

$$\begin{aligned}&\texttt{psfis}\ (\texttt{sp},\texttt{sts},\texttt{lambda})\ \texttt{f}\ = \\ &\quad \texttt{IMAGE}\ (\lambda\ (\texttt{s},\texttt{a},\texttt{c}).\ \texttt{pos\_simple\_fn\_integral}\ (\texttt{sp},\texttt{sts},\texttt{lambda})\ \texttt{s}\ \texttt{a}\ \texttt{c}) \\ &\qquad (\texttt{psfs}\ (\texttt{sp},\texttt{sts},\texttt{lambda})\ \texttt{f}).\end{aligned}$$

[7] Although only the values f takes on sp are of concern, f must be a total function in HOL4 and might take negative values when applied outside of sp. For the sake of convenience f is required to be non-negative for all inputs; this is done without any loss of generality. The same approach is taken with the second condition in the formalisation.

[8] More precisely, the final condition is that the union of the a i's is sp which together with their disjointness makes them a partition.

Before moving on to look at integration for positive measurable functions, we will briefly review some useful and reassuring properties that have been proved about the formalisations developed above.

HOL Theorem 1 (uniqueness)**.** *The integral of a positive simple function is* unique *regardless of the choice of representation using coefficients and measurable sets.*

$$\forall\ \mathtt{m\ f\ a\ b}.\ \mathtt{measure\_space\ m}\ \wedge\ \mathtt{a\ IN\ psfis\ m\ f}\ \wedge\ \mathtt{b\ IN\ psfis\ m\ f}\ \Rightarrow\ (\mathtt{a} = \mathtt{b}).$$

HOL Theorem 2 (additivity)**.** *The integral of the function defined by adding two positive simple functions is the addition of their integrals.*

$$\forall\ \mathtt{m\ f\ g\ a\ b}.\ \mathtt{measure\_space\ m}\ \wedge\ \mathtt{a\ IN\ psfis\ m\ f}\ \wedge\ \mathtt{b\ IN\ psfis\ m\ g}$$
$$\Rightarrow\ \mathtt{a} + \mathtt{b\ IN\ psfis\ m}\ (\lambda\ \mathtt{x}.\ \mathtt{f\ x} + \mathtt{g\ x}).$$

Note that this property can be generalised to any finite linear combination of positive simple functions; this generalised additivity property has also been proved.

HOL Theorem 3 (multiplicativity)**.** *The integral of the function defined by multiplying a positive simple function f by a non-negative constant c is the integral of f multiplied by c.*

$$\forall\ \mathtt{m\ f\ a\ c}.\ \mathtt{measure\_space\ m}\ \wedge\ \mathtt{a\ IN\ psfis\ m\ f}\ \wedge\ 0 \leq \mathtt{c}$$
$$\Rightarrow\ \mathtt{c} * \mathtt{a\ IN\ psfis\ m}\ (\lambda\ \mathtt{x}.\ \mathtt{c} * (\mathtt{f\ x})).$$

HOL Theorem 4 (monotonicity)**.** *If f is a positive simple function which is pointwise less than or equal to another positive simple function g, then the integral of f is less than or equal to the integral of g.*

$$\forall\ \mathtt{m\ f\ g\ a\ b}.\ \mathtt{measure\_space\ m}\ \wedge\ \mathtt{a\ IN\ psfis\ m\ f}\ \wedge\ \mathtt{b\ IN\ psfis\ m\ g}\ \wedge$$
$$(\forall\ \mathtt{x}.\ \mathtt{f\ x} \leq \mathtt{g\ x})\ \Rightarrow\ \mathtt{a} \leq \mathtt{b}.$$

HOL Theorem 5 (commonality)**.** *If f and g are positive simple functions on the same measure space m, then there is a set of measurable sets of m which, paired with a set of coefficients for f and a set of coefficients for g, can be used to characterise both f and g. The statement of this theorem in HOL4 is omitted here for the sake of brevity, but can be found in Appendix D as* `psfis_present`*.*

Theorem 1 (convergence of integrals of positive simple functions)**.** *Consider a measure space $(S, \mathbb{S}, \lambda)$ and a non-negative function f from S to $\mathbb{R}$. Let $\{f_i\}$ be a pointwise monotone-increasing sequence of positive simple functions such that*

$$\forall\ x \in S.\ \lim_{i \to \infty} f_i(x) \to f(x).$$

If $\lim_{i\to\infty} \int_S f_i\ d\lambda \to r$ for some r, then for any positive simple function g such that $\forall x.\ g(x) \leq f(x)$

$$\int_S g\ d\lambda \leq r.$$

In terms of subsequent developments in this chapter, Theorem 1 is the most important property that has been proved about integration for positive simple functions. Since the statement of that theorem is intricate and the proof lengthy, it appears here only in mathematical notation; the statement of Theorem 1 in HOL notation can be found in Appendix D under the name `psfis_mono_conv_mono`.

2.4.2 Integration of positive measurable functions

Having looked at integration for positive simple functions in the previous section, the next step towards Lebesgue integration for measurable functions is integration of *positive* measurable functions. We will now review a textbook definition for the integral of a positive measurable function, followed by its formalisation in higher-order logic.

Definition 22 (integral of a positive measurable function)**.** *Let $(S, \mathbb{S}, \lambda)$ be a measure space, $(S', \mathbb{S}')$ a measurable space, and f a non-negative real-valued function that is measurable from $(S, \mathbb{S})$ to $(S', \mathbb{S}')$. If there exists a pointwise monotone-increasing sequence of positive simple functions $\{f_i\}$ such that the f_i converge pointwise to f i.e.*

$$\forall\ x \in S.\ \lim_{i \to \infty} f_i(x) \to f(x)$$

and the sequence of integrals of the f_i's converges to a value y i.e.

$$\lim_{i \to \infty} \int_S f_i\ d\lambda \to y,$$

then the integral of f on S with respect to λ is defined to be y:

$$\int_S f\ d\lambda = y.$$

Notice that $\int_S f\ d\lambda$ is not defined for all f because a convergent sequence of step functions and a corresponding convergent sequence of integrals might not exist for a particular f. Such is the case when the integral of f diverges to positive or negative infinity. As with step functions, the *set of integrals* of a positive measurable function f has been formalised, allowing the integral of f to be referred to directly, without explicitly mentioning the sequence of approximating step functions. The set of integrals of f contains a single unique element when the integral exists and is empty otherwise.

Formalisation 17 (integral of a positive measurable function)**.**

```
nnfis (sp, sts, lambda) f =
  {y | ∃fi xi. mono_convergent fi f sp ∧
       ∀n. (xi n) IN psfis (sp, sts, lambda) (fi n) ∧
       xi → y}.
```

As with the integrals of step functions, the uniqueness, multiplicativity, additivity, and monotonicity of the integrals of non-negative measurable functions has been formalised. These theorems are not stated here as they are very similar to HOL Theorems 1–4, but they can be found in Appendix D as `nnfis_uniqe`, `nnfis_times`, `nnfis_add`, and `nnfis_mono`.

Building on Theorem 1 above, the Beppo-Levi monotone convergence theorem has been proved. This theorem is of great importance for subsequent formalisations, most notably for the proof simplifying the definition of Lebesgue integration for discrete spaces. The statement of this theorem is presented below in mathematical notation for the sake of brevity and readability; see Appendix D for the statement in HOL notation as `nnfis_mon_conv`.

Theorem 2 (Beppo-Levi monotone convergence theorem)**.** *Let $(S, \mathbb{S}, \lambda)$ be a measure space and f be a non-negative function from S to $\mathbb{R}$. Let $\{f_i\}$ be a pointwise monotone-increasing sequence of non-negative functions converging pointwise to f, i.e.*

$$\forall\ x \in S.\ \lim_{i \to \infty} f_i(x) \to f(x),$$

for which the integrals of all the f_i's exist and the sequence of integrals of the f_i's converges to a value y i.e.

$$\lim_{i \to \infty} \int_S f_i\ d\lambda \to y,$$

then

$$\int_S f\ d\lambda = y.$$

If a sequence of non-negative functions converge to a function f and their integrals converge to a value y, then Theorem 2 guarantees that the integral of f exists and is y.

2.4.3 Lebesgue integration of measurable functions

Having studied integration of positive simple functions and positive measurable functions above, we can now take the final step and examine the Lebesgue integral of a measurable function. We will now review a textbook definition of this integral.

Definition 23 (Lebesgue integral of a measurable function)**.** *Let* $(S, \mathbb{S}, \lambda)$ *be a measure space,* $(S', \mathbb{S}')$ *a measurable space, and* f *a real-valued measurable function from* $(S, \mathbb{S})$ *to* $(S', \mathbb{S}')$*. Let* $f^+(x) = \max(f(x), 0)$ *and* $f^-(x) = \max(-f(x), 0)$ *i.e.* f^+ *and* f^- *are the positive and negative portions of* f *respectively. Note that* $f(x) = f^+(x) - f^-(x)$ *and* f^+ *and* f^- *are both non-negative functions. Define the integral of* f *on* S *with respect to* λ *to be*

$$\int_S f \, d\lambda = \int_S f^+ \, d\lambda - \int_S f^- \, d\lambda.$$

Note that the integral of f *is well-defined iff* f^+ *and* f^- *are both measurable from* $(S, \mathbb{S})$ *to* $(S', \mathbb{S}')$ *and their integrals both exist.*

Seeing as the integral of a measurable function is simply the difference of the integrals of its positive and negative portions, it can be defined easily in terms of Formalisation 17.

Formalisation 18 (Lebesgue integral of a measurable function)**.**

```
integral (sp, sts, lambda) f =
  @x. x IN nnfis (sp, sts, lambda) (pos_part f) −
  @y. y IN nnfis (sp, sts, lambda) (neg_part f),
```

where `@x. P x` *is the Hilbert's choice operator equivalent to "any* `x` *for which* `P x` *holds" and* `pos_part f` *and* `neg_part f` *are the functions representing the absolute value of the positive and negative portions of* `f` *respectively.*

Formalisation 18 is undefined if the integral of either the positive or the negative portion of the function is infinite. In practice this is not often a limitation. To compare, Lester's formalisation [93] allows infinite values, but his formalisation stops with something akin to Formalisation 17 and does not allow for integration of functions taking both positive and negative values, which are commonly of interest.

Definition 23 generalises simpler definitions for finite and countable discrete spaces; Formalisation 18 has been proved equivalent to those simpler forms in the theorem-prover. Before looking at those proofs, let's examine HOL definitions of those simplified forms of the integral.

Formalisation 19 (integral of a measurable fn. on a countable, discrete space)**.**

```
countable_space_integral (sp, sts, lambda) f =
  let e = enumerate (IMAGE f sp) in
  suminf (λ i. e i * lambda (PREIMAGE f {e i} INTER sp)),
```

where `enumerate s` *is a bijection from the natural numbers to the elements of set* `s`*;* `enumerate s` *is well-defined only when* `s` *is a countably-infinite set.* `suminf f` *is the infinite summation of a real-valued function* `f` *on the natural numbers and is equivalent to the mathematical notation* $\sum_{i=0}^{\infty} f(i)$ *when this sum converges and is undefined otherwise.*

Because `enumerate` in Formalisation 19 is only well-defined on countably-*infinite* sets, it is necessary to formalise a separate definition for *finite* sets using a simpler finite summation.

Formalisation 20 (integral of a measurable fn. on a finite discrete space)**.**

```
finite_space_integral (sp, sts, lambda) f =
  SIGMA (λ r. r * lambda (PREIMAGE f {r} INTER sp)) (IMAGE f sp).
```

Clearly, it is simpler and more intuitive to define integration as a summation (as in Formalisations 19 and 20) than as a difference of limits of sums of approximating functions (as in Formalisation 18). Let's take a look at the HOL theorems showing the equivalence of the general defintion of Lebesgue integration to its simplified forms for discrete spaces; those proof are a useful contribution of this section and allow many general definitions in Section 2.5 to be simplified when considering *discrete* probability spaces.

HOL Theorem 6 (equivalence of Forms. 18 and 20 for finite, discrete spaces)**.**

```
∀ (sp, sts, lambda) f. measure_space (sp, sts, lambda) ∧
  f IN borel_measurable (sp, sts) ∧ FINTE sp ⇒
  (integral (sp, sts, lambda) f =
   finite_space_integral (sp, sts, lambda) f),
```

Notice that the function `f` is required to be borel-measurable for the equivalence above to hold. This does not pose any limitation because the existence of the the integral of `f` implies its borel-measurability.[9]

Now let's look at the case of countably-infinite spaces.

HOL Theorem 7 (equivalence of Forms. 18 and 19 for countable, discrete spaces)**.**

```
∀ (sp, sts, lambda) f p n. measure_space (sp, sts, lambda) ∧
  f IN borel_measurable (sp, sts) ∧ countable (IMAGE f sp) ∧
  ¬FINITE (IMAGE (pos_part f) sp) ∧ ¬FINITE (IMAGE (neg_part f) sp) ∧
  (λ r. r * lambda (PREIMAGE (pos_part f) {r} INTER sp)) ∘
      (enumerate (IMAGE (pos_part f) sp)) sums p ∧
  (λ r. r * lambda (PREIMAGE (neg_part f) {r} INTER sp)) ∘
      (enumerate (IMAGE (neg_part f) sp)) sums n ⇒
  (integral (sp, sts, lambda) f = p − n),
```

where `sums f r` *denotes that the infinite summation of* `f` *(as defined by* `suminf`*) converges to* `r` *and* $\circ$ *denotes the function composition operator.*

A number of useful properties of the Lebesgue integral have been proved in HOL. An example of one such useful property is that the integral of the indicator function of a measurable set is the measure of that set. The statement of that theorem in HOL4 and in mathematical notation can be found below.

HOL Theorem 8 (integral of the indicator fn. of a measurable set)**.**

```
∀ (sp, sts, lambda) s. measure_space (sp, sts, lambda) ∧ s IN sts ⇒
  (integral (indicator_fn s) = lambda s).
```

Equivalently in mathematical notation: Let $(S, \mathbb{S}, \lambda)$ *be a measure space.*

$$\forall\ a \in \mathbb{S}.\ \int_S 1_a\ d\lambda = \lambda(a).$$

2.4.4 Radon-Nikodým derivatives

The Radon-Nikodým derivative of one measure with respect to another measure is needed to define some of the information-theoretic concepts that will be presented in Chapter 3. Let's review a definition of the Radon-Nikodým derivative and then examine its formalisation in higher-order logic.

Definition 24 (Radon-Nikodým derivative)**.** *Let* $(S, \mathbb{S}, \mu)$ *and* $(S, \mathbb{S}, \nu)$ *be measure spaces. Notice that* μ *and* ν *are measures on the same space. Define the* **Radon-Nikodým derivative** *of* ν *with respect to* $(S, \mathbb{S}, \mu)$ *to be the borel-measurable function* f *on* S *such that*

$$\forall\ a \in \mathbb{S}.\ \nu(a) = \int_a f(a) d\mu,$$

where the integral above is a Lebesgue integral. The function f*, representing the Radon-Nikodým derivative of* ν *with respect to* μ*, is often denoted by* $\frac{d\nu}{d\mu}$*.*

Formalisation 21 (Radon-Nikodým derivative)**.**

```
RN_deriv (sp, sts, mu) nu =
  @f. measure_space (sp, sts, mu) ∧ measure_space (sp, sts, nu) ∧
      f IN borel_measurable (sp, sts) ∧
      ∀ a. a IN sts ⇒
        (integral (sp, sts, mu) (λ x. f x * indicator_fn a x) = nu a),
```

[9] The theorem stating that integrability implies borel-measurability can be found in Appendix D as `integral_borel_measurable`.

where @x. P x *denotes the Hilbert's choice operator; recall that* @x. P x *can be read informally as* "any x such that P x".

The Radon-Nikodým theorem guarantees the existence of such a derivative and its uniqueness, up to μ-null sets[10], for any absolutely continuous ν. More details about the Radon-Nikodým theorem can be found in standard textbooks addressing measure and integration [54, 100, 155]. The proof of that theorem is nontrivial and best left until a larger body of integrability theorems have been proved. However, the reduction of Formalisation 21 needed for the developments in Chapter 3 does not rely on a general proof of the Radon-Nikodým theorem. In the case of finite, discrete spaces, the R.-N. derivative of two measures of a non-null set reduces to a division of the two measures. This reduction is stated more rigourously below.

HOL Theorem 9 (Radon-Nikodým derivative of finite, discrete spaces)**.**

$$\begin{array}{l}\forall\ \texttt{sp sts mu nu.}\ \texttt{FINITE sp}\ \wedge\ \texttt{measure\_space}\ (\texttt{sp},\texttt{sts},\texttt{mu})\ \wedge \\ \quad \texttt{measure\_space}\ (\texttt{sp},\texttt{sts},\texttt{nu})\ \wedge \\ \quad (\forall\ \texttt{x.}\ (\texttt{mu}\ \{\texttt{x}\} = 0)\ \Rightarrow\ (\texttt{nu}\ \{\texttt{x}\} = 0))\ \Rightarrow \\ \quad (\forall\ \texttt{x.}\ \texttt{x IN sp}\ \wedge\ (\texttt{mu}\ \{\texttt{x}\} \neq 0)\ \Rightarrow \\ \qquad (\texttt{RN\_deriv}\ (\texttt{sp},\texttt{sts},\texttt{mu})\ \texttt{nu x} = \texttt{nu}\ \{\texttt{x}\}/\texttt{mu}\ \{\texttt{x}\})).\end{array}$$

This reduction of the Radon-Nikodým derivative of ν with respect to μ to $\frac{\nu(x)}{\mu(x)}$ is a nice justification of typical denotation as $\frac{d\nu}{d\mu}(x)$.

2.4.5 Product measures

The concept of a *product measure* on a *product space* is needed for some definitions from information theory. This topic really belongs under the domain of measure theory (Section 2.3), but its presentation has been delayed until now because Lebesgue integration is required. A thorough formalisation of theorems relating to product measures is not in the scope of the present work and is left as a useful area for future work. We will now review a general definition of product measure spaces, followed by the HOL formalisation of that definition and theorems needed in subsequent chapters.

Definition 25 (product measure space)**.** *Let $(S_1, \mathbb{S}_1, \mu_1)$ and $(S_2, \mathbb{S}_2, \mu_2)$ be measure spaces. Let $S = S_1 \times S_2$ be the space defined by the cross product of S_1 and S_2 and $\mathbb{S}$ be the class of subsets in the smallest σ-algebra generated by the cross products of sets in $\mathbb{S}_1$ and $\mathbb{S}_2$ respectively. If μ is a measure on $(S, \mathbb{S})$ such that*

$$\forall a_1 \in \mathbb{S}_1\ a_2 \in \mathbb{S}_2.\ \mu(a_1 \times a_2) = \mu_1(a_1)\mu_2(a_2),$$

then $(S, \mathbb{S}, \mu)$ is a **product measure space** *and μ is its* **product measure***.*

A product measure μ for two measure spaces $(S_1, \mathbb{S}_1, \mu_1)$ and $(S_2, \mathbb{S}_2, \mu_2)$ can be constructed as

$$\begin{array}{rcl}\mu(a) & = & \int_{S_1}(\lambda x_1.\ \mu_2((\lambda x_2.\ (x_1, x_2))^{-1}[a]))\ d\mu_1 \\ & = & \int_{S_2}(\lambda x_2.\ \mu_1((\lambda x_1.\ (x_1, x_2))^{-1}[a]))\ d\mu_2,\end{array}$$

where $(\lambda\ x_2.\ (x_1, x_2))^{-1}[a]$ denotes the inverse image of $(\lambda\ x_2.\ (x_1, x_2))$ on a. The existence and uniqueness of the product measure are guaranteed by the Fubini-Lebesgue theorem. A detailed presentation of the Fubini-Lebesgue theorem can be found in standard textbooks covering measure theory and integration [54, 100, 155]. The construction of the product measure described above can be defined in higher-order logic using the formalisation of Lebesgue integration developed earlier.

Formalisation 22 (product measure)**.**

$$\begin{array}{l}\texttt{prod\_measure}\ (\texttt{sp1},\texttt{sts1},\texttt{mu1})\ (\texttt{sp2},\texttt{sts2},\texttt{mu2})\ = \\ \quad (\lambda\ \texttt{a.}\ \texttt{integral}\ (\texttt{sp1},\texttt{sts1},\texttt{mu1}) \\ \qquad\quad (\lambda\ \texttt{x1.}\ \texttt{mu2}\ (\texttt{PREIMAGE}\ (\lambda\ \texttt{x2.}\ (\texttt{x1},\texttt{x2}))\ \texttt{a}))).\end{array}$$

Building on this definition, product measure spaces can be formalised in HOL as follows.

[10] sets which are not measurable with respect to μ or whose measure is zero

Formalisation 23 (product measure space).

```
prod_measure_space (sp1, sts1, mu1) (sp2, sts2, mu2) =
  (sp1 CROSS sp2,
   subsets (sigma (sp1 CROSS sp2) (prod_sets sts1 sts2)),
   prod_measure (sp1, sts1, mu1) (sp2, sts2, mu2)).
```

A general proof of the Fubini-Lebesgue theorem is outside the scope of the present work and will not be undertaken here. However, the equivalence of the product measure to the product of the respective measures has been proved for a class of finite, discrete spaces. The statement of that theorem and that the product measure space forms a valid measure space for that class of spaces can be found in Appendix D as `finite_POW_prod_measure_reduce` and `measure_space_finite_prod_measure_POW1` respectively.

2.5 Probability theory formalised in HOL4

2.5.1 A general formalisation of probability theory

Many of the foundational definitions in probability theory are simply specific instances of corresponding definitions in measure theory. By building on the formalisation of measure theory presented in Section 2.3, it is straightforward to generalise Hurd's formalisation of probability theory in HOL. This section examines such a formalisation. The specialisation of a measure space to define a probability space serves as a natural starting point. Thus, we will begin by reviewing a textbook definition for a *probability space* and then studying the HOL formalisation of that definition.

Definition 26 (probability space). *Let $(S, \mathbb{S}, \mu)$ be a measure space. $(S, \mathbb{S}, \mu)$ is a* **probability space** *iff $\mu(S) = 1$. For a probability space $(S, \mathbb{S}, \mu)$, we refer to S, $\mathbb{S}$, and μ respectively as the* space, events, *and* probability *function of $(S, \mathbb{S}, \mu)$.*

Definition 26 can be formalised as extension of the HOL definition for measure spaces (Formalisation 7).

Formalisation 24 (probability space).

```
probability_space (sp, sts, mu) =
  measurable_space (sp, sts, mu) ∧ (mu (sp) = 1).
```

The restriction on Hurd's formalisation of probability spaces [84] is precisely his restriction on measure spaces carried forward; that restriction was explained in detail in Sections 2.2 and 2.3 and will not be readdressed here.

One of the most important properties that can hold between two events in a probability space is *independence*; much of Hurd's formalisation work focused on independence results for events and functions on probability spaces. We will now look at a HOL definition for the independence of two events; first we review a textbook definition of this property.

Definition 27 (independent events). *Let $(S, \mathbb{S}, \mu)$ be a probability space. Then $s \in \mathbb{S}$ and $s' \in \mathbb{S}$ are* **independent events** *of $(S, \mathbb{S}, \mu)$ iff*

$$\mu(s \cap s') = (\mu(s))(\mu(s')).$$

Note that $s \cap s'$ is guaranteed to be an event of $(S, \mathbb{S}, \mu)$, when s and s' are both events of $(S, \mathbb{S}, \mu)$, because $(S, \mathbb{S})$ defines a σ-algebra and therefore $\mathbb{S}$ is closed under intersection.

Formalisation 25 (independent events).

```
indep (sp, sts, mu) s s' =
s IN sts ∧ s' IN sts ∧ (mu (s INTER s') = mu s * mu s'),
```

assuming `probability_space (sp, sts, mu)` *holds.*

This concludes our look at HOL formalisations generalising Hurd's for probability theory. As with measure theory, the majority of the effort was not devoted to formalising the definitions, but to proving that they satisfy appropriate properties. That effort does not appear here; however, those theorems can be found in HOL notation in Appendix E.

2.5.2 Extensions to the formalisation of probability theory

Hurd's formalisations [84] did not include a number of basic definitions from probability theory that are needed for information theory. For example, he did not formalise a general measure-theoretic definition of expected value, which would have required him to formalise Lebesgue integration. Having examined a generalisations of Hurd's constructions for probability theory, we will now move on to study extensions including definitions from probability theory that are necessary for the development of information theory in Chapter 3.

In the formalisations below, general measure-theoretic definitions have been used in order to maintain flexibility for future applications and to provide a mathematically sound basis. Wherever practical, the general definitions have been proved equivalent to simpler forms for discrete probability spaces. Although the general measure-theoretic definitions are sufficient by themselves, these proofs make the formalisation much easier to use by replacing more complex constructs (e.g. Lebesgue integration) with simpler ones (e.g. summation) where possible.

Discrete probability spaces are sufficient for the intended applications of this work, so focus has been placed on developments for those spaces. Users of the formalisation benefit from the simplifications that have been proved, without a loss of generality and applicability to continuous spaces and other more unusual spaces. Furthermore, probability spaces with a finite space are sufficient for the eventual applications of this work (Chapters 4 and 5). As discussed in Section 2.4, technical details of the HOL4 formalisations require separate reductions for finite and countably-infinite, discrete probability spaces. This can be seen as a useful feature, allowing finite (rather than infinite) summations to be used in definitions for finite spaces. Considering the overall aims of this text, the presentation below will focus on developments for finite, discrete probability spaces in greater detail than for countably-infinite probability spaces; however, both formalisations will be explained thoroughly.

We begin our investigation of the extensions to the constructions of Section 2.5.1 by reviewing a textbook definition for a *random variable* followed by the formalisation of that definition in higher-order logic.

Definition 28 (random variable)**.** *Let $\mathcal{X}$ be a* measurable function *from $(S, \mathbb{S})$ to $(S', \mathbb{S}')$. If $(S, \mathbb{S})$ is equipped with a probability measure μ, i.e. $(S, \mathbb{S}, \mu)$ is a probability space, then $\mathcal{X}$ is called a* **random variable** *on $(S, \mathbb{S}, \mu)$.*

Assuming the appropriate measurability requirements hold, the joint random variable *of two random variables $\mathcal{X}$ and $\mathcal{Y}$ is the function mapping an input to the pair of values taken by $\mathcal{X}$ and $\mathcal{Y}$ for that input i.e.*

$$(\mathcal{X}, \mathcal{Y})(x) = (\mathcal{X}(x), \mathcal{Y}(x)).$$

Typically, $(\mathcal{X}, \mathcal{Y})$ is used to denote the joint random variable of $\mathcal{X}$ and $\mathcal{Y}$, emphasising that it takes a pair of values.

Definition 28 is formalised simply by referring to a measurable function as a random variable in probability-theoretic contexts.

Formalisation 26 (random variable)**.**

```
random_variable X (sp, sts, mu) (sp', sts') =
  prob_space (sp, sts, mu) ∧ X IN measurable (sp, sts) (sp', sts').
```

The class of Borel-measurable real-valued random variables is of particular importance, so a definition for that class of random variable has also been formalised. A specific formalisation for joint random variables is unnecessary; the joint random variable of `X` and `Y` can be define simply as (λ `x`. (`X x`, `Y x`)).

Formalisation 27 (real-valued random variable)**.**

```
real_random_variable X (sp, sts, mu) =
  prob_space (sp, sts, mu) ∧ X IN borel_measurable (sp, sts).
```

All random variables implicitly define a probability measure over the σ-algebra onto which they are measurable. This measure is known as the *distribution* or *law* of the random variable and lies at the core of many of the definitions that follow. We will now review a general definition for the distribution of a random variable and then examine some specific definitions for discrete and continuous random variables. We will then go on to study the HOL construction for those definitions.

Definition 29 (distribution of a random variable)**.** *Let $\mathcal{X}$ be a random variable from a probability space $(S, \mathbb{S}, \mu)$ to a σ algebra $(S', \mathbb{S}')$. The* **distribution** *of $\mathcal{X}$ is defined to be*

$$\mathrm{P}(\mathcal{X} \in A) = \mu(\mathcal{X}^{-1}[A]),$$

for any $A \in \mathbb{S}'$. Note that $(S', \mathbb{S}', (\lambda\ A.\ \mathrm{P}(\mathcal{X} \in A)))$ is a probability space. Thus, $\mathrm{P}(\mathcal{X} \in A)$ intuitively defines the probability of $\mathcal{X}$ taking a value in A. The distribution of $\mathcal{X}$ is often denoted by $\mathrm{P}_{\mathcal{X}}$.

In the case that $\mathcal{X}$ is a discrete *random variable taking values $\{x_0, x_1, \ldots\}$, the term* **probability mass function (pmf)** *is used to refer to $\mathrm{P}(\mathcal{X} = x_i) = \mathrm{P}(\mathcal{X} \in \{x_i\})$. If $\mathcal{X}$ is a* continuous*, real-valued random variable, the term* **cumulative distribution function (cdf)** *is used to refer to $\mathrm{P}(\mathcal{X} \leq x) = \mathrm{P}(\mathcal{X} \in \{y \mid y \leq x\})$. The* **probability density function (pdf)** *of $\mathcal{X}$ is defined to be $\mathrm{P}(a \leq \mathcal{X} \leq b) = \mathrm{P}(\mathcal{X} \in \{y \mid a \leq y \leq b\})$, when it exists. The pdf of a random variable can be defined as the Radon-Nikodým derivative of its distribution with respect to some reference measure.*

Recall that for a probability space $(S, \mathbb{S}, \mu)$, $\forall\ A \in \mathbb{S}.\ \mu(A) = \int_S 1_A\ d\mu$. (HOL Theorem 8). Thus,

$$\mathrm{P}(\mathcal{X} \in A) = \int_S 1_{\mathcal{X}^{-1}[A]}\ d\mu = \int_{S'} 1_A\ d(\lambda\ A.\ \mathrm{P}(\mathcal{X} \in A)).$$

Formalisation 28 (distribution of a random variable)**.**

`distribution (sp,sts,mu) X Y = (λ A. mu (PREIMAGE X A INTER sp))`.

For convenience the distribution of a joint random variable has also been formalised.

Formalisation 29 (distribution of a joint random variable)**.**

```
joint_distribution (sp,sts,mu) X =
  (λ A. mu (PREIMAGE (λ x. (X x, Y x)) A INTER sp))
```

The statement of HOL theorems showing that the distribution of a random variable defines a probability measure on its range space along with the relationships of the distribution to the Lebesgue integral outlined at the end of Definition 29 can be found in Appendix E as `distribution_prob_space`, `distribution_lebesgue_thm1`, and `distribution_lebesgue_thm2` respectively.

Having examined the definition of a random variable and its distribution, we now move on to the notion of the *expected value* or *expectation* of a random variable. It is here that we will see the efforts of Section 2.4 come to fruition. Let us begin by reviewing the definition of the expected value of a random variable.

Definition 30 (expected value)**.** *The* **expected value** *or* **expectation** *of a real-valued random variable $\mathcal{X}$ on probability space $(S, \mathbb{S}, \mu)$ is the Lebesgue intregral of $\mathcal{X}$ on S with respect to μ:*

$$\mathbb{E}(\mathcal{X}) = \int_S \mathcal{X}\ d\mu.$$

If $\mathcal{X}$ is a discrete random variable taking values $\{x_0, x_1, \ldots\}$, then the expected value of $\mathcal{X}$ is typically expressed in the simpler form

$$\mathbb{E}(\mathcal{X}) = \sum_{x_i} (x_i) \mathrm{P}(\mathcal{X} = x_i).$$

Intuitively, $\mathbb{E}(\mathcal{X})$ is the mean *of the values taken by $\mathcal{X}$ on S, weighted by their respective probabilities.*

The HOL formalisation of Definition 30 is simply a matter of referring to the Lebesgue integral as expectation in probability-theoretic contexts.

Formalisation 30 (expected value)**.**

`expectation = integral`.

The practice of proving simplifying equivalences for HOL constructions has been carried forward to the definition of expectation; this includes a proof that the formalisation captures the intuitive notion of expectation for discrete random variables.

HOL Theorem 10 (reduction of expecation of a r.v. on a finite space)**.**

$$\begin{array}{l}\forall\ (\texttt{sp},\texttt{sts},\texttt{mu})\ \texttt{X}.\ \texttt{real\_random\_variable}\ \texttt{X}\ (\texttt{sp},\texttt{sts},\texttt{mu})\ \wedge\ \texttt{FINITE sp}\ \Rightarrow \\ \quad (\texttt{expectation}\ (\texttt{sp},\texttt{sts},\texttt{mu})\ \texttt{X} = \\ \quad\ \texttt{SIGMA}\ (\lambda\ \texttt{r}.\ \texttt{r}\ *\ \texttt{distribution}\ (\texttt{sp},\texttt{sts},\texttt{mu})\ \texttt{X}\ \{\texttt{r}\})\ (\texttt{IMAGE X sp})).\end{array}$$

This theorem captures the simplification of expectation to the intuitive notion of expectation for a discrete *random variable presented at the end of Definition 30. The proof follows from HOL Theorem 6 and basic definitions.*

One of the final extensions to the formalisation of probability theory that we will examine is the *conditional expectation* of a random variable. This construction will not be used for information-theoretic definitions in Chapter 3, so it is not developed in as much detail; however there are many useful applications for conditional expectation, which motivate a brief presentation of its formalisation. We will begin by reviewing a definition of conditional expectation and other notions of conditional probability based thereupon. We will then proceed to study their development in HOL.

Definition 31 (conditional expectation)**.** *Let $\mathcal{X}$ and $\mathcal{Y}$ be real-valued random variables from a probability space $(S, \mathbb{S}, \mu)$ to σ-algebras $(S', \mathbb{S}')$ and $(S'', \mathbb{S}'')$ respectively. Let $A \subseteq \mathbb{S}$ be a class of subsets of S. The* **conditional expectation** *of $\mathcal{X}$ with respect to A, $\mathbb{E}(\mathcal{X}|A)$, is defined to be the borel-measurable random variable f from S to $\mathbb{R}$ such that*

$$\forall a \in A.\ \int_S (\lambda x.\ f(x)1_a(x))\ d\mu = \int_S (\lambda x.\ \mathcal{X}(x)1_a(x))\ d\mu.$$

The **conditional expectation** *of $\mathcal{X}$ given $\mathcal{Y}$ forms a random variable on $(S'', \mathbb{S}'', \mathrm{P}_{\mathcal{Y}})$ and is defined by*

$$\mathbb{E}(\mathcal{X}|\mathcal{Y}) = \mathbb{E}(\mathcal{X}|\mathcal{Y}^{-1}[\mathbb{S}'']).$$

If $\mathcal{X}$ and $\mathcal{Y}$ are discrete random variables taking values $\{x_0, x_1, \ldots\}$ and $\{y_0, y_1, \ldots\}$ respectively, then the conditional expectation of $\mathcal{X}$ and $\mathcal{Y}$ can be expressed in the simpler form

$$\mathbb{E}(\mathcal{X}|\mathcal{Y} = y_j) = \sum_{x_i} (x_i)\mathrm{P}(\mathcal{X} = x_i|\mathcal{Y} = y_j).$$

Finally, the **conditional probability** *of an event $e_1 \in S$ given an event $e_2 \in S$ can be defined most generally in terms of conditional expectation as $\mathrm{P}(\mathrm{e}_1|\mathrm{e}_2) = \mathbb{E}(1_{\mathrm{e}_1}|\mathrm{e}_2)$.*

The definitions for conditional expectation and conditional probability have been formalised in HOL4 as follows:

Formalisation 31 (conditional expectation of a random variable and a set)**.**

$$\begin{array}{l}\texttt{conditional\_expectation}\ (\texttt{sp},\texttt{sts},\texttt{mu})\ \texttt{X}\ \texttt{A}\ = \\ \quad \texttt{@f}.\ \texttt{real\_random\_variable f}\ (\texttt{sp},\texttt{sts},\texttt{mu})\ \wedge \\ \qquad \forall\ \texttt{a}.\ \texttt{a IN A}\ \Rightarrow \\ \qquad (\texttt{integral}\ (\texttt{sp},\texttt{sts},\texttt{mu})\ (\lambda\texttt{x}.\ \texttt{f x}\ *\ \texttt{indicator\_fn a x})\ = \\ \qquad\ \texttt{integral}\ (\texttt{sp},\texttt{sts},\texttt{mu})\ (\lambda\texttt{x}.\ \texttt{X x}\ *\ \texttt{indicator\_fn a x})).\end{array}$$

Recall that `@x. P x` *denotes the Hilbert's choice operator, where* `@x. P x` *can be read informally as* "any `x` such that `P x`"*.*

Formalisation 32 (conditional expectation of two random variables)**.**

$$\begin{array}{l}\texttt{rv\_conditional\_expectation}\ (\texttt{sp},\texttt{sts},\texttt{mu})\ (\texttt{sp}',\texttt{sts}')\ \texttt{X}\ \texttt{Y}\ = \\ \quad \texttt{conditional\_expecation}\ (\texttt{sp},\texttt{sts},\texttt{mu})\ \texttt{X} \\ \qquad (\texttt{IMAGE}\ (\lambda\texttt{a}.\ (\texttt{PREIMAGE Y a})\ \texttt{INTER sp})\ \texttt{sts}').\end{array}$$

Formalisation 33 (conditional probability)**.**

$$\begin{array}{l}\texttt{conditional\_prob}\ (\texttt{sp},\texttt{sts},\texttt{mu})\ \texttt{e1}\ \texttt{e2} = \\ \quad \texttt{conditional\_expecation}\ (\texttt{sp},\texttt{sts},\texttt{mu})\ (\texttt{indicator\_fn e1})\ \texttt{e2}.\end{array}$$

2.6 Summary

This chapter has explained formalisations for measure theory, Lebesgue integration, and probability theory, within the framework of the HOL4 theorem-prover. This work surpasses previous related work in a number of respects. Firstly, the HOL constructions make explicit the space involved in various definitions, allowing general measure-theoretic definitions to be formalised and then proved equivalent to simpler definitions for discrete or continuous spaces. The formalisation presented above also extends previous work by including additional definitions and theorems, such as the development of conditional expectation, product measures, and Radon-Nikodým derivatives. The constructs developed in this chapter serve as essential groundwork for the development of information theory in Chapter 3.

Chapter 3

Information, Entropy, and Uncertainty

"Information is the resolution of uncertainty."
– Claude. E. Shannon, A mathematical theory of communication

3.1 Background

While working at Bell Labs, Claude E. Shannon published a technical report which began the branch of applied mathematics now known as information theory. Shannon's report [138] provided a theory for reasoning about the transmission of signals over a *noisy channel*, i.e. one which is capable of losing or distorting the flow of information. He argued that the signals transmitted over a noisy channel could be viewed as uncertain events in the sense of probability theory. Shannon then established that the *entropy* of the distribution over possible transmissions determines the minimum number of bits needed (on average) in order to transmit on a given channel without loss of information. That development, known as Shannon's source coding theorem, is one of the two most important results from Shannon's 1948 report. The second of these two results is Shannon's noisy channel coding theorem, which is concerned with *rates* of transmission. With that theorem, Shannon defined the (average) maximum rate at which transmissions can be made over a noisy channel without losing information; that maximum rate is referred to as the *channel capacity*.

Considering his place of employment, it seems natural that Shannon should have been interested in developing a theory about transmissions over unreliable media for potential applications to telephony and other communications technologies. Such a theory should prove useful when working to improve the robustness and reliability of communications systems. Not surprisingly, there are a great number of applications of information theory in the domain of coding theory, which is the application area closest to Shannon's original aims. The development of *error-correcting codes* is an excellent example. Error-correcting codes are used to encode a piece of information with sufficient redundancy such that errors introduced by noise during transmission can be detected and corrected with a high probability. There are numerous applications for error-correcting codes ranging from those that many people encounter unknowingly in their daily lives (e.g. the encoding of data on compact discs so they can withstand minor scratches without loss of information) to far more exotic applications (e.g. NASA's deep-space telecommunications). Hamming's book [75] provides an excellent introduction to coding theory and its many applications.

3.2 Motivation

Diverging from Shannon's initial aims, information theory has applications in a wide range of fields including that of computer security analysis. Within that domain, Shannon's source coding theorem is often used contrary to its original use; rather than using his theorem to determine how much redundancy is necessary to overcome some amount of noise (i.e. randomness) in a transmission, it can be used to

determine how much randomness must be present in a transmission in order to ensure that an eavesdropper cannot determine the information transmitted significantly better than random guessing. A closely related application of information theory is its use for analysis of *information flow* in computer software. Examples of this type of utilisation are Denning's seminal work on security models for information flow [42], subsequent work by Gray [72] and McLean [106] on the non-interference of programs, and Millen's work on covert (side-channel) information flow [107]. These uses of information theory make it a valuable tool for reasoning about the potential for leakage of sensitive information in a computer system. It is this application of information theory to information leakage analysis that motivates the HOL formalisation of information theory presented in this chapter; the application of those developments to information leakage analysis warrants the space of its own chapter (Chapter 4). The advantage of formalising information theory in a theorem-prover is that any proofs or analysis performed using such a framework can be guaranteed to be logically and mathematically consistent by the mechanised proof-assistant as discussed in Section 1.2.

3.3 Related work and novel contributions

Below we will examine the formalisation of Shannon's information theory in higher-order logic, developed in the context of the HOL4 theorem-prover. To the best of my knowledge, this is the only formalisation of information theory that has been developed in a theorem-prover. As such is the case, *all* of the formalisation work in this chapter represents a novel contribution. This is not to say that any new theory is developed here; information theory has become mature and well developed in the nearly sixty years since Shannon's initial report. The work presented in this chapter involves the formalisation of standard textbook definitions in a higher-order logic theorem-prover; however, this constitutes a significant contribution in that it provides a rigorous framework in which information theory can be applied and analyzed, whatever the application domain. One such application will be presented in Chapter 4, where the formalisations developed here are used to construct a framework for reasoning about information leakage. There are numerous uses for information theory and as a result there is a large body of work on the application of information theory to various domains. A thorough review of that body of literature is outside of the scope of the present text; however, previous work applying information theory to the domains of anonymous communications and information leakage is discussed in Sections 1.1 and 4.5 respectively.

The formalisation of probability theory studied in Section 2.5 serves as the starting point for the formalisation of information theory developed in this chapter. Familiarity with the content of Chapter 2 is assumed, but an understanding of probability and measure theories is sufficient to understand most of the work below, with the exception of some of the technicalities of the HOL4 constructions. Definitions from information theory will be reviewed briefly, prior to their formalisations, in order facilitate easier reading; more detailed presentations of those definitions can be found in introductory textbooks on information theory [34, 69]. In general, the definitions from information theory presented in this chapter are Gray's definitions [69] restated to respect the notational conventions used here, so credit should be directed where due; however, the formalisations of those definitions are my own contribution. The presentation of the HOL definitions examined below generally aims to be independent of their implementation in the HOL4 theorem-prover; implementation details are included whenever they are of particular interest or are necessary for the reader's understanding. Where such is the case, the definitions are explained sufficiently such that prior knowledge of HOL4 syntax is not needed to understand them.

The next section of this chapter is intended to provide an intuitive introduction of information-theoretic concepts. If you have a strong background in information theory, you may wish to proceed directly to Section 3.5, where the HOL formalisations of information-theoretic definitions are examined. As definitions are encountered, useful properties that have been proved about them in HOL4 will also be presented. Showing that the formalisations maintain appropriate properties reinforces the equivalence between standard textbook definitions and the HOL definitions.

3.4 A gentle introduction to information theory

This section provides a gentle introduction to information theory, emphasising an intuitive understanding of information-theoretic concepts rather than a high degree of precision. The lack of rigour in this section

will be more than made up for by the formal definitions provided in Section 3.5. Where applicable, specific definitions for the case of finite, discrete spaces will be studied in order to aid understanding.

3.4.1 Information

Before addressing more sophisticated concepts, we must begin by examining the most fundamental notion of information theory, the definition of *information* itself. In this text, the term information is used specifically to refer to the information learned through the occurrence of an event sampled from a probability space, which is equipped with a corresponding probability distribution.

To demonstrate with a concrete example, lets look at Shannon's original application domain: telephony. Consider that there is a system for *transmitting* information, such as a string with paper cups at each end. A child named Sebastian at one end of the string is trying to use this *channel* to communicate a secret word to a child named Rupert, who is at the other end of the string. The children have agreed that the word will be spelled out letter by letter, but sometimes their low fidelity telephone makes it difficult for Rupert to determine which letter he has heard.

Being both an eager and a clever child, Rupert would like to learn the secret word as quickly and as accurately as possible. He also knows that certain letters occur more frequently in words than others; thus, he considers it more *probable* the he will hear a letter which occurs in words more often. If the first letter Rupert hears is "E", a letter which occurs frequently, then he has not learned very much about the secret word, because there are many words that start with the letter "E". Conversely, if he were to here he letter "Z", a letter which occurs infrequently, he would have learned a great deal about the secret word, because there are relatively few words beginning with the letter "Z".

In this simplistic example, there is a probability distribution over the letters of the Roman alphabet implied by the frequency of use of individual letters; an event consists of Sebastian saying a particular letter. Examining this example, it is clear that the *information* of each event, sampled from the distribution over the letters, represents *how much* is learned through that observation. Also, note that the information of an event is related to a number of other concepts. For instance, the information of a particular event is related to its *uncertainty*. If an event is certain to occur (i.e. has probability 1), then no information is gained by observing that event. On the contrary, if a very unlikely event is observed, much information is received. Thus, the information of an event is inversely related to its probability.

Now let's take a slightly more formal approach and review a standard definition of the information of an event from a probability space.

Definition 32 (information)**.** *Let $(S, \mathbb{S}, \mu)$ be a probability space and let e be an event of $(S, \mathbb{S}, \mu)$ i.e. $e \in \mathbb{S}$. The* **information** *of event e is defined to be $-\log_b(\mu(e))$.*

Note that the base of the logarithm, b, in the definition of information is left undetermined. The correct choice of base is dependent on the application. In the case of a discrete probability space, where information is to be measured in binary digits (bits), $\log_2$ is the natural choice. Also, observe that information is undefined for events with probability 0. This corresponds to intuition about the definition of information: what can be learned from the observation of an event that will never occur?

Recall from Chapter 2, that real-valued random variables induce a probability distribution over the values they take. Information is itself a real-valued random variable on $(S, \mathbb{S}, \mu)$. Thus, one may equivalently refer to the information of a random variable $\mathcal{X}$ taking value in A, which is by definition $-\log_b(\mathrm{P}(\mathcal{X} \in A))$.

Having reviewed the definition of the information of an event in a probability space, we can now move on to more complex concepts from information theory.

3.4.2 Entropy

The *entropy* of a probability distribution is one of the core notions of information theory. You might be familiar with the use of the term entropy to describe the amount of disorder in a physical system (e.g. the second law of thermodynamics). This usage is analogous to information-theoretic entropy and serves as a useful starting pointing for an intuitive understanding.[1] Entropy captures the average amount of *uncertainty* as to which outcome will occur from a particular probability distribution: the greater the uncertainty the higher the entropy. Conversely, a probability distribution with one certain event (i.e. an

[1] An in depth discussion relating physical entropy and information-theoretic entropy is outside the scope of this text; some general textbooks touch upon the subject and there are also texts devoted primarily to that purpose [63, 73].

event with probability 1) has zero entropy i.e. contains no uncertainty. Alternatively, if all the events are evenly distributed (i.e. equiprobable), then the entropy of the distribution is maximal. Entropy can also be seen as a measure of the *uniformity* of a distribution.

Informally, the entropy of a distribution can be defined as the mean information of its events, weighted by their respective probabilities. In the case of a standard, discrete space with events $\{e_0, \ldots, e_n\}$ and probability measure μ, the entropy of μ is

$$\mathbb{H}_\mu = -\sum_{i=0}^{n} \mu(e_i) \log(\mu(e_i)),$$

i.e. the expected value of the information of the events. In the case of an absolutely continuous distribution, the probability of each singleton event is 0; thus the entropy of such a distribution, even if a limiting definition is used, will always be zero. As a result, entropy can only be seen as a measure of uncertainty in the case of discrete distributions. Despite this shortcoming, other measures based on entropy (such as *mutual information*) are meaningful even in the case of continuous distributions.

As with information, it is common to refer to the entropy of a discrete random variable.[2] The entropy of a random variable $\mathcal{X}$ is simply the entropy of the probability distribution induced by $\mathcal{X}$ i.e. $\mathbb{H}_{\mathrm{P}_\mathcal{X}}$, where $\mathrm{P}_\mathcal{X}$ is used to denote the distribution of $\mathcal{X}$.

3.4.3 Conditional entropy

Conditional entropy is a useful stepping stone to defining mutual information, a concept which will feature heavily in the remainder of the chapter.

Informally, the conditional entropy of a random variable $\mathcal{X}$, with respect to another random variable $\mathcal{Y}$, measures the mean uncertainty of the outcome of $\mathcal{X}$, given that the outcome of $\mathcal{Y}$ is known. To some extent, conditional entropy is a measure of the *independence* of $\mathcal{X}$ and $\mathcal{Y}$. For example, the conditional entropy of random variable $\mathcal{X}$ conditioned on itself is zero, because $\mathcal{X}$ is completely determined by itself; thus there is no uncertainty about the outcome of $\mathcal{X}$ given knowledge of the outcome of $\mathcal{X}$. On the other hand, if $\mathcal{X}$ and $\mathcal{Y}$ are independent, then knowledge of the outcome of $\mathcal{Y}$ does not change one's uncertainty about the outcome of $\mathcal{X}$; thus, the conditional entropy of $\mathcal{X}$ given $\mathcal{Y}$ is simply the entropy of $\mathcal{X}$.

In the case that $\mathcal{X}$ and $\mathcal{Y}$ are random variables taking values $\{x_0, \ldots, x_m\}$ and $\{y_0, \ldots, y_n\}$ respectively, then the conditional entropy of $\mathcal{X}$ with respect to $\mathcal{Y}$ can be defined by

$$\mathbb{H}(\mathcal{X}|\mathcal{Y}) = -\sum_{i=0}^{m}\sum_{j=0}^{n} \mathrm{P}(\mathcal{X} = x_i; \mathcal{Y} = y_j) \log\Big(\frac{\mathrm{P}(\mathcal{X} = x_i; \mathcal{Y} = y_j)}{\mathrm{P}(\mathcal{Y} = y_j)}\Big),$$

i.e. as the expected value, with respect to the joint probability mass function (pmf) of $\mathcal{X}$ and $\mathcal{Y}$, of the information of the conditional pmf of $\mathcal{X}$ and $\mathcal{Y}$.

3.4.4 Mutual information

The mutual information of two random variables $\mathcal{X}$ and $\mathcal{Y}$ measures the *correlation* of $\mathcal{X}$ and $\mathcal{Y}$: the greater the correlation, the greater the mutual information. Informally, the mutual information of $\mathcal{X}$ and $\mathcal{Y}$ can be seen as measuring the amount of information that can be learned about $\mathcal{X}$ by observing $\mathcal{Y}$ and vice versa. Mutual information can be defined using entropy and conditional entropy as

$$\mathbb{I}(\mathcal{X}; \mathcal{Y}) = \mathbb{H}(\mathcal{X}) - \mathbb{H}(\mathcal{X}|\mathcal{Y}) = \mathbb{H}(\mathcal{Y}) - \mathbb{H}(\mathcal{Y}|\mathcal{X}) = \mathbb{I}(\mathcal{Y}; \mathcal{X}).$$

This formulation aligns with the initial intuitive definition of mutual information provided above; it could be read as, "the correlation of $\mathcal{X}$ and $\mathcal{Y}$ is the uncertainty about the outcome of $\mathcal{X}$ minus the uncertainty about the outcome of $\mathcal{X}$ given knowledge of $\mathcal{Y}$ i.e. how much observation of $\mathcal{Y}$ reduces the uncertainty of the outcome of $\mathcal{X}$. Equivalently, mutual information can be defined directly as

$$\mathbb{I}(\mathcal{X}; \mathcal{Y}) = \sum_{i=0}^{m}\sum_{j=0}^{n} \mathrm{P}(\mathcal{X} = x_i, \mathcal{Y} = y_j) \log\Big(\frac{\mathrm{P}(\mathcal{X} = x_i, \mathcal{Y} = y_j)}{\mathrm{P}(\mathcal{X} = x_i)\mathrm{P}(\mathcal{Y} = y_j)}\Big),$$

[2] More precisely, it is the entropy of the probability space implicitly defined by a random variable and its distribution.

when $\mathcal{X}$ and $\mathcal{Y}$ are finite, discrete random variables taking values $\{x_0, \ldots, x_m\}$ and $\{y_0, \ldots, y_n\}$ respectively.

Notice that mutual information is symmetric. Also, take note that the entropy of a random variable $\mathcal{X}$ can be defined as the self mutual information of $\mathcal{X}$ i.e. $\mathbb{H}(\mathcal{X}) = \mathbb{I}(\mathcal{X}; \mathcal{X})$; in the first formulation, this follows directly from the fact that the conditional entropy of a random variable with respect to itself is always zero (or equivalently in the second formulation, from the fact that the joint distribution of a random variable with itself is simply the distribution of that random variable). The fact that entropy can be defined using mutual information will be used in Section 3.5 for the formalisation of entropy in higher-order logic.

3.4.5 Conditional mutual information

We conclude our informal investigation of information theory by examining the concept of conditional mutual information. The conditional mutual information of random variables $\mathcal{X}$ and $\mathcal{Y}$, with respect to random variable $\mathcal{Z}$, measures the correlation of $\mathcal{X}$ and $\mathcal{Y}$ given knowledge of the outcome of $\mathcal{Z}$. Conditional mutual information can be defined using conditional entropy as

$$\mathbb{I}(\mathcal{X}; \mathcal{Y} | \mathcal{Z}) = \mathbb{H}(\mathcal{X} | \mathcal{Z}) - \mathbb{H}(\mathcal{X} | (\mathcal{Y}, \mathcal{Z}))$$

or equivalently using mutual information as

$$\mathbb{I}(\mathcal{X}; \mathcal{Y} | \mathcal{Z}) = \mathbb{I}(\mathcal{X}; (\mathcal{Y}, \mathcal{Z})) - \mathbb{I}(\mathcal{X}; \mathcal{Z}),$$

where $(\mathcal{Y}, \mathcal{Z})$ denotes the joint random-variable defined by $\mathcal{Y}$ and $\mathcal{Z}$. Informally, the conditional mutual information of $\mathcal{X}$ and $\mathcal{Y}$ with respect to $\mathcal{Z}$ can be seen as the amount that knowledge of the outcome of $\mathcal{Y}$ reduces one's uncertainty about the outcome of $\mathcal{X}$, given that the outcome of $\mathcal{Z}$ is already known.

Having completed our brief introduction to information theory, we will now take a more rigourous approach and begin to study the formalisation of information theory in higher-order logic.

3.5 Information theory formalised in HOL4

In this section, we will examine the formalisation of information theory in HOL4, which builds on the formalisations for probability theory developed in Chapter 2. Before doing so, it is worth revisiting some of the underlying decisions made for the formalisations of Chapter 2 that are pertinent to the work appearing below.

Firstly, recall that the real numbers were used in the developments of Chapter 2, rather than the extended reals (i.e. the real numbers extended to include elements for positive and negative infinity). The repercussions of that decision are that any definition involving an integration (or equivalently an infinite summation) is undefined in the case that the integral (or sum) diverges to infinity. One advantage of this design decision is that arithmetic is greatly simplified, as one need not be concerned with difficulties such as the undefinedness of the sum of positive and negative infinity.

Additionally, a powerful approach of formalising general *measure-theoretic* definitions of probability-theoretic notions was used in Chapter 2. These general definitions can then be proved equivalent to simpler forms for the continuous and discrete case, thereby providing a mathematically rigorous connection between the simpler definitions and allowing them to be used in place of the general definition. This practice has been carried forward to the formalisations for information theory and the same benefits are received.

In *The Mathematical Theory of Communication*, Shannon [139] wrote

> *"We will not attempt, in the continuous case, to obtain our results with the greatest generality, or with the extreme rigour of pure mathematics, since this would involve a great deal of abstract measure theory and would obscure the main thread of the analysis."*

Although Shannon's initial report on information theory gave birth to the subject in a remarkably mature form, the lack of definitions generalising the cases of discrete and absolutely continuous random variables could be seen as a major shortcoming of his work [138, 139]. Fortunately, in the nearly sixty years since Shannon's report was first published, a great deal of work has been done to generalise Shannon's

information theoretic concepts and the ergodic theories that underpin his coding theories. The research of Kolmogorov [90] and of Pinkser [122] are two such examples. A thorough review of the development of information theory is outside of the scope of the present text; however, Gray [69] provides a nice overview of major developments in information theory in the prologue of his book on the subject.

In the HOL development of information theory presented below, I have not aimed for the greatest possible generality because the additional complications are not justified by the applications in subsequent chapters. However, the formalisations are sufficiently general to account for standard, discrete and absolutely continuous random variables, providing a solid grounding for information theory in the formalisation of probability theory developed above and a strong connection between definitions for the discrete and continuous cases. Thus, the definitions below are more general than in Shannon's original presentation, as can be deduced from his words in the preceding paragraph. As in Chapter 2, the general definitions have been proved equivalent to simpler forms for finite random variables, allowing for easier use of the formalisations; similar reductions could be proved for countably-infinite, discrete random variables and absolutely-continuous random variables.

As mentioned above, finite, discrete random variables are sufficient for the applications presented in Chapters 4 and 5. Thus, emphasis has been placed on simplifying proofs for that class of random variables. Although the focus of the *presentation* is on finite, discrete random variables, this is without a loss of generality for the definitions and their applicability to countably-infinite, discrete random variables and absolutely-continuous random variables.

Having dispensed with preliminary discussion and equipped with the HOL definitions for probability theory from Chapter 2, we can now begin to investigate the formalisation of information theory in HOL4.

3.5.1 Kullback-Leibler divergence

As hinted at in Section 3.4, the formalisation of information-theoretic concepts in this chapter will begin with the definition of mutual information; entropy and conditional mutual information can then be defined. Mutual information will be defined in terms of Kullback-Leibler divergence, as is done in standard textbooks on information theory [34, 69]. Thus, we will begin our study of information theory formalised in higher-order logic by reviewing a standard definition of Kullback-Leibler divergence.

Informally, the Kullback-Leibler divergence of two probability distributions can be seen as a measure of the *difference* or *distance* between the two distributions. Taking a slightly more rigourous view, Kullback-Leibler divergence is not actually a distance or a metric, because it neither is symmetric nor satisfies the triangle inequality. Without further delay, we will now review a formal definition of Kullback-Leibler divergence.

Definition 33 (Kullback-Leibler divergence)**.** *Let* $(S, \mathbb{S}, \mu)$ *and* $(S, \mathbb{S}, \nu)$ *be measure spaces. Define the* **Kullback-Leibler divergence** *of* μ *from* ν *with respect to* $(S, \mathbb{S})$ *as*

$$\mathbb{D}_b(\mu\|\nu) = \int_S \log_b\left(\frac{d\nu}{d\mu}\right) d\mu,$$

where the integral above is the Lebesgue integral with respect to $(S, \mathbb{S}, \mu)$ *and* $\frac{d\nu}{d\mu}$ *denotes the Radon-Nikodým derivative of* ν *with respect to* $(S, \mathbb{S}, \mu)$*.*

Recall from above that Kullback-Leibler divergence is not symmetric, so it is not necessarily the case the $\mathbb{D}_b(\mu\|\nu) = \mathbb{D}_b(\nu\|\mu)$. Also note that the base of the logarithm, b, in the definition above is left undetermined. The correct choice of base is dependent on the application. In the case of a discrete probability space, where information is to be measure in binary digits (bits), $\log_2$ is the natural choice.

When developing the HOL definition for Kullback-Leibler divergence, I decided to formalise Definition 33 directly, rather than defining a more general measure of divergence[3] and then defining the Kullback-Leibler divergence as a limit of the more general measure. A more general approach is not need for any of the applications in this text, so the additional complications of such could not be justified. Building on the HOL definitions presented in Chapter 2, it is a relatively straightforward matter to translate Definition 33 into higher-order logic.

[3] For example, the Rényi measure [127] of divergence generalises Kullback-Leibler divergence.

Formalisation 34 (Kullback-Leibler divergence)**.**

```
KL_divergence b (sp, sts) mu nu =
  integral (sp, sts, mu) (λx. logr b ((RN_deriv (sp, sts, nu) mu) x)),
```

Note that the HOL definition of Kullback-Leibler divergence has retained the generality of an arbitrary base for the logarithm as in Definition 33 above. In the applications of this formalisation in Chapters 4 and 5, the base 2 will always be used. As a result, a body of theorems relating to properties of $\log_2$ have been proved in HOL4 in order to support this formalisation. One aspect of that work which was surprisingly challenging was the proof of the (positive) concavity of the natural logarithm function, which involved use of Harrison's formalisation of gauge integration. That proof, though necessary to establish bounds on entropy via Jensen's inequality, is tangential to the work presented here and so is omitted for the sake of brevity.

We can now move on to study the HOL definition of mutual information, the keystone to the formalisation of information theory.

3.5.2 Mutual information

By defining mutual information in terms of Kullback-Leibler divergence, the definition can be made sufficiently general to account for the cases of both discrete and absolutely continuous random variables. Recall from Section 3.4.4 that the mutual information of two random variables $\mathcal{X}$ and $\mathcal{Y}$ measures the correlation of $\mathcal{X}$ and $\mathcal{Y}$. Informally, this is a measure of how much can be learned about $\mathcal{X}$ by observing $\mathcal{Y}$ (and vice versa, because mutual information is symmetric).

Definition 34 (Mutual information)**.** *Let $(S, \mathbb{S}, \mu)$ be a probability space and $\mathcal{X}$ and $\mathcal{Y}$ be random variables from $(S, \mathbb{S})$ to $(S', \mathbb{S}')$ and $(S'', \mathbb{S}'')$ respectively. Let $\mathrm{P}_{\mathcal{X}}$, $\mathrm{P}_{\mathcal{Y}}$, $\mathrm{P}_{\mathcal{X}\mathcal{Y}}$ denote the distribution of $\mathcal{X}$, the distribution of $\mathcal{Y}$, and their joint distribution. Let $S_{\mathcal{X}} \times S_{\mathcal{Y}}$ denote the product space of $(S', \mathbb{S}', \mathrm{P}_{\mathcal{X}})$ and $(S'', \mathbb{S}'', \mathrm{P}_{\mathcal{Y}})$ and $\mathrm{P}_{\mathcal{X}\times\mathcal{Y}}$ denote the product measure of $S_{\mathcal{X}} \times S_{\mathcal{Y}}$. Define the* **mutual information** *of $\mathcal{X}$ and $\mathcal{Y}$ using Kullback-Leibler divergence with respect to $S_{\mathcal{X}} \times S_{\mathcal{Y}}$ as*

$$\mathbb{I}_b(\mathcal{X}; \mathcal{Y}) = \mathbb{D}_b(\mathrm{P}_{\mathcal{X}\mathcal{Y}} \| \mathrm{P}_{\mathcal{X}\times\mathcal{Y}}).$$

In the case that $\mathcal{X}$ and $\mathcal{Y}$ are *discrete* random variables taking values $\{x_0, \ldots, x_m\}$ and $\{y_0, \ldots, y_n\}$ respectively, the definition of mutual information above can be reduced to

$$\mathbb{I}_b(\mathcal{X}; \mathcal{Y}) = \sum_{i=0}^{m} \sum_{j=0}^{n} \mathrm{P}(\mathcal{X} = x_i, \mathcal{Y} = y_j) \log_b \Big(\frac{\mathrm{P}(\mathcal{X} = x_i, \mathcal{Y} = y_j)}{\mathrm{P}(\mathcal{X} = x_i)\mathrm{P}(\mathcal{Y} = y_j)} \Big),$$

as in Section 3.4.4.

Having developed HOL definitions for product measures in Chapter 2 and Kullback-Leibler divergence in Section 3.5.1, the formalisation of Definition 34 in HOL becomes a straightforward translation.

Formalisation 35 (Mutual information)**.**

```
mutual_information b (sp, sts, mu) (sp′, sts′) (sp″, sts″) X Y =
  let (psp, psts, pmu) =
      prod_measure_space (sp′, sts′, distribution p X)
                         (sp″, sts″, distribution p Y)
  in
      KL_divergence b (psp, psts) (joint_distribution p X Y) pmu.
```

Now that we have a general formalisation of mutual information, it would be useful to be able to simplify this general definition to the finite, discrete form specified after Definition 34; such an equivalence has been proved in HOL4.

HOL Theorem 11 (Reduction of mutual info. for finite, discrete spaces)**.**

```
∀ b sp mu X Y. FINITE sp ∧
  random_variable X (sp, POW sp, mu) (IMAGE X sp, POW (IMAGE X sp)) ∧
  random_variable Y (sp, POW sp, mu) (IMAGE Y sp, POW (IMAGE Y sp)) ⇒
  (mutual_information b (sp, POW sp, mu)
                        (IMAGE X sp, POW (IMAGE X sp))
                        (IMAGE Y sp, POW (IMAGE Y sp))
                        X Y =
   SIGMA (λ(x, y). joint_distribution (sp, POW sp, mu) X Y {(x, y)} *
           logr b (joint_distribution (sp, POW sp, mu) X Y {(x, y)} /
                   (pmf (sp, POW sp, mu) X {x} *
                    pmf (sp, POW sp, mu) Y {y})))
        ((IMAGE X sp) CROSS (IMAGE Y sp)).
```

The proof of the theorem above follows from the reductions of the Lebesgue integral from Section 2.4, properties of the product measure, properties of finite summations, and the basic definitions. In particular, some of the most important lemmas for the proof can be found in the appendices as the HOL theorems `measure_space_finite_prod_measure_POW2` *in Appendix D and* `finite_marginal_product_space_POW` *in Appendix E.*

Note that the symmetry of the formalisation of mutual information follows directly from the theorem above, the commutativity of set intersection, and commutativity of multiplication.

While the proof of the theorem above was routine, it did require a considerable effort. Fortunately, since all other information-theoretic definitions have been formalised in terms of mutual information, the effort necessary for additional equivalence proofs was greatly decreased. This is can be seen in the formalisation of entropy that we will now examine.

3.5.3 Entropy

This subsection presents a HOL definition for the *entropy* of a probability distribution, one of the core notions of information theory. Recall from Section 3.4.2 that the entropy of a random variable is a measure of the *uncertainty* of its outcome, or equivalently, of the *uniformity* of the distribution over its outcomes. Informally, the entropy of a distribution is the mean information of its events, weighted by their respective probabilities. Also remember that the entropy of a random variable can be defined as the self mutual information of the random variable, as was explained in Section 3.4.2. We will now review a more rigourous definition of the entropy of a random variable.

Definition 35 (entropy)**.** *Let* $(S, \mathbb{S}, \mu)$ *be a probability space and* $\mathcal{X}$ *be a random variable from* $(S, \mathbb{S})$ *to* $(S', \mathbb{S}')$*. Define the* **entropy** *of* $\mathcal{X}$ *with respect to a base* b *to be*

$$\mathbb{H}_b(\mathcal{X}) = \mathbb{I}_b(\mathcal{X}; \mathcal{X}).$$

In the case that $\mathcal{X}$ is a *discrete* random variable taking values from $\{x_0, \ldots, x_n\}$, the definition of entropy given above can be reduced to

$$\mathbb{H}_b(\mathcal{X}) = \sum_{i=0}^{n} \mathrm{P}(\mathcal{X} = x_i) \log_b(\mathrm{P}(\mathcal{X} = x_i)),$$

as in Section 3.4.2.

It is worthwhile examining some properties of the bounds on the entropy of a finite, discrete random variable, as this provides some intuition as to the meaning of entropy. Consider a random variable $\mathcal{X}$ which takes values $\{x_1, \ldots, x_n\}$, each with non-zero probability i.e. $\mathrm{P}(\mathcal{X} = x_i) > 0$ for any i such that $1 \leq i \leq n$. The entropy of $\mathcal{X}$ with respect to a base b lies in the range

$$0 \leq \mathbb{H}_b(\mathcal{X}) \leq \log_b n.$$

These are in fact strict bounds, as the entropy of $\mathcal{X}$ can take both the value 0 and the value $\log_b n$ depending on the distribution induced by $\mathcal{X}$. The entropy of $\mathcal{X}$ is zero iff there is an x_i that occurs with

probability 1 i.e. is certain. Equivalently, in mathematical notation,

$$(\mathbb{H}_b(\mathcal{X}) = 0) = (\exists i.\ 1 \leq i \leq n \wedge \mathrm{P}(\mathcal{X} = x_i) = 1).$$

On the other hand, the entropy of $\mathcal{X}$ is $\log_b n$ iff the x_i's are uniformly distributed i.e. occur with equal probability. Equivalently, in mathematical notation,

$$(\mathbb{H}_b(\mathcal{X}) = \log_b n) = (\forall i \in \{1, \ldots, n\}\ j \in \{1, \ldots, n\}.\ \mathrm{P}(\mathcal{X} = x_i) = \mathrm{P}(\mathcal{X} = x_j)).$$

Without further delay, let's move on to the definition of entropy in higher-order logic. This formulation constitutes a simple translation of Definition 35 using Formalisation 35.

Formalisation 36 (entropy)**.**

```
entropy b (sp, sts, mu) (sp', sts') X =
  mutual_information b (sp, sts, mu) (sp', sts') (sp', sts') X X.
```

As with the other HOL definitions developed above, it is useful to prove an equivalence reducing the definition for entropy to the simpler form given below Definition 35; such a reduction has been proved in HOL4 and the statement of that theorem follows.

HOL Theorem 12 (Reduction of entropy for finite, standard spaces)**.**

```
∀ b sp mu X. FINITE sp ∧
  random_variable X (sp, POW sp, mu) (IMAGE X sp, POW (IMAGE X sp)) ⇒
  (entropy b (sp, POW sp, mu) (IMAGE X sp, POW (IMAGE X sp)) X =
   −SIGMA (λx. distribution (sp, POW sp, mu) X {x} *
               logr b (distribution (sp, POW sp, mu) X {x})
          (IMAGE X sp)).
```

This captures the intuitive definition of entropy as mean information.

When formalising textbook definitions in a theorem-prover, it is common practice to prove some well-known properties for each definition as a sanity check for the formalisation; if the formalisation satisfies a number of properties satisfied by the definition, some reassurance is provided that it properly captures the definition. To this end, the bounds on entropy for finite, discrete spaces mentioned above have been proved in HOL, along with proofs showing that the general definitions reduce to the simpler definitions under appropriate conditions. The statement of those theorems in HOL notation can be found in Appendix F as `finite_entropy_certainty_eq_0`, `finite_entropy_le_card`, and `finite_entropy_uniform_max`; they are omitted here for the sake of space. The most interesting of those proofs was the proof of the upper bound on entropy, which required a proof of Jensen's inequality[4] and the (positive) concavity of the natural logarithm. An outline of the HOL proof of this upper bound would read much the same as in standard textbooks on information theory [34].

3.5.4 Conditional mutual information

Conditional mutual information is the most complicated of the information-theoretic notions addressed here, but it will also be the most important for the developments in Chapter 4. The conditional mutual information of random variables $\mathcal{X}$ and $\mathcal{Y}$, with respect to random variable $\mathcal{Z}$, can be seen as a measure of the correlation of $\mathcal{X}$ and $\mathcal{Y}$, given knowledge of the outcome of $\mathcal{Z}$. Recall the definition for conditional mutual information given in Section 3.4.5 as a difference of mutual information:

$$\mathbb{I}(\mathcal{X}; \mathcal{Y} | \mathcal{Z}) = \mathbb{I}(\mathcal{X}; (\mathcal{Y}, \mathcal{Z})) - \mathbb{I}(\mathcal{X}; \mathcal{Z}),$$

[4] Jensen's inequality guarantees that, for a probability space $(S, \mathbb{S}, \mu)$, a function g that is integrable with respect to $(S, \mathbb{S}, \mu)$, and a convex measurable function on the reals ϕ

$$\phi\Big(\int_S g\ d\mu\Big) \leq \int_S \phi \circ g\ d\mu.$$

The inequality is reversed in the case that ϕ is concave. More details on Jensen's inequality can be found in standard texts on measure theory and probability [54].

where $(\mathcal{Y}, \mathcal{Z})$ denotes the joint random variable defined by $\mathcal{Y}$ and $\mathcal{Z}$. This definition of conditional mutual information, which is based on Kolmogorov's formula[5], has been used as the basis for the HOL formalisation presented below.

Notice that the definition of conditional mutual information above is only well-defined when it is not indeterminate, e.g. $\infty - \infty$. In the framework of the HOL definitions developed above, this means that conditional mutual information will be well-defined only when both of the mutual information values in the definition are well-defined (i.e. any integrals involved converge and both values are finite). As was explained for the formalisation of the Lebesgue integral in Chapter 2, this is a slight restriction of the mathematical definition in which only one side of the difference needs to be finite. This is a direct result of using the real numbers (without the inclusion of an infinity element) for the formalisation and is beneficial in that indeterminacy is not a concern.

Up to this point, the definitions used for the formalisations in this chapter have closely followed those from Gray's book [69]; the definition of conditional mutual information above diverges from Gray. Gray presents that definition as an alternative but chooses instead to define conditional mutual information directly using Kullback-Leibler divergence. The definition based on Kolmogorov's formula is less general than Gray's precisely because the result might be indeterminate, as discussed above; however, this generality is not required for my applications. Gray's definition [69] can be stated roughly as

$$\mathbb{I}(\mathcal{X}; \mathcal{Y}|\mathcal{Z}) = \mathbb{D}(\mathrm{P}_{\mathcal{X}\mathcal{Y}\mathcal{Z}} \| \mathrm{P}_{\mathcal{X}\times\mathcal{Y}|\mathcal{Z}}),$$

where $\mathrm{P}_{\mathcal{X}\mathcal{Y}\mathcal{Z}}$ and $\mathrm{P}_{\mathcal{X}\times\mathcal{Y}|\mathcal{Z}}$ denote the marginal distribution and the conditional product measure of $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z}$ respectively. The conditional product measure used in Gray's definition does not seem be a feature in standard texts on measure, probability, or information theories and so its precise meaning is not obvious.[6] Thus, it seems safer to use the definition of conditional mutual information based on Kolmogorov's formula for the formalisation below. Furthermore, defining conditional mutual information using mutual information allows theorems about mutual information to be reused when proving properties of conditional mutual information.

With out further delay we will now examine a more rigourous version of the definition of conditional mutual information chosen above.

Definition 36 (conditional mutual information)**.** *Let $(S, \mathbb{S}, \mu)$ be a probability space. Let $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z}$ be random variables from $(S, \mathbb{S})$ to $(S', \mathbb{S}')$, $(S'', \mathbb{S}'')$, and $(S''', \mathbb{S}''')$ respectively. Define the* **conditional mutual information** *of $\mathcal{X}$ and $\mathcal{Y}$, with respect to $\mathcal{Z}$ and some base b, as*

$$\mathbb{I}_b(\mathcal{X}; \mathcal{Y}|\mathcal{Z}) = \mathbb{I}_b(\mathcal{X}; (\mathcal{Y}, \mathcal{Z})) - \mathbb{I}_b(\mathcal{X}; \mathcal{Z}),$$

where $(\mathcal{Y}, \mathcal{Z})$ denotes the joint random variable defined by $\mathcal{X}$ and $\mathcal{Y}$.

In the case that $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z}$ are *discrete* random variables taking values $\{x_0, \ldots, x_m\}$, $\{y_0, \ldots, y_n\}$, and $\{z_0, \ldots, z_r\}$ respectively, the definition of conditional mutual information above can be reduced to

$$\begin{aligned}
\mathbb{I}_b(\mathcal{X}; \mathcal{Y}|\mathcal{Z}) &= -\sum_{i=0}^{m}\sum_{k=0}^{r} \mathrm{P}_{\mathcal{X}\mathcal{Z}}(x_i, z_k) \log_b \Big(\frac{\mathrm{P}_{\mathcal{X}\mathcal{Z}}(x_i, z_k)}{\mathrm{P}_{\mathcal{Z}}(z_k)}\Big) \\
&- -\sum_{i=0}^{m}\sum_{j=0}^{n}\sum_{k=0}^{r} \mathrm{P}_{\mathcal{X}\mathcal{Y}\mathcal{Z}}(x_i, y_j, z_k) \log_b \Big(\frac{\mathrm{P}_{\mathcal{X}\mathcal{Y}\mathcal{Z}}(x_i, y_j, z_k)}{\mathrm{P}_{\mathcal{Y}\mathcal{Z}}(y_j, z_k)}\Big) \\
&= \mathbb{H}_b(\mathcal{X}|\mathcal{Z}) - \mathbb{H}_b(\mathcal{X}|\mathcal{Y}, \mathcal{Z}),
\end{aligned}$$

as in Section 3.4.5.

As with entropy, conditional mutual information can be defined in HOL relatively easily, since mutual information has already been formalised.

[5] Gray [69] states Kolmogorov's formula as

$$\mathbb{I}(\mathcal{X}; \mathcal{Y}|\mathcal{Z}) + \mathbb{I}(\mathcal{Y}; \mathcal{Z}) = \mathbb{I}(\mathcal{Y}; (\mathcal{X}, \mathcal{Z})).$$

[6] A recent paper by Swart [146] does attempt to precisely define this concept of conditional product measures; however, it is generally good practice to use well established definitions when constructing formalisations.

Formalisation 37 (Conditional mutual information)**.**

```
conditional_mutual_information b (sp, sts, mu) (sp′, sts′)
                                 (sp″, sts″) (sp‴, sts‴)
                                 X Y Z =
  let (psp, psts, pmu) =
      prod_measure_space (sp″, sts″, distribution p Y)
                         (sp‴, sts‴, distribution p Z)
  in
      mutual_information b (sp, sts, mu)
        (sp′, sts′) (psp, psts) X (λx. (Y x, Z x)) −
      mutual_information b (sp, sts, mu) (sp′, sts′) (sp‴, sts‴) X Z.
```

Just like the other definitions above, the HOL formalisation of conditional mutual information has been proven equivalent to the simplified form for finite, discrete spaces given at the end of its mathematical definition. The statement of this theorem in higher-order logic follows.

HOL Theorem 13 (Reduction of conditional mutual information)**.**

```
∀ b sp mu X Y Z. FINITE sp ∧
  random_variable X (sp, POW sp, mu) (IMAGE X sp, POW (IMAGE X sp)) ∧
  random_variable Y (sp, POW sp, mu) (IMAGE Y sp, POW (IMAGE Y sp)) ∧
  random_variable Z (sp, POW sp, mu) (IMAGE Z sp, POW (IMAGE Z1 sp)) ⇒
  (conditional_mutual_information b (sp, POW sp, mu)
                                    (IMAGE X sp, POW (IMAGE X sp))
                                    (IMAGE Y sp, POW (IMAGE Y sp))
                                    (IMAGE Z sp, POW (IMAGE Z sp))
                                    X Y Z =
   −SIGMA (λ(x, z). joint_distribution (sp, POW sp, mu) X Z {(x, z)} ∗
                  logr b (joint_distribution (sp, POW sp, mu)
                                             X Z {(x, z)} /
                          distribution (sp, POW sp, mu) Z {z}))
          ((IMAGE X sp) CROSS (IMAGE Z sp)) −
   −SIGMA (λ(x, y, z). joint_distribution (sp, POW sp, mu)
                         X (λx. (Y x, Z x)) {(x, y, z)} ∗
                    logr b (joint_distribution (sp, POW sp, mu)
                                X (λx. (Y x, Z x)) {(x, y, z)} /
                            distribution (sp, POW sp, mu)
                              (λx. (Y x, Z x)) {(y, z)}))
          ((IMAGE X sp) CROSS (IMAGE (λx. (Y x, Z x)) sp))).
```

The proof of HOL Theorem 13 was surprisingly lengthy and tedious, mostly due to the nested summations involved and obligations to prove properties of various distributions and summations thereof; the proof was greatly aided by the use of HOL Theorem 11.

3.6 Summary

This chapter explained the formalisation of information theory in HOL. The definitions used above were sufficiently general to characterise both the discrete and absolutely continuous case. Furthermore, each of these general definitions has been proved to reduce to a simpler definition for the case of finite, discrete spaces; similar proofs for the continuous and other cases could be proved in the future. This formalisation of information theory in a mechanised theorem-prover allows mathematically rigorous, machine-verified analysis to be performed using information theory. The developments in this chapter serve as necessary groundwork for the framework presented in Chapter 4.

Chapter 4

Programs, Probabilism, and Information Leakage

"Security isn't a dirty word, Blackadder...
Leak is a positively disgusting word."
– Stephen Fry as General Melchett, Blackadder Goes Forth

The primary contribution of this chapter is to demonstrate how the formalisation of information theory developed in Chapter 3 can be used to reason about the quantity of sensitive information that is leaked by a program. Two of the core requirements for that proof framework are that it quantifies the amount of information leaked and that it can be applied to programs exhibiting probabilistic behaviour; both of those requirements will be motivated below. Throughout this chapter, examples will be used to motivate the formalised definition of information leakage. Those examples also exhibit the flexibility of the framework and the range of scenarios in which it can be applied. In addition to examining formalised definitions for information leakage, we will also look at some concrete examples of those HOL formalisations in use and tool support that simplifies the proofs. The examples in this chapter will be followed by their formalised versions; the simple correspondence between these examples and formalisations, as well as their ease of proof using the tool support developed, will demonstrate that formal analysis need not be inherently difficult or arcane.

4.1 Motivation

Controlling and tracking the flow of information is a very old concern, predating the digital era. The techniques used to address these issues have changed greatly as technologies have developed, but the underlying goals remain the same. Since many of these developments have been motivated by military applications, let's consider the evolution of information control within military contexts to gain insight into the primary concerns and techniques relating to information flow.

Imagine the dilemma of an emperor who must communicate orders over vast distances while maintaining control over who has access to what information. He can easily dispatch a messenger to the edge of his lands, but there is always a risk that the messenger will be seized and confidential messages will fall into the hands of an enemy. The information could be protected using physical strength by sending a convoy to defend the messenger, but that approach is costly and a strong enemy might still overpower the convoy. Alternatively, confidential information could be sent in a form that can be understood only by the intended recipients. Then, the enemy gains nothing by capturing a messenger; even if he is tortured, he cannot reveal anything about the message he carries. Such *cryptographic* approaches to controlling information flow have been used for centuries[1] and remain one of the primary techniques today.

[1] In his biography of Julius Caesar (100 BC – 44 BC), Suetonius explained a simple cipher that the emperor used to communicate military plans to his generals [92]; the name "Caesar Cipher" remains associated with this type of cipher. The approach Caesar used was to shift each letter in his message forward three letters in the alphabet, with the end of the alphabet wrapping around to cover the beginning. Caesar's is the first recorded use of this type of shift cipher, but earlier examples of message encryption do exist.

Jump forward nearly two millennia to the Cold War Era and the size of military organisations has grown tremendously, as has the quantity of information that must be managed. Deciding access rights on an individual basis between each person and piece of information had become impossible, so NATO countries began using *multi-level* security classifications for information control. This approach allows each piece of information to be classified once, based on its sensitivity, and each individual to be classified once, based on their trustworthiness. Information at a specified level (e.g. "Secret") was only to be accessed by an individual of equal or higher level (e.g. "Secret" or "Top Secret"). Advantages and disadvantages of this approach, as well as alternatives and historical perspective, can be found in Anderson's book [5].

Moving on to the present day, the advent of modern computers and telecommunications technologies has given rise to numerous means of storing and disseminating information. As a result, new techniques for controlling the flow of information have been developed: the more sophisticated the underlying technologies, the more advanced the techniques. The ways in which information is gathered and used have also changed and concern about information flow is no longer limited to military manoeuvres.

Corporations hold vast quantities of sensitive information about their customers and their own trade secrets. For example, the research and development team at a company may want to share some features of a product with the marketing team, without revealing how those features are implemented. The research team might be concerned that the salespeople will accidently leak those details when pitching the product. Government organisations also store vast amounts of information for non-military applications. One example is the use of national databases to store health records. These databases have enormous potential for privacy violations and proper access control is not easy to achieve. For example, researchers would like to use these records to correlate certain factors with the incidence of a given disease, but only a few factors are needed to identify an individual from their record, even if names and similar identifiers are removed. Denning [43] and others began studying this area of information *inference* in the 1970's.[2] Their work serves as a foundation for the framework developed in this chapter. An excellent introduction to issues of information control for health records, as well as data de-identification, can be found in Anderson's book [5]; he also presents details about issues of deploying national health databases [4].

One difficulty when controlling information flow is that some amount of flow may be necessary. Consider the case of password checkers; rejecting a login attempt leaks something about the correct password (i.e. the password is not the guess). This kind of information leakage is inherent in the system and cannot be avoided. The medical records example is more difficult. Researchers need any correlations in the data to be statistically significant, yet the probability of compromising the identity of a patient must be very low. How this balance might be achieved is not obvious and privacy-preserving data mining remains an active area of research; Evfimievski [56] presents a number of current issues in that area.

Another difficulty when attempting to prevent information leakage is the possibility of covert channels, hidden ways of communicating information. Covert channels could be used by a malicious insider to transfer data from a high-security area to a low-security area. This might be high-tech (e.g. setting the lock value on a publicly visible file) or low-tech (e.g. leaving a desk lamp on at night). Since it is difficult to eliminate covert channels altogether, the *rate* of leakage for these channels is often of primary concern. For example, the desk lamp channel is capable of transmitting one bit per night; whether or not this constitutes a risk would depend on the situation. A nice overview of covert channels and research in that area can be found in Anderson's book [5].

Given the complexity of modern solutions to controlling information flow, it is important to have ways of rigorously analysing the amount of control offered by those systems. Despite the many complications for managing information flow mentioned above, nearly all scenarios can be characterised as the transfer of some information from an acceptable location, which we'll call *high* security, to an unacceptable location, which is *low* security. Any flow of information from high to low constitutes *information leakage*. Since partial leakage may or may not be acceptable, a framework for analysis should *quantify* information leakage, rather than simply stating whether or not it can occur. Many algorithms, particularly those that involve information hiding, use randomness to achieve the desired results; thus, it is also beneficial for a framework to be applicable to systems involving *probabilistic* behaviour.

[2] Their original concerns were similar, but applied to census data rather than health records.

4.2 Information leakage formalised in HOL4

In this chapter, information leakage will be characterised by a flow of information from variables specified as *high* security to variables that are *low* security. Intuitively, a program leaks information if an attacker, who knows or can control the low-security inputs to the program, can infer something about the high-security inputs by observing the program's outputs. Recall the definition of conditional mutual information from Section 3.4.5: $\mathbb{I}(\mathcal{X};\mathcal{Y}|\mathcal{Z})$ measures how much information can be learned about $\mathcal{Y}$ by observing $\mathcal{X}$ (and vice versa), given that the value taken by $\mathcal{Z}$ is known. Notice that $\mathbb{I}(\mathcal{O};\mathcal{H}|\mathcal{L})$ captures the intuitive definition of information leakage given above, assuming that $\mathcal{O}$, $\mathcal{H}$, and $\mathcal{L}$ are random variables ranging over possible program outputs, high-security inputs, and low-security inputs respectively. The amount of information that can be learned about $\mathcal{H}$ by observing $\mathcal{O}$, given that the attacker knows or can control $\mathcal{L}$, is precisely $\mathbb{I}(\mathcal{O};\mathcal{H}|\mathcal{L})$. This is the formulation of information leakage used by Clark et al. [29] for deterministic programs and will be adopted in this chapter for that scenario. The distributions of $\mathcal{H}$ and $\mathcal{L}$ capture the likelihood of different sets of inputs. For a deterministic program $\mathcal{M}$, the distribution of $\mathcal{O}$ is completely determined by $\mathcal{M}$ and the distributions of $\mathcal{H}$ and $\mathcal{L}$.

This simple two-level approach is sufficient to characterise multi-level systems, since the amount of flow can be determined pointwise between each layer. Similarly, this approach can be used to analyse the security of multi-lateral data sharing between different departments or functional units, rather than between varying levels of classification. Again, leakage can be examined pointwise between each department, with high representing the department leaking information and low representing the department receiving that information.

4.2.1 Adding probabilistic behaviour

As mentioned earlier in this chapter, it is often desirable or necessary to introduce probabilistic behaviour into algorithms, particularly those involving some amount of information hiding. The dining cryptographers protocol (Chapter 5) is an excellent example of this, since it involves a number of coin tosses and its correctness relies on the randomness of those coin tosses. Fortunately, it is not difficult to introduce probabilistic nondeterminism to the model of information leakage developed so far.

Programs involving probabilistic behaviour can be modelled by including a third "random" portion of input state, in addition to the high and low-security portions. These inputs resolve the nondeterminism in a given run of the program and the distribution over possible random inputs characterises the probabilistic behaviour of the program. Let's look at an example of this sort of translation. High-security variables will be denoted by an overline and low-security variables by an underline, such as $\overline{high}$ and $\underline{low}$. The probabilistic algorithm

$$\text{if } heads \text{ then } \underline{out} := \overline{high} \text{ else } \underline{out} := \underline{low},$$

which uses a coin flip and has one high-security input and one low-security input, can be modelled as the deterministic algorithm

$$\text{if } r = 1 \text{ then } \underline{out} := \overline{high} \text{ else } \underline{out} := \underline{low},$$

which has an additional random input r taking the value 1 with probability $\frac{1}{2}$. Note that, in the case of a probabilistic assignment within a loop, a separate random input is required for *each* cycle of the loop.

The simple extension of the input state outlined above is all that is needed to model probabilistic behaviour in programs being analysed. However, it remains to be decided what role the *random* portion of the input state should play in the evaluation of information leakage. We will now examine a number of factors that affect this decision.

Two views on probabilism in information-leakage

There are two possible views of probabilistic behaviour that correspond to different types of systems and threat models. One possibility is that the attacker cannot directly observe the resolution of nondeterminism in a particular run of the system, but knows that it will follow a given distribution. For example, the attacker knows that a coin is flipped and it will come up heads or tails with equal probability, but he cannot observe the outcome of the coin flip for a given execution of the program. In this case, we will consider the random inputs to the program to be *hidden* from the attacker. Alternatively, the resolution of the probabilism in a particular execution could be *visible* to the attacker. In that case, he can observe

whether heads or tails has been flipped, but still cannot influence the outcome of the flip; the random input continues to follow the appropriate distribution and cannot be affected by the attacker.

In the sample probabilistic program above, the attacker learns all or none of the secret with equal probability, assuming that $\overline{high}$ and $\underline{low}$ cannot take the same value. Notice that the *visibility* of the outcome of the coin toss does not affect the leakage in that case. Leakage and lack thereof occur with equal probability, as long as the attacker cannot influence the *outcome* of the coin flip. In this text we will assume that the attacker cannot affect the function of programs in that way. (If such behaviour is possible, then such "random" inputs should really be modelled as low-security inputs over which the attacker has control.) Knowledge of the outcome of the coin flip is not needed for the attacker to obtain the secret in this program. However, there are cases where the security of the secret depends on the resolution of probabilistic behaviour being hidden from the attacker. One such example (Example 6) is provided in Section 4.4.

If the outcome of the coin flip is not visible to the attacker and $\overline{high}$ and $\underline{low}$ can take the same value, then the situation is slightly more complicated. In this case, the secret is only leaked when the output does not match the low input, the frequency of which is determined by the distribution over input states. There is no additional confusion when $\overline{high}$ and $\underline{low}$ can take the same value and the coin flip is visible, because the attacker knows which path of execution has been taken.

Let's look at a simpler example to clarify the difference between visible and hidden probabilistic behaviour. Consider the program

$$\text{if } r = \top \text{ then } \underline{out} := \overline{high} \text{ else } \underline{out} := \neg\overline{high},$$

where the variables are boolean and r and $\overline{high}$ are evenly distributed. If the outcome of the coin flip (i.e. the value assigned to r) is know to the attacker, the entire secret is leaked. Otherwise, the attacker learns nothing about the secret.

Having examined two possible views of probabilistic behaviour, information leakage for probabilistic programs must be defined, taking into account those two options. Malacaria [99] defined information leakage for programs involving probabilistic nondeterminism as

$$\mathbb{I}(\mathcal{O}; \mathcal{H} | (\mathcal{L}, \mathcal{R})),$$

where $\mathcal{R}$ is a random variable ranging over possible values for the random portion of the input state. Malacaria's work does not consider the distinction between visible and hidden random inputs, which is crucial in the present discussion. Recall the informal definition of conditional mutual information: $\mathbb{I}(\mathcal{X}; \mathcal{Y} | \mathcal{Z})$ measures how much can be learned about $\mathcal{Y}$ by observing $\mathcal{X}$, given knowledge of $\mathcal{Z}$. Malacaria's definition of leakage captures the case of *visible* probabilistic behaviour i.e. how much can be learned about the high-security inputs by observing the outputs, given knowledge of both the low and random inputs. Thus, the definition of leakage as $\mathbb{I}(\mathcal{O}; \mathcal{H} | (\mathcal{L}, \mathcal{R}))$ will be used for the case of programs exhibiting visible probabilistic behaviour. In practice, this can also be achieved by including such visible random inputs as part of the low inputs and using the definition of leakage as $\mathbb{I}(\mathcal{O}; \mathcal{H} | \mathcal{L})$.

If a program involves hidden probabilistic behaviour, Malacaria's definition will overestimate the leakage; this will be demonstrated with Example 6 in Section 4.4. Instead, I propose that leakage for programs exhibiting *hidden* probabilistic behaviour should be defined as

$$\mathbb{I}(\mathcal{O}; \mathcal{H} | \mathcal{L})$$

i.e. the same definition as for deterministic programs, which is not conditioned on the random inputs. This definition captures the meaning of the resolution of probabilistic behaviour being hidden; $\mathbb{I}(\mathcal{O}; \mathcal{H} | \mathcal{L})$ measures how much can be learned about the high-security inputs by observing the outputs, given knowledge of the low inputs (but not of the random inputs). As was mentioned above, visible random inputs can be modelled by simply including them with the low-security state. Thus, the definition of leakage as $\mathbb{I}(\mathcal{O}; \mathcal{H} | \mathcal{L})$ is sufficiently flexible to capture both visible and hidden probabilistic behaviour and even programs involving both. Since the probabilistic behaviour in the case study in Chapter 5 is of the hidden sort, such probabilism will be of primary concern in this chapter in preparation for that example.

Having defined information leakage using conditional mutual information above and formalised a definition of conditional mutual information in Chapter 3, most of the work in formalising a definition of information leakage is already done. However, a number of decisions must be made about how to model programs that are to be analysed for information leakage. We'll now look at a number of these issues, beginning with modelling program state.

4.2.2 Modelling program state

Programs that are to be analysed can be seen as transforming some initial program state to a final program state; these states characterise the values of all the program variables at a particular time in the program execution cycle. Program states can be modelled as mappings from variable names to values. For example, a high security program state, in which program variables $\overline{h1}$ and $\overline{h2}$ are assigned the values 0 and 1 respectively, can be modelled as the mapping $\langle \overline{h1} \mapsto 0, \overline{h2} \mapsto 1 \rangle$. Translating this concept into a HOL formalisation, program states have been modelled as HOL functions from variable names of type *string* to values of an arbitrary type α, which can be instantiated as desired. More formally, the HOL type : α *state* has been defined as

$$\alpha \ state = string \rightarrow \alpha.$$

Recall, that HOL functions must be total, so the functions characterising program states are necessarily total. As a result, states are infinite and must assign a value to every possible string-named variable. In practice, this is not a serious problem, since a default value can be assigned to all program variables that are not of interest. For example, the real-valued program state $\langle \underline{low} \mapsto 3.5 \rangle$ can be captured by the HOL function (λ`s`. `if s = "low" then 3.5 else 0`), which has type : *real state* and assigns the default value 0 to any program variable that is not of interest. This approach keeps the number of states to be considered finite, whenever there are a finite number of variables and values of interest.

4.2.3 Formalising program definitions

Prior to the research presented in this text, I developed a technique for analysing information leakage using a formalisation of the probabilistic Guarded Command Language (pGCL) [105] embedded in HOL4 by Hurd [85].[3] That approach seemed promising in theory, but in practice the embedded language and semantics became overly cumbersome in the theorem-proving environment. For example, it took me nearly a year to prove basic correctness and termination properties of the dining cryptographers case study (Chapter 5) using the pGCL approach; all of those proofs are necessary groundwork before proofs about information leakage can be undertaken.

As a result of my experience using formalised pGCL, I decided that programs to be analysed for information leakage should be modelled directly as HOL functions. This approach greatly reduces the difficulty in proving properties of modelled programs. For example, the correctness proofs for the dining cryptographers protocol took about a week's time when the protocol was modelled directly as a HOL function; that is a stark contrast to the year taken to prove the same correctness properties and termination using pGCL embedded in HOL4. As an added benefit of modelling programs as HOL functions, termination can generally be proved automatically by the HOL4 system; this was the case for the dining cryptographers case study.

Programs were described above as transforming an initial program state into a final program state. Let's take a closer look at how that concept is captured when formalising programs as HOL functions. Recall, that program variables must be specified to be of one of three types: high-security, low-security, or random. This can be achieved by dividing the program state into three separate parts: the high-security state, the low-security state, and the random state, each mapping variable names to values. Thus, programs have been modelled as HOL functions from a triple of states, representing the high, low, and random input states, to a low-security output state. Formally, the HOL types for program input-states, : *prog_state*, and programs, : *prog* have been defined as

$$(\alpha, \beta, \gamma) \ prog\_state = (\alpha \ state) * (\beta \ state) * (\gamma \ state)$$

and

$$(\alpha, \beta, \gamma, \delta) \ prog = (\alpha, \beta, \gamma) \ prog\_state \rightarrow \delta \ state,$$

where $\alpha * \beta$ denotes the type of pairs of the form $(x : \alpha, y : \beta)$.

By explicitly including the type of each program variable, this approach maintains flexibility and eliminates ambiguity. Let's look at an example of how a program updating its state would be formalised.

[3] The probabilistic Guarded Command Language is an extension of Dijkstra's Guarded Command Language [51] to include probabilistic assignment; it is a simple Turing-complete language which is capable of modelling probabilistic and nondeterministic behaviour, as well as non-termination.

A program that updates the low-security input state, to map a variable named $l1$ to the value 3, and then outputs the updated low-security state would be formalised as the HOL function

$$(\lambda(\texttt{high}, \texttt{low}, \texttt{random}).\ (\lambda \texttt{s}.\ \texttt{if s} = \text{“}\texttt{l1}\text{”}\ \texttt{then 3 else low s})).$$

This strategy of updating the low-security state and then outputting the updated state seems the usual behaviour that one would wish to model. Notice that the same variable name can be used for high-security, low-security, and random program variables. Since the appropriate part of the state must be mentioned when referencing program variables, this is not a problem. For example, `low` “`l1`” and `high` “`l1`” cannot be confused in that the first must be a program variable in the low-security state and the second in the high-security state. This technique of splitting the state is much more flexible and robust than any relying on a single state and naming conventions.

Bounds and rates of leakage for non-terminating looping constructs were one of the primary contributions of Malacaria's recent work [99]. Since HOL functions must be proved to terminate, one limitation of modelling programs as HOL functions is that non-terminating programs cannot be analysed; such analysis would require an embedding similar to the work with pGCL described above. Considering the benefits of modelling programs as HOL functions mentioned earlier, I decided that this limitation to terminating programs was acceptable.

4.2.4 Random variables over portions of program states

Recall the definition of information leakage decided upon above: $\mathbb{I}(\mathcal{O}; \mathcal{H}|\mathcal{L})$. Having decided upon a method for formalising programs and program states, only the random variables $\mathcal{O}, \mathcal{H}$ and $\mathcal{L}$ remain to be defined before formalising the definition of leakage. Since the input state is divided into its high-security, low-security, and random components, $\mathcal{H}$ and $\mathcal{L}$ are defined simply by identifying the correct portion of the input state.

Formalisation 38 (Random variable on high-security input states)**.**

$$\texttt{H} = (\lambda((\texttt{h}, \texttt{l}), \texttt{r}).\ \texttt{h}),$$

where $\texttt{h} : \alpha$ *state,* $\texttt{l} : \beta$ *state, and* $\texttt{r} : \gamma$ *state.*

Formalisation 39 (Random variable on low-security input states)**.**

$$\texttt{L} = (\lambda((\texttt{h}, \texttt{l}), \texttt{r}).\ \texttt{l}),$$

where $\texttt{h} : \alpha$ *state,* $\texttt{l} : \beta$ *state, and* $\texttt{r} : \gamma$ *state.*

Similarly, a random variable on the random input state can be defined. This is required for analysis of programs involving only visible probabilistic behaviour using the definition $\mathbb{I}(\mathcal{O}; \mathcal{H}|(\mathcal{L}, \mathcal{R}))$ as discussed above.

Formalisation 40 (Random variable on random input states)**.**

$$\texttt{R} = (\lambda((\texttt{h}, \texttt{l}), \texttt{r}).\ \texttt{r}),$$

where $\texttt{h} : \alpha$ *state,* $\texttt{l} : \beta$ *state, and* $\texttt{r} : \gamma$ *state.*

Finally, a program modelled as a HOL function as described above defines the random variable on low-security program outputs, so an explicit definition of $\mathcal{O}$ is unnecessary.

4.2.5 Formalised information leakage

Using the random variables defined above and the formalisation of conditional mutual information from Chapter 3 (Formalisation 37), the definition of information leakage as $\mathbb{I}(\mathcal{O}; \mathcal{H}|\mathcal{L})$ can easily be formalised in HOL.

Formalisation 41 (Info. leakage of programs with hidden probabilism)**.**

```
leakage (sp, sts, mu) f =
    conditional_mutual_information 2 (sp, sts, mu)
        (IMAGE f sp, POW (IMAGE f sp))
        (IMAGE H sp, POW (IMAGE H sp))
        (IMAGE L sp, POW (IMAGE L sp))
        f H L,
```

where (sp, sts, mu) *is a probability space, as formalised in Chapter 2, characterising the probability distribution over input states. The constant factor* 2 *in the definition identifies the base for the logarithm in the conditional mutual information. Reasons for this choice of base were presented in Chapter 3.*

Formalisation 41 defines leakage for programs exhibiting hidden probabilistic behaviour and will be of primary concern in the rest of this text; however, a similar formalisation for programs exhibiting only visible probabilistic behaviour has been included below for the sake of completeness.

Formalisation 42 (Info. leakage of programs with visible probabilism)**.**

```
visible_leakage (sp, sts, mu) f =
    conditional_mutual_information 2 (sp, sts, mu)
        (IMAGE f sp, POW (IMAGE f sp))
        (IMAGE H sp, POW (IMAGE H sp))
        (IMAGE (λs. (L s, R s)) sp, POW (IMAGE (λs. (L s, R s)) sp))
        f H (λs. (L s, R s)),
```

where (λs. L s, R s) *is the joint random variable defined by* L *and* R*.*

4.3 Assistance for information leakage analysis in HOL4

The decision to model programs as HOL functions greatly simplifies the modelling step for information leakage analysis. However, there are other task that must be undertaken. The foremost of these is modelling the distribution on input states. Formidable examples will probably require a substantial amount of effort to define the probability space characterising the distribution on initial states; furthermore, the space defined must be proved to form a probability space in accordance with the formalised definition.

Fortunately, it is possible to define different classes of probability spaces that characterise different distributions over input states. This is beneficial because the class of spaces as a whole can be proved to define proper probability spaces; in future applications, the class can be instantiated with a particular distribution and the proof does not need to be repeated.

One of the most useful classes of input distributions is the uniform distribution on inputs, in which all input states (from a set of valid states) are equally likely. The class of uniform input distributions has been formalised in HOL and support has been developed to ease proofs of information (non)-leakage involving such distributions. We will now examine some of those developments.

The techniques examined below are not limited to the uniform distribution and similar assistance could be formalised for other distributions; I have focused on the uniform distribution in this text because of its wide range of uses and specifically for its use in the dining cryptographers case study (Chapter 5).

4.3.1 unif_prog_space

Consider a situation where there are sets of valid high, low, and random input states that are deemed possible inputs to a particular program. Let's call these sets *high*, *low*, and *random* respectively. Assume that any initial program state that is composed of a high-security input state from *high*, a low security input state from *low*, and a random input from *random* is valid. If any valid initial state is equally likely, i.e. the initial states are uniformly distributed, then each initial program state has a probability of

$$\frac{1}{(|high|)(|low|)(|random|)},$$

where $|S|$ denotes the cardinality of set S. Since the probability of all initial states is a constant value, the definition of information leakage can be optimised to a simpler form for this class of input distribution.

The probability space that captures the uniform distribution on initial states outlined above has been formalised in HOL, taking *high*, *low* and *random* as parameters. That formalisation is presented below in two steps: first, the uniform point distribution on valid program inputs, then the probability space for that distribution.

Formalisation 43 (Uniform point distribution on initial states)**.**

```
unif_prog_dist high low random =
    (λs. if s IN high CROSS low CROSS random then
             1/(&(CARD(high CROSS low CROSS random)))
         else 0),
```

where &n *converts a natural number* n *into the equivalent real number. The function* unif_prog_dist *takes a value of type prog state and returns a real number.*

Formalisation 44 (Uniform probability space for initial states)**.**

```
unif_prog_space high low random =
    (high CROSS low CROSS random,
     POW (high CROSS low CROSS random),
     (λs. SIGMA (unif_prog_dist high low random) s)).
```

The probability space characterising the uniform distribution on initial program states, `unif_prog_space`, has been defined on the space $high \times low \times random$ and the set of events $\mathcal{P}(high \times low \times random)$, with $S \times R$ denoting the set cross product operation and $\mathcal{P}(S)$ the powerset of S. This specifies that any set of initial program states $S \subseteq (high \times low \times random)$ is an *event* and has a well-defined probability. Since the initial program states are evenly distributed, any set of states $S \subseteq (high \times low \times random)$ has probability

$$\frac{|S|}{(|high|)(|low|)(|random|)}.$$

The space defined by `unif_prog_space high low random` has been proved to form a valid probability space, assuming that `high`, `low`, and `random` are finite and that their cross product is a non-empty set; the statement of that theorem in HOL notation can be found in Appendix G as `prob_space_unif_prog_space`. That proof allows programs involving a uniform input distribution to be specified and analysed more easily. Using `unif_prog_space` to specify a space with a uniform distribution, the resulting space has already been proved to define a proper probability space. Moreover, the probability of a set of states S in `unif_prog_space` has been proved to be `(CARD S)/(CARD high * CARD low * CARD random)`, as explained in the preceding paragraph; that HOL theorem appears in Appendix G as `prob_unif_prog_space`.

4.3.2 Simplified leakage computation for `unif_prog_space`

As discussed earlier, the definition of information leakage for programs using the `unif_prog_space` distribution can be optimised to allow simpler proofs. First of all, `unif_prog_space` defines a finite, discrete probability space, so the conditional mutual information in Formalisation 41 can be simplified according to HOL Theorem 13. The definition can be reduced further by simplifying the probabilities involved; recall that the probability of any state is a constant value determined by the cardinality of the valid input sets. This reduces the definition of information leakage for programs using `unif_prog_space` to the form specified by `unif_prog_space_leakage_computation_reduce` in Appendix G.

The simplifications mentioned above greatly reduce the effort needed to prove the information leakage of programs modelled in HOL. In the case of more sophisticated examples involving inductive proofs, that means that the interactive proofs are less difficult. In the case of simple examples, the proofs can be dispatched with a single application of a HOL4 conversion that I have developed, followed by a few lines of script to simplify an equality involving arithmetic and $\log_2$; the HOL4 conversion automatically simplifies the leakage quantity down to an arithmetic expression, provided with appropriate lemmas about the program and its inputs. While this tool support is not fully automatic and requires the user to supply several inputs, it makes it much easier to calculate and prove the quantity of information leakage when

compared to a pen-and-paper approach. For instance, the examples in Section 4.4 were proved using this automated approach. Those proofs each involved between two and eight lines of proof script, which could easily be written in ten minutes or less. For those simple examples, the time for the proofs to be performed by the system was a minute or less for each, once the proof scripts were written.

A similar approach was used to prove the non-leakage of the dining cryptographers protocol (Chapter 5), for the case of three cryptographers. That example is much more complex than any of the examples in this chapter and the HOL system takes about eighteen minutes to process that proof; it is quite likely that such an approach would become intractable for larger examples. Despite that limitation, the ability to calculate the information leakage of a program modelled in HOL, even for small example cases, is useful for gaining insight into the leakage characteristics of a particular program. Even the simple examples we will examine in this chapter quickly become tedious to prove and the analysis error-prone when done on paper. In comparison, the computation in HOL is painless.

While the simple examples in this chapter could easily processed by the proof system without much tool support, the dining cryptographers example is more complicated and required more care in order to mechanise the computation using the proof system. The application of this automated approach for proving information leakage to the dining cryptographers protocol motivated a number minor tool developments; these made the computation easier and more efficient and reduced the amount of interaction that was necessary. That tool assistance is general and should be useful for many future applications.

4.4 Formalised information leakage examples

Let's apply the definition of information leakage developed above to some simple examples in order to gain intuition about the definition; some of the examples below were inspired by those found in Denning's book [43]. High-security variables will be denoted by an overline and low-security variables by an underline, such as $\overline{high}$ and $\underline{low}$.

Example 1 (Total leakage of addition)**.** Consider the program

$$\mathcal{M}1 \equiv \underline{out} := \overline{high} + \underline{low}$$

which has a high-security input $\overline{high}$, a low-security input $\underline{low}$, and a low-security output $\underline{out}$.

Each portion of the program state has only one variable ($\overline{high}$, $\underline{low}$, and $\underline{out}$ respectively). As a result, the probability of a random variable over one of those portions of the program state taking a particular value is simply the probability of the corresponding program variable taking that value. For example, $\mathbb{P}(\mathcal{H} = \langle\overline{high} \mapsto 1\rangle)$ is equivalent to $\mathbb{P}(\overline{high} = 1)$, where $\langle\overline{high} \mapsto 1\rangle$ is used to represent the high-security input state which has one variable $\overline{high}$ that is assigned the value 1. This simplification of notation will be used when a portion of the state contains a single program variable.

Assume that the distribution on the high and low security input states is such that

$$\mathbb{P}(\overline{high} = i) = \begin{cases} \frac{1}{4} & 0 \leq i \leq 3 \\ 0 & \text{otherwise,} \end{cases}$$

$$\mathbb{P}(\underline{low} = i) = \begin{cases} \frac{1}{4} & 0 \leq i \leq 3 \\ 0 & \text{otherwise.} \end{cases}$$

The distribution over $\underline{out}$ is implicitly defined by these two distributions and $\mathcal{M}1$.

Using the definition of information leakage above and Definitions 36 and 34 for conditional mutual information and mutual information, the leakage of $\mathcal{M}1$ for the distributions on $\overline{high}$ and $\underline{low}$ given above

is

$$\begin{aligned}
\mathbb{I}(\mathcal{O};\mathcal{H}|\mathcal{L}) &= \mathbb{I}(\mathcal{O};(\mathcal{H},\mathcal{L})) - \mathbb{I}(\mathcal{O};\mathcal{L}) \\
&= \sum_{o=0}^{6}\sum_{h=0}^{3}\sum_{l=0}^{3} \mathbb{P}(\underline{out} = o; (\overline{high}, \underline{low}) = (h,l)) \\
&\qquad \left(\log_2 \tfrac{\mathbb{P}(\underline{out}=o;(\overline{high},\underline{low})=(h,l))}{\mathbb{P}(\underline{out}=o)\mathbb{P}((\overline{high},\underline{low})=(h,l))}\right) \\
&- \sum_{o=0}^{6}\sum_{l=0}^{3} \mathbb{P}(\underline{out} = o; \underline{low} = l) \log_2 \left(\tfrac{\mathbb{P}(\underline{out}=o;\underline{low}=l)}{\mathbb{P}(\underline{out}=o)\mathbb{P}(\underline{low}=l)}\right) \\
&= \tfrac{1}{16}(0+32) \\
&= 2.
\end{aligned}$$

The result is that $\mathcal{M}1$ leaks 2 bits of the secret. Since the secret is the value of $\overline{high}$, which can be captured in 2 bits, $\mathcal{M}1$ leaks *all* of the secret. Knowing the result of an addition ($\underline{out}$) and one of its operands ($\underline{low}$), the other operand ($\overline{high}$) is completely determined. This intuition aligns with the leakage result above.

Now let's look at how Example 1 can be formalised and proved in HOL using the tools developed in Section 4.3. The first step is to model the program $\mathcal{M}1$ as a HOL function of type $(num, num, num, num)\ prog$.

```
M1 = (λs : ((num, num, num) prog_state).
        (λs'. if s' = "out" then
                 (H s "high") + (L s "low")
              else 0))
```

The next step is to define the probability space characterising the distribution on inputs. Since the distribution in Example 1 is uniform, `unif_prog_space` can be used to define the probability space. First, the sets of valid high, low, and random inputs need to be defined. These will be defined inductively, so the optimisations described in Section 4.3.2 can be used for the leakage proof.

```
high 0       = {(λs. if s = "high" then 0 else 0)} ∧
high (SUC n) = (λs. if s = "high" then SUC n else 0) INSERT (high n)

low 0        = {(λs. if s = "low" then 0 else 0)} ∧
low (SUC n)  = (λs. if s = "low" then SUC n else 0) INSERT (low n)

random       = {(λs. 0)}
```

Notice how the default value 0 has been used for variables not used in the program, thereby keeping the number of valid states finite. Similarly, since there are no random inputs, a default random input that is everywhere 0 has been used.

The goal to be proved, stating the quantity of information leakage for `M1`, is
`leakage (unif_prog_space (high 3) (low 3) random) M1 = 2`. The first step of the proof is to prove two lemmas used for testing equality between input states; they are `lem1`

```
∀s n. ((λs'. (if s' = s then n else 0)) = (λs'. 0)) = (n = 0)
```

and `lem2`

```
∀s n m. ((λs'. (if s' = s then n else 0)) =
         (λs'. (if s' = s then m else 0))) =
        (n = m),
```

which were proved automatically by the Metis tool in the HOL4 system. The next step for the proof is to define how the HOL tool for leakage computation should unfold the definitions for the input states. Since this is an easy example, a single simplification with arithmetic rules and the definitions and lemmas above is sufficient for any of the input three states:

```
example1_conv = SIMP_CONV arith_ss [high, low, random, lem1, lem2].
```

The next step is to define how the tool can decide the (in)equality of different inputs to and outputs from the program. In this case, `example1_conv` is also sufficient for this purpose for the three input states and the outputs.

At last, the HOL4 conversion to compute the quantity of information leakage can be used:

```
LEAKAGE_COMPUTE_CONV (“high 3”, “low 3”, “random”)
    [high, low, random, lem1, lem2]
    [M1, H_def, L_def, FST, SND]
    example1_conv example1_conv example1_conv
    example1_conv example1_conv example1_conv
    example1_conv.
```

The arguments to the tool are: three terms defining the high, low, and random input states; two lists of lemmas to be used in relation to the inputs and the program definition; three conversions to unwind the definitions of the high, low, and random input states; four conversions for determining (in)equality of the high, low, and random input states and the output states.

Using `LEAKAGE_COMPUTE_CONV` reduces the goal to

$$\texttt{inv}\ 16 * (0 - (-\texttt{lg}\ 4 + (-\texttt{lg}\ 4 + (-\texttt{lg}\ 4 + (-\texttt{lg}\ 4 + (-\texttt{lg}\ 4 + (-\texttt{lg}\ 4 + (-\texttt{lg}\ 4 + (-\texttt{lg}\ 4 + (-\texttt{lg}\ 4 + (-\texttt{lg}\ 4 + (-\texttt{lg}\ 4 + (-\texttt{lg}\ 4 + (-\texttt{lg}\ 4 + (-\texttt{lg}\ 4 + (-\texttt{lg}\ 4 + -\texttt{lg}\ 4)))))))))))))))) = 2.$$

The simplifier can then prove the lemma $\texttt{lg}\ 4 = 2$, given a few lemmas about logarithms. Rewriting with that lemma and some basic properties of real arithmetic, the goal becomes

$$32/16 = 2.$$

Finally, by instantiating a lemma about real division, the goal is proved. □

Knowing that the bitwise exclusive-or (XOR) operation is used in cryptrographic algorithms, one might try to eliminate the leakage in Example 1 by changing the addition to bitwise XOR. The following example illustrates the insecurity of that naïve approach.

Example 2 (Total leakage of bitwise exclusive-or)**.** Consider the program

$$\mathcal{M}2 \equiv \underline{out} := \overline{high} \oplus \underline{low}$$

where $\oplus$ represents the bitwise exclusive-or operation.

Assume that the distribution on the high and low security input states is the same as in Example 1. The distribution over $\underline{out}$ is implicitly defined by $\mathcal{M}2$ and the distribution over the inputs.

It is again the case that $\mathbb{I}(\mathcal{O};\mathcal{H}|\mathcal{L}) = 2$, even though the addition in Example 1 has been replaced with bitwise exclusive-or. As with addition, knowing the result of a bitwise exclusive-or and one of its operands, the other operand is completely determined.

Now let's look at how this example can be modelled in HOL; proofs will not be presented for this or any of the remaining examples, as they are very similar to the proof in Example 1. The program $\mathcal{M}2$ can be modelled as the HOL function

```
M2 = (λs : ((num, num, num) prog_state).
        (λs'. if s' = “out” then
                w2n((n2w(H s “high”))??(n2w(L s “low”)))
              else 0)),
```

where `n2w` and `w2n` respectively turn a natural number into the corresponding bit string (n-bit word) and back again and ?? is the bitwise XOR operation on n-bit words. The lists of valid inputs are the same as in Example 1 and the goal is nearly the same except with `M2` substituted for `M1`. □

The leakage in the examples above may seem trivially obvious, so let's move on to an example with a more subtle leakage.

Example 3 (Partial leakage of bitwise exclusive-or)**.** Consider the program

$$\mathcal{M}3 \equiv \underline{out} := \overline{h1} \oplus \overline{h2}$$

which has two high-security inputs $\overline{h1}$ and $\overline{h2}$, no low-security inputs, and a low-security output $\underline{out}$.

Assume that the distribution on the high security input states is such that

$$\mathbb{P}(\overline{h1} = i, \overline{h2} = j) = \begin{cases} \frac{1}{16} & 0 \leq i, j \leq 3 \\ 0 & \text{otherwise,} \end{cases}$$

The distribution over $\underline{out}$ is implicitly defined by this distribution and $\mathcal{M}3$.

As in the examples above, it is again the case that $\mathbb{I}(\mathcal{O}; \mathcal{H}|\mathcal{L}) = 2$. However, the secret is the value of $\overline{h1}$ and $\overline{h2}$, each of which is two bits, so this constitutes leakage of three quarters of the secret. This may seem counterintuitive, since one can not determine the value of either of the operands in a bitwise exclusive-or from the result thereof. However, a great deal is leaked about the values of $\overline{h1}$ and $\overline{h2}$ together, namely which bits of each are the same and which are different. Initially, an attacker knows there are 16 possible assignments of $\overline{h1}$ and $\overline{h2}$, but with knowledge of their exclusive-or, he can eliminate three quarters of those possibilities.

The program `M3` can be modelled in HOL as

```
M3 = (λs : ((num, num, num) prog_state).
        (λs'. if s' = "out" then
                 w2n((n2w(H s "h1"))??(n2w(H s "h2")))
              else 0)),
```

which is quite similar to `M2`. This time, the sets of valid high, low, and random inputs must be redefined in accordance with the description above. They are modelled in HOL, with the high inputs defined in two stages, as

```
h1 0          =  {(λs. if s = "h1" then 0 else 0)} ∧
h1 (SUC n)    =  (λs. if s = "h1" then SUC n else 0) INSERT (h1 n)

h2 l 0        =  IMAGE (λs. (λs'. if s' = "h2" then 0 else s s')) l ∧
h2 l (SUC n)  =  (IMAGE (λs. (λs'. if s' = "h2" then (SUC n) else s s')) l)
                 UNION (h2 l n)

high n        =  h2 (h1 n) n

low           =  {(λs. 0)}

random        =  {(λs. 0)}.                                          □
```

The first step in defining the high inputs uses the function `h1` to inductively define a set states with valid assignments for the variable "`h1`". In the second step, the function `h2` inductively constructs the set of valid high inputs by including each of the states generated by `h1`, updated with each of the valid assignments of the variable "`h2`".

Later on we'll look at an example of a probabilistic algorithm involving bitwise exclusive-or that does not result in any information leakage. For now, let's look at one more example that uses addition.

Example 4 (Partial-leakage of addition)**.** Consider the program

$$\mathcal{M}4 \equiv \underline{out} := \overline{h1} + \overline{h2}$$

which has two high-security inputs $\overline{h1}$ and $\overline{h2}$, no low-security inputs, and a low-security output $\underline{out}$. Assume that the distribution on the high-security input states is the same as in Example 3. The distribution over $\underline{out}$ is implicitly defined by this distribution and $\mathcal{M}4$.

The calculation of information leakage for $\mathcal{M}4$ provides a slightly cryptic value: $\mathbb{I}(\mathcal{O}; \mathcal{H}|\mathcal{L}) = \frac{1}{8}(26 - 3\log_2 3) \approx 2.66$. This is equivalent to a leakage between 2 and 3 bits; the reason that a non-integer value can result is that the relative probability of different inputs is taken into account in addition to their

resulting leakage. (Recall that the intuitive definitions of many of the information-theoretic concepts in Chapter 3 involve a *weighted* average).

To gain some insight into this result, let's look at some of the possible inputs, their relative probabilities, and the resulting leakages. In the case that $\underline{out} = 0$, all 4 bits of the secret are leaked, since $\overline{h1}$ and $\overline{h2}$ must both be 0. However, this output has a probability of only $\frac{1}{16}$, corresponding to the probability of the single input that can cause it. On the other hand, if $\underline{out} = 4$, then 8 of the 16 assignments of $\overline{h1}$ and $\overline{h2}$ are possible, resulting in half of the secret (2 bits) being leaked. Since half of the possible inputs result in this output, the probability of this leakage is $\frac{1}{2}$. In between these two extremes, it might be the case that $\underline{out} = 1$. Then one of inputs must be 1 and the other 0, so the only bit that is not leaked is which value is which. The probability of this output is $\frac{1}{8}$, since there are two possible inputs that can result in it.

The program $\mathcal{M}4$ can be formalised in HOL as

$$\begin{aligned}\texttt{M4} = (\lambda \texttt{s} : ((\texttt{num}, \texttt{num}, \texttt{num})\ \texttt{prog\_state}).\\ (\lambda \texttt{s}'.\ \texttt{if s}' = \text{“}\texttt{out}\text{”}\ \texttt{then}\\ (\texttt{H s “h1”}) + (\texttt{H s “h2”})\\ \texttt{else 0})),\end{aligned}$$

The distribution over inputs is modelled the same as in Example 3. □

4.4.1 Handling intermediate values

Up until this point, we have only considered programs that involve a single assignment to any program variable. If multiple (overwriting) assignments to a variable occur in a program to be analysed, some decisions must be made about the attacker model being considered. On the one hand, the program could be viewed as a black box; the attacker can observe the inputs entering and the outputs leaving, but cannot observe intermediate values, even for low-security variables. Alternatively, the attacker could be allowed to see all the intermediate values taken by low-security variables during program execution, but not the intermediate values of high-security variables. To gain some understanding of the importance of this distinction, let's look at another example.

Example 5 (Leakage through an intermediate value)**.** Consider the program

$$\begin{aligned}\mathcal{M}5 \equiv\ & \underline{out} := \overline{high};\\ & \underline{out} := \underline{low},\end{aligned}$$

where a semicolon denotes sequential composition.

Only a quick examination of $\mathcal{M}5$ is needed to see that the attacker learns all of the secret if he can observe the intermediate value of $\underline{out}$ after the first assignment. However, he learns nothing about the secret if he can only see the final value of $\underline{out}$. This is the case regardless of the distribution on $\underline{low}$ and $\overline{high}$ (assuming $\overline{high}$ can take more than one value i.e. is not already known certainly). □

As Example 5 illustrates, careful consideration is necessary when deciding how to handle intermediate values, since the same program can leak all or none of the secret depending on the choice of viewpoint. It is important to recognise that neither view is universally correct; each corresponds to different capabilities of a potential attacker. The correct choice is the one that reflects the capabilities of the attacker and system considered. An incorrect choice will result either in an overestimation of leakage or in potential leakage going unidentified.

The approach to characterising information leakage outlined above is sufficiently flexible to model both views of intermediate values with little additional effort. Use of the framework as demonstrated in the examples above already captures the black box approach to intermediate values. For example, analysis that does not consider intermediate values would be performed by modelling $\mathcal{M}5$ in HOL as

follows.

$$\begin{aligned}
&\texttt{assign1} = (\lambda \texttt{s} : ((\texttt{num}, \texttt{num}, \texttt{num})\ \texttt{prog_state}).\\
&\qquad (\lambda \texttt{s}'.\ \texttt{if}\ \texttt{s}' = \text{“}\texttt{out}\text{”}\ \texttt{then}\\
&\qquad\qquad (\texttt{H}\ \texttt{s}\ \text{“}\texttt{high}\text{”})\\
&\qquad\quad \texttt{else}\ 0))
\end{aligned}$$

$$\begin{aligned}
&\texttt{assign2} = (\lambda \texttt{s} : ((\texttt{num}, \texttt{num}, \texttt{num})\ \texttt{prog_state}).\\
&\qquad (\lambda \texttt{s}'.\ \texttt{if}\ \texttt{s}' = \text{“}\texttt{out}\text{”}\ \texttt{then}\\
&\qquad\qquad (\texttt{L}\ \texttt{s}\ \text{“}\texttt{low}\text{”})\\
&\qquad\quad \texttt{else}\ 0))
\end{aligned}$$

$$\begin{aligned}
&\texttt{M5} = (\lambda \texttt{s} : ((\texttt{num}, \texttt{num}, \texttt{num})\ \texttt{prog_state}).\\
&\qquad \texttt{assign2}\ ((\texttt{H}\ \texttt{s}, \texttt{assign1}\ \texttt{s}), \texttt{R}\ \texttt{s}))
\end{aligned}$$

For the input distribution defined in Example 1, M5 has been proved non-leaking.

Using the same framework, knowledge of intermediate values can be modelled by changing all assignments in a program from overwriting operations to operations that add the current value of the variable to a list of values taken by that variable. For example, $\mathcal{M}5$ could be changed to

$$\begin{aligned}
\mathcal{M}5' \equiv\ & \underline{out} := (HD(\overline{high})) :: \underline{out};\\
& \underline{out} := (HD(\underline{low})) :: \underline{out},
\end{aligned}$$

where $h :: tl$ adds the element h to the front of list tl and HD selects the first element from a list. Notice that only the *current* value of a variable, i.e. the one at the front of the list, should be added to the list of another variable when an assignment occurs. The final value of $\underline{out}$ in $\mathcal{M}5'$, $\underline{out} = [\underline{low}, \underline{high}, \text{initial value of } \underline{out}]$, reflects all the intermediate values taken by $\underline{out}$. Thus, the same definition for information leakage applied to $\mathcal{M}5$ and $\mathcal{M}5'$ would capture their respective non-leakage and total leakage, corresponding to the differing views of attacker capabilities. Where initialisation of $\underline{out}$ is not necessary or appropriate, its initial value can be set to the nil list. This method of handling different views of intermediate values is extremely flexible and can even be used to model situations in which some intermediate values should be considered visible and others hidden; care should be taken in such complex scenarios in order to maintain a consistent handling of those variables/values that should be considered visible and those that should not.

Considering intermediate leakage, the program $\mathcal{M}5'$ would be formalised in HOL as follows.

$$\begin{aligned}
&\texttt{state_update name value} = (\lambda \texttt{s}.\ (\lambda \texttt{n}.\ \texttt{if n} = \texttt{name then value else s n}))\\
&\\
&\texttt{state_append name value} = (\lambda \texttt{s}.\ \texttt{state_update name}\\
&\quad ((\texttt{if value} = [] \texttt{ then } [] \texttt{ else } [\texttt{HD value}]) + + (\texttt{s name}))\ \texttt{s})\\
&\\
&\texttt{assign1}' = (\lambda \texttt{s}.\ \texttt{state_update}\ \text{“}\texttt{out}\text{”}\ (\texttt{H s}\ \text{“}\texttt{high}\text{”})\ (\texttt{L s}))\\
&\\
&\texttt{assign2}' = (\lambda \texttt{s}.\ \texttt{state_append}\ \text{“}\texttt{out}\text{”}\ (\texttt{L s}\ \text{“}\texttt{low}\text{”})\ (\texttt{L s}))\\
&\\
&\texttt{M5}' = (\lambda \texttt{s} : ((\texttt{num}, \texttt{num}, \texttt{num})\ \texttt{prog_state}).\\
&\qquad \texttt{assign2}'\ ((\texttt{H s}, \texttt{assign1}'\ \texttt{s}), \texttt{R s}))
\end{aligned}$$

Since the HOL function HD is only well-defined for non-empty lists, the behaviour in $\mathcal{M}5'$ must be modelled using the if-then construction and the list concatenation function ++ as seen above. The input states must also be redefined so the states are num list-valued rather than num-valued.

$$\begin{aligned}
\texttt{high 0} &= \{(\lambda \texttt{s}.\ \texttt{if s} = \text{“}\texttt{high}\text{”}\ \texttt{then}\ [0]\ \texttt{else}\ [])\} \wedge\\
\texttt{high (SUC n)} &= (\lambda \texttt{s}.\ \texttt{if s} = \text{“}\texttt{high}\text{”}\ \texttt{then}\ [\texttt{SUC n}]\ \texttt{else}\ [])\ \texttt{INSERT (high n)}\\
&\\
\texttt{low 0} &= \{(\lambda \texttt{s}.\ \texttt{if s} = \text{“}\texttt{low}\text{”}\ \texttt{then}\ [0]\ \texttt{else}\ [])\} \wedge\\
\texttt{low (SUC n)} &= (\lambda \texttt{s}.\ \texttt{if s} = \text{“}\texttt{low}\text{”}\ \texttt{then}\ [\texttt{SUC n}]\ \texttt{else}\ [])\ \texttt{INSERT (low n)}\\
&\\
\texttt{random} &= \{(\lambda \texttt{s}.\ [])\}
\end{aligned}$$

This formalisation captures the case of intermediate information leakage in $\mathcal{M}5$ and M5′ has been proved to leak 2 bits, i.e. all of the secret, for the input distribution modelled above.

4.4.2 Security through hidden probabilistic behaviour

Example 6 (Non-leakage of bitwise exclusive-or)**.** Consider the program

$$\mathcal{M}6 \equiv \underline{out} := \overline{h} \oplus r$$

which has one high-security input $\overline{h}$, a random input r, no low-security inputs, and a low-security output $\underline{out}$.

Assume that the distribution on the high-security and random input states is such that

$$\mathbb{P}(\overline{h} = i, r = j) = \begin{cases} \frac{1}{16} & 0 \leq i, j \leq 3 \\ 0 & \text{otherwise,} \end{cases}$$

The distribution over $\underline{out}$ is implicitly defined by this distribution and $\mathcal{M}6$.

If the outcome of r is considered to be visible to the attacker, the program is equivalent to $\mathcal{M}2$ from Example 2 and all 2 bits of the secret are leaked. In contrast, if the value taken by r in a particular execution of $\mathcal{M}6$ cannot be observed by the attacker, then no information about $\overline{h}$ is leaked.

This example is worth remembering because it is analogous to the use of bitwise exclusive-or in the dining cryptographers protocol studied in Chapter 5; an understanding of Example 6 provides useful insight into the reason for the correctness of the dining cryptographers protocol.

The scenario involving $\mathcal{M}6$ and the distribution outlined above can be modelled in HOL as follows.

```
M6 = (λs : ((num, num, num) prog_state).
        (λs'. if s' = "out" then
                 w2n ((n2w (H s "h"))??(n2w (R s "r")))
              else 0)).
```

The sets of inputs are formalised as

```
high 0          =  {(λs. if s = "h" then 0 else 0)} ∧
high (SUC n)    =  (λs. if s = "h" then SUC n else 0) INSERT (high n)

low             =  {(λs. 0)}

random 0        =  {(λs. if s = "r" then 0 else 0)} ∧
random (SUC n)  =  (λs. if s = "r" then SUC n else 0) INSERT (random n)
```

In the event that r is deemed hidden, then the goal to be proved is very similar to the form in Example 1,

```
leakage (unif_prog_space (high 3) (low 3) random) M6 = 0.
```

This non-leakage of `M6`, when r is deemed hidden, has been proved using the HOL4 tool. Similarly, the total leakage of `M6`, when r is visible, has been proved in HOL4; that goal is the same as for the hidden case except that `leakage` is replaced by `visible_leakage` and the quantity is `2` rather than `0`. □

4.4.3 *What* is being leaked?

The primary aim of the work presented in this chapter is to *quantify* the amount of information leaked by a program. Underlying the framework discussed above is the assumption that any leakage of high-security information is equally disastrous, so it is not of importance which particular pieces of information are leaked, only the quantity. However, there may be some cases were it is desirable to determine if a particular piece of information has been leaked. This type of analysis can be accommodated within the approach presented above by isolating the high-security input of interest and treating other high-security inputs as hidden random inputs. To see how this can be used, let's revisit Example 3.

Recall the program in Example 3

$$\mathcal{M}3 \equiv \underline{out} := \overline{h1} \oplus \overline{h2},$$

which has two high-security inputs $\overline{h1}$ and $\overline{h2}$. Assuming that $\overline{h1}$ and $\overline{h2}$ take values from $\{0, \ldots, 3\}$ with equal probability, $\mathcal{M}3$ leaks two bits of the secret; an attacker can eliminate three quarters of the

possible assignments of $\overline{h1}$ and $\overline{h2}$ because they do not yield the appropriate exclusive-or. Despite this leakage, the individual values of $\overline{h1}$ and $\overline{h2}$ are not leaked. In order to prove that this is the case, the amount of $\overline{h1}$ leaked by $\mathcal{M}3$ and the amount of $\overline{h2}$ leaked by $\mathcal{M}3$ can be proved separately. This is done as described above. To isolate $\overline{h1}$ as the only high input and calculate how much of it is leaked, the other high-security inputs ($\overline{h2}$) are modelled as hidden random inputs. This makes $\mathcal{M}3$ equivalent to $\mathcal{M}6$, which does not leak any information. Similarly, $\overline{h2}$ can be isolated in order to prove that its value is not leaked by $\mathcal{M}3$.

It may seem counterintuitive that a program which does not leak the value of either of the two secret inputs still leaks half of the secret. As mentioned, this is because information is leaked about the relationship between these two values. There is no universally correct decision as to whether such information is sufficient to mount an attack, so the leakage results would have to be considered within the particular context. However, there are numerous examples of xor-ciphers being broken because of reuse of a key, demonstrating that such leakage can be significant. One such example is the breaking of the German Lorenz Cipher by analysts at Bletchley Park during WWII [81]; that attack on the cipher system was possible because the same key was reused once to transmit very similar messages.

The framework in this chapter provides a flexible setting to investigate and prove the information leakage of a program; where more detail is needed or desired, one can determine the information leaked about particular variables using the technique explained in this section.

4.4.4 Information flow from low to high-security

Another possible type of leakage results from an assignment from a low-security variable to a high-security variable. Let's examine this more carefully by studying another example.

Example 7 (Information flow from low to high)**.**

$$\begin{aligned} \mathcal{M}7 \equiv \quad & \overline{high} := \underline{low}; \\ & \underline{out} := \underline{low}. \end{aligned}$$

□

Whether $\mathcal{M}7$ is a security violation or not depends again on the threat model. If one is concerned with intermediate and final values of $\overline{high}$, then the program would leak information, since the attacker knows the final value of $\overline{high}$. For example, if $\overline{high}$ were a password, then setting the password to a known value might be sufficient for the attacker's purposes, even if the initial value of $\overline{high}$ is never obtained by the attacker. On the other hand, if the attacker's objective is to obtain the initial value of $\overline{high}$ that was input, then $\mathcal{M}7$ does not leak any information.

Information flow from low inputs to high intermediate values can be modelled in a similar manner to the intermediate flow from high to low studied earlier. Again, one must consider the capabilities of the attacker modelled and how intermediate values should be handled. Even though intermediate values of the high-security variables are not know to an attacker, they may still be a concern. This issue is illustrated by the following example.

Example 8 (Intermediate information flow from low to high security)**.** Consider the program

$$\begin{aligned} \overline{h1} &:= \underline{low}; \\ \overline{h1} &:= \overline{h2}. \end{aligned}$$

If the attacker can halt or delay the execution of the program after the first assignment, then he knows the value of $\overline{h1}$ at that time and can make use of that knowledge; however, if this is not within the attacker's power, then the program leaks nothing about the initial value of $\overline{h1}$ and its final value remains unknown, since the value of $\overline{h2}$ is unknown. □

If intermediate values for high are to be considered, i.e. the attacker could halt or delay execution of the program after a particular instruction, then the values taken by the high-security variables should be treated as a list, just as intermediate values of low security variables were handled above. If only the initial and final values of high-security variables are a concern, then only that pair of values should be considered; assignments would be treated as updates (rather than additions to a list), as is done for low security variables when not considering intermediate values. In order to model information flow from low

to high security, the random variable $\mathcal{H}$ needs to be redefined to range over the lists of values (or pairs of final and initial values) taken by the high-security variables, rather than initial values only. This would involve modelling programs as HOL functions from a triple of states, representing the high, low, and random input states, to a pair of states representing the resulting high and low security output states; a random output state would not needed because the random state is inherently *read-only* and should not be updated by programs. Leakage from low to high security will not be considered in the remainder of this text, but has been mentioned since the framework can easily be adapted to accommodate it.

4.5 Related work and novel contributions

There is a vast amount of literature on the topic of information leakage and a complete survey is not possible within the scope of this chapter. In the following review, emphasis will be placed on early foundational work and those developments encountered in a linear progression from the field's inception to the research presented in this chapter.

4.5.1 Towards probabilistic, quantitative analysis

Some of the earliest work in the area of information flow and leakage was Denning's [43] work on information inference using census data. Building on the work of Denning and others, Goguen and Meseguer [65, 66] provided the first definitions for information leakage as *(non)-interference* in deterministic state machines. Subsequent formative research on information leakage and non-interference can be divided into two primary categories: nondeterminism [101, 103, 145, 156, 157] and probabilistic behaviour [71, 72, 70, 106, 132, 153], as in information theory. Gray [71, 72, 70] and McLean [106] conducted some of the most important early work on information leakage in the probabilistic setting. The framework developed in this chapter is quantitative rather than qualitative in order to provide meaningful analysis in the case of partial leakage. Millen's work [107] was foundational for quantitative analysis of information leakage; subsequent work in the area includes that of Clark et al. [28] and Di Pierro et al. [121].

4.5.2 Formal methods

A substantial amount of work has been done applying formal methods to information leakage using process algebras [3, 58, 59, 60, 130, 129, 131]; unfortunately, none of that work was quantitative or probabilistic. However, there has been noteworthy work in that area providing quantitative analysis. Lowe has consistently pioneered the use of model-checkers for formal analysis of security properties and his work on information leakage [95, 96, 97, 98] is no exception. Lowe used the process algebra CSP to model information flow in systems and automated that analysis using the model-checker FDR. While his approach was quantitative and based on information theory, it was strictly nondeterministic and did not allow for analysis of probabilistic algorithms. Another piece of work that includes tool support for verification is the PicNIc project by Crafa et al. [35]. Their tool uses models in the π-calculus and allows for qualitative non-intereference to be automatically checked for finite systems.

The framework presented in this chapter is the first that has been developed for analysing information leakage within a theorem-prover. Furthermore, this is the first tool-assisted framework that allows for a quantitative, probabilistic approach to information leakage. The use of automatic model-checkers in previous frameworks has limited their applicability to finite-state systems. In contrast, the mechanised proof approach developed above is viable for infinite-state systems. A more detailed comparison of model-checking and theorem-proving approaches is provided in Chapter 1.

4.5.3 Programming language approaches

Clark, Hunt, and Malacaria [28, 29, 30, 99] have developed language-based analysis for information leakage using Shannon's information theory. Much of their work has focused on quantitative analysis for programs involving probabilistic nondeterminism; that work has been one of the major sources of inspiration for the framework developed in this chapter and a number of features have been borrowed from their approach. Recently, they have developed a semantics-based approach to characterising information leakage for a simple Turing-complete language including while loops [29, 99]; that work included analysis of rates and bounds on leakage in loops, even loops that may not terminate.

One of the major criticisms of Clark's approach is that the intermediate values of high and low security variables are not considered when calculating leakage, only their initial and final values. This makes it unsuitable for analysis of timing attacks, except for global timing attacks. Though modelled after Clark's work, the framework presented above overcomes this weakness and allows intermediate values to be considered in the leakage computation if desired. Aiming to address the weakness in their analysis, Clark and Hunt [27] have recently developed an approach for analysing information leakage in an interactive setting; unfortunately, that approach is restricted to qualitative analysis and deterministic programs. O'Neill et al. [118] have done similar work to develop a method of qualitative analysis in the interactive setting; their analysis differs from Clark and Hunt's in that it is valid for programs involving both probabilistic and nondeterministic operations.

Bossi et al. [16] have also taken a language-based approach to information leakage. They developed a framework for qualitative analysis of information flow based on bisimulation; they then went on to define proof techniques for this approach, leveraging the Tarski decidability of first-order formulae on the real numbers.

An alternative approach has been developed by McIver and Morgan [104] which uses the probabilistic Guarded Command Language. The strength of their framework is that *intermediate* values of high and low variables are kept track of and the security of the high variables is maintained, though their values may change during program execution; there approach allows for probabilistic algorithms, but the analysis is qualitative and cannot be used to reason about partial leakage.

Clarkson et al. [31] have proposed that information leakage should be not be based on the reduction of uncertainty in an attacker's beliefs, when attacker is making (possibly incorrect) assumptions about the distribution of inputs. They argue that leakage should be based on the amount of change in the attacker's beliefs rather than the uncertainty. Their belief-based approach differs from all the others mentioned in this section. The strength of any analysis taking this view would seem heavily dependent on how accurately the prospective attacker's beliefs can be predicted.

At the more applied end of the spectrum, Hansen and Probst [76] have looked at using information flow analysis to verify non-interference of Java Card bytecode. Their approach allowed for qualitative analysis based on observational equivalence, using a simple language modelling the Java Card bytecode language.

4.5.4 Computational-complexity approaches

Recent work by Backes and Pfitzmann [6, 7, 8] has provided a means for quantitative analysis of information flow in a reactive setting. Their proofs are of the pen-and-paper variety, but they allow for polynomial time complexity-theoretic proofs as typically used by cryptographers. This approach makes their framework particularly suited for proofs involving real cryptographic algorithms and helps to bridge the gap between formal analysis and deployed systems. Backes considers the introduction of information-theoretic analysis to their framework as future work [6]. A computational-complexity approach to verification in an interactive theorem-prover has not been undertaken, so using a framework similar to Backes's would be premature. However, Blanchet [14] has recently developed a tool for automatic proof of security protocols in the computational-complexity model. It would be interesting to see if such an approach could be developed within the framework of an interactive theorem-prover and then to develop Backes's theories on non-interference in that setting.

4.6 Summary

This chapter explained how the formalisation of information theory from Chapter 3 can be used to analyse the information leakage of programs modelled as HOL functions. The framework outlined is sufficiently general to model various scenarios and threat models, a number of which were discussed above.

The decision to model programs as HOL functions is important, because it allows for more natural specification and proof. Additionally, this approach allows automated tools to be developed to minimise the amount of interaction required for simple examples. To demonstrate these benefits, some tool support has been developed for the case of uniformly distributed inputs and applied to a number of examples. These examples and their formalisations demonstrated use of the proof framework.

The work in this chapter will be the foundation for the proof of the dining cryptographers protocol in Chapter 5. That chapter examines the use of the framework developed here to prove a well-known case

study. Looking in detail at the example in Chapter 5, the application of the theory in this chapter will become more concrete.

A number of scenarios and threat models were considered above, but one question that was not addressed is how encryption can be handled in that analysis. This is because most encryption algorithms are not information-theoretically secure[4] — there is necessarily a correspondence between the inputs and outputs allowing for subsequent decryption; the security of many modern encryption algorithms relies on this correspondence being *difficult* to compute. As a result, encryption cannot be generalised easily in the information-theoretic setting: either the information-theoretic properties need to be specified in detail in order to appropriately model a particular algorithm, or else the requirements will be too severe to be satisfied by the implemented encryption algorithm. Nonetheless, the analysis approach in this chapter has numerous applications that do not involve encryption and can still be used in settings involving simple encryption techniques such as XOR. This can be seen in the examples in this chapter and the case study in Chapter 5. Futhermore, information-theoretic analysis of encryption algorithms themselves represents a potential application for the techniques presented in this chapter.

[4]Shannon's work [138] shows that an information-theoretically secure encryption algorithm requires a key the length of the plain text e.g. a one time pad.

Chapter 5

Anonymity, Cryptographers, and Gastronomy

"When I get a little money I buy books; if any is left, I buy food..."
– Desiderius Erasmus

The advent of systems for anonymous electronic communication has created a need for formal methods of analysing the relative strength of those systems. This chapter will demonstrate how privacy guarantees can be seen as information non-leakage properties. The framework presented in Chapter 4 can then be used to prove the amount of anonymity provided by a system. After a brief discussion of these basic ideas, the remainder of this chapter will be devoted to proving the anonymity guarantees of Chaum's dining cryptographers (DC) protocol [22]. Using the framework from Chapter 4, the DC protocol has been proved to preserve total anonymity for an unbounded number of participants. This case study illustrates a practical application of the formalisations developed earlier in this text.

5.1 Motivation

Since it was first proposed in 1988, the dining cryptographers protocol has been one of the most frequently used examples for tools aimed at analysing privacy or anonymity guarantees [10, 18, 26, 41, 40, 74, 80, 91, 109, 133, 152]; this continues to be the case with recent research [18, 26, 41, 80, 91, 109]. The simplicity of this mature and well-understood protocol, as well as the wide range of tools that have been applied to it, make it an ideal benchmark. Using the DC protocol as a common case study ensures that the capabilities and limitations of different approaches to analysis are assessed with respect to a common basis. This guarantees that any differences exhibited are a result of the analysis approach rather than the choice of example. That is not to say that more sophisticated examples are not important, since they demonstrate the scalability of an analysis approach. However, the dining cryptographers remains the quintessential example and first proof of concept for any approach to analysing privacy or anonymity properties.

5.2 Anonymity as information non-leakage

While the contribution might seem trivial in retrospect, treating privacy and anonymity in terms of information leakage is of great practical benefit; that insight allows developments from the more mature area of information leakage to be applied to privacy. In this chapter, the framework developed in Chapter 4 will be adopted for information leakage analysis. By modelling any identity-linked values used in a system as high-security inputs, the information flow of the system captures the extent to which privacy or anonymity is compromised. Recall from Chapter 4 that the information leakage of a program quantifies how many bits of the high-security inputs are leaked by the outputs — in this case, how many bits of identifying information are leaked.

Though it may not be possible to characterise all systems by modelling identifying information as high-security inputs, that approach is sufficiently flexible to address many of the major system designs in the literature. This can be seen with the dining cryptographers example in the remainder of this chapter,

where the identity of the cryptographer that has paid is treated as a high-security input. Mix networks can be analysed by treating the mapping between messages, senders, and receivers as a high-security input and the messages entering, being rerouted, and ultimately delivered as the visible outputs of the system; the view of the attacker can be specified by which links between mixes are treated as visible outputs. A similar approach could be taken to analysing a system like Crowds [125]. As discussed earlier in this text, handling encryption is a difficult issue in information-theoretic analysis; however, it is worth noting that those difficulties are not inherent to the treatment of anonymity properties as information leakage.

Without further delay, we will now move on to examine a concrete application of the ideas developed above to the dining cryptographers protocol. First we'll look at a simple explanation of that protocol.

5.3 The dining cryptographers protocol

In Chaum's original presentation of the dining cryptographers problem [22], a group of cryptographers sit down to dinner and are immediately informed by the Maître d'hôtel that the bill has already been paid. They come to the conclusion that either one of their party has paid, or the bill has been paid by some external agency such as the US National Security Agency (NSA). They would like to determine which of these has occurred while preserving the anonymity of the payer, in the event that one of the cryptographers has paid. The following solution, known as the dining cryptographers (DC) protocol, is suggested. Each cryptographer flips a fair coin under the table and shares the outcome of the coin flip with the cryptographer to his left. All the cryptographers then announce whether or not the coins they saw matched; if one of the cryptographers has paid, then he should announce the opposite of what he has actually seen. If the number of "don't match" announcements is odd, then one of the cryptographers has paid. This can be expressed mathematically as each cryptographer announcing the bitwise exclusive-or (XOR) of the two coins he has seen (his own and that of the cryptographer to the right) and whether or not he has paid. The result is then computed as the XOR of the announcements; if the XOR of these announcements is true, then one of the cryptographers has paid, otherwise some outside agency has paid. Figure 5.1 shows a sample execution of the protocol in which one of the cryptographers has paid.

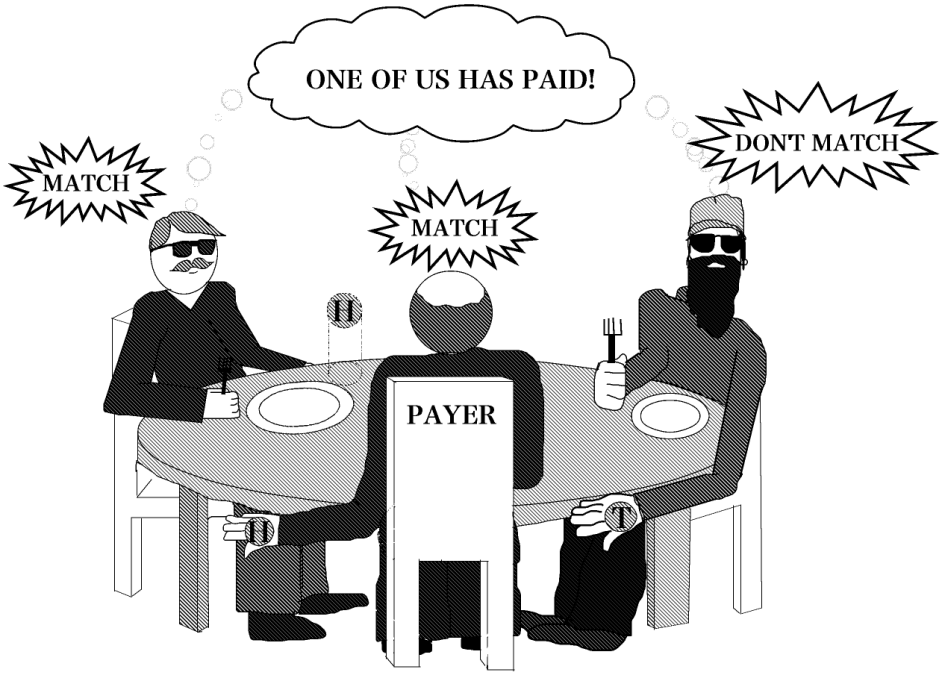

Figure 5.1: An execution of the dining cryptographers protocol.

In its original form, the DC protocol might seem quite restricted in applicability, but it can be generalised to more useful forms. For instance, the ring structure imposed by the cryptographers sit-

ting around a table can be relaxed to an arbitrary graph structure. Moreover, repeated execution of the protocol serves as the basis for an anonymous broadcast protocol — each iteration of the protocol anonymously broadcasts one bit.[1]

The anonymity of the dining cryptographers protocol relies on the fairness of the coins and properties of XOR; Chaum proved the total anonymity guarantee of the protocol in his original presentation. Note that the probabilistic behaviour inherent in the coin flipping of the DC protocol is that which was classified as *hidden* probabilism in the previous chapter — the coins are flipped under the table where they cannot be observed.

5.4 The dining cryptographers protocol in HOL4

Before analysing the anonymity of the dining cryptographers protocol, it is first necessary to model the protocol as a HOL function of the type specified in Chapter 4. The HOL definition of the protocol has been constructed inductively and has been parameterised on the number of cryptographers and whether or not the NSA has paid; this allows properties of the protocol to be proved for an unbounded number of cryptographers and is the most natural way of specifying the protocol in HOL. The protocol has been modelled specifically for the original case of a ring-structured network and has not been generalised to arbitrary graphs.[2]

Taking the outcome of the cryptographers' coin flips as "random" inputs to the system, the protocol can be seen as having two stages — the cryptographers make their announcements based on the results of the coin flips and then compute the outcome of the protocol based on those announcements. Below we will examine how each of these stages can be defined as HOL functions and then composed to form the entire protocol.

All of the values involved in the protocol are boolean (eg. heads/tails and paid/didn't pay), so the *states* of the system have been modelled as mappings from string-type variable names to boolean values. The system takes as its input three states: a high input state describing who has paid, a low input state that is not used, and a random input state containing the results of the cryptographers' coin flips. The output state of the system includes the announcement of each cryptographer as well as the result of the protocol.

5.4.1 Setting the cryptographers' announcements

Let's look at how the first stage of the protocol, setting the cryptographers' announcements, has been modelled in HOL. The output state of this stage is simply the low input state updated to include each cryptographer's announcement; those values are stored in variables "`announces 0`", ..., "`announces n`", where there are `n + 1` cryptographers. As a convention, the $n + 1$ cryptographers have been indexed $0, \ldots, n$. The announcement of cryptographer $i + 1$ is computed as the exclusive-or of his own coin toss (variable "`coin (i + 1)`" in the random input state), his neighbour's coin toss ("`coin i`" in the random input state), and whether or not he has paid ("`pays (i+1)`" in the high input state). Thus, cryptographer 0 looks at his own coin and that of cryptographer n in order to complete the ring. The HOL function modelling this stage of the protocol is defined inductively — first the base case of cryptographer 0 and then the step case of cryptographer $i + 1$.

[1] The situation is slightly more complex as some overhead is required to ensure that two "cryptographers" aren't trying to broadcast a message at the same time; a simplistic approach, allowing messages to be broadcast simultaneously, would result in garbled transmissions.

[2] A formal proof of anonymity for the case of arbitrary graphs is a difficult problem; mechanising such a proof would first require some amount of graph theory to be formalised.

Formalisation 45 (setting the cryptographers' announcements)**.**

```
set_announcements high low random n 0 s =
    if (s = "announces 0") then
        (high "pays 0") xor (random "coin 0") xor (random "coin n")
    else low s

set_announcements high low random n (SUC i) s =
    if (s = "announces (SUC i)") then
        (high "pays (SUC i)") xor
        (random "coin (SUC i)") xor (random "coin i")
    else (set_announcements high low random n i) s
```

where `high`*,* `low`*, and* `random` *are the input states,* `n + 1` *is the number of cryptographers, and* `i + 1` *is the index of the cryptographer whose announcement is being set. The variable* `s` *is a string-type variable name — the result of* `set_announcements high low random n i` *is an updated output state that takes a variable name* `s` *and returns its value in the updated state.*

5.4.2 Computing the outcome

Having updated the output state to include each cryptographer's announcement, the next step is to compute the result of the protocol: whether one of the cryptographers has paid or not. This is done by XOR-ing all of the cryptographers' announcements together. The HOL definition is formalised in two steps: the result is computed and then the variable "`result`" in the output state is updated to that value. It is easier to define this stage of the protocol in two steps in HOL, because it is more natural to define the XOR-ing of the announcements inductively.

Formalisation 46 (XOR-ing the cryptographers' announcements)**.**

```
XOR_announces low 0       =  low "announces 0"
XOR_announces low (SUC i) =  (low "announces (SUC i)") xor
                             (XOR_announces low i)
```

The HOL function `XOR_announces low n` computes the XOR of the cryptographers' announcements, assuming `low` is a state containing those announcements in variables "`announces 0`", . . . , "`announces n`" and there are `n + 1` cryptographers.

Formalisation 47 (setting the result of the protocol)**.**

```
compute_result low n s =
    if (s = "result") then XOR_announces low n else low s
```

The HOL function `compute_result` updates the output state such that the variable "`result`" contains the value computed by `XOR_announces`.

5.4.3 Putting it all together

The final step in formalising the dining cryptographers protocol is to compose the functions `set_announcements` and `compute_result` correctly to create the whole protocol. Having already defined the two stages of the protocol, this is a simple task, but some care is needed to ensure that the right indices are used for each of the stages. Since the protocol is only valid for three or more cryptographers, it has been parameterised such that the number of cryptographers can only be set to an appropriate value.

Formalisation 48 (the dining cryptographers protocol in HOL)**.**

```
dcprog (SUC (SUC (SUC n))) =
    (λ((high, low), random). compute_result
        (set_announcements high low random (SUC (SUC n)) (SUC(SUC n)))
        (SUC (SUC n))
```

The HOL function `dcprog (SUC (SUC SUC n))` defines the dining cryptographers protocol for `n + 3` cryptographers, where $0 \leq$ `n`. This function takes an input of type *bool prog_state* — a triple of high, low and random input states — and returns a low-security output state of type *bool state*.

5.4.4 Defining the distribution on input states

Having formalised a HOL definition of the DC protocol above, all that remains to be done before analysing the protocol is to formalise the distribution over combinations of high, low, and random input states. Assuming that the attacker has no additional clues, he will consider any of the cryptographers to be equally likely to have paid. Therefore, the high input states should be distributed evenly over each of the cryptographers paying. Similarly, the coins are meant to be fair, so any assignment of heads and tails to the coin flips in the random input state should be equally likely. There aren't any low inputs, so a default low input state can be specified. Since the distribution on input states is uniform over combinations from (finite) sets of valid states, the probability space characterising that distribution can be formalised using `unif_prog_space` from Chapter 4; this requires the sets of valid high, low, and random inputs to be defined in the theorem-prover.

Let's start by looking at the set of valid high-security input states. Assuming that there are `n` cryptographers and one of the cryptographers has paid, a valid high-security input state is a state for which one (and only one) of the variables "`pays 0`", ..., "`pays (n − 1)`" is set to be true; as a convention, all other variables are initialised to false. In the case that the NSA has paid, the convention will be to set "`pays n`" to be true and all other variables to be false.

The set of valid high-security input states is defined in two stages. First the case when one of the cryptographers has paid is formalised, followed by a parameterised definition including the case when the NSA has paid. Recall from Chapter 4 that it is useful to define the sets of valid inputs inductively, as this makes it easier to automatically compute the information-leakage of a program.

Formalisation 49 (valid high states when a cryptographer has paid)**.**

```
dc_high_states_set 0       =  {(λs. s = "pays 0")}
dc_high_states_set (SUC i) =  (λs. s = "pays (SUC i)")
                              INSERT (dc_high_states_set i)
```

Assuming that one of the cryptographers has paid, Formalisation 49 defines the set of valid high-security input states corresponding to the different possible assignments of the payer. The overall set of valid high-security input states, including the case when the NSA has paid, is formalised as follows.

Formalisation 50 (the set of valid high-security input states)**.**

```
dc_high_states nsapays (SUC (SUC n)) =
    if nsapays then {(λs. s = "pays (SUC (SUC n))")}
    else dc_high_states_set (SUC n)
```

Since there aren't any low-security inputs to the protocol, the set of valid low-security inputs can simply be defined to be a single default value; here the state mapping all variables to false has been chosen.

Formalisation 51 (the set of valid low-security input states)**.**

$$\texttt{dc\_low\_states} = \{(\lambda \texttt{s}.\ \bot)\}$$

Finally, the set of valid random inputs must be formalised. Assuming there are n cryptographers, this consists of the set of states with each of the possible assignments of the variables "`coin 0`", ..., "`coin (n − 1)`" and all other variables set to a default value; false has been chosen as the default value to maintain the convention in this chapter. The formalisation of this set of input states is defined much the same as for the high-security inputs.

Formalisation 52 (the set of valid random input states)**.**

```
dc_random_states 0 = {(λs. s = "coin 0"); (λs.⊥)}
dc_random_states (SUC i) =
    (IMAGE (λs. (λx. if x = "coin (SUC i)" then ⊤ else s x))
           (dc_random_states i)) UNION
    (IMAGE (λs. (λx. if x = "coin (SUC i)" then ⊥ else s x))
           (dc_random_states i))
```

The probability distribution on input states has been defined using a uniform distribution, specifically using the `unif_prog_space` described in Chapter 4. This ensures that the assumption of fair coins is modelled appropriately as well as the assumption that any of the cryptographers is considered equally likely to have paid. Having defined the sets of valid high, low, and random input states, formalising the probability space is a simple task, but care is required so that the appropriate indices are used for each definition.

Formalisation 53 (the distribution on input states)**.**

```
dc_prog_space (SUC (SUC n)) nsapays =
    unif_prog_space (dc_high_states nsapays (SUC (SUC n)))
                    dc_low_states
                    (dc_random_states (SUC n))
```

Formalisations 48 and 53, respectively defining the dining cryptographers protocol and the distribution on inputs, together characterise the HOL model of the DC protocol.

5.5 Proof of the dining cryptographers protocol in HOL4

Chaum's original presentation of the DC protocol [22] included informal, pen-and-paper proofs of total anonymity both from the perspective of an outside observer and from the perspective of one of the cryptographers. Since then, there have been numerous proofs of Chaum's result. Some of those extend his results to include different graph structures [18], while others mechanise the analysis using model-checking approaches [133] or demonstrate variations that exhibit partial leakage [40].

The proof outlined in Section 5.5.1 is the first machine-checked proof of the DC protocol for an unbounded number of cryptographers. It focuses on anonymity from the perspective of an outside observer and is based on the techniques developed earlier in this chapter and in Chapter 4. Previous machine-assisted proofs using model-checking have been limited to specific instances of the protocol, rather than general proofs.

In Section 5.5.2 we will see how the tools developed in Chapter 4 can be used to automate the proof of anonymity for small instances of the protocol. Anonymity against a corrupt cryptographer and variations on the protocol involving biased coins are examined in Section 5.5.3.

5.5.1 Interactive, parameterised proof of anonymity

Before undertaking any proofs about the anonymity properties of the DC protocol, basic correctness properties of the HOL model of the protocol must be proved.

Proofs of correctness properties

To begin, some basic properties of the sets of valid input states need to be formalised. Examples of such properties are the cardinalities of the sets and that they are finite and non-empty. The statements of those properties in HOL notation can be found in Appendix H and their proofs are straightforward. Since the sets of high and random input states were defined inductively, it is also useful to formalise theorems stating the membership criteria for those sets. In the case of the high-security inputs that theorem is formalised as follows:

```
∀n x. x IN (dc_high_states_set n) =
    ∃i. i ≤ n ∧ (x = (λs. s = "pays i")).
```

The theorem characterising the membership criteria for the set of random input states is very similar and can be found in Appendix H as `IN_dc_random_states`; each of those theorems can be proved easily by induction.

The next step is to formalise correctness properties of the HOL model of the protocol. Naturally, the first of these relate to the initial announcement-setting stage of the protocol, as captured by Formalisation 45. For example,

```
∀h l r n i. i ≤ (SUC (SUC n)) ⇒
    (set_announcements h l r (SUC (SUC n))) (SUC (SUC n)) "announces 0"
     = (h "pays 0") xor (r "coin 0") xor (r "coin (SUC (SUC n))"))
```

states that the updated output state returned by `set_announcements` maps the variable "`announces 0`" to the XOR of the values of "`coin 0`" and "`coin n+2`" from the random input state and the value of "`pays 0`" from the high input state, assuming there are `n+3` cryptographers. Other theorems capture that the value of variable "`anounces i`", for $0 < i \leq n$, is updated appropriately and all other variables remain mapped to the same value as in the low input state. Those theorems appear as `dc_set_announcements_result1` – `dc_set_announcements_result6` in Appendix H; again, the proofs are routine.

Moving on, the correctness of Formalisation 46, which XORs the cryptographers announcements to compute the result, was proved to behave as desired (`dc_XOR_announces_result1` – `dc_XOR_announces_result5`). The culmination of these proofs captures that the appropriate result of the protocol execution is stored in the output variable "`result`" and the rest of the output variables are unchanged by the final stage of the protocol (`dcprog_result1` – `dcprog_result6`). Emphasis has been placed on results for executions of the protocol when one of the cryptographers has paid; no anonymity guarantees are required when the NSA has paid, so that scenario is of less interest.

Proofs of key lemmas

With the necessary correctness properties of the protocol formalised, we can move on to the more interesting proof of total anonymity. The crux of that proof lies in two lemmas establishing the number of different outputs produced by the protocol and the number of random input states that can result in a particular output, given certain high and low input states. Let's look at the development of those results in some detail.

The first critical lemma states that the cardinality of the set of outputs produced by the protocol is 2^n, assuming there are $n+1$ cryptographers. An informal argument for this is as follows: Assuming one of the cryptographers has paid, the variable "`result`" in the output is set to true, the bitwise XOR of the variables "`announces 0`", ..., "`announces n`" in the output is also true, and other variables take the value false by default; these properties are all supported by the correctness results discussed earlier. Since the value of "`result`" is fixed, the only variation in the output states must come from the assignment of the "`announces 0`", ..., "`announces n`" variables. Elementary combinatorics tells us that there are 2^n ways that boolean-valued variables "`announces 0`", ..., "`announces n − 1`" can be assigned. Once those values are set, there is only one assignment of "`announces n`" that will result in the bitwise XOR of the announcements being true, namely the inverse of the bitwise XOR of "`announces 0`", ..., "`announces n − 1`".

The proof of this lemma proceeds in several stages. Firstly, the membership criteria for the set of valid output states described above must be formalised. This results in a proof that

```
∀n. (IMAGE (dcprog (SUC (SUC (SUC n))))
           ((dc_high_states ⊥ (SUC (SUC (SUC n))) CROSS dc_low_states)
            CROSSdc_random_states (SUC (SUC n))) =
     {s | (s "result" = XOR_announces s (SUC (SUC n)) ∧
          XOR_announces s (SUC (SUC n)) ∧
          ∀x. (x ≠ "result") ∧ (∀i. i ≤ (SUC (SUC n)) ⇒ (x ≠ "announces i"))
               ⇒ ¬ s x}.
```

The next step is to inductively define how the set of valid outputs can be constructed; the inductive definition of this set makes it much easier to prove its cardinality. Then an equivalence must be proved between the inductively defined set of outputs and the set of outputs produced by the protocol. Finally, these lemmas can be put together to prove a theorem stating the cardinality of the set of outputs.

HOL Theorem 14 (cardinality of the set of valid outputs)**.**

```
∀n. CARD (IMAGE (dcprog (SUC (SUC (SUC n))))
           ((dc_high_states ⊥ (SUC (SUC (SUC n))) CROSS dc_low_states) =
     2 pow (SUC (SUC n)).
```

The second critical lemma states that, for any valid output state and high and low input states, there are precisely two random input states that could produce that output. Informally, the argument is as follows: There is a choice between two ways to assign the value of the first "`coin`" variable in the random input state. Once, this coin's value has been set, the values of the adjacent coins are already determined

because they must XOR together with the value of the "pays" variables (whose values are already fixed in the high input state) to produce the right announcement from the output state. This assignment propagates to each coin in turn such that the values of all the coins are determined simply by fixing the value of one of the coins.

The first step in the HOL proof of this lemma is to formalise the conditions under which an assignment of the "coin" variables is valid with respect to a given high input state (i.e. selection of which cryptographer has paid) and output state (i.e. assignment of the "announces" variables). This property is defined inductively as follows.

Formalisation 54 (validity of a coin assignment w.r.t. a payer and output)**.**

```
valid_coin_assignment r out h n 0 =
    (r "coin 0″ =
     (r "coin (SUC (SUC n))) xor (XOR_announces out 0) xor (0 ≥ h))
valid_coin_assignment r out h n (SUC i) =
    (r "coin (SUC i)″ =
     (r "coin (SUC (SUC n))) xor (XOR_announces out (SUC i)) xor
     ((SUC i) ≥ h))
```

where h *is the index of the payer and there are* n + 3 *cryptographers.*

The inductive definition above is then proved equivalent to a more natural definition in set notation (valid_coin_set_eq_valid_coin_assignment in Appendix H). Next, a function is defined that produces a correct assignment of the coin variables for a given payer and output; the parameter choice below selects whether the first coin should be set to true or false in that assignment of the coins variables.

Formalisation 55 (generating a coin assignment given a payer and output)**.**

```
coin_assignment out h n choice 0 =
    (λs. if s = "coin 0" then
         choice xor (XOR_announces out 0) xor (0 ≥ h)
         else ⊥)
coin_assignment out h n choice (SUC i) =
    (λs. if s = "coin (SUC i)" then
         choice xor (XOR_announces out (SUC i)) xor ((SUC i) ≥ h)
         else coin_assignment out h n choice i s)
```

The set of valid random input states for a given payer and output is the two element set containing coin_assignment out p n ⊤ (SUC(SUC n)) and coin_assignment out p n ⊥ (SUC(SUC n)).

Formalisation 56 (a valid random input state given a payer and output)**.**

```
coin_assingment_set out p n =
    {coin_assignment out p n ⊤ (SUC(SUC n));
     coin_assignment out p n ⊥ (SUC(SUC n))}
```

Finally, the set generated by coin_assingment_set is proved equivalent to the set of random input states satisfying valid_coin_assignment, for a given output state and payer (valid_coin_assignment_eq_2_element_set in Appendix H). The number of random input states that can generate a given output from a given high input state follows directly from the lemmas and definitions above.

Proof of total anonymity

Using the correctness results mentioned earlier and the critical lemmas explained above, the most interesting aspects of the anonymity proof for the DC protocol are complete. The rest of proof relies upon lemmas relating to the unif_prog_space developed in Chapter 4 and properties of summations of real-valued functions over finite sets; these portions of the proof are straightforward, albeit requiring a fair amount of effort. We'll now examine a high-level outline of the anonymity proof as a whole.

The goal of the total-anonymity proof is that the there is no information leakage for the model of the protocol and distribution on inputs described above, whenever one of the cryptographers has paid. Formally,

$$\forall \mathtt{n}.\ \mathtt{leakage}\ (\mathtt{dc\_prog\_space}\ (\mathtt{n}+3)\ \bot)\ (\mathtt{dcprog}\ (\mathtt{n}+3) = 0.$$

The first step of the proof is to expand the definition of information leakage to a simplified form specifically for the `unif_prog_space` (using the lemma `unif_prog_space_leakage_computation_reduce` in Appendix G). After further simplification with lemmas about the cardinalities of the high, low, and random input sets ($\mathtt{n}+3$, $\mathtt{1}$, and $2^{\mathtt{n}+3}$ respecitively), the goal becomes

$$\begin{aligned}
&\tfrac{2^{\mathtt{n}+3}}{\mathtt{n}+3}\Big(\mathtt{SIGMA}\ (\lambda(\mathtt{out},\mathtt{h},\mathtt{l}).\ (\lambda\mathtt{x}.\ (\mathtt{x})\big(\mathtt{lg}\big(\tfrac{\mathtt{x}}{2^{\mathtt{n}+3}}\big)\big))\\
&\qquad\qquad (\mathtt{SIGMA}\ (\lambda\mathtt{r}.\ \mathtt{if\ dcprog}\ (\mathtt{n}+3)\ ((\mathtt{h},\mathtt{l}),\mathtt{r}) = \mathtt{out}\\
&\qquad\qquad\qquad \mathtt{then\ 1\ else\ 0})\\
&\qquad\qquad\qquad (\mathtt{dc_random_states}\ (\mathtt{n}+2))))\\
&\qquad (\mathtt{IMAGE}\ (\lambda\mathtt{s}.\ (\mathtt{dcprog}\ (\mathtt{n}+3)\ \mathtt{s}, \mathtt{FST\ s}))\\
&\qquad\qquad (\mathtt{dc_high_states_set}\ (\mathtt{n}+2) \times \mathtt{dc_low_states}\\
&\qquad\qquad \times \mathtt{dc_random_states}\ (\mathtt{n}+2)))-\\
&\mathtt{SIGMA}\ (\lambda(\mathtt{out},\mathtt{l}).\ (\lambda\mathtt{x}.\ \mathtt{x}\big(\mathtt{lg}\big(\tfrac{\mathtt{x}}{(\mathtt{n}+3)(2^{\mathtt{n}+3})}\big)\big))\\
&\qquad\qquad (\mathtt{SIGMA}\ (\lambda(\mathtt{h},\mathtt{r}).\ \mathtt{if\ dcprog}\ (\mathtt{n}+3)\ ((\mathtt{h},\mathtt{l}),\mathtt{r}) = \mathtt{out}\\
&\qquad\qquad\qquad \mathtt{then\ 1\ else\ 0})\\
&\qquad\qquad\qquad (\mathtt{dc_high_states_set}\ (\mathtt{n}+2)\\
&\qquad\qquad\qquad \times \mathtt{dc_random_states}\ (\mathtt{n}+2))))\\
&\qquad (\mathtt{IMAGE}\ (\lambda\mathtt{s}.\ (\mathtt{dcprog}\ (\mathtt{n}+3)\ \mathtt{s}, \mathtt{SND(FST\ s)}))\\
&\qquad\qquad (\mathtt{dc_high_states_set}\ (\mathtt{n}+2) \times \mathtt{dc_low_states}\\
&\qquad\qquad \times \mathtt{dc_random_states}\ (\mathtt{n}+2)))\Big) = 0,
\end{aligned}$$

where some mathematical notation has been used for easier reading. It is sufficient to prove that the two summations in the goal above are equal.

A close examination of the inner summation of the first sum reveals that it is equivalent to the cardinality of the set of random input states that produce output `out` with high input state `h` and low input state `l`. Recall from the critical lemmas discussed above that the cardinality of that set is 2, for any valid choice of `out`, `h`, and `l`. Thus, the first sum is reduced to

$$\mathtt{SIGMA}\ (\lambda(\mathtt{out},\mathtt{h},\mathtt{l}).\ (2)\big(\mathtt{lg}\big(\frac{2}{2^{\mathtt{n}+3}}\big)\big))\ \ldots$$

Since the function being summed is a constant value, the summation reduces to that constant times the cardinality of the set over which the function is being summed. The first sum then becomes

$$\begin{aligned}
&\Big((2)\big(\mathtt{lg}\big(\tfrac{2}{2^{\mathtt{n}+3}}\big)\big)\Big)(\mathtt{CARD}\ (\mathtt{IMAGE}\ (\lambda\mathtt{s}.\ (\mathtt{dcprog}\ (\mathtt{n}+3)\ \mathtt{s}, \mathtt{FST\ s}))\\
&\qquad\qquad (\mathtt{dc_high_states_set}\ (\mathtt{n}+2) \times \mathtt{dc_low_states}\\
&\qquad\qquad \times \mathtt{dc_random_states}\ (\mathtt{n}+2))).
\end{aligned}$$

The inner summation of the second sum in the goal is equivalent to the number of pairs of high and random input states that yield output `out` for low input `l`. Using the critical lemmas described above, there are $\mathtt{n}+3$ valid high input states and for each of these there are 2 random inputs that yield output `out`; thus, the inner summation is equal to $2^{\mathtt{n}+3}$. The second outer summation can now be reduced as was the first, because the function being summed is constant-valued. This results in the following goal:

$$\begin{aligned}
&\Big((2)\big(\mathtt{lg}\big(\tfrac{2}{2^{\mathtt{n}+3}}\big)\big)\Big)(\mathtt{CARD}\ (\mathtt{IMAGE}\ (\lambda\mathtt{s}.\ (\mathtt{dcprog}\ (\mathtt{n}+3)\ \mathtt{s}, \mathtt{FST\ s}))\\
&\qquad\qquad (\mathtt{dc_high_states_set}\ (\mathtt{n}+2) \times \mathtt{dc_low_states}\\
&\qquad\qquad \times \mathtt{dc_random_states}\ (\mathtt{n}+2))) =\\
&\Big((2^{\mathtt{n}+3})\big(\mathtt{lg}\big(\tfrac{1}{\mathtt{n}+3}\big)\big)\Big)(\mathtt{CARD}\ (\mathtt{IMAGE}\ (\lambda\mathtt{s}.\ (\mathtt{dcprog}\ (\mathtt{n}+3)\ \mathtt{s}, \mathtt{SND(FST\ s)}))\\
&\qquad\qquad (\mathtt{dc_high_states_set}\ (\mathtt{n}+2) \times \mathtt{dc_low_states}\\
&\qquad\qquad \times \mathtt{dc_random_states}\ (\mathtt{n}+2))).
\end{aligned}$$

The set on the left side of the equation above is the set of triples of the form $(\mathtt{out},\mathtt{h},\mathtt{l})$, where `out` is a valid output of the protocol produced by high and low inputs `h` and `l` and some valid random input. From the key lemmas above, we know that there are $2^{\mathtt{n}+2}$ possible outputs and that any of the $\mathtt{n}+3$ combinations of high and low inputs can produce any one of those outputs; thus, the cardinality of the set is $(\mathtt{n}+3)(2^{\mathtt{n}+2})$. The set on the right side of the equation above is the set of pairs of valid outputs

and low inputs producing those outputs. Since there is only one low input and it can produce any of the outputs, the cardinality of the set is simply that of the set of valid outputs: 2^{n+2}. The rest of the proof follows by arithmetic and properties of $\log_2$. Though the proof steps may seem simple, substantial effort is required for each step in order to manipulate the sets and summations involved using appropriate lemmas.

5.5.2 Automatic computation of leakage for finite instances

The automation developed in Chapter 4 for proving information-leakage has been used to automatically prove the anonymity of the DC protocol for the case of three cryptographers. That proof requires about 15 lines of script to define HOL conversions for unrolling the definitions of the input states and deciding equivalence between states. The system takes approximately 18 minutes to complete the proof. The efficiency of this automation certainly cannot compare to that of model-checking approaches and it probably won't scale to significantly larger instances of the protocol. Nevertheless it has its use. If one intends to prove the information leakage of some program using a theorem-prover, he might first want to gain some intuition by checking small instances of the program using a model-checker. The automation developed here provides this automatic computation of leakage using the model of the program already present in the theorem-prover; this reduces the effort required, since the program does not need to be remodelled.

5.5.3 Proofs of variations of the protocol

Having examined a general proof of the DC protocol for any number of participants, it is worth considering some variations on the protocol. Due to time constraints, the proofs of these variations are only for the case of three cryptographers, but they could be extended to larger cases or more general proofs. For the sake of brevity, these additional examples are not presented in detail, but the definitions and theorems can be found in Appendix H.

Proof of anonymity against an insider

The proof of the DC protocol explained above only considered an outside observer in the threat model. It might also be desirable to see if there is any leakage of anonymity given the knowledge of one of the cryptographers. This can easily be modelled by making one pair of coins (i.e. the knowledge of one of the cryptographers) a low-security input rather than a random input. Since the ring structure in the protocol is symmetric and the numbering of the cryptographers/coins is arbitrary, it is possible to pick coin 0 and coin n to be visible and this is equivalent to picking any of the other pairs of coins. The total anonymity of the protocol, even with knowledge of one of the coins, can be proved more-or-less automatically as described in Section 5.5.2.

Partial leakage through biased coins

Another variation on the protocol is to consider a corrupt cryptographer, who injects biased coins into the protocol in an attempt to gain information about who has paid. Deng et al. [40] have used the probabilistic model-checker PRISM to show that a partial loss of anonymity can result from the use of biased coins in the DC protocol. I have proved that no leakage occurs if the attacker is able to bias only one of the coins he sees, but a partial leakage occurs if he is able to bias the outcome of the coin he does not see. This demonstrates the applicability of the techniques in this text to analysing *partial* anonymity.

5.6 Related work and novel contributions

This section provides an overview of prominent work at the intersection of formal methods and anonymous communications. Particular focus will be placed on approaches providing tool support, but important pen-and-paper work will also be discussed; any tool-support for the methods following will be specifically mentioned. A great deal of work has been done to understand various systems for anonymous communications by informally reasoning through as many potential attacks as possible; while very fruitful, that

work will not be addressed here. An overview of anonymous communication in general can be found in Chapter 1.

5.6.1 Formal definitions of anonymity

One of the earliest formal treatments of anonymity properties was Schneider and Sidiropoulos's [133] trace-based approach using CSP. That work could not be used to reason about partial vulnerabilities, but could be applied to programs exhibiting nondeterministic behaviour. Their approach is also noteworthy because the analysis was mechanised using the model-checker FDR; however, the use of a model-checker limits its applicability to specific instances of systems. Schneider and Sidiropoulos illustrated their technique by applying it to the dining crytographers protocol for the cases of three and four cryptographers and to some variants of the protocol involving two-headed coins.

Syverson and Stubblebine were also forerunners in formalising anonymity, but used epistemic logic [148]. Their approach was also possibilistic rather than probabilistic and all-or-nothing rather than quantitative. They demonstrated their approach on a model of the Anonimizer web proxy. Recently, Garcia et al. [62] have also proposed an epistemic approach to formalising anonymity, defining the anonymity of a system in terms of observational equivalence between runs of the system. Similarly, their method is possibilistic rather than probabilistic and all-or-nothing rather than quantitative. They used the Crowds system and Onion Routing as examples for their approach.

In their description of the Crowds system, Reiter and Rubin [125] were among the first to observe that there can be varying levels of anonymity, which can be used to quantify the amount of anonymity provided by a system. Later, Pfitzmann and Köhntopp [120] attempted to standardise the terminology for those levels of anonymity and different metrics for measuring them.

Serjantov and Danezis [36, 134, 136] and Díaz et al. [45, 50] revolutionised the formalisation of anonymity by proposing the use of information-theoretic metrics to measure anonymity; specifically they suggested the use of entropy as a measure of anonymity.[3] Their work serves as inspiration for the information-theoretic approach taken in this text. Variants of their metrics have been suggested for specific scenarios and threat models [55, 150, 151]. Steinbrecher and Köpsell [144] and Berman et al. [9] have extended this line of work to include information-theoretic definitions for unlinkability in addition to anonymity; Franz et al. [61] have developed a definition of unlinkability that captures contextual information.

Recently, Chatzikokolakis [18, 19, 20] advanced the use of information-theoretic metrics for anonymity by examining a number of metrics other than entropy, such as mutual information, channel capacity, and Bayes risk. His work is closely related to that presented in this text, but was more concerned with investigating possible metrics for anonymity and efficient means of computing or approximating those metrics. Clauß and Schiffer [32] have investigated the selection of different information-theoretic metrics for entropy varying depending on threat model and whether the application layer or network layer is being considered. They provide a comparison of Rényi entropy and Shannon entropy as measures of anonymity and suggest models in which each is more appropriate. Recently, Smith [143] has also proposed the use of Rényi entropy rather than Shannon entropy for measuring information flow, when considering an attacker that makes only a single guess at the secret.

Hughes and Shmatikov [83] proposed a formalisation for anonymity based on the concept of a function view, representing partial knowledge of a function. Their approach is closely related to the epistemic approach of Syverson and Stubblebine and work involving observational equivalence. As with most epistemic approaches, their framework is limited to possibilistic rather than probabilistic definitions of anonymity.

Halpern and O'Neill [74] were the first to provide a formalisation of anonymity capable of dealing with probabilistic definitions. That advancement made it possible for them to formalise definitions for the varying levels of anonymity suggested by Reiter and Rubin, Pfitzmann and Köhntopp, and others. However, they did not consider any information-theoretic metrics for anonymity. Their approach was based on the runs and systems framework and they used the dining cryptographers protocol as a case study. Halpern and O'Neill [74] also provide an excellent comparison of their work with previous formalisations and strong motivation for the use of probabilistic rather than possibilistic definitions of anonymity.

[3] Serjantov and Danezis and Díaz et al. developed their definitions independently, but they were published at nearly the same. Their definitions of anonymity are nearly identical except for a normalisation factor.

Mauw et al. [102] developed a formal model of Onion Routing in a process-algebraic setting. They used a possibilistic metric for anonymity, defining anonymity in terms of trace equivalence. Also amongst process-algrebraic treatments is Bhargava and Palamidessi's [10] formalisation of the dining cryptographers protocol in the probabilistic π-calculus. The framework they developed is suitable for analysis of models involving both probabilistic and nondeterministic behaviour. Their analysis was by-hand, but they noted that a model-checker for the probabilistic π-calculus is under development. Deng et al. [41] have used a probabilistic version of the process algebra CCS to analyse the dining cryptographers protocol using an information-theoretic definition of anonymity. Their analysis was without tool support, but they intend to develop support via integration with the PRISM model-checker. Chothia [25] used the π-calculus with an approach based on bi-simulation to analyse the anonymous file-sharing system MUTE, which resulted in the identification of a bug. That work did not include any tool support, but he has shown how his approach can be automated using the μCRL tool [26].

Camenisch and Lysyanskaya [17] have developed a formalisation of anonymity in the Universally-Composable framework and demonstrated their approach by modelling Onion Routing; this approach allows for proofs of anonymity that are similar to how proofs for cryptographic protocols are often handled, reducing a given security property to a set of assumptions under which it is guaranteed to hold.

Morgan [109] has proposed a treatment of anonymity and privacy based on abstraction and refinement. He suggests an epistemic approach, modelling properties as Hoare triples. Morgan's approach uses a possibilistic rather than probabilistic definition of anonymity and he demonstrated his technique using the dining cryptographers protocol as a case study.

While not strictly "formal methods", there has been a sizeable amount of work devoted to developing general mathematical models of mix networks and their components. Díaz and Serjantov [49] developed a mathematical model for mixes, generalising all possible mixing strategies. That model allows for rigorous comparison of different mixing strategies [45, 46, 47, 48, 135]. Newman et al. [113] have developed a holistic model of mix networks for the purposes of analysing the affects of traffic analysis prevention mechanisms, such as dummy traffic. Their work focuses on the security of the network as a whole rather than from the viewpoint of a particular participant. Kesdogan et al. [88] have developed a model for mix networks involving a changing set of participants.

5.6.2 Tool-supported analysis of anonymity systems

Kawabe et al. [86] were the first to use theorem-proving in the anonymity domain and theirs is the only use in that domain apart from my own. They have taken a trace-based approach to formalising anonymity, modelling systems as I/O automata. The definition they propose for trace anonymity is based on the indistinguishability of traces in which different users have acted and is similar to observational equivalence. Kawabe et al. provide a pen-and-paper proof that trace anonymity is equivalent to the existence of a suitable forward/backward simulation; they then used the Larch Prover [2] in the IOA Toolkit [1] to prove the existence of appropriate simulations for some simple examples, thereby demonstrating their trace anonymity. Since then, they have extended their framework to include stronger threat models than a simple eavesdropper, such as a corrupt insider [87]. They demonstrated those extensions using the Crowds system as a case study.

When compared with my own work, a major drawback of Kawabe's initial approach to theorem-proving anonymity is that it cannot be applied to probabilistic systems. More recently, Hasuo and Kawabe [80] have proposed a probabilistic definition of trace anonymity, which can be applied to systems exhibiting probabilistic and nondeterministic behaviour, but cannot currently be applied to systems exhibiting both. That definition takes into account an attacker's a priori distribution on states (i.e. his initial suspicions) and is similar to Bhargava and Palamidessi's [10] definition of conditional anonymity. One difference between Hasuo and Kawabe's work and my own is that their definition for probabilistic anonymity is qualitative and mine is quantitative; by using information-theoretic metrics, my approach characterises the degree to which anonymity is preserved or violated, rather than solely stating whether anonymity is maintained. Appealing to coalgebraic theory, Hasuo and Kawabe proved that probabilistic trace anonymity is equivalent to the existence of an appropriate forward/backward probabilistic simulation and proved the anonymity of the dining cryptographers protocol (for the case of three cryptographers) by exhibiting such a simulation. Unfortunately, all of the work involving their probabilistic variant of trace anonymity was of the pen-and-paper variety; it is not clear if Kawabe's approach using the IOA Toolkit and Larch Prover can be used for their probabilistic definition of anonymity.

The work presented in this text and the efforts of Kawabe et al. [86, 87] constitute the only tool support for analysing anonymity properties using a theorem-prover. Most of the tool-supported applications of formal methods to anonymity systems make use of model-checkers. The earliest such example is Schneider and Sidiropoulos's [133] use of the model-checker FDR on their formalisation of the dining cryptographers protocol; that work was described in more detail in the previous subsection. One of the most noteworthy uses of model-checkers on anonymity systems was Shmatikov's [140, 141] use of the probabilistic model-checker PRISM on a model of the Crowds system; he identified an attack on the system that had not been previously noticed. Deng et al. [40] have used PRISM to analyse the dining cryptographers protocol and variations thereof involving biased coins for the case of three cryptographers. Van der Meyden and Su [152] have analysed the dining cryptographers protocol using symbolic model-checking techniques and an epistemic definition of anonymity; in contrast to the two previous approaches using PRISM, their definition of anonymity was possibilistic rather than probabilistic. Chothia et al. [26] used the μCRL tool to check the anonymity of the dining cryptographers protocol and the FOO 92 voting system; their analysis also used possibilistic rather than probabilistic metrics for anonymity and was based on bi-simulation rather than trace equivalence. One major innovation of that work was that the checking could be distributed over multiple machines, allowing them to check the dining cryptographers protocol for as many as 17 cryptographers, which was a record at that time. Finally, Legay et al. [91] have used the automated equivalence checker APEX to analyse the dining cryptographers protocol; this alternative to model-checking offers a huge performance benefit,[4] but the equivalence-based approach does not allow for a probabilistic definition of anonymity. There have also been a number of uses of ad hoc simulations to aid in understanding of various anonymous communications systems [15, 57, 142].

5.7 Summary

This chapter explained how anonymity guarantees can be proved in an interactive theorem-prover using the framework developed in Chapter 4. This approach builds from the observation that anonymity and privacy guarantees can be treated as information non-leakage properties. The usefulness of the techniques developed in this text was demonstrated by proving the total anonymity of the dining cryptographers protocol, which is a standard benchmark. This is the first machine-checked proof of the DC protocol for an unbounded number of participants. Proofs for small instances of the protocol can be generated automatically using the tools explained in Chapter 4. Finally, the applicability of those techniques to partial anonymity was demonstrated by applying them to variations on the DC protocol involving biased coins.

[4] Legay et al. [91] claim that, on the same system, PRISM takes over an hour to verify the case of 10 cryptographers while APEX can complete the case of 100 cryptographers in 25 seconds.

Chapter 6

Summary

"The temple bell stops
but I still hear the sound
coming out of the flowers"
– Bashō

The main purpose of this text was to demonstrate a framework for analysing and proving the privacy guarantees of communications systems. The overall approach was to treat privacy guarantees as information (non)-leakage properties, using the metric of conditional mutual information from Shannon's theory of information. That analysis was formalised within an interactive theorem-prover in order to provide a high assurance of mathematical and logic consistency for all proofs.

In order to take an information-theoretic approach to measuring information leakage, the theories of measure, probability, and information had first to be formalised in the theorem-prover. Though formalisations for measure and probability theories already existed in the HOL4 system, the formalisations presented in this text improve upon previous work; by using more general formalisations for probability and measure theories, quantities such as expectation can be defined generally, linking both the discrete and continuous cases under one unifying theory. That undertaking required the formalisation of Lebesgue integration, for which no prior formalisation existed in HOL4. Similarly, Chapter 3 presents the first formalisation of information theory in an interactive theorem-prover.

Chapter 4 showed how information leakage can be measured for programs modelled as HOL functions, using the formalisation of conditional mutual information developed earlier. A number of examples were used to illustrate the applicability of the framework to different threat models and to algorithms exhibiting partial information leakage. That work provides the first framework for proving information leakage guarantees using a theorem-prover and information-theoretic metrics capable of handling partial leakage. Tool support and automation were implemented to reduce the effort required to prove information-leakage properties.

Finally, the dining cryptographers protocol was used as a case study to illustrate the applicability of the framework to proving privacy guarantees of systems. The DC protocol serves as a classic benchmark for analysing anonymity and privacy properties. The proof explained in Chapter 5 is both the first machine-checked proof of the DC protocol for an unbounded number of cryptographers and the first application of an interactive theorem-prover in the privacy/anonymity domain for probabilistic systems and quantitative metrics. A number of variations on the protocol were discussed to illustrate different threat models and the framework's applicability to systems providing *partial* anonymity.

6.1 Future Work

As with any project there are countless small extensions and improvements I would have liked to make, but there simply wasn't enough time. Those present numerous opportunities for future improvements. More importantly, I see a number of potential areas for large-scale research projects building upon this work.

Beginning with the fundamentals, my new formalisations for probability, measure, and Lebesgue integration present a number of opportunities for formal methods work on probabilistic algorithms. Verifying

software and systems exhibiting random behaviour involving *continuous* distributions is an active area of research and Hasan et al. have already begun building on my formalisation of Lebesgue integration to verify expectation properties of typical continuous random variables; no reference is provided for that work because it is ongoing and has not yet been published. There is much work that can be done proceeding along that path. Another piece of work would be to formalise an equivalence between my formalisation of the Lebesgue integral and Harrison's formalisation [77] of the gauge integral.

Since Chapter 3 presents the first formalisation of Shannon's theory of information in an interactive theorem-prover, it too presents a range of opportunities for future work. While the definitions from Shannon's theory were formalised in Chapter 3 along with a number of useful lemmas, Shannon's coding theorems were not proved. Formalising a number of key results from Shannon's 1948 report [138] would extend the domain in which a theorem-prover can be used for formal analysis. For example, proofs about coding algorithms could then be performed; those could have various applications such as error-correcting codes and compression algorithms.

The framework for proving information leakage properties presented in Chapter 4 could also be applied to some more examples. At the same time, it would be useful to develop additional tool support. As was mentioned earlier, most encryption algorithms do not provide information-theoretically perfect security and as a result their behaviour is difficult to generalise in an information-theoretic setting. Proving the information-theoretic properties of specific encryption algorithms is a challenging area of open research; the work presented in this text could form a foundation for formalising information-theoretic proofs about encryption algorithms. Once properties of a particular encryption scheme are proved, information leakage could be analysed for systems making use of that encryption algorithm.

As discussed earlier, a number of different information-theoretic metrics for anonymity have been suggested in recent literature, each being most appropriate in a certain scenario. Some such examples are channel capacity [18], Bayes risk [21], and the use of Rényi entropy rather than Shannon entropy [143]. Formalising those metrics for anonymity should be a straightforward extension of the work presented here, but would provide a useful contribution for future analysis.

Finally, I demonstrated my approach to proving privacy and anonymity guarantees using the dining cryptographers protocol as a case study; however, the long term aim for this work is to analyse deployed systems for anonymous communications. With the groundwork done, it would be interesting to see what can be proved about deployed privacy-enhancing technologies like Crowds and Tor.

Appendix A

Glossary of HOL4 notation

HOL4 notation	Description	Mathematical notation
{}	*empty set*	$\emptyset$
[]	*nil/empty list*	
l1 ++ l2	*list concatenation*	
hd :: tl	*list construction*	
(f ∘ g) x	*function composition*	$f(g(x))$
BIGINTER S	*set intersection*	$\bigcap_{x \in S} x$
BIGUNION S	*set union*	$\bigcup_{x \in S} x$
COMPL S	*set complementation*	$\overline{S}$
R CROSS S	*Cartesian product of sets*	$R \times S$
R DIFF S	*set difference*	$R - S$ or $R \setminus S$
DISJOINT R S	*disjoint sets*	$R \cap S = \emptyset$
enumerate S	*enumeration of countable set*	
IMAGE f S	*function image*	$f[S] = \bigcup_{x \in S} f(x)$
x IN S	*set membership*	$x \in S$
x INSERT S	*set construction*	$\{x\} \cup S$
R INTER S	*set intersection*	$R \cap S$
logr b r	*logarithm*	$\log_b r$
POW S	*powerset*	$\mathcal{P}(S)$ or 2^S
PREIMAGE f S	*inverse-function image*	$f^{-1}[S] = \{x \mid f(x) \in S\}$
set l	*defines a set based on list* l	
R SUBSET S	*subset*	$R \subseteq S$
SIGMA f S	*finite summation*	$\sum_{x \in S} f(x)$
SUC n	*successor of a natural number*	$n + 1$
suminf f	*infinite summation*	$\lim_{n \to \infty} \sum_{i=0}^{n} f(i)$
f sums r	*infinite summation*	$\lim_{n \to \infty} \sum_{i=0}^{n} f(i) \to r$
sup S	*supremum of set*	$\sup S$
R UNION S	*set union*	$R \cup S$
(UNIV : α → bool)	*universal set*	U or $\{x : \alpha \mid \top\}$

Appendix B

measureTheory

```
[additive_def]  Definition

   |- !m.
        additive m =
        !s t.
          s IN measurable_sets m /\ t IN measurable_sets m /\
          DISJOINT s t ==>
          (measure m (s UNION t) = measure m s + measure m t)

[algebra_def]  Definition

   |- !a.
        algebra a =
        subset_class (space a) (subsets a) /\ {} IN subsets a /\
        (!s. s IN subsets a ==> space a DIFF s IN subsets a) /\
        !s t.
          s IN subsets a /\ t IN subsets a ==> s UNION t IN subsets a

[closed_cdi_def]  Definition

   |- !p.
        closed_cdi p =
        subset_class (space p) (subsets p) /\
        (!s. s IN subsets p ==> space p DIFF s IN subsets p) /\
        (!f.
           f IN (UNIV -> subsets p) /\ (f 0 = {}) /\
           (!n. f n SUBSET f (SUC n)) ==>
           BIGUNION (IMAGE f UNIV) IN subsets p) /\
        !f.
          f IN (UNIV -> subsets p) /\
          (!m n. ~(m = n) ==> DISJOINT (f m) (f n)) ==>
          BIGUNION (IMAGE f UNIV) IN subsets p

[countably_additive_def]  Definition

   |- !m.
        countably_additive m =
        !f.
          f IN (UNIV -> measurable_sets m) /\
          (!m n. ~(m = n) ==> DISJOINT (f m) (f n)) /\
          BIGUNION (IMAGE f UNIV) IN measurable_sets m ==>
```

```
          measure m o f sums measure m (BIGUNION (IMAGE f UNIV))

[countably_subadditive_def]  Definition

   |- !m.
        countably_subadditive m =
        !f.
          f IN (UNIV -> measurable_sets m) /\
          BIGUNION (IMAGE f UNIV) IN measurable_sets m /\
          summable (measure m o f) ==>
          measure m (BIGUNION (IMAGE f UNIV)) <= suminf (measure m o f)

[increasing_def]  Definition

   |- !m.
        increasing m =
        !s t.
          s IN measurable_sets m /\ t IN measurable_sets m /\
          s SUBSET t ==>
          measure m s <= measure m t

[indicator_fn_def]  Definition

   |- !s. indicator_fn s = (\x. (if x IN s then 1 else 0))

[inf_measure_def]  Definition

   |- !m s.
        inf_measure m s =
        inf
          {r |
           ?f.
             f IN (UNIV -> measurable_sets m) /\
             (!m n. ~(m = n) ==> DISJOINT (f m) (f n)) /\
             s SUBSET BIGUNION (IMAGE f UNIV) /\ measure m o f sums r}

[lambda_system_def]  Definition

   |- !gen lam.
        lambda_system gen lam =
        {l |
         l IN subsets gen /\
         !s.
           s IN subsets gen ==>
           (lam (l INTER s) + lam ((space gen DIFF l) INTER s) = lam s)}

[m_space_def]  Definition

   |- !sp sts mu. m_space (sp,sts,mu) = sp

[measurable_def]  Definition

   |- !a b.
        measurable a b =
        {f |
         sigma_algebra a /\ sigma_algebra b /\
```

```
          f IN (space a -> space b) /\
          !s. s IN subsets b ==> PREIMAGE f s INTER space a IN subsets a}

[measurable_sets_def]  Definition

   |- !sp sts mu. measurable_sets (sp,sts,mu) = sts

[measure_def]  Definition

   |- !sp sts mu. measure (sp,sts,mu) = mu

[measure_preserving_def]  Definition

   |- !m1 m2.
        measure_preserving m1 m2 =
        {f |
         f IN
         measurable (m_space m1,measurable_sets m1)
           (m_space m2,measurable_sets m2) /\ measure_space m1 /\
         measure_space m2 /\
         !s.
           s IN measurable_sets m2 ==>
           (measure m1 (PREIMAGE f s INTER m_space m1) = measure m2 s)}

[measure_space_def]  Definition

   |- !m.
        measure_space m =
        sigma_algebra (m_space m,measurable_sets m) /\ positive m /\
        countably_additive m

[outer_measure_space_def]  Definition

   |- !m.
        outer_measure_space m =
        positive m /\ increasing m /\ countably_subadditive m

[positive_def]  Definition

   |- !m.
        positive m =
        (measure m {} = 0) /\
        !s. s IN measurable_sets m ==> 0 <= measure m s

[sigma_algebra_def]  Definition

   |- !a.
        sigma_algebra a =
        algebra a /\
        !c.
          countable c /\ c SUBSET subsets a ==> BIGUNION c IN subsets a

[sigma_def]  Definition

   |- !sp st.
        sigma sp st =
```

```
        (sp,BIGINTER {s | st SUBSET s /\ sigma_algebra (sp,s)})

[smallest_closed_cdi_def]  Definition

   |- !a.
        smallest_closed_cdi a =
        (space a,
         BIGINTER {b | subsets a SUBSET b /\ closed_cdi (space a,b)})

[space_def]  Definition

   |- !x y. space (x,y) = x

[subadditive_def]  Definition

   |- !m.
        subadditive m =
        !s t.
          s IN measurable_sets m /\ t IN measurable_sets m ==>
          measure m (s UNION t) <= measure m s + measure m t

[subset_class_def]  Definition

   |- !sp sts. subset_class sp sts = !x. x IN sts ==> x SUBSET sp

[subsets_def]  Definition

   |- !x y. subsets (x,y) = y

[ADDITIVE]  Theorem

   |- !m s t u.
        additive m /\ s IN measurable_sets m /\
        t IN measurable_sets m /\ DISJOINT s t /\ (u = s UNION t) ==>
        (measure m u = measure m s + measure m t)

[ADDITIVE_INCREASING]  Theorem

   |- !m.
        algebra (m_space m,measurable_sets m) /\ positive m /\
        additive m ==>
        increasing m

[ADDITIVE_SUM]  Theorem

   |- !m f n.
        algebra (m_space m,measurable_sets m) /\ positive m /\
        additive m /\ f IN (UNIV -> measurable_sets m) /\
        (!m n. ~(m = n) ==> DISJOINT (f m) (f n)) ==>
        (sum (0,n) (measure m o f) =
         measure m (BIGUNION (IMAGE f (count n))))

[ALGEBRA_ALT_INTER]  Theorem

   |- !a.
        algebra a =
```

```
        subset_class (space a) (subsets a) /\ {} IN subsets a /\
        (!s. s IN subsets a ==> space a DIFF s IN subsets a) /\
        !s t.
          s IN subsets a /\ t IN subsets a ==> s INTER t IN subsets a

[ALGEBRA_COMPL]  Theorem

   |- !a s. algebra a /\ s IN subsets a ==> space a DIFF s IN subsets a

[ALGEBRA_DIFF]  Theorem

   |- !a s t.
        algebra a /\ s IN subsets a /\ t IN subsets a ==>
        s DIFF t IN subsets a

[ALGEBRA_EMPTY]  Theorem

   |- !a. algebra a ==> {} IN subsets a

[ALGEBRA_FINITE_UNION]  Theorem

   |- !a c.
        algebra a /\ FINITE c /\ c SUBSET subsets a ==>
        BIGUNION c IN subsets a

[ALGEBRA_INTER]  Theorem

   |- !a s t.
        algebra a /\ s IN subsets a /\ t IN subsets a ==>
        s INTER t IN subsets a

[ALGEBRA_INTER_SPACE]  Theorem

   |- !a s.
        algebra a /\ s IN subsets a ==>
        (space a INTER s = s) /\ (s INTER space a = s)

[ALGEBRA_SPACE]  Theorem

   |- !a. algebra a ==> space a IN subsets a

[ALGEBRA_SUBSET_LAMBDA_SYSTEM]  Theorem

   |- !m.
        algebra (m_space m,measurable_sets m) /\ positive m /\
        increasing m /\ additive m ==>
        measurable_sets m SUBSET
        lambda_system (m_space m,POW (m_space m)) (inf_measure m)

[ALGEBRA_UNION]  Theorem

   |- !a s t.
        algebra a /\ s IN subsets a /\ t IN subsets a ==>
        s UNION t IN subsets a

[CARATHEODORY]  Theorem
```

```
   |- !m0.
        algebra (m_space m0,measurable_sets m0) /\ positive m0 /\
        countably_additive m0 ==>
        ?m.
          (!s.
             s IN measurable_sets m0 ==>
             (measure m s = measure m0 s)) /\
          ((m_space m,measurable_sets m) =
           sigma (m_space m0) (measurable_sets m0)) /\ measure_space m

[CARATHEODORY_LEMMA]  Theorem

   |- !gsig lam.
        sigma_algebra gsig /\
        outer_measure_space (space gsig,subsets gsig,lam) ==>
        measure_space (space gsig,lambda_system gsig lam,lam)

[CLOSED_CDI_COMPL]  Theorem

   |- !p s.
        closed_cdi p /\ s IN subsets p ==> space p DIFF s IN subsets p

[CLOSED_CDI_DISJOINT]  Theorem

   |- !p f.
        closed_cdi p /\ f IN (UNIV -> subsets p) /\
        (!m n. ~(m = n) ==> DISJOINT (f m) (f n)) ==>
        BIGUNION (IMAGE f UNIV) IN subsets p

[CLOSED_CDI_DUNION]  Theorem

   |- !p s t.
        {} IN subsets p /\ closed_cdi p /\ s IN subsets p /\
        t IN subsets p /\ DISJOINT s t ==>
        s UNION t IN subsets p

[CLOSED_CDI_INCREASING]  Theorem

   |- !p f.
        closed_cdi p /\ f IN (UNIV -> subsets p) /\ (f 0 = {}) /\
        (!n. f n SUBSET f (SUC n)) ==>
        BIGUNION (IMAGE f UNIV) IN subsets p

[COUNTABLY_ADDITIVE]  Theorem

   |- !m s f.
        countably_additive m /\ f IN (UNIV -> measurable_sets m) /\
        (!m n. ~(m = n) ==> DISJOINT (f m) (f n)) /\
        (s = BIGUNION (IMAGE f UNIV)) /\ s IN measurable_sets m ==>
        measure m o f sums measure m s

[COUNTABLY_ADDITIVE_ADDITIVE]  Theorem

   |- !m.
        algebra (m_space m,measurable_sets m) /\ positive m /\
```

```
        countably_additive m ==>
        additive m

[COUNTABLY_SUBADDITIVE]  Theorem

   |- !m f s.
        countably_subadditive m /\ f IN (UNIV -> measurable_sets m) /\
        summable (measure m o f) /\ (s = BIGUNION (IMAGE f UNIV)) /\
        s IN measurable_sets m ==>
        measure m s <= suminf (measure m o f)

[COUNTABLY_SUBADDITIVE_SUBADDITIVE]  Theorem

   |- !m.
        algebra (m_space m,measurable_sets m) /\ positive m /\
        countably_subadditive m ==>
        subadditive m

[INCREASING]  Theorem

   |- !m s t.
        increasing m /\ s SUBSET t /\ s IN measurable_sets m /\
        t IN measurable_sets m ==>
        measure m s <= measure m t

[INCREASING_ADDITIVE_SUMMABLE]  Theorem

   |- !m f.
        algebra (m_space m,measurable_sets m) /\ positive m /\
        increasing m /\ additive m /\
        f IN (UNIV -> measurable_sets m) /\
        (!m n. ~(m = n) ==> DISJOINT (f m) (f n)) ==>
        summable (measure m o f)

[INF_MEASURE_AGREES]  Theorem

   |- !m s.
        algebra (m_space m,measurable_sets m) /\ positive m /\
        countably_additive m /\ s IN measurable_sets m ==>
        (inf_measure m s = measure m s)

[INF_MEASURE_CLOSE]  Theorem

   |- !m s e.
        algebra (m_space m,measurable_sets m) /\ positive m /\ 0 < e /\
        s SUBSET m_space m ==>
        ?f l.
          f IN (UNIV -> measurable_sets m) /\
          s SUBSET BIGUNION (IMAGE f UNIV) /\
          (!m n. ~(m = n) ==> DISJOINT (f m) (f n)) /\
          measure m o f sums l /\ l <= inf_measure m s + e

[INF_MEASURE_COUNTABLY_SUBADDITIVE]  Theorem

   |- !m.
        algebra (m_space m,measurable_sets m) /\ positive m /\
```

```
        increasing m ==>
        countably_subadditive (m_space m,POW (m_space m),inf_measure m)

[INF_MEASURE_EMPTY]  Theorem

   |- !m.
        algebra (m_space m,measurable_sets m) /\ positive m ==>
        (inf_measure m {} = 0)

[INF_MEASURE_INCREASING]  Theorem

   |- !m.
        algebra (m_space m,measurable_sets m) /\ positive m ==>
        increasing (m_space m,POW (m_space m),inf_measure m)

[INF_MEASURE_LE]  Theorem

   |- !m s x.
        algebra (m_space m,measurable_sets m) /\ positive m /\
        increasing m /\
        x IN
        {r |
         ?f.
           f IN (UNIV -> measurable_sets m) /\
           s SUBSET BIGUNION (IMAGE f UNIV) /\ measure m o f sums r} ==>
        inf_measure m s <= x

[INF_MEASURE_NONEMPTY]  Theorem

   |- !m g s.
        algebra (m_space m,measurable_sets m) /\ positive m /\
        s IN measurable_sets m /\ g SUBSET s ==>
        measure m s IN
        {r |
         ?f.
           f IN (UNIV -> measurable_sets m) /\
           (!m n. ~(m = n) ==> DISJOINT (f m) (f n)) /\
           g SUBSET BIGUNION (IMAGE f UNIV) /\ measure m o f sums r}

[INF_MEASURE_OUTER]  Theorem

   |- !m.
        algebra (m_space m,measurable_sets m) /\ positive m /\
        increasing m ==>
        outer_measure_space (m_space m,POW (m_space m),inf_measure m)

[INF_MEASURE_POS]  Theorem

   |- !m g.
        algebra (m_space m,measurable_sets m) /\ positive m /\
        g SUBSET m_space m ==>
        0 <= inf_measure m g

[INF_MEASURE_POS0]  Theorem

   |- !m g x.
```

```
        algebra (m_space m,measurable_sets m) /\ positive m /\
        x IN
        {r |
         ?f.
           f IN (UNIV -> measurable_sets m) /\
           (!m n. ~(m = n) ==> DISJOINT (f m) (f n)) /\
           g SUBSET BIGUNION (IMAGE f UNIV) /\ measure m o f sums r} ==>
        0 <= x

[INF_MEASURE_POSITIVE]  Theorem

   |- !m.
        algebra (m_space m,measurable_sets m) /\ positive m ==>
        positive (m_space m,POW (m_space m),inf_measure m)

[IN_MEASURABLE]  Theorem

   |- !a b f.
        f IN measurable a b =
        sigma_algebra a /\ sigma_algebra b /\
        f IN (space a -> space b) /\
        !s. s IN subsets b ==> PREIMAGE f s INTER space a IN subsets a

[IN_MEASURE_PRESERVING]  Theorem

   |- !m1 m2 f.
        f IN measure_preserving m1 m2 =
        f IN
        measurable (m_space m1,measurable_sets m1)
          (m_space m2,measurable_sets m2) /\ measure_space m1 /\
        measure_space m2 /\
        !s.
          s IN measurable_sets m2 ==>
          (measure m1 (PREIMAGE f s INTER m_space m1) = measure m2 s)

[IN_SIGMA]  Theorem

   |- !sp a x. x IN a ==> x IN subsets (sigma sp a)

[LAMBDA_SYSTEM_ADDITIVE]  Theorem

   |- !g0 lam l1 l2.
        algebra g0 /\ positive (space g0,subsets g0,lam) ==>
        additive (space g0,lambda_system g0 lam,lam)

[LAMBDA_SYSTEM_ALGEBRA]  Theorem

   |- !g0 lam.
        algebra g0 /\ positive (space g0,subsets g0,lam) ==>
        algebra (space g0,lambda_system g0 lam)

[LAMBDA_SYSTEM_CARATHEODORY]  Theorem

   |- !gsig lam.
        sigma_algebra gsig /\
        outer_measure_space (space gsig,subsets gsig,lam) ==>
```

```
        !f.
          f IN (UNIV -> lambda_system gsig lam) /\
          (!m n. ~(m = n) ==> DISJOINT (f m) (f n)) ==>
          BIGUNION (IMAGE f UNIV) IN lambda_system gsig lam /\
          lam o f sums lam (BIGUNION (IMAGE f UNIV))

[LAMBDA_SYSTEM_COMPL]  Theorem

   |- !g0 lam l.
        algebra g0 /\ positive (space g0,subsets g0,lam) /\
        l IN lambda_system g0 lam ==>
        space g0 DIFF l IN lambda_system g0 lam

[LAMBDA_SYSTEM_EMPTY]  Theorem

   |- !g0 lam.
        algebra g0 /\ positive (space g0,subsets g0,lam) ==>
        {} IN lambda_system g0 lam

[LAMBDA_SYSTEM_INCREASING]  Theorem

   |- !g0 lam.
        increasing (space g0,subsets g0,lam) ==>
        increasing (space g0,lambda_system g0 lam,lam)

[LAMBDA_SYSTEM_INTER]  Theorem

   |- !g0 lam l1 l2.
        algebra g0 /\ positive (space g0,subsets g0,lam) /\
        l1 IN lambda_system g0 lam /\ l2 IN lambda_system g0 lam ==>
        l1 INTER l2 IN lambda_system g0 lam

[LAMBDA_SYSTEM_POSITIVE]  Theorem

   |- !g0 lam.
        positive (space g0,subsets g0,lam) ==>
        positive (space g0,lambda_system g0 lam,lam)

[LAMBDA_SYSTEM_STRONG_ADDITIVE]  Theorem

   |- !g0 lam g l1 l2.
        algebra g0 /\ positive (space g0,subsets g0,lam) /\
        g IN subsets g0 /\ DISJOINT l1 l2 /\
        l1 IN lambda_system g0 lam /\ l2 IN lambda_system g0 lam ==>
        (lam ((l1 UNION l2) INTER g) =
         lam (l1 INTER g) + lam (l2 INTER g))

[LAMBDA_SYSTEM_STRONG_SUM]  Theorem

   |- !g0 lam g f n.
        algebra g0 /\ positive (space g0,subsets g0,lam) /\
        g IN subsets g0 /\ f IN (UNIV -> lambda_system g0 lam) /\
        (!m n. ~(m = n) ==> DISJOINT (f m) (f n)) ==>
        (sum (0,n) (lam o (\s. s INTER g) o f) =
         lam (BIGUNION (IMAGE f (count n)) INTER g))
```

```
[MEASUBABLE_BIGUNION_LEMMA]  Theorem

   |- !a b f.
        sigma_algebra a /\ sigma_algebra b /\
        f IN (space a -> space b) /\
        (!s. s IN subsets b ==> PREIMAGE f s IN subsets a) ==>
        !c.
          countable c /\ c SUBSET IMAGE (PREIMAGE f) (subsets b) ==>
          BIGUNION c IN IMAGE (PREIMAGE f) (subsets b)

[MEASURABLE_BIGUNION_PROPERTY]  Theorem

   |- !a b f.
        sigma_algebra a /\ sigma_algebra b /\
        f IN (space a -> space b) /\
        (!s. s IN subsets b ==> PREIMAGE f s IN subsets a) ==>
        !c.
          c SUBSET subsets b ==>
          (PREIMAGE f (BIGUNION c) = BIGUNION (IMAGE (PREIMAGE f) c))

[MEASURABLE_COMP]  Theorem

   |- !f g a b c.
        f IN measurable a b /\ g IN measurable b c ==>
        g o f IN measurable a c

[MEASURABLE_COMP_STRONG]  Theorem

   |- !f g a b c.
        f IN measurable a b /\ sigma_algebra c /\
        g IN (space b -> space c) /\
        (!x.
           x IN subsets c ==>
           PREIMAGE g x INTER IMAGE f (space a) IN subsets b) ==>
        g o f IN measurable a c

[MEASURABLE_COMP_STRONGER]  Theorem

   |- !f g a b c t.
        f IN measurable a b /\ sigma_algebra c /\
        g IN (space b -> space c) /\ IMAGE f (space a) SUBSET t /\
        (!s. s IN subsets c ==> PREIMAGE g s INTER t IN subsets b) ==>
        g o f IN measurable a c

[MEASURABLE_DIFF_PROPERTY]  Theorem

   |- !a b f.
        sigma_algebra a /\ sigma_algebra b /\
        f IN (space a -> space b) /\
        (!s. s IN subsets b ==> PREIMAGE f s IN subsets a) ==>
        !s.
          s IN subsets b ==>
          (PREIMAGE f (space b DIFF s) = space a DIFF PREIMAGE f s)

[MEASURABLE_I]  Theorem
```

```
   |- !a. sigma_algebra a ==> I IN measurable a a

[MEASURABLE_LIFT]  Theorem

   |- !f a b.
        f IN measurable a b ==>
        f IN measurable a (sigma (space b) (subsets b))

[MEASURABLE_POW_TO_POW]  Theorem

   |- !m f.
        measure_space m /\ (measurable_sets m = POW (m_space m)) ==>
        f IN measurable (m_space m,measurable_sets m) (UNIV,POW UNIV)

[MEASURABLE_POW_TO_POW_IMAGE]  Theorem

   |- !m f.
        measure_space m /\ (measurable_sets m = POW (m_space m)) ==>
        f IN
        measurable (m_space m,measurable_sets m)
          (IMAGE f (m_space m),POW (IMAGE f (m_space m)))

[MEASURABLE_PROD_SIGMA]  Theorem

   |- !a a1 a2 f.
        sigma_algebra a /\ FST o f IN measurable a a1 /\
        SND o f IN measurable a a2 ==>
        f IN
        measurable a
          (sigma (space a1 CROSS space a2)
             (prod_sets (subsets a1) (subsets a2)))

[MEASURABLE_RANGE_REDUCE]  Theorem

   |- !m f s.
        measure_space m /\
        f IN measurable (m_space m,measurable_sets m) (s,POW s) /\
        ~(s = {}) ==>
        f IN
        measurable (m_space m,measurable_sets m)
          (s INTER IMAGE f (m_space m),
           POW (s INTER IMAGE f (m_space m)))

[MEASURABLE_SETS_SUBSET_SPACE]  Theorem

   |- !m s.
        measure_space m /\ s IN measurable_sets m ==> s SUBSET m_space m

[MEASURABLE_SIGMA]  Theorem

   |- !f a b sp.
        sigma_algebra a /\ subset_class sp b /\ f IN (space a -> sp) /\
        (!s. s IN b ==> PREIMAGE f s INTER space a IN subsets a) ==>
        f IN measurable a (sigma sp b)

[MEASURABLE_SIGMA_PREIMAGES]  Theorem
```

```
   |- !a b f.
        sigma_algebra a /\ sigma_algebra b /\
        f IN (space a -> space b) /\
        (!s. s IN subsets b ==> PREIMAGE f s IN subsets a) ==>
        sigma_algebra (space a,IMAGE (PREIMAGE f) (subsets b))

[MEASURABLE_SUBSET]  Theorem

   |- !a b.
        measurable a b SUBSET measurable a (sigma (space b) (subsets b))

[MEASURABLE_UP_LIFT]  Theorem

   |- !sp a b c f.
        f IN measurable (sp,a) c /\ sigma_algebra (sp,b) /\
        a SUBSET b ==>
        f IN measurable (sp,b) c

[MEASURABLE_UP_SIGMA]  Theorem

   |- !a b.
        measurable a b SUBSET measurable (sigma (space a) (subsets a)) b

[MEASURABLE_UP_SUBSET]  Theorem

   |- !sp a b c.
        a SUBSET b /\ sigma_algebra (sp,b) ==>
        measurable (sp,a) c SUBSET measurable (sp,b) c

[MEASURE_ADDITIVE]  Theorem

   |- !m s t u.
        measure_space m /\ s IN measurable_sets m /\
        t IN measurable_sets m /\ DISJOINT s t /\ (u = s UNION t) ==>
        (measure m u = measure m s + measure m t)

[MEASURE_COMPL]  Theorem

   |- !m s.
        measure_space m /\ s IN measurable_sets m ==>
        (measure m (m_space m DIFF s) =
         measure m (m_space m) - measure m s)

[MEASURE_COUNTABLE_INCREASING]  Theorem

   |- !m s f.
        measure_space m /\ f IN (UNIV -> measurable_sets m) /\
        (f 0 = {}) /\ (!n. f n SUBSET f (SUC n)) /\
        (s = BIGUNION (IMAGE f UNIV)) ==>
        measure m o f --> measure m s

[MEASURE_COUNTABLY_ADDITIVE]  Theorem

   |- !m s f.
        measure_space m /\ f IN (UNIV -> measurable_sets m) /\
```

```
        (!m n. ~(m = n) ==> DISJOINT (f m) (f n)) /\
        (s = BIGUNION (IMAGE f UNIV)) ==>
        measure m o f sums measure m s

[MEASURE_DOWN]  Theorem

   |- !m0 m1.
        sigma_algebra (m_space m0,measurable_sets m0) /\
        measurable_sets m0 SUBSET measurable_sets m1 /\
        (measure m0 = measure m1) /\ measure_space m1 ==>
        measure_space m0

[MEASURE_EMPTY]  Theorem

   |- !m. measure_space m ==> (measure m {} = 0)

[MEASURE_PRESERVING_LIFT]  Theorem

   |- !m1 m2 a f.
        measure_space m1 /\ measure_space m2 /\
        (measurable_sets m2 = subsets (sigma (m_space m2) a)) /\
        f IN measure_preserving m1 (m_space m2,a,measure m2) ==>
        f IN measure_preserving m1 m2

[MEASURE_PRESERVING_SUBSET]  Theorem

   |- !m1 m2 a.
        measure_space m1 /\ measure_space m2 /\
        (measurable_sets m2 = subsets (sigma (m_space m2) a)) ==>
        measure_preserving m1 (m_space m2,a,measure m2) SUBSET
        measure_preserving m1 m2

[MEASURE_PRESERVING_UP_LIFT]  Theorem

   |- !m1 m2 f.
        measure_space m1 /\
        f IN measure_preserving (m_space m1,a,measure m1) m2 /\
        sigma_algebra (m_space m1,measurable_sets m1) /\
        a SUBSET measurable_sets m1 ==>
        f IN measure_preserving m1 m2

[MEASURE_PRESERVING_UP_SIGMA]  Theorem

   |- !m1 m2 a.
        measure_space m1 /\
        (measurable_sets m1 = subsets (sigma (m_space m1) a)) ==>
        measure_preserving (m_space m1,a,measure m1) m2 SUBSET
        measure_preserving m1 m2

[MEASURE_PRESERVING_UP_SUBSET]  Theorem

   |- !m1 m2.
        measure_space m1 /\ a SUBSET measurable_sets m1 /\
        sigma_algebra (m_space m1,measurable_sets m1) ==>
        measure_preserving (m_space m1,a,measure m1) m2 SUBSET
        measure_preserving m1 m2
```

```
[MEASURE_REAL_SUM_IMAGE]  Theorem

   |- !m s.
        measure_space m /\ s IN measurable_sets m /\
        (!x. x IN s ==> {x} IN measurable_sets m) /\ FINITE s ==>
        (measure m s = SIGMA (\x. measure m {x}) s)

[MEASURE_SPACE_ADDITIVE]  Theorem

   |- !m. measure_space m ==> additive m

[MEASURE_SPACE_REDUCE]  Theorem

   |- !m. (m_space m,measurable_sets m,measure m) = m

[MEASURE_SPACE_SUBSET]  Theorem

   |- !s s' m.
        s' SUBSET s /\ measure_space (s,POW s,m) ==>
        measure_space (s',POW s',m)

[MONOTONE_CONVERGENCE]  Theorem

   |- !m s f.
        measure_space m /\ f IN (UNIV -> measurable_sets m) /\
        (!n. f n SUBSET f (SUC n)) /\ (s = BIGUNION (IMAGE f UNIV)) ==>
        measure m o f --> measure m s

[OUTER_MEASURE_SPACE_POSITIVE]  Theorem

   |- !m. outer_measure_space m ==> positive m

[POW_ALGEBRA]  Theorem

   |- algebra (sp,POW sp)

[POW_SIGMA_ALGEBRA]  Theorem

   |- sigma_algebra (sp,POW sp)

[SIGMA_ALGEBRA]  Theorem

   |- !p.
        sigma_algebra p =
        subset_class (space p) (subsets p) /\ {} IN subsets p /\
        (!s. s IN subsets p ==> space p DIFF s IN subsets p) /\
        !c.
          countable c /\ c SUBSET subsets p ==> BIGUNION c IN subsets p

[SIGMA_ALGEBRA_ALGEBRA]  Theorem

   |- !a. sigma_algebra a ==> algebra a

[SIGMA_ALGEBRA_ALT]  Theorem
```

```
   |- !a.
        sigma_algebra a =
        algebra a /\
        !f.
          f IN (UNIV -> subsets a) ==>
          BIGUNION (IMAGE f UNIV) IN subsets a

[SIGMA_ALGEBRA_ALT_DISJOINT]  Theorem

   |- !a.
        sigma_algebra a =
        algebra a /\
        !f.
          f IN (UNIV -> subsets a) /\
          (!m n. ~(m = n) ==> DISJOINT (f m) (f n)) ==>
          BIGUNION (IMAGE f UNIV) IN subsets a

[SIGMA_ALGEBRA_ALT_MONO]  Theorem

   |- !a.
        sigma_algebra a =
        algebra a /\
        !f.
          f IN (UNIV -> subsets a) /\ (f 0 = {}) /\
          (!n. f n SUBSET f (SUC n)) ==>
          BIGUNION (IMAGE f UNIV) IN subsets a

[SIGMA_ALGEBRA_COUNTABLE_UNION]  Theorem

   |- !a c.
        sigma_algebra a /\ countable c /\ c SUBSET subsets a ==>
        BIGUNION c IN subsets a

[SIGMA_ALGEBRA_ENUM]  Theorem

   |- !a f.
        sigma_algebra a /\ f IN (UNIV -> subsets a) ==>
        BIGUNION (IMAGE f UNIV) IN subsets a

[SIGMA_ALGEBRA_FN]  Theorem

   |- !a.
        sigma_algebra a =
        subset_class (space a) (subsets a) /\ {} IN subsets a /\
        (!s. s IN subsets a ==> space a DIFF s IN subsets a) /\
        !f.
          f IN (UNIV -> subsets a) ==>
          BIGUNION (IMAGE f UNIV) IN subsets a

[SIGMA_ALGEBRA_FN_DISJOINT]  Theorem

   |- !a.
        sigma_algebra a =
        subset_class (space a) (subsets a) /\ {} IN subsets a /\
        (!s. s IN subsets a ==> space a DIFF s IN subsets a) /\
        (!s t.
```

```
           s IN subsets a /\ t IN subsets a ==>
           s UNION t IN subsets a) /\
         !f.
           f IN (UNIV -> subsets a) /\
           (!m n. ~(m = n) ==> DISJOINT (f m) (f n)) ==>
           BIGUNION (IMAGE f UNIV) IN subsets a
```

[SIGMA_ALGEBRA_SIGMA] Theorem

```
   |- !sp sts. subset_class sp sts ==> sigma_algebra (sigma sp sts)
```

[SIGMA_POW] Theorem

```
   |- !s. sigma s (POW s) = (s,POW s)
```

[SIGMA_PROPERTY] Theorem

```
   |- !sp p a.
        subset_class sp p /\ {} IN p /\ a SUBSET p /\
        (!s. s IN p INTER subsets (sigma sp a) ==> sp DIFF s IN p) /\
        (!c.
           countable c /\ c SUBSET p INTER subsets (sigma sp a) ==>
           BIGUNION c IN p) ==>
        subsets (sigma sp a) SUBSET p
```

[SIGMA_PROPERTY_ALT] Theorem

```
   |- !sp p a.
        subset_class sp p /\ {} IN p /\ a SUBSET p /\
        (!s. s IN p INTER subsets (sigma sp a) ==> sp DIFF s IN p) /\
        (!f.
           f IN (UNIV -> p INTER subsets (sigma sp a)) ==>
           BIGUNION (IMAGE f UNIV) IN p) ==>
        subsets (sigma sp a) SUBSET p
```

[SIGMA_PROPERTY_DISJOINT] Theorem

```
   |- !sp p a.
        algebra (sp,a) /\ a SUBSET p /\
        (!s. s IN p INTER subsets (sigma sp a) ==> sp DIFF s IN p) /\
        (!f.
           f IN (UNIV -> p INTER subsets (sigma sp a)) /\ (f 0 = {}) /\
           (!n. f n SUBSET f (SUC n)) ==>
           BIGUNION (IMAGE f UNIV) IN p) /\
        (!f.
           f IN (UNIV -> p INTER subsets (sigma sp a)) /\
           (!m n. ~(m = n) ==> DISJOINT (f m) (f n)) ==>
           BIGUNION (IMAGE f UNIV) IN p) ==>
        subsets (sigma sp a) SUBSET p
```

[SIGMA_PROPERTY_DISJOINT_LEMMA] Theorem

```
   |- !sp a p.
        algebra (sp,a) /\ a SUBSET p /\ closed_cdi (sp,p) ==>
        subsets (sigma sp a) SUBSET p
```

```
[SIGMA_PROPERTY_DISJOINT_LEMMA1]  Theorem

   |- !a.
        algebra a ==>
        !s t.
          s IN subsets a /\ t IN subsets (smallest_closed_cdi a) ==>
          s INTER t IN subsets (smallest_closed_cdi a)

[SIGMA_PROPERTY_DISJOINT_LEMMA2]  Theorem

   |- !a.
        algebra a ==>
        !s t.
          s IN subsets (smallest_closed_cdi a) /\
          t IN subsets (smallest_closed_cdi a) ==>
          s INTER t IN subsets (smallest_closed_cdi a)

[SIGMA_PROPERTY_DISJOINT_WEAK]  Theorem

   |- !sp p a.
        algebra (sp,a) /\ a SUBSET p /\ subset_class sp p /\
        (!s. s IN p ==> sp DIFF s IN p) /\
        (!f.
           f IN (UNIV -> p) /\ (f 0 = {}) /\
           (!n. f n SUBSET f (SUC n)) ==>
           BIGUNION (IMAGE f UNIV) IN p) /\
        (!f.
           f IN (UNIV -> p) /\
           (!m n. ~(m = n) ==> DISJOINT (f m) (f n)) ==>
           BIGUNION (IMAGE f UNIV) IN p) ==>
        subsets (sigma sp a) SUBSET p

[SIGMA_REDUCE]  Theorem

   |- !sp a. (sp,subsets (sigma sp a)) = sigma sp a

[SIGMA_SUBSET]  Theorem

   |- !a b.
        sigma_algebra b /\ a SUBSET subsets b ==>
        subsets (sigma (space b) a) SUBSET subsets b

[SIGMA_SUBSET_MEASURABLE_SETS]  Theorem

   |- !a m.
        measure_space m /\ a SUBSET measurable_sets m ==>
        subsets (sigma (m_space m) a) SUBSET measurable_sets m

[SIGMA_SUBSET_SUBSETS]  Theorem

   |- !sp a. a SUBSET subsets (sigma sp a)

[SMALLEST_CLOSED_CDI]  Theorem

   |- !a.
        algebra a ==>
```

```
        subsets a SUBSET subsets (smallest_closed_cdi a) /\
        closed_cdi (smallest_closed_cdi a) /\
        subset_class (space a) (subsets (smallest_closed_cdi a))

[SPACE]  Theorem

   |- !a. (space a,subsets a) = a

[SPACE_SIGMA]  Theorem

   |- !sp a. space (sigma sp a) = sp

[SPACE_SMALLEST_CLOSED_CDI]  Theorem

   |- !a. space (smallest_closed_cdi a) = space a

[STRONG_MEASURE_SPACE_SUBSET]  Theorem

   |- !s s'.
        s' SUBSET m_space s /\ measure_space s /\
        POW s' SUBSET measurable_sets s ==>
        measure_space (s',POW s',measure s)

[SUBADDITIVE]  Theorem

   |- !m s t u.
        subadditive m /\ s IN measurable_sets m /\
        t IN measurable_sets m /\ (u = s UNION t) ==>
        measure m u <= measure m s + measure m t

[UNIV_SIGMA_ALGEBRA]  Theorem

   |- sigma_algebra (UNIV,UNIV)

[finite_additivity_sufficient_for_finite_spaces]  Theorem

   |- !s m.
        sigma_algebra s /\ FINITE (space s) /\
        positive (space s,subsets s,m) /\
        additive (space s,subsets s,m) ==>
        measure_space (space s,subsets s,m)

[finite_additivity_sufficient_for_finite_spaces2]  Theorem

   |- !m.
        sigma_algebra (m_space m,measurable_sets m) /\
        FINITE (m_space m) /\ positive m /\ additive m ==>
        measure_space m
```

Appendix C

borelTheory

```
[measure] Parent theory of "borel"

[string] Parent theory of "borel"

[borel_measurable_def]  Definition

   |- !a. borel_measurable a = measurable a borel_space

[borel_space_def]  Definition

   |- borel_space = sigma UNIV (IMAGE (\a. {x | x <= a}) UNIV)

[mono_convergent_def]  Definition

   |- !u f s.
        mono_convergent u f s =
        (!x m n. m <= n /\ x IN s ==> u m x <= u n x) /\
        !x. x IN s ==> (\i. u i x) --> f x

[real_rat_set_def]  Definition

   |- real_rat_set =
      IMAGE (\(a,b). & a / & b) (UNIV CROSS UNIV) UNION
      IMAGE (\(a,b). ~(& a / & b)) (UNIV CROSS UNIV)

[IN_BOREL_MEASURABLE]  Theorem

   |- !f s.
        f IN borel_measurable s =
        sigma_algebra s /\
        !s'.
          s' IN subsets (sigma UNIV (IMAGE (\a. {x | x <= a}) UNIV)) ==>
          PREIMAGE f s' INTER space s IN subsets s

[NON_NEG_REAL_RAT_DENSE]  Theorem

   |- !x y. 0 <= x /\ x < y ==> ?m n. x < & m / & n /\ & m / & n < y

[REAL_RAT_DENSE]  Theorem

   |- !x y. x < y ==> ?i. i IN real_rat_set /\ x < i /\ i < y
```

```
[affine_borel_measurable]  Theorem

   |- !m g.
        measure_space m /\
        g IN borel_measurable (m_space m,measurable_sets m) ==>
        !a b.
          (\x. a + g x * b) IN
          borel_measurable (m_space m,measurable_sets m)

[borel_measurable_SIGMA_borel_measurable]  Theorem

   |- !m f s.
        measure_space m /\ FINITE s /\
        (!i.
           i IN s ==>
           f i IN borel_measurable (m_space m,measurable_sets m)) ==>
        (\x. SIGMA (\i. f i x) s) IN
        borel_measurable (m_space m,measurable_sets m)

[borel_measurable_const]  Theorem

   |- !s c. sigma_algebra s ==> (\x. c) IN borel_measurable s

[borel_measurable_eq_borel_measurable]  Theorem

   |- !m f g.
        measure_space m /\
        f IN borel_measurable (m_space m,measurable_sets m) /\
        g IN borel_measurable (m_space m,measurable_sets m) ==>
        {w | w IN m_space m /\ (f w = g w)} IN measurable_sets m

[borel_measurable_ge_iff]  Theorem

   |- !m.
        measure_space m ==>
        !f.
          f IN borel_measurable (m_space m,measurable_sets m) =
          !a. {w | w IN m_space m /\ a <= f w} IN measurable_sets m

[borel_measurable_gr_iff]  Theorem

   |- !m.
        measure_space m ==>
        !f.
          f IN borel_measurable (m_space m,measurable_sets m) =
          !a. {w | w IN m_space m /\ a < f w} IN measurable_sets m

[borel_measurable_indicator]  Theorem

   |- !s a.
        sigma_algebra s /\ a IN subsets s ==>
        indicator_fn a IN borel_measurable s

[borel_measurable_le_iff]  Theorem
```

```
   |- !m.
        measure_space m ==>
        !f.
          f IN borel_measurable (m_space m,measurable_sets m) =
          !a. {w | w IN m_space m /\ f w <= a} IN measurable_sets m

[borel_measurable_leq_borel_measurable]  Theorem

   |- !m f g.
        measure_space m /\
        f IN borel_measurable (m_space m,measurable_sets m) /\
        g IN borel_measurable (m_space m,measurable_sets m) ==>
        {w | w IN m_space m /\ f w <= g w} IN measurable_sets m

[borel_measurable_less_borel_measurable]  Theorem

   |- !m f g.
        measure_space m /\
        f IN borel_measurable (m_space m,measurable_sets m) /\
        g IN borel_measurable (m_space m,measurable_sets m) ==>
        {w | w IN m_space m /\ f w < g w} IN measurable_sets m

[borel_measurable_less_iff]  Theorem

   |- !m.
        measure_space m ==>
        !f.
          f IN borel_measurable (m_space m,measurable_sets m) =
          !a. {w | w IN m_space m /\ f w < a} IN measurable_sets m

[borel_measurable_neq_borel_measurable]  Theorem

   |- !m f g.
        measure_space m /\
        f IN borel_measurable (m_space m,measurable_sets m) /\
        g IN borel_measurable (m_space m,measurable_sets m) ==>
        {w | w IN m_space m /\ ~(f w = g w)} IN measurable_sets m

[borel_measurable_plus_borel_measurable]  Theorem

   |- !m f g.
        measure_space m /\
        f IN borel_measurable (m_space m,measurable_sets m) /\
        g IN borel_measurable (m_space m,measurable_sets m) ==>
        (\x. f x + g x) IN
        borel_measurable (m_space m,measurable_sets m)

[borel_measurable_square]  Theorem

   |- !m f g.
        measure_space m /\
        f IN borel_measurable (m_space m,measurable_sets m) ==>
        (\x. f x pow 2) IN
        borel_measurable (m_space m,measurable_sets m)

[borel_measurable_sub_borel_measurable]  Theorem
```

```
   |- !m f g.
        measure_space m /\
        f IN borel_measurable (m_space m,measurable_sets m) /\
        g IN borel_measurable (m_space m,measurable_sets m) ==>
        (\x. f x - g x) IN
        borel_measurable (m_space m,measurable_sets m)

[borel_measurable_times_borel_measurable]  Theorem

   |- !m f g.
        measure_space m /\
        f IN borel_measurable (m_space m,measurable_sets m) /\
        g IN borel_measurable (m_space m,measurable_sets m) ==>
        (\x. f x * g x) IN
        borel_measurable (m_space m,measurable_sets m)

[countable_real_rat_set]  Theorem

   |- countable real_rat_set

[mono_convergent_borel_measurable]  Theorem

   |- !u m f.
        measure_space m /\
        (!n. u n IN borel_measurable (m_space m,measurable_sets m)) /\
        mono_convergent u f (m_space m) ==>
        f IN borel_measurable (m_space m,measurable_sets m)

[sigma_algebra_borel_space]  Theorem

   |- sigma_algebra borel_space

[sigma_ge_gr]  Theorem

   |- !f A.
        sigma_algebra A /\
        (!a. {w | w IN space A /\ a <= f w} IN subsets A) ==>
        !a. {w | w IN space A /\ a < f w} IN subsets A

[sigma_gr_le]  Theorem

   |- !f A.
        sigma_algebra A /\
        (!a. {w | w IN space A /\ a < f w} IN subsets A) ==>
        !a. {w | w IN space A /\ f w <= a} IN subsets A

[sigma_le_less]  Theorem

   |- !f A.
        sigma_algebra A /\
        (!a. {w | w IN space A /\ f w <= a} IN subsets A) ==>
        !a. {w | w IN space A /\ f w < a} IN subsets A

[sigma_less_ge]  Theorem
```

```
|- !f A.
     sigma_algebra A /\
     (!a. {w | w IN space A /\ f w < a} IN subsets A) ==>
     !a. {w | w IN space A /\ a <= f w} IN subsets A
```

Appendix D

lebesgueTheory

```
[RN_deriv_def]  Definition

   |- !m v.
        RN_deriv m v =
        @f.
          measure_space m /\
          measure_space (m_space m,measurable_sets m,v) /\
          f IN borel_measurable (m_space m,measurable_sets m) /\
          !a.
            a IN measurable_sets m ==>
            (integral m (\x. f x * indicator_fn a x) = v a)

[countable_space_integral_def]  Definition

   |- !m f.
        countable_space_integral m f =
        (let e = enumerate (IMAGE f (m_space m)) in
           suminf
             ((\r. r * measure m (PREIMAGE f {r} INTER m_space m)) o e))

[finite_space_integral_def]  Definition

   |- !m f.
        finite_space_integral m f =
        SIGMA (\r. r * measure m (PREIMAGE f {r} INTER m_space m))
          (IMAGE f (m_space m))

[integrable_def]  Definition

   |- !m f.
        integrable m f =
        measure_space m /\ (?x. x IN nnfis m (pos_part f)) /\
        ?y. y IN nnfis m (neg_part f)

[integral_def]  Definition

   |- !m f.
        integral m f =
        (@i. i IN nnfis m (pos_part f)) - @j. j IN nnfis m (neg_part f)

[mon_upclose_def]  Definition
```

```
   |- !u m. mon_upclose u m = mon_upclose_help m u m

[mon_upclose_help_def]  Definition

   |- (!u m. mon_upclose_help 0 u m = u m 0) /\
      !n u m.
        mon_upclose_help (SUC n) u m =
        upclose (u m (SUC n)) (mon_upclose_help n u m)

[mono_increasing_def]  Definition

   |- !f. mono_increasing f = !m n. m <= n ==> f m <= f n

[neg_part_def]  Definition

   |- !f. neg_part f = (\x. (if 0 <= f x then 0 else ~f x))

[nnfis_def]  Definition

   |- !m f.
        nnfis m f =
        {y |
         ?u x.
           mono_convergent u f (m_space m) /\
           (!n. x n IN psfis m (u n)) /\ x --> y}

[nonneg_def]  Definition

   |- !f. nonneg f = !x. 0 <= f x

[pos_fn_integral_def]  Definition

   |- !m f.
        pos_fn_integral m f =
        sup {r | ?g. r IN psfis m g /\ !x. g x <= f x}

[pos_part_def]  Definition

   |- !f. pos_part f = (\x. (if 0 <= f x then f x else 0))

[pos_simple_fn_def]  Definition

   |- !m f s a x.
        pos_simple_fn m f s a x =
        (!t. 0 <= f t) /\
        (!t.
           t IN m_space m ==>
           (f t = SIGMA (\i. x i * indicator_fn (a i) t) s)) /\
        (!i. i IN s ==> a i IN measurable_sets m) /\ (!i. 0 <= x i) /\
        FINITE s /\
        (!i j. i IN s /\ j IN s /\ ~(i = j) ==> DISJOINT (a i) (a j)) /\
        (BIGUNION (IMAGE a s) = m_space m)

[pos_simple_fn_integral_def]  Definition
```

```
   |- !m s a x.
        pos_simple_fn_integral m s a x =
        SIGMA (\i. x i * measure m (a i)) s

[prod_measure_def]  Definition

   |- !m0 m1.
        prod_measure m0 m1 =
        (\a. integral m0 (\s0. measure m1 (PREIMAGE (\s1. (s0,s1)) a)))

[prod_measure_space_def]  Definition

   |- !m0 m1.
        prod_measure_space m0 m1 =
        (m_space m0 CROSS m_space m1,
         subsets
           (sigma (m_space m0 CROSS m_space m1)
              (prod_sets (measurable_sets m0) (measurable_sets m1))),
         prod_measure m0 m1)

[psfis_def]  Definition

   |- !m f.
        psfis m f =
        IMAGE (\(s,a,x). pos_simple_fn_integral m s a x) (psfs m f)

[psfs_def]  Definition

   |- !m f. psfs m f = {(s,a,x) | pos_simple_fn m f s a x}

[upclose_def]  Definition

   |- !f g. upclose f g = (\t. max (f t) (g t))

[IN_psfis]  Theorem

   |- !m r f.
        r IN psfis m f ==>
        ?s a x.
          pos_simple_fn m f s a x /\
          (r = pos_simple_fn_integral m s a x)

[borel_measurable_mon_conv_psfis]  Theorem

   |- !m f.
        measure_space m /\
        f IN borel_measurable (m_space m,measurable_sets m) /\
        (!t. t IN m_space m ==> 0 <= f t) ==>
        ?u x.
          mono_convergent u f (m_space m) /\ !n. x n IN psfis m (u n)

[countable_space_integral_reduce]  Theorem

   |- !m f p n.
        measure_space m /\ positive m /\
        f IN borel_measurable (m_space m,measurable_sets m) /\
```

```
        countable (IMAGE f (m_space m)) /\
        ~FINITE (IMAGE (pos_part f) (m_space m)) /\
        ~FINITE (IMAGE (neg_part f) (m_space m)) /\
        (\r.
           r * measure m (PREIMAGE (neg_part f) {r} INTER m_space m)) o
        enumerate (IMAGE (neg_part f) (m_space m)) sums n /\
        (\r.
           r * measure m (PREIMAGE (pos_part f) {r} INTER m_space m)) o
        enumerate (IMAGE (pos_part f) (m_space m)) sums p ==>
        (integral m f = p - n)

[finite_POW_RN_deriv_reduce]  Theorem

   |- !m v.
        measure_space m /\ FINITE (m_space m) /\
        measure_space (m_space m,measurable_sets m,v) /\
        (POW (m_space m) = measurable_sets m) /\
        (!x. (measure m {x} = 0) ==> (v {x} = 0)) ==>
        !x.
          x IN m_space m /\ ~(measure m {x} = 0) ==>
          (RN_deriv m v x = v {x} / measure m {x})

[finite_POW_prod_measure_reduce]  Theorem

   |- !m0 m1.
        measure_space m0 /\ measure_space m1 /\ FINITE (m_space m0) /\
        FINITE (m_space m1) /\
        (POW (m_space m0) = measurable_sets m0) /\
        (POW (m_space m1) = measurable_sets m1) ==>
        !a0 a1.
          a0 IN measurable_sets m0 /\ a1 IN measurable_sets m1 ==>
          (prod_measure m0 m1 (a0 CROSS a1) =
           measure m0 a0 * measure m1 a1)

[finite_integral_on_set]  Theorem

   |- !m f a.
        measure_space m /\ FINITE (m_space m) /\
        f IN borel_measurable (m_space m,measurable_sets m) /\
        a IN measurable_sets m ==>
        (integral m (\x. f x * indicator_fn a x) =
         SIGMA (\r. r * measure m (PREIMAGE f {r} INTER a)) (IMAGE f a))

[finite_space_POW_integral_reduce]  Theorem

   |- !m f.
        measure_space m /\ (POW (m_space m) = measurable_sets m) /\
        FINITE (m_space m) ==>
        (integral m f = SIGMA (\x. f x * measure m {x}) (m_space m))

[finite_space_integral_reduce]  Theorem

   |- !m f.
        measure_space m /\
        f IN borel_measurable (m_space m,measurable_sets m) /\
        FINITE (m_space m) ==>
```

```
        (integral m f = finite_space_integral m f)

[indicator_fn_split]  Theorem

   |- !r s b.
        FINITE r /\ (BIGUNION (IMAGE b r) = s) /\
        (!i j.
           i IN r /\ j IN r /\ ~(i = j) ==> DISJOINT (b i) (b j)) ==>
        !a.
          a SUBSET s ==>
          (indicator_fn a =
           (\x. SIGMA (\i. indicator_fn (a INTER b i) x) r))

[integral_add]  Theorem

   |- !m f g.
        integrable m f /\ integrable m g ==>
        integrable m (\t. f t + g t) /\
        (integral m (\t. f t + g t) = integral m f + integral m g)

[integral_borel_measurable]  Theorem

   |- !m f.
        integrable m f ==>
        f IN borel_measurable (m_space m,measurable_sets m)

[integral_cmul_indicator]  Theorem

   |- !m s c.
        measure_space m /\ s IN measurable_sets m ==>
        (integral m (\x. c * indicator_fn s x) = c * measure m s)

[integral_indicator_fn]  Theorem

   |- !m a.
        measure_space m /\ a IN measurable_sets m ==>
        (integral m (indicator_fn a) = measure m a) /\
        integrable m (indicator_fn a)

[integral_mono]  Theorem

   |- !m f g.
        integrable m f /\ integrable m g /\
        (!t. t IN m_space m ==> f t <= g t) ==>
        integral m f <= integral m g

[integral_times]  Theorem

   |- !m f a.
        integrable m f ==>
        integrable m (\t. a * f t) /\
        (integral m (\t. a * f t) = a * integral m f)

[integral_zero]  Theorem

   |- !m. measure_space m ==> (integral m (\x. 0) = 0)
```

```
[markov_ineq]  Theorem

   |- !m f a n.
        integrable m f /\ 0 < a /\
        integrable m (\x. abs (f x) pow n) ==>
        measure m (PREIMAGE f {y | a <= y} INTER m_space m) <=
        integral m (\x. abs (f x) pow n) / a pow n

[measure_space_finite_prod_measure_POW1]  Theorem

   |- !m0 m1.
        measure_space m0 /\ measure_space m1 /\ FINITE (m_space m0) /\
        FINITE (m_space m1) /\
        (POW (m_space m0) = measurable_sets m0) /\
        (POW (m_space m1) = measurable_sets m1) ==>
        measure_space (prod_measure_space m0 m1)

[measure_space_finite_prod_measure_POW2]  Theorem

   |- !m0 m1.
        measure_space m0 /\ measure_space m1 /\ FINITE (m_space m0) /\
        FINITE (m_space m1) /\
        (POW (m_space m0) = measurable_sets m0) /\
        (POW (m_space m1) = measurable_sets m1) ==>
        measure_space
          (m_space m0 CROSS m_space m1,
           POW (m_space m0 CROSS m_space m1),prod_measure m0 m1)

[measure_split]  Theorem

   |- !r b m.
        measure_space m /\ FINITE r /\
        (BIGUNION (IMAGE b r) = m_space m) /\
        (!i j. i IN r /\ j IN r /\ ~(i = j) ==> DISJOINT (b i) (b j)) /\
        (!i. i IN r ==> b i IN measurable_sets m) ==>
        !a.
          a IN measurable_sets m ==>
          (measure m a = SIGMA (\i. measure m (a INTER b i)) r)

[mon_upclose_psfis]  Theorem

   |- !m u.
        measure_space m /\ (!n n'. ?a. a IN psfis m (u n n')) ==>
        ?c. !n. c n IN psfis m (mon_upclose u n)

[mono_increasing_converges_to_sup]  Theorem

   |- !f r.
        (!i. 0 <= f i) /\ mono_increasing f /\ f --> r ==>
        (r = sup (IMAGE f UNIV))

[nnfis_add]  Theorem

   |- !f g m a b.
        measure_space m /\ a IN nnfis m f /\ b IN nnfis m g ==>
```

```
        a + b IN nnfis m (\w. f w + g w)

[nnfis_borel_measurable]  Theorem

   |- !m f a.
        measure_space m /\ a IN nnfis m f ==>
        f IN borel_measurable (m_space m,measurable_sets m)

[nnfis_dom_conv]  Theorem

   |- !m f g b.
        measure_space m /\
        f IN borel_measurable (m_space m,measurable_sets m) /\
        b IN nnfis m g /\ (!t. t IN m_space m ==> f t <= g t) /\
        (!t. t IN m_space m ==> 0 <= f t) ==>
        ?a. a IN nnfis m f /\ a <= b

[nnfis_integral]  Theorem

   |- !m f a.
        measure_space m /\ a IN nnfis m f ==>
        integrable m f /\ (integral m f = a)

[nnfis_minus_nnfis_integral]  Theorem

   |- !m f g a b.
        measure_space m /\ a IN nnfis m f /\ b IN nnfis m g ==>
        integrable m (\t. f t - g t) /\
        (integral m (\t. f t - g t) = a - b)

[nnfis_mon_conv]  Theorem

   |- !f m h x y.
        measure_space m /\ mono_convergent f h (m_space m) /\
        (!n. x n IN nnfis m (f n)) /\ x --> y ==>
        y IN nnfis m h

[nnfis_mono]  Theorem

   |- !f g m a b.
        measure_space m /\ a IN nnfis m f /\ b IN nnfis m g /\
        (!w. w IN m_space m ==> f w <= g w) ==>
        a <= b

[nnfis_pos_on_mspace]  Theorem

   |- !f m a.
        measure_space m /\ a IN nnfis m f ==>
        !x. x IN m_space m ==> 0 <= f x

[nnfis_times]  Theorem

   |- !f m a z.
        measure_space m /\ a IN nnfis m f /\ 0 <= z ==>
        z * a IN nnfis m (\w. z * f w)
```

```
[nnfis_unique]  Theorem

   |- !f m a b.
        measure_space m /\ a IN nnfis m f /\ b IN nnfis m f ==> (a = b)

[pos_part_neg_part_borel_measurable]  Theorem

   |- !f m.
        measure_space m /\
        f IN borel_measurable (m_space m,measurable_sets m) ==>
        pos_part f IN borel_measurable (m_space m,measurable_sets m) /\
        neg_part f IN borel_measurable (m_space m,measurable_sets m)

[pos_part_neg_part_borel_measurable_iff]  Theorem

   |- !f m.
        measure_space m ==>
        (f IN borel_measurable (m_space m,measurable_sets m) =
         pos_part f IN borel_measurable (m_space m,measurable_sets m) /\
         neg_part f IN borel_measurable (m_space m,measurable_sets m))

[pos_psfis]  Theorem

   |- !r m f. measure_space m /\ r IN psfis m f ==> 0 <= r

[pos_simple_fn_integral_REAL_SUM_IMAGE]  Theorem

   |- !m f s a x P.
        measure_space m /\
        (!i. i IN P ==> pos_simple_fn m (f i) (s i) (a i) (x i)) /\
        FINITE P ==>
        ?s' a' x'.
          pos_simple_fn m (\t. SIGMA (\i. f i t) P) s' a' x' /\
          (pos_simple_fn_integral m s' a' x' =
           SIGMA (\i. pos_simple_fn_integral m (s i) (a i) (x i)) P)

[pos_simple_fn_integral_add]  Theorem

   |- !m f s a x g s' b y.
        measure_space m /\ pos_simple_fn m f s a x /\
        pos_simple_fn m g s' b y ==>
        ?s'' c z.
          pos_simple_fn m (\x. f x + g x) s'' c z /\
          (pos_simple_fn_integral m s a x +
           pos_simple_fn_integral m s' b y =
           pos_simple_fn_integral m s'' c z)

[pos_simple_fn_integral_indicator]  Theorem

   |- !m A.
        measure_space m /\ A IN measurable_sets m ==>
        ?s a x.
          pos_simple_fn m (indicator_fn A) s a x /\
          (pos_simple_fn_integral m s a x = measure m A)

[pos_simple_fn_integral_mono]  Theorem
```

```
   |- !m f s a x g s' b y.
        measure_space m /\ pos_simple_fn m f s a x /\
        pos_simple_fn m g s' b y /\ (!x. f x <= g x) ==>
        pos_simple_fn_integral m s a x <=
        pos_simple_fn_integral m s' b y

[pos_simple_fn_integral_mono_on_mspace]  Theorem

   |- !m f s a x g s' b y.
        measure_space m /\ pos_simple_fn m f s a x /\
        pos_simple_fn m g s' b y /\
        (!x. x IN m_space m ==> f x <= g x) ==>
        pos_simple_fn_integral m s a x <=
        pos_simple_fn_integral m s' b y

[pos_simple_fn_integral_mult]  Theorem

   |- !m f s a x.
        measure_space m /\ pos_simple_fn m f s a x ==>
        !z.
          0 <= z ==>
          ?s' b y.
            pos_simple_fn m (\x. z * f x) s' b y /\
            (pos_simple_fn_integral m s' b y =
             z * pos_simple_fn_integral m s a x)

[pos_simple_fn_integral_present]  Theorem

   |- !m f s a x g s' b y.
        measure_space m /\ pos_simple_fn m f s a x /\
        pos_simple_fn m g s' b y ==>
        ?z z' c k.
          (!t.
             t IN m_space m ==>
             (f t = SIGMA (\i. z i * indicator_fn (c i) t) k)) /\
          (!t.
             t IN m_space m ==>
             (g t = SIGMA (\i. z' i * indicator_fn (c i) t) k)) /\
          (pos_simple_fn_integral m s a x =
           pos_simple_fn_integral m k c z) /\
          (pos_simple_fn_integral m s' b y =
           pos_simple_fn_integral m k c z') /\ FINITE k /\
          (!i j.
             i IN k /\ j IN k /\ ~(i = j) ==> DISJOINT (c i) (c j)) /\
          (!i. i IN k ==> c i IN measurable_sets m) /\
          (BIGUNION (IMAGE c k) = m_space m) /\ (!i. 0 <= z i) /\
          !i. 0 <= z' i

[pos_simple_fn_integral_unique]  Theorem

   |- !m f s a x s' b y.
        measure_space m /\ pos_simple_fn m f s a x /\
        pos_simple_fn m f s' b y ==>
        (pos_simple_fn_integral m s a x =
         pos_simple_fn_integral m s' b y)
```

```
[psfis_REAL_SUM_IMAGE]  Theorem

   |- !m f a P.
        measure_space m /\ (!i. i IN P ==> a i IN psfis m (f i)) /\
        FINITE P ==>
        SIGMA a P IN psfis m (\t. SIGMA (\i. f i t) P)

[psfis_add]  Theorem

   |- !m f g a b.
        measure_space m /\ a IN psfis m f /\ b IN psfis m g ==>
        a + b IN psfis m (\x. f x + g x)

[psfis_borel_measurable]  Theorem

   |- !m f a.
        measure_space m /\ a IN psfis m f ==>
        f IN borel_measurable (m_space m,measurable_sets m)

[psfis_equiv]  Theorem

   |- !f g a m.
        measure_space m /\ a IN psfis m f /\ (!t. 0 <= g t) /\
        (!t. t IN m_space m ==> (f t = g t)) ==>
        a IN psfis m g

[psfis_indicator]  Theorem

   |- !m A.
        measure_space m /\ A IN measurable_sets m ==>
        measure m A IN psfis m (indicator_fn A)

[psfis_intro]  Theorem

   |- !m a x P.
        measure_space m /\ (!i. i IN P ==> a i IN measurable_sets m) /\
        (!i. 0 <= x i) /\ FINITE P ==>
        SIGMA (\i. x i * measure m (a i)) P IN
        psfis m (\t. SIGMA (\i. x i * indicator_fn (a i) t) P)

[psfis_mono]  Theorem

   |- !m f g a b.
        measure_space m /\ a IN psfis m f /\ b IN psfis m g /\
        (!x. x IN m_space m ==> f x <= g x) ==>
        a <= b

[psfis_mono_conv_mono]  Theorem

   |- !m f u x y r s.
        measure_space m /\ mono_convergent u f (m_space m) /\
        (!n. x n IN psfis m (u n)) /\ (!m n. m <= n ==> x m <= x n) /\
        x --> y /\ r IN psfis m s /\
        (!a. a IN m_space m ==> s a <= f a) ==>
        r <= y
```

```
[psfis_mult]  Theorem

   |- !m f a.
        measure_space m /\ a IN psfis m f ==>
        !z. 0 <= z ==> z * a IN psfis m (\x. z * f x)

[psfis_nnfis]  Theorem

   |- !m f a. measure_space m /\ a IN psfis m f ==> a IN nnfis m f

[psfis_nonneg]  Theorem

   |- !m f a. a IN psfis m f ==> nonneg f

[psfis_present]  Theorem

   |- !m f g a b.
        measure_space m /\ a IN psfis m f /\ b IN psfis m g ==>
        ?z z' c k.
          (!t.
             t IN m_space m ==>
             (f t = SIGMA (\i. z i * indicator_fn (c i) t) k)) /\
          (!t.
             t IN m_space m ==>
             (g t = SIGMA (\i. z' i * indicator_fn (c i) t) k)) /\
          (a = pos_simple_fn_integral m k c z) /\
          (b = pos_simple_fn_integral m k c z') /\ FINITE k /\
          (!i j.
             i IN k /\ j IN k /\ ~(i = j) ==> DISJOINT (c i) (c j)) /\
          (!i. i IN k ==> c i IN measurable_sets m) /\
          (BIGUNION (IMAGE c k) = m_space m) /\ (!i. 0 <= z i) /\
          !i. 0 <= z' i

[psfis_unique]  Theorem

   |- !m f a b.
        measure_space m /\ a IN psfis m f /\ b IN psfis m f ==> (a = b)

[upclose_psfis]  Theorem

   |- !f g a b m.
        measure_space m /\ a IN psfis m f /\ b IN psfis m g ==>
        ?c. c IN psfis m (upclose f g)
```

Appendix E

probabilityTheory

```
[conditional_expectation_def]  Definition

   |- !p X s.
        conditional_expectation p X s =
        @f.
          real_random_variable f p /\
          !g.
            g IN s ==>
            (integral p (\x. f x * indicator_fn g x) =
             integral p (\x. X x * indicator_fn g x))

[conditional_prob_def]  Definition

   |- !p e1 e2.
        conditional_prob p e1 e2 =
        conditional_expectation p (indicator_fn e1) e2

[distribution_def]  Definition

   |- !p X.
        distribution p X = (\s. prob p (PREIMAGE X s INTER p_space p))

[events_def]  Definition

   |- events = measurable_sets

[expectation_def]  Definition

   |- expectation = integral

[indep_def]  Definition

   |- !p a b.
        indep p a b =
        a IN events p /\ b IN events p /\
        (prob p (a INTER b) = prob p a * prob p b)

[indep_families_def]  Definition

   |- !p q r.
        indep_families p q r = !s t. s IN q /\ t IN r ==> indep p s t
```

```
[joint_distribution_def]  Definition

   |- !p X Y.
        joint_distribution p X Y =
        (\a. prob p (PREIMAGE (\x. (X x,Y x)) a INTER p_space p))

[p_space_def]  Definition

   |- p_space = m_space

[possibly_def]  Definition

   |- !p e. possibly p e = e IN events p /\ ~(prob p e = 0)

[prob_def]  Definition

   |- prob = measure

[prob_preserving_def]  Definition

   |- prob_preserving = measure_preserving

[prob_space_def]  Definition

   |- !p. prob_space p = measure_space p /\ (measure p (p_space p) = 1)

[probably_def]  Definition

   |- !p e. probably p e = e IN events p /\ (prob p e = 1)

[random_variable_def]  Definition

   |- !X p s.
        random_variable X p s =
        prob_space p /\ X IN measurable (p_space p,events p) s

[real_random_variable_def]  Definition

   |- !X p.
        real_random_variable X p =
        prob_space p /\ X IN borel_measurable (p_space p,events p)

[rv_conditional_expectation_def]  Definition

   |- !p s X Y.
        rv_conditional_expectation p s X Y =
        conditional_expectation p X
          (IMAGE (\a. PREIMAGE Y a INTER p_space p) (subsets s))

[ABS_1_MINUS_PROB]  Theorem

   |- !p s.
        prob_space p /\ s IN events p /\ ~(prob p s = 0) ==>
        abs (1 - prob p s) < 1
```

```
[ABS_PROB]  Theorem

   |- !p s.
        prob_space p /\ s IN events p ==> (abs (prob p s) = prob p s)

[ADDITIVE_PROB]  Theorem

   |- !p.
        additive p =
        !s t.
          s IN events p /\ t IN events p /\ DISJOINT s t ==>
          (prob p (s UNION t) = prob p s + prob p t)

[COUNTABLY_ADDITIVE_PROB]  Theorem

   |- !p.
        countably_additive p =
        !f.
          f IN (UNIV -> events p) /\
          (!m n. ~(m = n) ==> DISJOINT (f m) (f n)) /\
          BIGUNION (IMAGE f UNIV) IN events p ==>
          prob p o f sums prob p (BIGUNION (IMAGE f UNIV))

[EVENTS]  Theorem

   |- !a b c. events (a,b,c) = b

[EVENTS_ALGEBRA]  Theorem

   |- !p. prob_space p ==> algebra (p_space p,events p)

[EVENTS_COMPL]  Theorem

   |- !p s.
        prob_space p /\ s IN events p ==> p_space p DIFF s IN events p

[EVENTS_COUNTABLE_INTER]  Theorem

   |- !p c.
        prob_space p /\ c SUBSET events p /\ countable c /\
        ~(c = {}) ==>
        BIGINTER c IN events p

[EVENTS_COUNTABLE_UNION]  Theorem

   |- !p c.
        prob_space p /\ c SUBSET events p /\ countable c ==>
        BIGUNION c IN events p

[EVENTS_DIFF]  Theorem

   |- !p s t.
        prob_space p /\ s IN events p /\ t IN events p ==>
        s DIFF t IN events p

[EVENTS_EMPTY]  Theorem
```

```
   |- !p. prob_space p ==> {} IN events p

[EVENTS_INTER]  Theorem

   |- !p s t.
        prob_space p /\ s IN events p /\ t IN events p ==>
        s INTER t IN events p

[EVENTS_SIGMA_ALGEBRA]  Theorem

   |- !p. prob_space p ==> sigma_algebra (p_space p,events p)

[EVENTS_SPACE]  Theorem

   |- !p. prob_space p ==> p_space p IN events p

[EVENTS_UNION]  Theorem

   |- !p s t.
        prob_space p /\ s IN events p /\ t IN events p ==>
        s UNION t IN events p

[INCREASING_PROB]  Theorem

   |- !p.
        increasing p =
        !s t.
          s IN events p /\ t IN events p /\ s SUBSET t ==>
          prob p s <= prob p t

[INDEP_EMPTY]  Theorem

   |- !p s. prob_space p /\ s IN events p ==> indep p {} s

[INDEP_REFL]  Theorem

   |- !p a.
        prob_space p /\ a IN events p ==>
        (indep p a a = (prob p a = 0) \/ (prob p a = 1))

[INDEP_SPACE]  Theorem

   |- !p s. prob_space p /\ s IN events p ==> indep p (p_space p) s

[INDEP_SYM]  Theorem

   |- !p a b. prob_space p /\ indep p a b ==> indep p b a

[INTER_PSPACE]  Theorem

   |- !p s. prob_space p /\ s IN events p ==> (p_space p INTER s = s)

[POSITIVE_PROB]  Theorem

   |- !p.
```

```
        positive p =
        (prob p {} = 0) /\ !s. s IN events p ==> 0 <= prob p s

[PROB]  Theorem

   |- !a b c. prob (a,b,c) = c

[PROB_ADDITIVE]  Theorem

   |- !p s t u.
        prob_space p /\ s IN events p /\ t IN events p /\
        DISJOINT s t /\ (u = s UNION t) ==>
        (prob p u = prob p s + prob p t)

[PROB_COMPL]  Theorem

   |- !p s.
        prob_space p /\ s IN events p ==>
        (prob p (p_space p DIFF s) = 1 - prob p s)

[PROB_COMPL_LE1]  Theorem

   |- !p s r.
        prob_space p /\ s IN events p ==>
        (prob p (p_space p DIFF s) <= r = 1 - r <= prob p s)

[PROB_COUNTABLY_ADDITIVE]  Theorem

   |- !p s f.
        prob_space p /\ f IN (UNIV -> events p) /\
        (!m n. ~(m = n) ==> DISJOINT (f m) (f n)) /\
        (s = BIGUNION (IMAGE f UNIV)) ==>
        prob p o f sums prob p s

[PROB_COUNTABLY_SUBADDITIVE]  Theorem

   |- !p f.
        prob_space p /\ IMAGE f UNIV SUBSET events p /\
        summable (prob p o f) ==>
        prob p (BIGUNION (IMAGE f UNIV)) <= suminf (prob p o f)

[PROB_COUNTABLY_ZERO]  Theorem

   |- !p c.
        prob_space p /\ countable c /\ c SUBSET events p /\
        (!x. x IN c ==> (prob p x = 0)) ==>
        (prob p (BIGUNION c) = 0)

[PROB_EMPTY]  Theorem

   |- !p. prob_space p ==> (prob p {} = 0)

[PROB_EQUIPROBABLE_FINITE_UNIONS]  Theorem

   |- !p s.
        prob_space p /\ s IN events p /\
```

```
        (!x. x IN s ==> {x} IN events p) /\ FINITE s /\
        (!x y. x IN s /\ y IN s ==> (prob p {x} = prob p {y})) ==>
        (prob p s = & (CARD s) * prob p {CHOICE s})

[PROB_EQ_BIGUNION_IMAGE]  Theorem

   |- !p.
        prob_space p /\ f IN (UNIV -> events p) /\
        g IN (UNIV -> events p) /\
        (!m n. ~(m = n) ==> DISJOINT (f m) (f n)) /\
        (!m n. ~(m = n) ==> DISJOINT (g m) (g n)) /\
        (!n. prob p (f n) = prob p (g n)) ==>
        (prob p (BIGUNION (IMAGE f UNIV)) =
         prob p (BIGUNION (IMAGE g UNIV)))

[PROB_EQ_COMPL]  Theorem

   |- !p s t.
        prob_space p /\ s IN events p /\ t IN events p /\
        (prob p (p_space p DIFF s) = prob p (p_space p DIFF t)) ==>
        (prob p s = prob p t)

[PROB_FINITELY_ADDITIVE]  Theorem

   |- !p s f n.
        prob_space p /\ f IN (count n -> events p) /\
        (!a b. a < n /\ b < n /\ ~(a = b) ==> DISJOINT (f a) (f b)) /\
        (s = BIGUNION (IMAGE f (count n))) ==>
        (sum (0,n) (prob p o f) = prob p s)

[PROB_INCREASING]  Theorem

   |- !p s t.
        prob_space p /\ s IN events p /\ t IN events p /\ s SUBSET t ==>
        prob p s <= prob p t

[PROB_INCREASING_UNION]  Theorem

   |- !p s f.
        prob_space p /\ f IN (UNIV -> events p) /\
        (!n. f n SUBSET f (SUC n)) /\ (s = BIGUNION (IMAGE f UNIV)) ==>
        prob p o f --> prob p s

[PROB_INDEP]  Theorem

   |- !p s t u.
        indep p s t /\ (u = s INTER t) ==>
        (prob p u = prob p s * prob p t)

[PROB_LE_1]  Theorem

   |- !p s. prob_space p /\ s IN events p ==> prob p s <= 1

[PROB_ONE_INTER]  Theorem

   |- !p s t.
```

```
        prob_space p /\ s IN events p /\ t IN events p /\
        (prob p t = 1) ==>
        (prob p (s INTER t) = prob p s)

[PROB_POSITIVE]  Theorem

   |- !p s. prob_space p /\ s IN events p ==> 0 <= prob p s

[PROB_PRESERVING]  Theorem

   |- !p1 p2.
        prob_preserving p1 p2 =
        {f |
         f IN
         measurable (p_space p1,events p1) (p_space p2,events p2) /\
         measure_space p1 /\ measure_space p2 /\
         !s.
           s IN events p2 ==>
           (prob p1 (PREIMAGE f s INTER p_space p1) = prob p2 s)}

[PROB_PRESERVING_LIFT]  Theorem

   |- !p1 p2 a f.
        prob_space p1 /\ prob_space p2 /\
        (events p2 = subsets (sigma (m_space p2) a)) /\
        f IN prob_preserving p1 (m_space p2,a,prob p2) ==>
        f IN prob_preserving p1 p2

[PROB_PRESERVING_SUBSET]  Theorem

   |- !p1 p2 a.
        prob_space p1 /\ prob_space p2 /\
        (events p2 = subsets (sigma (p_space p2) a)) ==>
        prob_preserving p1 (p_space p2,a,prob p2) SUBSET
        prob_preserving p1 p2

[PROB_PRESERVING_UP_LIFT]  Theorem

   |- !p1 p2 f.
        prob_space p1 /\
        f IN prob_preserving (p_space p1,a,prob p1) p2 /\
        sigma_algebra (p_space p1,events p1) /\ a SUBSET events p1 ==>
        f IN prob_preserving p1 p2

[PROB_PRESERVING_UP_SIGMA]  Theorem

   |- !p1 p2 a.
        prob_space p1 /\
        (events p1 = subsets (sigma (p_space p1) a)) ==>
        prob_preserving (p_space p1,a,prob p1) p2 SUBSET
        prob_preserving p1 p2

[PROB_PRESERVING_UP_SUBSET]  Theorem

   |- !p1 p2.
        prob_space p1 /\ a SUBSET events p1 /\
```

```
        sigma_algebra (p_space p1,events p1) ==>
        prob_preserving (p_space p1,a,prob p1) p2 SUBSET
        prob_preserving p1 p2

[PROB_REAL_SUM_IMAGE]  Theorem

   |- !p s.
        prob_space p /\ s IN events p /\
        (!x. x IN s ==> {x} IN events p) /\ FINITE s ==>
        (prob p s = SIGMA (\x. prob p {x}) s)

[PROB_REAL_SUM_IMAGE_FN]  Theorem

   |- !p f e s.
        prob_space p /\ e IN events p /\
        (!x. x IN s ==> e INTER f x IN events p) /\ FINITE s /\
        (!x y. x IN s /\ y IN s /\ ~(x = y) ==> DISJOINT (f x) (f y)) /\
        (BIGUNION (IMAGE f s) INTER p_space p = p_space p) ==>
        (prob p e = SIGMA (\x. prob p (e INTER f x)) s)

[PROB_SPACE]  Theorem

   |- !p.
        prob_space p =
        sigma_algebra (p_space p,events p) /\ positive p /\
        countably_additive p /\ (prob p (p_space p) = 1)

[PROB_SPACE_ADDITIVE]  Theorem

   |- !p. prob_space p ==> additive p

[PROB_SPACE_COUNTABLY_ADDITIVE]  Theorem

   |- !p. prob_space p ==> countably_additive p

[PROB_SPACE_INCREASING]  Theorem

   |- !p. prob_space p ==> increasing p

[PROB_SPACE_POSITIVE]  Theorem

   |- !p. prob_space p ==> positive p

[PROB_SUBADDITIVE]  Theorem

   |- !p s t u.
        prob_space p /\ t IN events p /\ u IN events p /\
        (s = t UNION u) ==>
        prob p s <= prob p t + prob p u

[PROB_UNIV]  Theorem

   |- !p. prob_space p ==> (prob p (p_space p) = 1)

[PROB_ZERO_UNION]  Theorem
```

```
   |- !p s t.
        prob_space p /\ s IN events p /\ t IN events p /\
        (prob p t = 0) ==>
        (prob p (s UNION t) = prob p s)

[PSPACE]  Theorem

   |- !a b c. p_space (a,b,c) = a

[distribution_lebesgue_thm1]  Theorem

   |- !X p s A.
        random_variable X p s /\ A IN subsets s ==>
        (distribution p X A =
         integral p (indicator_fn (PREIMAGE X A INTER p_space p)))

[distribution_lebesgue_thm2]  Theorem

   |- !X p s A.
        random_variable X p s /\ A IN subsets s ==>
        (distribution p X A =
         integral (space s,subsets s,distribution p X) (indicator_fn A))

[distribution_prob_space]  Theorem

   |- !p X s.
        random_variable X p s ==>
        prob_space (space s,subsets s,distribution p X)

[distribution_x_eq_1_imp_distribution_y_eq_0]  Theorem

   |- !X p x.
        random_variable X p
          (IMAGE X (p_space p),POW (IMAGE X (p_space p))) /\
        (distribution p X {x} = 1) ==>
        !y. ~(y = x) ==> (distribution p X {y} = 0)

[finite_expectation]  Theorem

   |- !p X.
        FINITE (p_space p) /\ real_random_variable X p ==>
        (expectation p X =
         SIGMA (\r. r * distribution p X {r}) (IMAGE X (p_space p)))

[finite_expectation1]  Theorem

   |- !p X.
        FINITE (p_space p) /\ real_random_variable X p ==>
        (expectation p X =
         SIGMA (\r. r * prob p (PREIMAGE X {r} INTER p_space p))
           (IMAGE X (p_space p)))

[finite_marginal_product_space_POW]  Theorem

   |- !p X Y.
        (POW (p_space p) = events p) /\
```

```
        random_variable X p
          (IMAGE X (p_space p),POW (IMAGE X (p_space p))) /\
        random_variable Y p
          (IMAGE Y (p_space p),POW (IMAGE Y (p_space p))) /\
        FINITE (p_space p) ==>
        measure_space
          (IMAGE X (p_space p) CROSS IMAGE Y (p_space p),
           POW (IMAGE X (p_space p) CROSS IMAGE Y (p_space p)),
           (\a. prob p (PREIMAGE (\x. (X x,Y x)) a INTER p_space p)))

[finite_marginal_product_space_POW2]  Theorem

   |- !p s1 s2 X Y.
        (POW (p_space p) = events p) /\
        random_variable X p (s1,POW s1) /\
        random_variable Y p (s2,POW s2) /\ FINITE (p_space p) /\
        FINITE s1 /\ FINITE s2 ==>
        measure_space
          (s1 CROSS s2,POW (s1 CROSS s2),joint_distribution p X Y)

[prob_x_eq_1_imp_prob_y_eq_0]  Theorem

   |- !p x.
        prob_space p /\ {x} IN events p /\ (prob p {x} = 1) ==>
        !y. {y} IN events p /\ ~(y = x) ==> (prob p {y} = 0)
```

Appendix F

informationTheory

```
[KL_divergence_def]  Definition

   |- !b s u v.
        KL_divergence b s u v =
        integral (space s,subsets s,u)
          (\x. logr b (RN_deriv (space s,subsets s,v) u x))

[conditional_mutual_information_def]  Definition

   |- !b p s1 s2 s3 X Y Z.
        conditional_mutual_information b p s1 s2 s3 X Y Z =
        (let prod_space =
               prod_measure_space (space s2,subsets s2,distribution p Y)
                 (space s3,subsets s3,distribution p Z)
         in
           mutual_information b p s1
             (p_space prod_space,events prod_space) X (\x. (Y x,Z x)) -
           mutual_information b p s1 s3 X Z)

[entropy_def]  Definition

   |- !b p s X. entropy b p s X = mutual_information b p s s X X

[mutual_information_def]  Definition

   |- !b p s1 s2 X Y.
        mutual_information b p s1 s2 X Y =
        (let prod_space =
               prod_measure_space (space s1,subsets s1,distribution p X)
                 (space s2,subsets s2,distribution p Y)
         in
           KL_divergence b (p_space prod_space,events prod_space)
             (joint_distribution p X Y) (prob prod_space))

[finite_conditional_mutual_information_reduce]  Theorem

   |- !b p X Y Z.
        (POW (p_space p) = events p) /\
        random_variable X p
          (IMAGE X (p_space p),POW (IMAGE X (p_space p))) /\
        random_variable Y p
```

```
          (IMAGE Y (p_space p),POW (IMAGE Y (p_space p))) /\
        random_variable Z p
          (IMAGE Z (p_space p),POW (IMAGE Z (p_space p))) /\
        FINITE (p_space p) ==>
        (conditional_mutual_information b p
           (IMAGE X (p_space p),POW (IMAGE X (p_space p)))
           (IMAGE Y (p_space p),POW (IMAGE Y (p_space p)))
           (IMAGE Z (p_space p),POW (IMAGE Z (p_space p))) X Y Z =
         ~SIGMA
           (\(x,z).
              joint_distribution p X Z {(x,z)} *
              logr b
                (joint_distribution p X Z {(x,z)} /
                 distribution p Z {z}))
           (IMAGE X (p_space p) CROSS IMAGE Z (p_space p)) -
         ~SIGMA
           (\(x,y,z).
              joint_distribution p X (\x. (Y x,Z x)) {(x,y,z)} *
              logr b
                (joint_distribution p X (\x. (Y x,Z x)) {(x,y,z)} /
                 distribution p (\x. (Y x,Z x)) {(y,z)}))
           (IMAGE X (p_space p) CROSS
            IMAGE (\x. (Y x,Z x)) (p_space p)))

[finite_entropy_certainty_eq_0]  Theorem

   |- !b p X.
        (POW (p_space p) = events p) /\
        random_variable X p
          (IMAGE X (p_space p),POW (IMAGE X (p_space p))) /\
        FINITE (p_space p) /\
        (?x. x IN IMAGE X (p_space p) /\ (distribution p X {x} = 1)) ==>
        (entropy b p (IMAGE X (p_space p),POW (IMAGE X (p_space p))) X =
         0)

[finite_entropy_le_card]  Theorem

   |- !b p X.
        1 <= b /\ (POW (p_space p) = events p) /\
        random_variable X p
          (IMAGE X (p_space p),POW (IMAGE X (p_space p))) /\
        FINITE (p_space p) ==>
        entropy b p (IMAGE X (p_space p),POW (IMAGE X (p_space p))) X <=
        logr b
          (&
             (CARD
                (IMAGE X (p_space p) INTER
                 {x | ~(distribution p X {x} = 0)})))

[finite_entropy_reduce]  Theorem

   |- !b p X.
        (POW (p_space p) = events p) /\
        random_variable X p
          (IMAGE X (p_space p),POW (IMAGE X (p_space p))) /\
        FINITE (p_space p) ==>
```

```
        (entropy b p (IMAGE X (p_space p),POW (IMAGE X (p_space p))) X =
         ~SIGMA
            (\x. distribution p X {x} * logr b (distribution p X {x}))
            (IMAGE X (p_space p)))

[finite_entropy_uniform_max]  Theorem

   |- !b p X.
        (POW (p_space p) = events p) /\
        random_variable X p
          (IMAGE X (p_space p),POW (IMAGE X (p_space p))) /\
        FINITE (p_space p) /\
        (!x y.
           x IN IMAGE X (p_space p) /\ y IN IMAGE X (p_space p) ==>
           (distribution p X {x} = distribution p X {y})) ==>
        (entropy b p (IMAGE X (p_space p),POW (IMAGE X (p_space p))) X =
         logr b (& (CARD (IMAGE X (p_space p)))))

[finite_mutual_information_reduce]  Theorem

   |- !b p s1 s2 X Y.
        (POW (p_space p) = events p) /\
        random_variable X p
          (IMAGE X (p_space p),POW (IMAGE X (p_space p))) /\
        random_variable Y p
          (IMAGE Y (p_space p),POW (IMAGE Y (p_space p))) /\
        FINITE (p_space p) ==>
        (mutual_information b p
           (IMAGE X (p_space p),POW (IMAGE X (p_space p)))
           (IMAGE Y (p_space p),POW (IMAGE Y (p_space p))) X Y =
         SIGMA
           (\(x,y).
              joint_distribution p X Y {(x,y)} *
              logr b
                (joint_distribution p X Y {(x,y)} /
                 (distribution p X {x} * distribution p Y {y})))
           (IMAGE X (p_space p) CROSS IMAGE Y (p_space p)))

[finite_mutual_information_reduce2]  Theorem

   |- !b p s1 s2 X Y Z.
        (POW (p_space p) = events p) /\
        random_variable X p
          (IMAGE X (p_space p),POW (IMAGE X (p_space p))) /\
        random_variable Y p
          (IMAGE Y (p_space p),POW (IMAGE Y (p_space p))) /\
        random_variable Z p
          (IMAGE Z (p_space p),POW (IMAGE Z (p_space p))) /\
        FINITE (p_space p) ==>
        (mutual_information b p
           (IMAGE X (p_space p),POW (IMAGE X (p_space p)))
           (IMAGE Y (p_space p) CROSS IMAGE Z (p_space p),
            POW (IMAGE Y (p_space p) CROSS IMAGE Z (p_space p))) X
           (\x. (Y x,Z x)) =
         SIGMA
           (\(x,y,z).
```

```
    joint_distribution p X (\x. (Y x,Z x)) {(x,y,z)} *
    logr b
      (joint_distribution p X (\x. (Y x,Z x)) {(x,y,z)} /
       (distribution p X {x} *
        distribution p (\x. (Y x,Z x)) {(y,z)})))
(IMAGE X (p_space p) CROSS
 (IMAGE Y (p_space p) CROSS IMAGE Z (p_space p))))
```

Appendix G

leakageTheory

```
[H_def]  Definition

   |- !s. H s = FST (FST s)

[L_def]  Definition

   |- !s. L s = SND (FST s)

[R_def]  Definition

   |- !s. R s = SND s

[leakage_def]  Definition

   |- !p f.
        leakage p f =
        conditional_mutual_information 2 p
          (IMAGE f (p_space p),POW (IMAGE f (p_space p)))
          (IMAGE H (p_space p),POW (IMAGE H (p_space p)))
          (IMAGE L (p_space p),POW (IMAGE L (p_space p))) f H L

[unif_prog_dist_def]  Definition

   |- !high low random.
        unif_prog_dist high low random =
        (\s.
           (if s IN high CROSS low CROSS random then
              1 / & (CARD (high CROSS low CROSS random))
            else
              0))

[unif_prog_space_def]  Definition

   |- !high low random.
        unif_prog_space high low random =
        (high CROSS low CROSS random,POW (high CROSS low CROSS random),
         (\s. SIGMA (unif_prog_dist high low random) s))

[visible_leakage_def]  Definition

   |- !p f.
```

```
        visible_leakage p f =
        conditional_mutual_information 2 p
          (IMAGE f (p_space p),POW (IMAGE f (p_space p)))
          (IMAGE H (p_space p),POW (IMAGE H (p_space p)))
          (IMAGE (\s. (L s,R s)) (p_space p),
           POW (IMAGE (\s. (L s,R s)) (p_space p))) f H (\s. (L s,R s))

[prob_space_unif_prog_space]  Theorem

   |- !high low random.
        FINITE high /\ FINITE low /\ FINITE random /\
        ~(high CROSS low CROSS random = {}) ==>
        prob_space (unif_prog_space high low random)

[prob_unif_prog_space]  Theorem

   |- !high low random P.
        FINITE high /\ FINITE low /\ FINITE random /\
        ~(high CROSS low CROSS random = {}) /\
        P SUBSET high CROSS low CROSS random ==>
        (prob (unif_prog_space high low random) P =
         & (CARD P) / & (CARD high * CARD low * CARD random))

[unif_prog_space_highlow_distribution]  Theorem

   |- !high low random f.
        FINITE high /\ FINITE low /\ FINITE random /\
        ~(high CROSS low CROSS random = {}) ==>
        !h l.
          h IN high /\ l IN low ==>
          (distribution (unif_prog_space high low random)
             (\x. (H x,L x)) {(h,l)} =
           1 / & (CARD high * CARD low))

[unif_prog_space_highlowrandom_distribution]  Theorem

   |- !high low random f.
        FINITE high /\ FINITE low /\ FINITE random /\
        ~(high CROSS low CROSS random = {}) ==>
        !h l r.
          h IN high /\ l IN low /\ r IN random ==>
          (distribution (unif_prog_space high low random)
             (\x. (H x,L x,R x)) {(h,l,r)} =
           1 / & (CARD high * CARD low * CARD random))

[unif_prog_space_leakage_LIST_TO_SET_computation_reduce]  Theorem

   |- !high low random f.
        ALL_DISTINCT high /\ ALL_DISTINCT low /\ ALL_DISTINCT random /\
        ~(high = []) /\ ~(low = []) /\ ~(random = []) ==>
        (leakage (unif_prog_space (set high) (set low) (set random)) f =
         1 / & (LENGTH high * LENGTH low * LENGTH random) *
         (REAL_SUM
            (MAP
               (\x.
                  (\(out,h,l).
```

```
                    (\s. s * lg (1 / & (LENGTH random) * s))
                      (REAL_SUM
                         (MAP
                            (\r. (if f ((h,l),r) = out then 1 else 0))
                            random))) x)
               (MAP_ALL_DISTINCT
                  (MAP (\s. (f s,FST s))
                     (LIST_COMBS (LIST_COMBS high low) random)))) -
            REAL_SUM
              (MAP
                 (\x.
                    (\(out,l).
                       (\s.
                          s *
                          lg (1 / & (LENGTH high * LENGTH random) * s))
                        (REAL_SUM
                           (MAP
                              (\(h,r).
                                 (if f ((h,l),r) = out then 1 else 0))
                              (LIST_COMBS high random)))) x)
                 (MAP_ALL_DISTINCT
                    (MAP (\s. (f s,SND (FST s)))
                       (LIST_COMBS (LIST_COMBS high low) random))))))

[unif_prog_space_leakage_computation_reduce]  Theorem

   |- !high low random f.
        FINITE high /\ FINITE low /\ FINITE random /\
        ~(high CROSS low CROSS random = {}) ==>
        (leakage (unif_prog_space high low random) f =
         1 / & (CARD high * CARD low * CARD random) *
         (SIGMA
            (\x.
               (\(out,h,l).
                  SIGMA (\r. (if f ((h,l),r) = out then 1 else 0))
                    random *
                  lg
                    (1 / & (CARD random) *
                     SIGMA (\r. (if f ((h,l),r) = out then 1 else 0))
                       random)) x)
              (IMAGE (\s. (f s,FST s)) (high CROSS low CROSS random)) -
            SIGMA
              (\x.
                 (\(out,l).
                    SIGMA (\(h,r). (if f ((h,l),r) = out then 1 else 0))
                      (high CROSS random) *
                    lg
                      (1 / & (CARD high * CARD random) *
                       SIGMA
                         (\(h,r). (if f ((h,l),r) = out then 1 else 0))
                         (high CROSS random))) x)
              (IMAGE (\s. (f s,SND (FST s)))
                 (high CROSS low CROSS random))))

[unif_prog_space_leakage_lemma1]  Theorem
```

```
   |- !high low random f.
        FINITE high /\ FINITE low /\ FINITE random /\
        ~(high CROSS low CROSS random = {}) ==>
        (SIGMA
           (\x.
              (\(x,z).
                 joint_distribution (unif_prog_space high low random) f
                   L {(x,z)} *
                 lg
                   (joint_distribution (unif_prog_space high low random)
                      f L {(x,z)} * & (CARD low))) x)
           (IMAGE (\s. (f s,SND (FST s)))
              (high CROSS low CROSS random)) =
         SIGMA
           (\x.
              (\(x,z).
                 1 / & (CARD high * CARD low * CARD random) *
                 SIGMA (\(h,r). (if f ((h,z),r) = x then 1 else 0))
                   (high CROSS random) *
                 lg
                   (1 / & (CARD high * CARD low * CARD random) *
                    SIGMA (\(h,r). (if f ((h,z),r) = x then 1 else 0))
                      (high CROSS random) * & (CARD low))) x)
           (IMAGE (\s. (f s,SND (FST s)))
              (high CROSS low CROSS random)))

[unif_prog_space_leakage_lemma2]  Theorem

   |- !high low random f.
        FINITE high /\ FINITE low /\ FINITE random /\
        ~(high CROSS low CROSS random = {}) ==>
        (SIGMA
           (\x.
              (\(x,y,z).
                 joint_distribution (unif_prog_space high low random) f
                   (\x. (H x,L x)) {(x,y,z)} *
                 lg
                   (joint_distribution (unif_prog_space high low random)
                      f (\x. (H x,L x)) {(x,y,z)} *
                    & (CARD high * CARD low))) x)
           (IMAGE (\s. (f s,FST s)) (high CROSS low CROSS random)) =
         SIGMA
           (\x.
              (\(out,h,l).
                 1 / & (CARD high * CARD low * CARD random) *
                 SIGMA (\r. (if f ((h,l),r) = out then 1 else 0))
                   random *
                 lg
                   (1 / & (CARD high * CARD low * CARD random) *
                    SIGMA (\r. (if f ((h,l),r) = out then 1 else 0))
                      random * & (CARD high * CARD low))) x)
           (IMAGE (\s. (f s,FST s)) (high CROSS low CROSS random)))

[unif_prog_space_leakage_lemma3]  Theorem

   |- !high low random f.
```

```
        FINITE high /\ FINITE low /\ FINITE random /\
        ~(high CROSS low CROSS random = {}) ==>
        (SIGMA
           (\x.
              (\(x,z).
                 joint_distribution (unif_prog_space high low random) f
                   L {(x,z)} *
                 lg
                   (joint_distribution (unif_prog_space high low random)
                      f L {(x,z)} * & (CARD low))) x)
           (IMAGE (\s. (f s,SND (FST s)))
              (high CROSS low CROSS random)) =
         1 / & (CARD high * CARD low * CARD random) *
         SIGMA
           (\x.
              (\(out,l).
                 SIGMA (\(h,r). (if f ((h,l),r) = out then 1 else 0))
                   (high CROSS random) *
                 lg
                   (1 / & (CARD high * CARD random) *
                    SIGMA (\(h,r). (if f ((h,l),r) = out then 1 else 0))
                      (high CROSS random))) x)
           (IMAGE (\s. (f s,SND (FST s)))
              (high CROSS low CROSS random)))

[unif_prog_space_leakage_lemma4]  Theorem

   |- !high low random f.
        FINITE high /\ FINITE low /\ FINITE random /\
        ~(high CROSS low CROSS random = {}) ==>
        (SIGMA
           (\x.
              (\(x,y,z).
                 joint_distribution (unif_prog_space high low random) f
                   (\x. (H x,L x)) {(x,y,z)} *
                 lg
                   (joint_distribution (unif_prog_space high low random)
                      f (\x. (H x,L x)) {(x,y,z)} *
                    & (CARD high * CARD low))) x)
           (IMAGE (\s. (f s,FST s)) (high CROSS low CROSS random)) =
         1 / & (CARD high * CARD low * CARD random) *
         SIGMA
           (\x.
              (\(out,h,l).
                 SIGMA (\r. (if f ((h,l),r) = out then 1 else 0))
                   random *
                 lg
                   (1 / & (CARD random) *
                    SIGMA (\r. (if f ((h,l),r) = out then 1 else 0))
                      random)) x)
           (IMAGE (\s. (f s,FST s)) (high CROSS low CROSS random)))

[unif_prog_space_leakage_reduce]  Theorem

   |- !high low random f.
        FINITE high /\ FINITE low /\ FINITE random /\
```

```
        ~(high CROSS low CROSS random = {}) ==>
        (leakage (unif_prog_space high low random) f =
         SIGMA
           (\x.
              (\(x,y,z).
                 joint_distribution (unif_prog_space high low random) f
                   (\x. (H x,L x)) {(x,y,z)} *
                 lg
                   (joint_distribution (unif_prog_space high low random)
                      f (\x. (H x,L x)) {(x,y,z)} *
                    & (CARD high * CARD low))) x)
           (IMAGE (\s. (f s,FST s)) (high CROSS low CROSS random)) -
         SIGMA
           (\x.
              (\(x,z).
                 joint_distribution (unif_prog_space high low random) f
                   L {(x,z)} *
                 lg
                   (joint_distribution (unif_prog_space high low random)
                      f L {(x,z)} * & (CARD low))) x)
           (IMAGE (\s. (f s,SND (FST s)))
              (high CROSS low CROSS random)))

[unif_prog_space_low_distribution]  Theorem

   |- !high low random f.
        FINITE high /\ FINITE low /\ FINITE random /\
        ~(high CROSS low CROSS random = {}) ==>
        !l.
          l IN low ==>
          (distribution (unif_prog_space high low random) L {l} =
           1 / & (CARD low))

[unif_prog_space_lowrandom_distribution]  Theorem

   |- !high low random f.
        FINITE high /\ FINITE low /\ FINITE random /\
        ~(high CROSS low CROSS random = {}) ==>
        !l r.
          l IN low /\ r IN random ==>
          (distribution (unif_prog_space high low random)
             (\x. (L x,R x)) {(l,r)} =
           1 / & (CARD low * CARD random))

[unif_prog_space_visible_leakage_LIST_TO_SET_computation_reduce]  Theorem

   |- !high low random f.
        ALL_DISTINCT high /\ ALL_DISTINCT low /\ ALL_DISTINCT random /\
        ~(high = []) /\ ~(low = []) /\ ~(random = []) ==>
        (visible_leakage
           (unif_prog_space (set high) (set low) (set random)) f =
         ~(1 / & (LENGTH high * LENGTH low * LENGTH random)) *
         REAL_SUM
           (MAP
              (\x.
                 (\(out,l,r).
```

```
                  (\s. s * lg (1 / & (LENGTH high) * s))
                    (REAL_SUM
                       (MAP (\h. (if f ((h,l),r) = out then 1 else 0))
                          high))) x)
              (MAKE_ALL_DISTINCT
                 (MAP (\s. (f s,SND (FST s),SND s))
                    (LIST_COMBS (LIST_COMBS high low) random)))))

[unif_prog_space_visible_leakage_computation_reduce]  Theorem

   |- !high low random f.
        FINITE high /\ FINITE low /\ FINITE random /\
        ~(high CROSS low CROSS random = {}) ==>
        (visible_leakage (unif_prog_space high low random) f =
         ~(1 / & (CARD high * CARD low * CARD random) *
           SIGMA
             (\x.
                (\(out,l,r).
                   SIGMA (\h. (if f ((h,l),r) = out then 1 else 0))
                     high *
                   lg
                     (1 / & (CARD high) *
                      SIGMA (\h. (if f ((h,l),r) = out then 1 else 0))
                        high)) x)
             (IMAGE (\s. (f s,SND (FST s),SND s))
                (high CROSS low CROSS random))))

[unif_prog_space_visible_leakage_lemma1]  Theorem

   |- !high low random f.
        FINITE high /\ FINITE low /\ FINITE random /\
        ~(high CROSS low CROSS random = {}) ==>
        (SIGMA
           (\x.
              (\(x,z).
                 joint_distribution (unif_prog_space high low random) f
                   (\s. (L s,R s)) {(x,z)} *
                 lg
                   (joint_distribution (unif_prog_space high low random)
                      f (\s. (L s,R s)) {(x,z)} *
                    & (CARD low * CARD random))) x)
           (IMAGE (\s. (f s,SND (FST s),SND s))
              (high CROSS low CROSS random)) =
         SIGMA
           (\x.
              (\(x,z).
                 1 / & (CARD high * CARD low * CARD random) *
                 SIGMA (\h. (if f ((h,FST z),SND z) = x then 1 else 0))
                   high *
                 lg
                   (1 / & (CARD high * CARD low * CARD random) *
                    SIGMA
                      (\h. (if f ((h,FST z),SND z) = x then 1 else 0))
                      high * & (CARD low * CARD random))) x)
           (IMAGE (\s. (f s,SND (FST s),SND s))
              (high CROSS low CROSS random)))
```

```
[unif_prog_space_visible_leakage_lemma2]  Theorem

   |- !high low random f.
        FINITE high /\ FINITE low /\ FINITE random /\
        ~(high CROSS low CROSS random = {}) ==>
        (SIGMA
           (\x.
              (\(x,y,z).
                 joint_distribution (unif_prog_space high low random) f
                   (\s. (H s,L s,R s)) {(x,y,z)} *
                 lg
                   (joint_distribution (unif_prog_space high low random)
                      f (\s. (H s,L s,R s)) {(x,y,z)} *
                    & (CARD high * CARD low * CARD random))) x)
           (IMAGE (\s. (f s,FST (FST s),SND (FST s),SND s))
              (high CROSS low CROSS random)) =
         SIGMA
           (\x.
              (\(out,h,l,r).
                 1 / & (CARD high * CARD low * CARD random) *
                 (if f ((h,l),r) = out then 1 else 0) *
                 lg
                   (1 / & (CARD high * CARD low * CARD random) *
                    (if f ((h,l),r) = out then 1 else 0) *
                    & (CARD high * CARD low * CARD random))) x)
           (IMAGE (\s. (f s,FST (FST s),SND (FST s),SND s))
              (high CROSS low CROSS random)))

[unif_prog_space_visible_leakage_lemma3]  Theorem

   |- !high low random f.
        FINITE high /\ FINITE low /\ FINITE random /\
        ~(high CROSS low CROSS random = {}) ==>
        (SIGMA
           (\x.
              (\(x,z).
                 joint_distribution (unif_prog_space high low random) f
                   (\s. (L s,R s)) {(x,z)} *
                 lg
                   (joint_distribution (unif_prog_space high low random)
                      f (\s. (L s,R s)) {(x,z)} *
                    & (CARD low * CARD random))) x)
           (IMAGE (\s. (f s,SND (FST s),SND s))
              (high CROSS low CROSS random)) =
         1 / & (CARD high * CARD low * CARD random) *
         SIGMA
           (\x.
              (\(out,l,r).
                 SIGMA (\h. (if f ((h,l),r) = out then 1 else 0)) high *
                 lg
                   (1 / & (CARD high) *
                    SIGMA (\h. (if f ((h,l),r) = out then 1 else 0))
                      high)) x)
           (IMAGE (\s. (f s,SND (FST s),SND s))
              (high CROSS low CROSS random)))
```

```
[unif_prog_space_visible_leakage_lemma4]  Theorem

   |- !high low random f.
        FINITE high /\ FINITE low /\ FINITE random /\
        ~(high CROSS low CROSS random = {}) ==>
        (SIGMA
           (\x.
              (\(x,y,z).
                 joint_distribution (unif_prog_space high low random) f
                   (\s. (H s,L s,R s)) {(x,y,z)} *
                 lg
                   (joint_distribution (unif_prog_space high low random)
                      f (\s. (H s,L s,R s)) {(x,y,z)} *
                    & (CARD high * CARD low * CARD random))) x)
           (IMAGE (\s. (f s,FST (FST s),SND (FST s),SND s))
              (high CROSS low CROSS random)) =
         0)

[unif_prog_space_visible_leakage_reduce]  Theorem

   |- !high low random f.
        FINITE high /\ FINITE low /\ FINITE random /\
        ~(high CROSS low CROSS random = {}) ==>
        (visible_leakage (unif_prog_space high low random) f =
         SIGMA
           (\x.
              (\(x,y,z).
                 joint_distribution (unif_prog_space high low random) f
                   (\s. (H s,L s,R s)) {(x,y,z)} *
                 lg
                   (joint_distribution (unif_prog_space high low random)
                      f (\s. (H s,L s,R s)) {(x,y,z)} *
                    & (CARD high * CARD low * CARD random))) x)
           (IMAGE (\s. (f s,FST (FST s),SND (FST s),SND s))
              (high CROSS low CROSS random)) -
         SIGMA
           (\x.
              (\(x,z).
                 joint_distribution (unif_prog_space high low random) f
                   (\s. (L s,R s)) {(x,z)} *
                 lg
                   (joint_distribution (unif_prog_space high low random)
                      f (\s. (L s,R s)) {(x,z)} *
                    & (CARD low * CARD random))) x)
           (IMAGE (\s. (f s,SND (FST s),SND s))
              (high CROSS low CROSS random)))
```

Appendix H

dining_cryptosTheory

```
[XOR_announces_def]  Definition

   |- (!low.
         XOR_announces low 0 = low (STRCAT "announces" (toString 0))) /\
      !low i.
        XOR_announces low (SUC i) =
        low (STRCAT "announces" (toString (SUC i))) xor
        XOR_announces low i

[biased_dc_prog_space2_primitive_def]  Definition

   |- biased_dc_prog_space2 =
      WFREC (@R'. WF R')
        (\biased_dc_prog_space2 a.
           case a of
              0 -> ARB
           || SUC 0 -> ARB
           || SUC (SUC n) ->
                I
                  (biased_high_states (SUC (SUC n)) CROSS
                   biased_low_states CROSS
                   biased_random_states (SUC (SUC n)),
                   POW
                     (biased_high_states (SUC (SUC n)) CROSS
                      biased_low_states CROSS
                      biased_random_states (SUC (SUC n))),
                   (\s.
                      SIGMA
                        (biased_dist2 (biased_high_states (SUC (SUC n)))
                           biased_low_states
                           (biased_random_states (SUC (SUC n)))) s)))

[biased_dc_prog_space_primitive_def]  Definition

   |- biased_dc_prog_space =
      WFREC (@R'. WF R')
        (\biased_dc_prog_space a.
           case a of
              0 -> ARB
           || SUC 0 -> ARB
           || SUC (SUC n) ->
```

```
                I
                  (biased_high_states (SUC (SUC n)) CROSS
                   biased_low_states CROSS
                   biased_random_states (SUC (SUC n)),
                   POW
                     (biased_high_states (SUC (SUC n)) CROSS
                      biased_low_states CROSS
                      biased_random_states (SUC (SUC n))),
                   (\s.
                      SIGMA
                        (biased_dist (biased_high_states (SUC (SUC n)))
                           biased_low_states
                           (biased_random_states (SUC (SUC n)))) s)))

[biased_dist2_def]  Definition

   |- !high low random.
        biased_dist2 high low random =
        (\s.
           (if R s (STRCAT "coin" (toString (SUC 0))) then
              1 / 2 * unif_prog_dist high low random s
            else
              3 / 2 * unif_prog_dist high low random s))

[biased_dist_def]  Definition

   |- !high low random.
        biased_dist high low random =
        (\s.
           (if L s = (\s. s = STRCAT "coin" (toString 0)) then
              1 / 2 * unif_prog_dist high low random s
            else
              3 / 2 * unif_prog_dist high low random s))

[biased_high_states_def]  Definition

   |- !n. biased_high_states n = insider_high_states_set n

[biased_low_states_def]  Definition

   |- biased_low_states =
      {(\s. s = STRCAT "coin" (toString 0)); (\s. F)}

[biased_random_states_def]  Definition

   |- (biased_random_states 0 = {(\s. F)}) /\
      !n.
        biased_random_states (SUC n) =
        IMAGE
          (\s x.
             (if x = STRCAT "coin" (toString (SUC n)) then T else s x))
          (biased_random_states n) UNION
        IMAGE
          (\s x.
             (if x = STRCAT "coin" (toString (SUC n)) then F else s x))
          (biased_random_states n)
```

```
[coin_assignment_def]  Definition

   |- (!out h n choice.
         coin_assignment out h n choice 0 =
         (\s.
            (if s = STRCAT "coin" (toString 0) then
               choice xor XOR_announces out 0 xor ~(0 < h)
             else
               F))) /\
      !out h n choice i.
        coin_assignment out h n choice (SUC i) =
        (\s.
           (if s = STRCAT "coin" (toString (SUC i)) then
              choice xor XOR_announces out (SUC i) xor ~(SUC i < h)
            else
              coin_assignment out h n choice i s))

[coin_assignment_set_def]  Definition

   |- !out p n.
        coin_assignment_set out p n =
        {coin_assignment out p n T (SUC (SUC n));
         coin_assignment out p n F (SUC (SUC n))}

[compute_result_def]  Definition

   |- !low n s.
        compute_result low n s =
        (if s = "result" then XOR_announces low n else low s)

[dc_high_states_curried_def]  Definition

   |- !x x1. dc_high_states x x1 = dc_high_states_tupled (x,x1)

[dc_high_states_set_def]  Definition

   |- (dc_high_states_set 0 = {(\s. s = STRCAT "pays" (toString 0))}) /\
      !n.
        dc_high_states_set (SUC n) =
        (\s. s = STRCAT "pays" (toString (SUC n))) INSERT
        dc_high_states_set n

[dc_high_states_tupled_primitive_def]  Definition

   |- dc_high_states_tupled =
      WFREC (@R'. WF R')
        (\dc_high_states_tupled a.
           case a of
              (nsapays,0) -> ARB
           || (nsapays,SUC 0) -> ARB
           || (nsapays,SUC (SUC n)) ->
                I
                  (if nsapays then
                     {(\s. s = STRCAT "pays" (toString (SUC (SUC n))))}
                   else
```

```
                    dc_high_states_set (SUC n)))

[dc_low_states_def]  Definition

   |- dc_low_states = {(\s. F)}

[dc_prog_space_curried_def]  Definition

   |- !x x1. dc_prog_space x x1 = dc_prog_space_tupled (x,x1)

[dc_prog_space_tupled_primitive_def]  Definition

   |- dc_prog_space_tupled =
      WFREC (@R'. WF R')
        (\dc_prog_space_tupled a.
           case a of
              (0,nsapays) -> ARB
           || (SUC 0,nsapays) -> ARB
           || (SUC (SUC n),nsapays) ->
                I
                  (unif_prog_space
                     (dc_high_states nsapays (SUC (SUC n)))
                     dc_low_states (dc_random_states (SUC n))))

[dc_random_states_def]  Definition

   |- (dc_random_states 0 =
       {(\s. s = STRCAT "coin" (toString 0)); (\s. F)}) /\
      !n.
        dc_random_states (SUC n) =
        IMAGE
          (\s x.
             (if x = STRCAT "coin" (toString (SUC n)) then T else s x))
          (dc_random_states n) UNION
        IMAGE
          (\s x.
             (if x = STRCAT "coin" (toString (SUC n)) then F else s x))
          (dc_random_states n)

[dc_random_witness_def]  Definition

   |- (!x.
         dc_random_witness x 0 =
         (\s.
            (if s = STRCAT "coin" (toString 0) then
               ~x (STRCAT "announces" (toString 0))
             else
               F))) /\
      !x i.
        dc_random_witness x (SUC i) =
        (\s.
           (if s = STRCAT "coin" (toString (SUC i)) then
              ~XOR_announces x (SUC i)
            else
              dc_random_witness x i s))
```

```
[dc_valid_outputs_def]  Definition

   |- !n.
        dc_valid_outputs n =
        {s |
         (s "result" = XOR_announces s n) /\ XOR_announces s n /\
         !x.
           ~(x = "result") /\
           (!i. i <= n ==> ~(x = STRCAT "announces" (toString i))) ==>
           ~s x}

[dc_valid_outputs_list_primitive_def]  Definition

   |- dc_valid_outputs_list =
      WFREC (@R'. WF R')
        (\dc_valid_outputs_list a.
           case a of
              0 -> ARB
           || SUC 0 -> ARB
           || SUC (SUC 0) -> ARB
           || SUC (SUC (SUC n)) ->
                I
                  (MAP
                     (\l s.
                        (s = "result") \/
                        (if
                           s =
                           STRCAT "announces" (toString (SUC (SUC n)))
                         then
                           ~XOR_announces l (SUC n)
                         else
                           l s)) (n_minus_1_announces_list (SUC n))))

[dcprog_primitive_def]  Definition

   |- dcprog =
      WFREC (@R'. WF R')
        (\dcprog a.
           case a of
              0 -> ARB
           || SUC 0 -> ARB
           || SUC (SUC 0) -> ARB
           || SUC (SUC (SUC n)) ->
                I
                  (\((high,low),random).
                     compute_result
                       (set_announcements high low random (SUC (SUC n))
                          (SUC (SUC n))) (SUC (SUC n))))

[insider_dc_prog_space_curried_def]  Definition

   |- !x x1.
        insider_dc_prog_space x x1 = insider_dc_prog_space_tupled (x,x1)

[insider_dc_prog_space_tupled_primitive_def]  Definition
```

```
   |- insider_dc_prog_space_tupled =
      WFREC (@R'. WF R')
        (\insider_dc_prog_space_tupled a.
           case a of
              (0,nsapays) -> ARB
           || (SUC 0,nsapays) -> ARB
           || (SUC (SUC n),nsapays) ->
                I
                  (unif_prog_space
                     (insider_high_states nsapays (SUC (SUC n)))
                     insider_low_states
                     (insider_random_states (SUC n))))

[insider_dcprog_primitive_def]  Definition

   |- insider_dcprog =
      WFREC (@R'. WF R')
        (\insider_dcprog a.
           case a of
              0 -> ARB
           || SUC 0 -> ARB
           || SUC (SUC 0) -> ARB
           || SUC (SUC (SUC n)) ->
                I
                  (\((high,low),random).
                     compute_result
                       (insider_set_announcements high low random
                          (SUC (SUC n)) (SUC (SUC n))) (SUC (SUC n))))

[insider_high_states_curried_def]  Definition

   |- !x x1.
        insider_high_states x x1 = insider_high_states_tupled (x,x1)

[insider_high_states_set_def]  Definition

   |- (insider_high_states_set 0 = {}) /\
      !n.
        insider_high_states_set (SUC n) =
        (\s. s = STRCAT "pays" (toString (SUC n))) INSERT
        insider_high_states_set n

[insider_high_states_tupled_primitive_def]  Definition

   |- insider_high_states_tupled =
      WFREC (@R'. WF R')
        (\insider_high_states_tupled a.
           case a of
              (nsapays,0) -> ARB
           || (nsapays,SUC 0) -> ARB
           || (nsapays,SUC (SUC n)) ->
                I
                  (if nsapays then
                     {(\s. s = STRCAT "pays" (toString (SUC (SUC n))))}
                   else
                     insider_high_states_set (SUC n)))
```

```
[insider_low_states_def]  Definition

   |- insider_low_states =
      {(\s. s = STRCAT "coin" (toString 0)); (\s. F)}

[insider_random_states_def]  Definition

   |- (insider_random_states 0 = {(\s. F)}) /\
      !n.
        insider_random_states (SUC n) =
        IMAGE
          (\s x.
             (if x = STRCAT "coin" (toString (SUC n)) then T else s x))
          (insider_random_states n) UNION
        IMAGE
          (\s x.
             (if x = STRCAT "coin" (toString (SUC n)) then F else s x))
          (insider_random_states n)

[insider_set_announcements_curried_def]  Definition

   |- !x x1 x2 x3 x4 x5.
        insider_set_announcements x x1 x2 x3 x4 x5 =
        insider_set_announcements_tupled (x,x1,x2,x3,x4,x5)

[insider_set_announcements_tupled_primitive_def]  Definition

   |- insider_set_announcements_tupled =
      WFREC
        (@R'.
           WF R' /\
           (!n random low high s.
              ~(s = STRCAT "announces" (toString (SUC 0))) ==>
              R' (high,low,random,n,0,s) (high,low,random,n,SUC 0,s)) /\
           !n random low high i s.
             ~(s = STRCAT "announces" (toString (SUC (SUC i)))) ==>
             R' (high,low,random,n,SUC i,s)
               (high,low,random,n,SUC (SUC i),s))
        (\insider_set_announcements_tupled a.
           case a of
              (high,low,random,n,0,s) ->
                I
                  (if s = STRCAT "announces" (toString 0) then
                     high (STRCAT "pays" (toString 0)) xor
                     low (STRCAT "coin" (toString 0)) xor
                     random (STRCAT "coin" (toString n))
                   else
                     low s)
           || (high,low,random,n,SUC 0,s) ->
                I
                  (if s = STRCAT "announces" (toString (SUC 0)) then
                     high (STRCAT "pays" (toString (SUC 0))) xor
                     random (STRCAT "coin" (toString (SUC 0))) xor
                     low (STRCAT "coin" (toString 0))
                   else
```

```
                    insider_set_announcements_tupled
                      (high,low,random,n,0,s))
          || (high,low,random,n,SUC (SUC i),s) ->
               I
                 (if
                    s = STRCAT "announces" (toString (SUC (SUC i)))
                  then
                    high (STRCAT "pays" (toString (SUC (SUC i)))) xor
                    random (STRCAT "coin" (toString (SUC (SUC i)))) xor
                    random (STRCAT "coin" (toString (SUC i)))
                  else
                    insider_set_announcements_tupled
                      (high,low,random,n,SUC i,s)))

[n_minus_1_announces_list_def]  Definition

   |- (n_minus_1_announces_list 0 =
       [(\s. s = STRCAT "announces" (toString 0)); (\s. F)]) /\
      !n.
        n_minus_1_announces_list (SUC n) =
        MAP
          (\s x.
             (if x = STRCAT "announces" (toString (SUC n)) then
                T
              else
                s x)) (n_minus_1_announces_list n) ++
        MAP
          (\s x.
             (if x = STRCAT "announces" (toString (SUC n)) then
                F
              else
                s x)) (n_minus_1_announces_list n)

[set_announcements_def]  Definition

   |- (!high low random n s.
         set_announcements high low random n 0 s =
         (if s = STRCAT "announces" (toString 0) then
            high (STRCAT "pays" (toString 0)) xor
            random (STRCAT "coin" (toString 0)) xor
            random (STRCAT "coin" (toString n))
          else
            low s)) /\
      !high low random n i s.
        set_announcements high low random n (SUC i) s =
        (if s = STRCAT "announces" (toString (SUC i)) then
           high (STRCAT "pays" (toString (SUC i))) xor
           random (STRCAT "coin" (toString (SUC i))) xor
           random (STRCAT "coin" (toString i))
         else
           set_announcements high low random n i s)

[valid_coin_assignment_def]  Definition

   |- (!r out h n.
         valid_coin_assignment r out h n 0 =
```

```
          (r (STRCAT "coin" (toString 0)) =
           r (STRCAT "coin" (toString (SUC (SUC n)))) xor
           XOR_announces out 0 xor ~(0 < h))) /\
       !r out h n i.
         valid_coin_assignment r out h n (SUC i) =
         (r (STRCAT "coin" (toString (SUC i))) =
          r (STRCAT "coin" (toString (SUC (SUC n)))) xor
          XOR_announces out (SUC i) xor ~(SUC i < h)) /\
         valid_coin_assignment r out h n i

[CARD_DISJOINT_UNION]  Theorem

   |- !P Q.
        FINITE P /\ FINITE Q /\ DISJOINT P Q ==>
        (CARD (P UNION Q) = CARD P + CARD Q)

[CARD_dc_high_states_set]  Theorem

   |- !n. CARD (dc_high_states_set n) = SUC n

[CARD_dc_low_states]  Theorem

   |- CARD dc_low_states = 1

[CARD_dc_random_states]  Theorem

   |- !n. CARD (dc_random_states n) = 2 ** SUC n

[CARD_dc_set_cross]  Theorem

   |- 1 /
      &
        (CARD
           (IMAGE (\s. (FST s,SND s,dcprog (SUC (SUC (SUC 0))) s))
              (dc_high_states_set (SUC (SUC 0)) CROSS
               dc_low_states CROSS dc_random_states (SUC (SUC 0))))) =
      1 / 24

[CARD_dc_valid_outputs]  Theorem

   |- !n.
        CARD
          (IMAGE (\(h,r). dcprog (SUC (SUC (SUC n))) ((h,(\s. F)),r))
             (dc_high_states F (SUC (SUC (SUC n))) CROSS
              dc_random_states (SUC (SUC n)))) =
        2 ** SUC (SUC n)

[IN_dc_high_states_set]  Theorem

   |- !n x.
        x IN dc_high_states_set n =
        ?i. i <= n /\ (x = (\s. s = STRCAT "pays" (toString i)))

[IN_dc_random_states]  Theorem

   |- !n s.
```

```
        s IN dc_random_states n =
        !x. ~(?i. i <= n /\ (x = STRCAT "coin" (toString i))) ==> ~s x

[biased_dc3_leakage_result]  Theorem

   |- leakage (biased_dc_prog_space (SUC (SUC 0)))
        (insider_dcprog (SUC (SUC (SUC 0)))) =
      0

[biased_dc3_leakage_result2]  Theorem

   |- leakage (biased_dc_prog_space2 (SUC (SUC 0)))
        (insider_dcprog (SUC (SUC (SUC 0)))) =
      3 / 4 * lg 3 - 1

[biased_dc_prog_space2_def]  Theorem

   |- biased_dc_prog_space2 (SUC (SUC n)) =
      (biased_high_states (SUC (SUC n)) CROSS biased_low_states CROSS
       biased_random_states (SUC (SUC n)),
       POW
         (biased_high_states (SUC (SUC n)) CROSS biased_low_states CROSS
          biased_random_states (SUC (SUC n))),
       (\s.
          SIGMA
            (biased_dist2 (biased_high_states (SUC (SUC n)))
               biased_low_states (biased_random_states (SUC (SUC n))))
            s))

[biased_dc_prog_space2_ind]  Theorem

   |- !P. P (SUC 0) /\ P 0 /\ (!n. P (SUC (SUC n))) ==> !v. P v

[biased_dc_prog_space_def]  Theorem

   |- biased_dc_prog_space (SUC (SUC n)) =
      (biased_high_states (SUC (SUC n)) CROSS biased_low_states CROSS
       biased_random_states (SUC (SUC n)),
       POW
         (biased_high_states (SUC (SUC n)) CROSS biased_low_states CROSS
          biased_random_states (SUC (SUC n))),
       (\s.
          SIGMA
            (biased_dist (biased_high_states (SUC (SUC n)))
               biased_low_states (biased_random_states (SUC (SUC n))))
            s))

[biased_dc_prog_space_ind]  Theorem

   |- !P. P (SUC 0) /\ P 0 /\ (!n. P (SUC (SUC n))) ==> !v. P v

[card_dc_high_states_set3]  Theorem

   |- & (CARD (dc_high_states_set (SUC (SUC 0)))) = 3

[coin_out_of_range_eq_zero_dc_random_states]  Theorem
```

```
   |- !n s.
        s IN dc_random_states n ==>
        !i. n < i ==> ~s (STRCAT "coin" (toString i))

[compute_result_alt]  Theorem

   |- !low n.
        compute_result low n =
        (\s. (if s = "result" then XOR_announces low n else low s))

[dc3_leakage_result]  Theorem

   |- leakage (dc_prog_space (SUC (SUC (SUC 0))) F)
        (dcprog (SUC (SUC (SUC 0)))) =
      0

[dc_XOR_announces_result1]  Theorem

   |- !high low random n i.
        i <= SUC (SUC n) /\ high (STRCAT "pays" (toString i)) /\
        (!j. ~(j = i) ==> ~high (STRCAT "pays" (toString j))) ==>
        !k.
          k < i ==>
          (XOR_announces
             (set_announcements high low random (SUC (SUC n))
                (SUC (SUC n))) k =
           random (STRCAT "coin" (toString k)) xor
           random (STRCAT "coin" (toString (SUC (SUC n)))))

[dc_XOR_announces_result2]  Theorem

   |- !high low random n i.
        i <= SUC (SUC n) /\ high (STRCAT "pays" (toString i)) /\
        (!j. ~(j = i) ==> ~high (STRCAT "pays" (toString j))) ==>
        (XOR_announces
           (set_announcements high low random (SUC (SUC n))
              (SUC (SUC n))) i =
         ~(random (STRCAT "coin" (toString i)) xor
           random (STRCAT "coin" (toString (SUC (SUC n))))))

[dc_XOR_announces_result3]  Theorem

   |- !high low random n i.
        i <= SUC (SUC n) /\ high (STRCAT "pays" (toString i)) /\
        (!j. ~(j = i) ==> ~high (STRCAT "pays" (toString j))) ==>
        !k.
          i <= k /\ k <= SUC (SUC n) ==>
          (XOR_announces
             (set_announcements high low random (SUC (SUC n))
                (SUC (SUC n))) k =
           ~(random (STRCAT "coin" (toString k)) xor
             random (STRCAT "coin" (toString (SUC (SUC n))))))

[dc_XOR_announces_result4]  Theorem
```

```
   |- !high low random n i.
        i <= SUC (SUC n) /\ high (STRCAT "pays" (toString i)) /\
        (!j. ~(j = i) ==> ~high (STRCAT "pays" (toString j))) ==>
        XOR_announces
          (set_announcements high low random (SUC (SUC n))
             (SUC (SUC n))) (SUC (SUC n))

[dc_XOR_announces_result5]  Theorem

   |- !high low random n.
        high IN dc_high_states F (SUC (SUC (SUC n))) ==>
        XOR_announces
          (set_announcements high low random (SUC (SUC n))
             (SUC (SUC n))) (SUC (SUC n))

[dc_high_states_def]  Theorem

   |- dc_high_states nsapays (SUC (SUC n)) =
      (if nsapays then
         {(\s. s = STRCAT "pays" (toString (SUC (SUC n))))}
       else
         dc_high_states_set (SUC n))

[dc_high_states_ind]  Theorem

   |- !P.
        (!nsapays. P nsapays (SUC 0)) /\ (!nsapays. P nsapays 0) /\
        (!nsapays n. P nsapays (SUC (SUC n))) ==>
        !v v1. P v v1

[dc_high_states_set_finite]  Theorem

   |- !n. FINITE (dc_high_states_set n)

[dc_high_states_set_not_empty]  Theorem

   |- !n. ~(dc_high_states_set n = {})

[dc_leakage_result]  Theorem

   |- !n.
        leakage (dc_prog_space (SUC (SUC (SUC n))) F)
          (dcprog (SUC (SUC (SUC n)))) =
        0

[dc_low_states_not_empty]  Theorem

   |- ~(dc_low_states = {})

[dc_prog_space_F_set_thm]  Theorem

   |- !n.
        dc_prog_space (SUC (SUC n)) F =
        unif_prog_space (dc_high_states_set (SUC n)) dc_low_states
          (dc_random_states (SUC n))
```

```
[dc_prog_space_T_set_thm]  Theorem

   |- !n.
        dc_prog_space (SUC (SUC n)) T =
        unif_prog_space
          {(\s. s = STRCAT "pays" (toString (SUC (SUC n))))} {(\s. F)}
          (dc_random_states (SUC n))

[dc_prog_space_def]  Theorem

   |- dc_prog_space (SUC (SUC n)) nsapays =
      unif_prog_space (dc_high_states nsapays (SUC (SUC n)))
        dc_low_states (dc_random_states (SUC n))

[dc_prog_space_ind]  Theorem

   |- !P.
        (!v5. P (SUC 0) v5) /\ (!v3. P 0 v3) /\
        (!n nsapays. P (SUC (SUC n)) nsapays) ==>
        !v v1. P v v1

[dc_random_states_not_empty]  Theorem

   |- !n. ~(dc_random_states n = {})

[dc_set_announcements_result1]  Theorem

   |- !h l r n i.
        i <= SUC (SUC n) ==>
        (set_announcements h l r (SUC (SUC n)) i
           (STRCAT "announces" (toString 0)) =
         h (STRCAT "pays" (toString 0)) xor
         r (STRCAT "coin" (toString 0)) xor
         r (STRCAT "coin" (toString (SUC (SUC n)))))

[dc_set_announcements_result2]  Theorem

   |- !h l r n.
        set_announcements h l r (SUC (SUC n)) (SUC (SUC n))
          (STRCAT "announces" (toString 0)) =
        h (STRCAT "pays" (toString 0)) xor
        r (STRCAT "coin" (toString 0)) xor
        r (STRCAT "coin" (toString (SUC (SUC n))))

[dc_set_announcements_result3]  Theorem

   |- !h l r n m i.
        SUC i <= m /\ m < SUC (SUC (SUC n)) ==>
        (set_announcements h l r (SUC (SUC n)) m
           (STRCAT "announces" (toString (SUC i))) =
         h (STRCAT "pays" (toString (SUC i))) xor
         r (STRCAT "coin" (toString (SUC i))) xor
         r (STRCAT "coin" (toString i)))

[dc_set_announcements_result4]  Theorem
```

```
   |- !h l r n i.
        SUC i < SUC (SUC (SUC n)) ==>
        (set_announcements h l r (SUC (SUC n)) (SUC (SUC n))
           (STRCAT "announces" (toString (SUC i))) =
         h (STRCAT "pays" (toString (SUC i))) xor
         r (STRCAT "coin" (toString (SUC i))) xor
         r (STRCAT "coin" (toString i)))

[dc_set_announcements_result5]  Theorem

   |- !h l r n m s.
        m <= SUC (SUC n) /\
        (!i. i <= m ==> ~(s = STRCAT "announces" (toString i))) ==>
        (set_announcements h l r (SUC (SUC n)) m s = l s)

[dc_set_announcements_result6]  Theorem

   |- !h l r n.
        set_announcements h l r (SUC (SUC n)) (SUC (SUC n))
          (STRCAT "announces" (toString (SUC (SUC n)))) =
        h (STRCAT "pays" (toString (SUC (SUC n)))) xor
        r (STRCAT "coin" (toString (SUC (SUC n)))) xor
        r (STRCAT "coin" (toString (SUC n)))

[dc_states3_cross_not_empty]  Theorem

   |- ~(dc_high_states_set (SUC (SUC 0)) CROSS dc_low_states CROSS
        dc_random_states (SUC (SUC 0)) =
        {})

[dc_states_cross_not_empty]  Theorem

   |- !n.
        ~(dc_high_states_set (SUC (SUC n)) CROSS dc_low_states CROSS
          dc_random_states (SUC (SUC n)) =
          {})

[dc_valid_outputs_eq_outputs]  Theorem

   |- !n.
        IMAGE (\(h,r). dcprog (SUC (SUC (SUC n))) ((h,(\s. F)),r))
          (dc_high_states F (SUC (SUC (SUC n))) CROSS
           dc_random_states (SUC (SUC n))) =
        dc_valid_outputs (SUC (SUC n))

[dc_valid_outputs_list_def]  Theorem

   |- dc_valid_outputs_list (SUC (SUC (SUC n))) =
      MAP
        (\l s.
           (s = "result") \/
           (if s = STRCAT "announces" (toString (SUC (SUC n))) then
              ~XOR_announces l (SUC n)
            else
              l s)) (n_minus_1_announces_list (SUC n))
```

```
[dc_valid_outputs_list_ind]  Theorem

   |- !P.
        P (SUC (SUC 0)) /\ P (SUC 0) /\ P 0 /\
        (!n. P (SUC (SUC (SUC n)))) ==>
        !v. P v

[dcprog_def]  Theorem

   |- dcprog (SUC (SUC (SUC n))) =
      (\((high,low),random).
         compute_result
           (set_announcements high low random (SUC (SUC n))
              (SUC (SUC n))) (SUC (SUC n)))

[dcprog_ind]  Theorem

   |- !P.
        P (SUC (SUC 0)) /\ P (SUC 0) /\ P 0 /\
        (!n. P (SUC (SUC (SUC n)))) ==>
        !v. P v

[dcprog_result1]  Theorem

   |- !high low random n.
        dcprog (SUC (SUC (SUC n))) ((high,low),random) "result" =
        XOR_announces
          (set_announcements high low random (SUC (SUC n))
             (SUC (SUC n))) (SUC (SUC n))

[dcprog_result2]  Theorem

   |- !high low random n x.
        ~(x = "result") ==>
        (dcprog (SUC (SUC (SUC n))) ((high,low),random) x =
         set_announcements high low random (SUC (SUC n)) (SUC (SUC n))
           x)

[dcprog_result3]  Theorem

   |- !high low random n i.
        XOR_announces (dcprog (SUC (SUC (SUC n))) ((high,low),random))
          i =
        XOR_announces
          (set_announcements high low random (SUC (SUC n))
             (SUC (SUC n))) i

[dcprog_result4]  Theorem

   |- !high low random n.
        XOR_announces (dcprog (SUC (SUC (SUC n))) ((high,low),random))
          (SUC (SUC n)) =
        dcprog (SUC (SUC (SUC n))) ((high,low),random) "result"

[dcprog_result5]  Theorem
```

```
   |- !high low random n.
        high IN dc_high_states F (SUC (SUC (SUC n))) ==>
        dcprog (SUC (SUC (SUC n))) ((high,low),random) "result"

[dcprog_result6]  Theorem

   |- !high low random n i.
        dcprog (SUC (SUC (SUC n))) ((high,low),random)
          (STRCAT "announces" (toString i)) =
        set_announcements high low random (SUC (SUC n)) (SUC (SUC n))
          (STRCAT "announces" (toString i))

[insider_dc3_leakage_result]  Theorem

   |- leakage (insider_dc_prog_space (SUC (SUC (SUC 0))) F)
        (insider_dcprog (SUC (SUC (SUC 0)))) =
      0

[insider_dc_prog_space_F_set_thm]  Theorem

   |- !n.
        insider_dc_prog_space (SUC (SUC n)) F =
        unif_prog_space (insider_high_states_set (SUC n))
          insider_low_states (insider_random_states (SUC n))

[insider_dc_prog_space_def]  Theorem

   |- insider_dc_prog_space (SUC (SUC n)) nsapays =
      unif_prog_space (insider_high_states nsapays (SUC (SUC n)))
        insider_low_states (insider_random_states (SUC n))

[insider_dc_prog_space_ind]  Theorem

   |- !P.
        (!v5. P (SUC 0) v5) /\ (!v3. P 0 v3) /\
        (!n nsapays. P (SUC (SUC n)) nsapays) ==>
        !v v1. P v v1

[insider_dcprog_def]  Theorem

   |- insider_dcprog (SUC (SUC (SUC n))) =
      (\((high,low),random).
         compute_result
           (insider_set_announcements high low random (SUC (SUC n))
              (SUC (SUC n))) (SUC (SUC n)))

[insider_dcprog_ind]  Theorem

   |- !P.
        P (SUC (SUC 0)) /\ P (SUC 0) /\ P 0 /\
        (!n. P (SUC (SUC (SUC n)))) ==>
        !v. P v

[insider_high_states_def]  Theorem

   |- insider_high_states nsapays (SUC (SUC n)) =
```

```
        (if nsapays then
           {(\s. s = STRCAT "pays" (toString (SUC (SUC n))))}
         else
           insider_high_states_set (SUC n))

[insider_high_states_ind]  Theorem

   |- !P.
        (!nsapays. P nsapays (SUC 0)) /\ (!nsapays. P nsapays 0) /\
        (!nsapays n. P nsapays (SUC (SUC n))) ==>
        !v v1. P v v1

[insider_set_announcements_alt]  Theorem

   |- !high low random n i.
        (insider_set_announcements high low random n 0 =
         (\s.
            (if s = STRCAT "announces" (toString 0) then
               high (STRCAT "pays" (toString 0)) xor
               low (STRCAT "coin" (toString 0)) xor
               random (STRCAT "coin" (toString n))
             else
               low s))) /\
        (insider_set_announcements high low random n (SUC 0) =
         (\s.
            (if s = STRCAT "announces" (toString (SUC 0)) then
               high (STRCAT "pays" (toString (SUC 0))) xor
               random (STRCAT "coin" (toString (SUC 0))) xor
               low (STRCAT "coin" (toString 0))
             else
               insider_set_announcements high low random n 0 s))) /\
        (insider_set_announcements high low random n (SUC (SUC i)) =
         (\s.
            (if s = STRCAT "announces" (toString (SUC (SUC i))) then
               high (STRCAT "pays" (toString (SUC (SUC i)))) xor
               random (STRCAT "coin" (toString (SUC (SUC i)))) xor
               random (STRCAT "coin" (toString (SUC i)))
             else
               insider_set_announcements high low random n (SUC i) s)))

[insider_set_announcements_def]  Theorem

   |- (!s random n low high.
         insider_set_announcements high low random n 0 s =
         (if s = STRCAT "announces" (toString 0) then
            high (STRCAT "pays" (toString 0)) xor
            low (STRCAT "coin" (toString 0)) xor
            random (STRCAT "coin" (toString n))
          else
            low s)) /\
      (!s random n low high.
         insider_set_announcements high low random n (SUC 0) s =
         (if s = STRCAT "announces" (toString (SUC 0)) then
            high (STRCAT "pays" (toString (SUC 0))) xor
            random (STRCAT "coin" (toString (SUC 0))) xor
            low (STRCAT "coin" (toString 0))
```

```
          else
            insider_set_announcements high low random n 0 s)) /\
      !s random n low i high.
        insider_set_announcements high low random n (SUC (SUC i)) s =
        (if s = STRCAT "announces" (toString (SUC (SUC i))) then
           high (STRCAT "pays" (toString (SUC (SUC i)))) xor
           random (STRCAT "coin" (toString (SUC (SUC i)))) xor
           random (STRCAT "coin" (toString (SUC i)))
         else
           insider_set_announcements high low random n (SUC i) s)

[insider_set_announcements_ind]  Theorem

   |- !P.
        (!high low random n s. P high low random n 0 s) /\
        (!high low random n s.
           (~(s = STRCAT "announces" (toString (SUC 0))) ==>
            P high low random n 0 s) ==>
           P high low random n (SUC 0) s) /\
        (!high low random n i s.
           (~(s = STRCAT "announces" (toString (SUC (SUC i)))) ==>
            P high low random n (SUC i) s) ==>
           P high low random n (SUC (SUC i)) s) ==>
        !v v1 v2 v3 v4 v5. P v v1 v2 v3 v4 v5

[new_dc3_leakage_result]  Theorem

   |- leakage (dc_prog_space (SUC (SUC (SUC 0))) F)
        (dcprog (SUC (SUC (SUC 0)))) =
      0

[prob_space_biased_dc_prog_space23]  Theorem

   |- prob_space (biased_dc_prog_space2 (SUC (SUC 0)))

[prob_space_biased_dc_prog_space3]  Theorem

   |- prob_space (biased_dc_prog_space (SUC (SUC 0)))

[set_announcements_alt]  Theorem

   |- !high low random n i.
        (set_announcements high low random n 0 =
         (\s.
            (if s = STRCAT "announces" (toString 0) then
               high (STRCAT "pays" (toString 0)) xor
               random (STRCAT "coin" (toString 0)) xor
               random (STRCAT "coin" (toString n))
             else
               low s))) /\
        (set_announcements high low random n (SUC i) =
         (\s.
            (if s = STRCAT "announces" (toString (SUC i)) then
               high (STRCAT "pays" (toString (SUC i))) xor
               random (STRCAT "coin" (toString (SUC i))) xor
               random (STRCAT "coin" (toString i))
```

```
              else
                set_announcements high low random n i s)))

[valid_coin_assignment_eq_2_element_set]  Theorem

   |- !n p out.
        p <= SUC (SUC n) /\
        out IN
        IMAGE (\(h,r). dcprog (SUC (SUC (SUC n))) ((h,(\s. F)),r))
          (dc_high_states F (SUC (SUC (SUC n))) CROSS
           dc_random_states (SUC (SUC n))) ==>
        ({r |
          r IN dc_random_states (SUC (SUC n)) /\
          valid_coin_assignment r out p n (SUC (SUC n))} =
         coin_assignment_set out p n)

[valid_coin_set_eq_valid_coin_assignment]  Theorem

   |- !n p out.
        p <= SUC (SUC n) /\
        out IN
        IMAGE (\(h,r). dcprog (SUC (SUC (SUC n))) ((h,(\s. F)),r))
          (dc_high_states F (SUC (SUC (SUC n))) CROSS
           dc_random_states (SUC (SUC n))) ==>
        ({r |
          r IN dc_random_states (SUC (SUC n)) /\
          (dcprog (SUC (SUC (SUC n)))
             (((\s. s = STRCAT "pays" (toString p)),(\s. F)),r) =
           out)} =
         {r |
          r IN dc_random_states (SUC (SUC n)) /\
          valid_coin_assignment r out p n (SUC (SUC n))})
```

Bibliography

[1] The IOA language and toolset. http://groups.csail.mit.edu/tds/ioa/.

[2] LP: The Larch prover. http://nms.lcs.mit.edu/larch/LP/overview.html.

[3] P.G. Allen. A comparison of non-interference and non-deducibility using CSP. In *Proceedings of the 4th IEEE Computer Security Foundations Workshop*, pages 43–54. IEEE Computer Society Press, June 1991.

[4] Ross Anderson. Confidentiality and connecting for health. *British Journal of General Practice*, 58(547):75–76, February 2008.

[5] Ross Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley, 2nd edition, April 2008.

[6] Michael Backes. Quantifying probabilistic information flow in computational reactive systems. In Sabrina De Capitani di Vimercati, Paul Syverson, and Dieter Gollmann, editors, *Proceedings of the 10th European Symposium on Research in Computer Security*, volume 3679 of *Lecture Notes in Computer Science*, pages 336–354, Milan, Italy, September 2005. Springer, Berlin.

[7] Michael Backes and Birgit Pfitzmann. Computational probabilistic non-interference. In Dieter Gollmann, Günther Karjoth, and Michael Waidner, editors, *Proceedings of the 7th European Symposium on Research in Computer Security*, volume 2502 of *Lecture Notes in Computer Science*, pages 1–25, Zurich, Switzerland, October 2002. Springer, Berlin.

[8] Michael Backes and Birgit Pfitzmann. Intransitive non-interference for cryptographic purposes. In *Proceedings of the 24th IEEE Symposium on Security and Privacy*, pages 140–152, Berkeley, California, USA, May 2003. IEEE Computer Society.

[9] Ron Berman, Amos Fiat, and Amnon Ta-Shma. Provable unlinkability against traffic analysis. In Ari Juels, editor, *Proceedings of the 8th International Conference on Financial Cryptography*, volume 3110 of *Lecture Notes in Computer Science*, pages 266–280, Key West, Florida, USA, February 2004. Springer, Berlin.

[10] Mohit Bhargava and Catuscia Palamidessi. Probabilistic anonymity. In Martín Abadi and Luca de Alfaro, editors, *Proceedings of the 16th International Conference on Concurrency Theory*, volume 3653 of *Lecture Notes in Computer Science*, pages 171–185, San Francisco, California, USA, August 2005. Springer, Berlin.

[11] Józef Białas. The σ-additive measure theory. *Journal of Formalized Mathematics*, 2(2):263–270, March–April 1991.

[12] Józef Białas. Properties of Caratheodor's measure. *Journal of Formalized Mathematics*, 3(1):67–70, 1992.

[13] George Dean Bissias, Marc Liberatore, David Jensen, and Brian Neil Levine. Privacy vulnerabilities in encrypted HTTP streams. In George Danezis and David Martin, editors, *Proceedings of the 5th International Workshop on Privacy Enhancing Technologies*, volume 3856 of *Lecture Notes in Computer Science*, pages 1–11, Cavtat, Croatia, May–June 2005. Springer, Berlin.

[14] Bruno Blanchet. A computationally sound mechanized prover for security protocols. *IEEE Transactions on Dependable and Secure Computing*, 5(4):193–207, October–December 2008.

[15] Nikita Borisov. An analysis of parallel mixing with attacker-controlled inputs. In George Danezis and David Martin, editors, *Proceedings of the 5th International Workshop on Privacy Enhancing Technologies*, volume 3856 of *Lecture Notes in Computer Science*, pages 12–25, Cavtat, Croatia, May–June 2005. Springer, Berlin.

[16] Annalisa Bossi, Carla Piazza, and Sabina Rossi. Compositional information flow security for concurrent programs. *Journal of Computer Security*, 15(3):373–416, August 2007.

[17] Jan Camenisch and Anna Lysyanskaya. A formal treatment of onion routing. In Victor Shoup, editor, *Proceedings of the 25th Annual International Cryptology Conference*, volume 3621 of *Lecture Notes in Computer Science*, pages 169–187, Santa Barbara, California, USA, August 2005. Springer, Berlin.

[18] Konstantinos Chatzikokolakis. *Probabilistic and Information-Theoretic Approaches to Anonymity*. PhD thesis, Laboratoire d'Informatique (LIX), École Polytechnique, Paris, France, October 2007.

[19] Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Prakash Panangaden. Anonymity protocols as noisy channels. In Ugo Montanari, Donald Sannella, and Roberto Bruni, editors, *Proceedings of the 2nd Symposium on Trustworthy Global Computing*, volume 4661 of *Lecture Notes in Computer Science*, pages 281–300, Lucca, Italy, November 2006. Springer, Berlin.

[20] Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Prakash Panangaden. Probability of error in information-hiding protocols. In *Proceedings of the 20th IEEE Computer Security Foundations Symposium*, pages 341–354, Venice, Italy, July 2007. IEEE Computer Society, Los Alamitos, California, USA.

[21] Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Prakash Panangaden. On the Bayes risk in information-hiding protocols. *Journal of Computer Security*, 16(5):531–571, 2008.

[22] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, January 1988.

[23] David Chaum, Peter Y.Ã. Ryan, and Steve Schneider. A practical voter-verifiable election scheme. In Sabrina De Capitani di Vimercati, Paul Syverson, and Dieter Gollmann, editors, *Proceedings of the 10th European Symposium on Research in Computer Security*, volume 3679 of *Lecture Notes in Computer Science*, pages 118–139, Milan, Italy, September 2005. Springer, Berlin.

[24] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, February 1981.

[25] Tom Chothia. Analysing the MUTE anonymous file-sharing system using the pi-calculus. In Elie Najm, Jean-François Pradat-Peyre, and Véronique Viguié Donzeau-Gouge, editors, *Proceedings of the 26th IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems*, volume 4229 of *Lecture Notes in Computer Science*, pages 115–130, Paris, France, September 2006. Springer, Berlin.

[26] Tom Chothia, Simona Orzan, Jun Pang, and Mohammad Torabi Dashti. A framework for automatically checking anonymity with μCRL. In Ugo Montanari, Donald Sannella, and Roberto Bruni, editors, *Proceedings of the 2nd Symposium on Trustworthy Global Computing*, volume 4661 of *Lecture Notes in Computer Science*, pages 301–318, Lucca, Italy, November 2006. Springer, Berlin.

[27] David Clark and Sebastian Hunt. Non-interference for deterministic interactive programs. In Pierpaolo Degano, Joshua Guttman, and Fabio Martinelli, editors, *Proceedings of the 5th International Workshop on Formal Aspects in Security and Trust*, volume 5491 of *Lecture Notes in Computer Science*, pages 50–66, Malaga, Spain, October 2008. Springer, Berlin.

[28] David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantitative analysis of the leakage of confidential data. In Alessandra Di Pierro and Herbert Wiklicky, editors, *Proceedings of the ACM Workshop on Quantitative Aspects of Programming Languages*, volume 59 of *Electronic Notes in Theoretical Computer Science*, pages 238–251, Firenze, Italy, September 2001. Elsevier.

[29] David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantified interference for a while language. In Antonio Cerone and Alessandra Di Pierro, editors, *Proceedings of the 2nd Workshop on Quantitative Aspects of Programming Languages*, volume 112 of *Electronic Notes in Theoretical Computer Science*, pages 149–166, Barcelona, Spain, March 2004. Elsevier.

[30] David Clark, Sebastian Hunt, and Pasquale Malacaria. A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security*, 15(3):321–371, August 2007.

[31] Michael R. Clarkson, Andrew C. Myers, and Fred B. Schneider. Belief in information flow. In *Proceedings of the 18th IEEE Workshop on Computer Security Foundations*, pages 31–45, Aix-en-Provence, France, June 2005. IEEE Computer Society, Los Alamitos, California, USA.

[32] Sebastian Clauß and Stefan Schiffner. Structuring anonymity metrics. In *Proceedings of the 2nd ACM Workshop on Digital Identity Management*, pages 55–62, Alexandria, Virginia, USA, November 2006. ACM Press, New York, New York, USA.

[33] Richard Clayton and George Danezis. Chaffinch: Confidentiality in the face of legal threats. In Fabien A. P. Petitcolas, editor, *Proceedings of the 5th International Workshop on Information Hiding*, volume 2578 of *Lecture Notes in Computer Science*, pages 70–86, Noordwijkerhout, The Netherlands, January 2002. Springer, Berlin.

[34] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley and Sons, Inc., 1991.

[35] S. Crafa, M. Mio, M. Miculan, C. Piazza, and S. Rossi. PicNIc - pi-calculus non-interference checker. In Jonathan Billington, Zhenhua Duan, and Maciej Koutny, editors, *Proceedings of the 8th International Conference on Application of Concurrency to System Design*, pages 33–38, Xi'an, China, June 2008. IEEE Press.

[36] George Danezis. *Better Anonymous Communications*. PhD thesis, University of Cambridge, Cambridge, United Kingdom, July 2004.

[37] George Danezis and Claudia Diaz. A survey of anonymous communication channels. Technical Report MSR-TR-2008-35, Microsoft Research, Cambridge, United Kingdom, January 2008.

[38] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a Type III anonymous remailer protocol. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pages 2–15, Berkeley, California, USA, May 2003. IEEE Computer Society, Los Alamitos, California, USA.

[39] George Danezis and Len Sassaman. Heartbeat traffic to counter (n-1) attacks: Red-green-black mixes. In Pierangela Samarati and Paul Syverson, editors, *Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society*, pages 88–93, Washington, DC, USA, October 2003. ACM Press, New York, New York, USA.

[40] Yuxin Deng, Catuscia Palamidessi, and Jun Pang. Weak probabilistic anonymity. In Michael Backes and Andre Scedrov, editors, *Proceedings of the International Workshop on Security and Concurrency*, volume 180 of *Electronic Notes in Theoretical Computer Science*, pages 55–76, San Francisco, California, USA, August 2005. Elsevier.

[41] Yuxin Deng, Jun Pang, and Peng Wu. Measuring anonymity with relative entropy. In Theo Dimitrakos, Fabio Martinelli, Peter Y.Ã. Ryan, and Steve Schneider, editors, *Proceedings of the 4th International Workshop on Formal Aspects in Security and Trust*, volume 4691 of *Lecture Notes in Computer Science*, pages 65–79, Hamilton, Ontario, Canada, August 2006. Springer, Berlin.

[42] Dorothy E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, May 1976.

[43] Dorothy Elizabeth Robling Denning. *Cryptography and Data Security*. Addison-Wesley Publishing Company, Inc., June 1982.

[44] A. K. Dewdney. Computer recreations: Of worms, viruses, and core war. *Scientific American*, page 110, March 1989.

[45] Claudia Díaz. *Anonymity and Privacy in Electronic Services*. PhD thesis, Katholieke Universiteit Leuven, Leuven, Belgium, December 2005.

[46] Claudia Díaz and Bart Preneel. Reasoning about the anonymity provided by pool mixes that generate dummy traffic. In Jessica Fridrich, editor, *Proceedings of the 6th International Workshop on Information Hiding*, volume 3200 of *Lecture Notes in Computer Science*, pages 309–325, Toronto, Canada, May 2004. Springer, Berlin.

[47] Claudia Díaz and Bart Preneel. Taxonomy of mixes and dummy traffic. In Yves Deswarte, Frédéric Cuppens, Sushil Jajodia, and Lingyu Wang, editors, *Information Security Management, Education and Privacy, Proceedings of the 18th IFIP World Computer Congress, I-NetSec04 3rd Working Conference on Privacy and Anonymity in Networked and Distributed Systems*, pages 217–232, Toulouse, France, August 2004. Kluwer Academic Publishers.

[48] Claudia Díaz, Len Sassaman, and Evelyne Dewitte. Comparison between two practical mix designs. In Pierangela Samarati, Peter Ryan, Dieter Gollmann, and Refik Molva, editors, *Proceedings of the 9th European Symposium on Research in Computer Security*, volume 3193 of *Lecture Notes in Computer Science*, pages 141–159, Sophia Antipolis, France, September 2004. Springer, Berlin.

[49] Claudia Díaz and Andrei Serjantov. Generalising mixes. In Roger Dingledine, editor, *Proceedings of the 3rd International Workshop on Privacy Enhancing Technologies*, volume 2760 of *Lecture Notes in Computer Science*, pages 18–31, Dresden, Germany, March 2003. Springer, Berlin.

[50] Claudia Díaz, Stefaan Seys, Joris Claessens, and Bart Preneel. Towards measuring anonymity. In Roger Dingledine and Paul Syverson, editors, *Proceedings of the 2nd International Workshop on Privacy Enhancing Technologies*, volume 2482 of *Lecture Notes in Computer Science*, pages 184–188, San Francisco, California, USA, April 2002. Springer, Berlin.

[51] Edsger W. Dijkstra. *A Discipline of Programming*. Prentice-Hall Series in Automatic Computation. Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1976.

[52] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, San Diego, California, USA, August 2004. USENIX Association, Berkeley, California, USA.

[53] Roger Dingledine, Vitaly Shmatikov, and Paul Syverson. Synchronous batching: From cascades to free routes. In David Martin and Andrei Serjantov, editors, *Proceedings of 4th International Workshop on Privacy Enhancing Technologies*, volume 3424 of *Lecture Notes in Computer Science*, pages 186–206, Toronto, Canada, May 2004. Springer, Berlin.

[54] J. L. Doob. *Measure Theory*. Number 143 in Graduate Texts in Mathematics. Springer-Verlag, New York, New York, USA, 1991.

[55] Matthew Edman, Fikret Sivrikaya, and Bülent Yener. A combinatorial approach to measuring anonymity. In Gheorghe Muresan, Tayfur Altiok, Benjamin Melamed, and Daniel Zeng, editors, *Proceedings of the 2007 IEEE Conference on Intelligence and Security Informatics*, pages 356–363, New Brunswick, New Jersey, USA, May 2007. IEEE.

[56] Alexandre Evfimievski, Johannes Gehrke, and Ramakrishnan Srikant. Limiting privacy breaches in privacy preserving data mining. In *Proceedings of the 22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systesm*, pages 211–222, San Diego, California, USA, June 2003. ACM Press, New York, New York, USA.

[57] Nick Feamster and Roger Dingledine. Location diversity in anonymity networks. In Sabrina De Capitani di Vimercati and Paul Syverson, editors, *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*, pages 66–76, Washington, DC, USA, October 2004. ACM Press, New York, New York, USA.

[58] Riccardo Focardi and Roberto Gorrieri. A classification of security properties for process algebras. *Journal of Computer Security*, 3(1):5–34, November 1994.

[59] Riccardo Focardi, Roberto Gorrieri, and Fabio Martinelli. Information flow analysis in a discrete-time process algebra. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, pages 170–184, Cambridge, United Kingdom, July 2000. IEEE Computer Society, Los Alamitos, California, USA.

[60] Richard Forster. *Non-Interference Properties for Nondeterministic Processes*. PhD thesis, University of Oxford, Oxford, United Kingdom, January 1999.

[61] Matthias Franz, Bernd Meyer, and Andreas Pashalidis. Attacking unlinkability: The importance of context. In Nikita Borisov and Philippe Golle, editors, *Proceedings of the 7th International Symposium on Privacy Enhancing Technologies*, volume 4776 of *Lecture Notes in Computer Science*, pages 1–16, Ottawa, Canada, June 2007. Springer, Berlin.

[62] Flavio D. Garcia, Ichiro Hasuo, Wolter Pieters, and Peter van Rossum. Provable anonymity. In Ralf Küster and John Mitchell, editors, *Proceedings of the 2005 ACM Workshop on Formal Methods in Security Engineering*, pages 63–72, Fairfax, Virginia, USA, November 2005. ACM Press, New York, New York, USA.

[63] Lila L. Gatlin. *Information Theory and the Living System*. Columbia University Press, New York, New York, USA, 1972.

[64] Sharad Goel, Mark Robson, Milo Polte, and Emin Gun Sirer. Herbivore: A scalable and efficient protocol for anonymous communication. Technical Report 2003-1890, Cornell University, Ithaca, New York, USA, February 2003.

[65] J. A. Goguen and J. Meseguer. Security policies and security models. In *Proceedings of the 1982 IEEE Symposium on Security and Privacy*, pages 11–20, Oakland, California, USA, April 1982. IEEE Computer Society Press.

[66] Joseph A. Goguen and José Meseguer. Unwinding and inference control. In *Proceedings of the 1984 IEEE Symposium on Security and Privacy*, pages 75–87, Oakland, California, USA, April 1984. IEEE Computer Society Press.

[67] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Hiding routing information. In Ross Anderson, editor, *Proceedings of the 1st International Workshop on Information Hiding*, volume 1174 of *Lecture Notes in Computer Science*, pages 137–150, Cambridge, United Kingdom, May 1996. Springer, Berlin.

[68] Mike Gordon. From LCF to HOL: a short history. In Gordon Plotkin, Colin Stirling, and Mads Tofte, editors, *Proof, Language, and Interaction: Essays in Honour of Robin Milner*, pages 169–185. MIT Press, Cambridge, Massachusetts, USA, May 2000.

[69] Robert M. Gray. *Entropy and Information Theory*. Springer-Verlag, New York, New York, USA, 1990.

[70] J. W. Gray III. Toward a mathematical foundation for information flow security. *Journal of Computer Security*, 1(3):255–295, 1992.

[71] James W. Gray III. Probabilistic inference. In *Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 170–179, Oakland, California, USA, May 1990. IEEE Computer Society Press, Los Alamitos, California, USA.

[72] James W. Gray III. Toward a mathematical foundation for information flow security. In *Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 21–34, Oakland, California, USA, May 1991. IEEE Computer Society Press, Los Alamitos, California, USA.

[73] Silviu Guiasu. *Information Theory with Applications*. McGraw-Hill, New York, New York, USA, November 1977.

[74] Joseph Y. Halpern and Kevin R. O'Neill. Anonymity and information hiding in multiagent systems. *Journal of Computer Security*, 13(3):483–514, May 2005.

[75] Richard Wesley Hamming. *Coding and Information Theory*. Prentice-Hall Inc., Upper Saddle River, New Jersey, USA, 1980.

[76] René Rydhof Hansen and Christian W. Probst. Non-interference and erasure policies for java card bytecode. In *Proceedings of the 6th International Workshop on Issues in the Theory of Security*, Vienna, Austria, March 2006.

[77] John Harrison. *Theorem Proving with the Real Numbers*. Distinguished Dissertations. Springer-Verlag, Berlin, June 1998.

[78] Osman Hasan and Sofiène Tahar. Formalization of continuous probability distributions. In Frank Pfenning, editor, *Proceedings of the 21st International Conference on Automated Deduction*, volume 4603 of *Lecture Notes in Computer Science*, pages 3–18, Bremen, Germany, July 2007. Spinger, Berlin.

[79] Osman Hasan Hasan and Sofiène Tahar. Verification of expectation properties for discrete random variables in HOL. In Klaus Schneider and Jens Brandt, editors, *Proceedings of the 20th International Conference on Theorem Proving in Higher Order Logics*, volume 4732 of *Lecture Notes in Computer Science*, pages 119–134, Kaiserslautern, Germany, September 2007. Springer, Berlin.

[80] Ichiro Hasuo and Yoshinobu Kawabe. Probabilistic anonymity via coalgebraic simulations. In Rocco De Nicola, editor, *Proceedings of the 16th European Symposium on Programming*, volume 4421 of *Lecture Notes in Computer Science*, pages 379–394, Braga, Portugal, March–April 2007. Springer, Berlin.

[81] F. H. Hinsley and Alan Stripp, editors. *Codebreakers: The Inside Story of Bletchley Park*. Oxford University Press, Oxford, United Kingdom, 1993.

[82] Andrew Hinton, Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM: A tool for automatic verification of probabilistic systems. In Holger Hermanns and Jens Palsberg, editors, *Proceedings of the 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 3920 of *Lecture Notes in Computer Science*, pages 441–444, Vienna, Austria, March–April 2006. Spinger, Berlin.

[83] Dominic Hughes and Vitaly Shmatikov. Information hiding, anonymity and privacy: A modular approach. *Journal of Computer Security*, 12(1):3–36, 2004.

[84] Joe Hurd. *Formal Verification of Probabilistic Algorithms*. PhD thesis, University of Cambridge, Cambridge, United Kingdom, 2002.

[85] Joe Hurd, Annabelle McIver, and Carroll Morgan. Probabilistic guarded commands mechanized in HOL. *Theoretical Computer Science*, 346(1):95–112, November 2005.

[86] Yoshinobu Kawabe, Ken Mano, Hideki Sakurada, and Yasuyuki Tsukada. Theorem-proving anonymity of infinite-state systems. *Information Processing Letters*, 101(1):46–51, January 2007.

[87] Yoshinobu Kawabe and Hideki Sakurada. A formal approach to designing anonymous software. In Haeng-Kon Kim, Jiro Tanaka, Bran Malloy, Roger Lee, Chisu Wu, and Doo Kwon Baik, editors, *Proceedings of the 5th ACIS International Conference on Software Engineering Research, Management, and Applications*, pages 203–212, Busan, South Korea, August 2007. IEEE Computer Society, Los Alamitos, California, USA.

[88] Dogan Kesdogan, Dakshi Agrawal, and Stefan Penz. Limits of anonymity in open environments. In Fabien A. P. Petitcolas, editor, *Proceedings of the 5th International Workshop on Information Hiding*, volume 2578 of *Lecture Notes in Computer Science*, pages 53–69, Noordwijkerhout, The Netherlands, October 2002. Spinger, Berlin.

[89] A. N. Kolmogorov. *Foundations of the Theory of Probability*. Chelsea Publishing Company, New York, New York, USA, second edition, 1956.

[90] Andrei N. Kolmogorov. On the Shannon theory of information transmission in the case of continuous signals. *IRE Transactions on Information Theory*, 2(4):102–108, December 1956.

[91] Axel Legay, Andrzej S. Murawski, Joël Ouaknine, and James Worrell. On automated verification of probabilistic programs. In C. R. Ramakrishnan and Jakob Rehof, editors, *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *Lecture Notes in Computer Science*, pages 173–187, Budapest, Hungary, March 2008. Spinger, Berlin.

[92] Albert C. Leighton. Secret communication among the greeks and romans. *Technology and Culture*, 10(2):139–154, 1969.

[93] David R Lester. Topology in PVS: continuous mathematics with applications. In John Rushby and Natarajan Shankar, editors, *Proceedings of the 2nd Workshop on Automated Formal Methods*, pages 11–20, Atlanta, Georgia, November 2007. ACM Press, New York, New York, USA.

[94] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In Tiziana Margaria and Bernhard Steffen, editors, *Proceedings of the 2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166, Passau, Germany, March 1996. Springer, Berlin.

[95] Gavin Lowe. Quantifying information flow. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pages 18–31, Cape Breton, Nova Scotia, Canada, June 2002. IEEE Computer Society, Los Alamitos, California, USA.

[96] Gavin Lowe. Defining information flow quantity. *Journal of Computer Security*, 12(3–4):619–653, 2004.

[97] Gavin Lowe. Semantic models for information flow. *Theoretical Computer Science*, 315(1):209–256, May 2004.

[98] Gavin Lowe. On information flow and refinement-closure. In *Proceedings of the 7th International Workshop on Issues in the Theory of Security*, Braga, Portugal, March 2007.

[99] Pasquale Malacaria. Assessing security threats of looping constructs. In *Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 225–235, French Riviera, Nice, France, January 2007. ACM Press, New York, New York, USA.

[100] Paul Malliavin. *Integration and Probability*, volume 157 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, New York, USA, 1995.

[101] Heiko Mantel. Unwinding possibilistic security properties. In Frédéric Cuppens, Yves Deswarte, Dieter Gollmann, and Michael Waidner, editors, *Proceedings of the 6th European Symposium on Research in Computer Security*, volume 1895 of *Lecture Notes in Computer Science*, pages 238–254, Toulouse, France, October 2000. Spinger, Berlin.

[102] S. Mauw, J. H. S. Verschuren, and E.P. de Vink. A formalization of anonymity and onion routing. In Pierangela Samarati, Peter Ryan, Dieter Gollmann, and Refik Molva, editors, *Proceedings of the 9th European Symposium on Research in Computer Security*, volume 3193 of *Lecture Notes in Computer Science*, pages 109–124, Sophia Antipolis, France, September 2004. Spinger, Berlin.

[103] Daryl McCullough. Specifications for multi-level security and a hook-up property. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 161–166, Oakland, California, USA, April 1987. IEEE Computer Society Press, Washington, DC, USA.

[104] Annabelle McIver and Carroll Morgan. A probabilistic approach to information hiding. In Annabelle McIver and Carroll Morgan, editors, *Programming Methodology*, Monographs in Computer Science, pages 441–460. Springer-Verlag, New York, New York, USA, 2003.

[105] Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Monographs in Computer Science. Springer-Verlag, New York, New York, USA, 2005.

[106] John McLean. Security models and information flow. In *Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 180–187, Oakland, California, USA, May 1990. IEEE Computer Society Press, Los Alamitos, California, USA.

[107] Jonathan K. Millen. Covert channel capacity. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 60–66, Oakland, California, USA, April 1987. IEEE Computer Society Press, Washington, DC, USA.

[108] Ulf Möller, Lance Cottrell, Peter Palfrader, and Len Sassaman. Mixmaster Protocol — Version 2. IETF Internet Draft, July 2003.

[109] Carroll Morgan. The shadow knows: Refinement of ignorance in sequential programs. In Tarmo Uustalu, editor, *Proceedings of the 8th International Conference on the Mathematics of Program Construction*, volume 4014 of *Lecture Notes in Computer Science*, pages 359–378, Kuressaare, Estonia, July 2006. Spinger, Berlin.

[110] Steven J. Murdoch and George Danezis. Low-cost traffic analysis of Tor. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 183–195, Oakland, California, USA, May 2005. IEEE Computer Society, Los Alamitos, California, USA.

[111] Andrzej Nędzusiak. σ-fields and probability. *Journal of Formalized Mathematics*, 1, 1989.

[112] Andrzej Nędzusiak. Probability. *Journal of Formalized Mathematics*, 1(4):745–749, 1990.

[113] Richard E. Newman, Ira S. Moskowitz, Paul Syverson, and Andrei Serjantov. Metrics for traffic analysis prevention. In Roger Dingledine, editor, *Proceedings of the 3rd International Workshop on Privacy Enhancing Technologies*, volume 2760 of *Lecture Notes in Computer Science*, pages 48–65, Dresden, Germany, March 2003. Spinger, Berlin.

[114] Michael Norrish and Konrad Slind. *The HOL System: Description*. http://prdownloads.sourceforge.net/hol/kananaskis-4-description.pdf, January 2007.

[115] Michael Norrish and Konrad Slind. *The HOL System: Logic*. http://prdownloads.sourceforge.net/hol/kananaskis-4-logic.pdf, January 2007.

[116] Michael Norrish and Konrad Slind. *The HOL System: Reference*. http://prdownloads.sourceforge.net/hol/kananaskis-4-reference.pdf, January 2007.

[117] Michael Norrish and Konrad Slind. *The HOL System: Tutorial*. http://prdownloads.sourceforge.net/hol/kananaskis-4-tutorial.pdf, January 2007.

[118] Kevin R. O'Neill, Michael R. Clarkson, and Stephen Chong. Information-flow security for interactive programs. In *Proceedings of the 19th IEEE Computer Security Foundations Workshop*, pages 12–23, Venice, Italy, July 2006. IEEE Computer Society, Los Alamitos, California, USA.

[119] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1–2):85–128, September 1998.

[120] Andreas Pfitzmann and Marit Köhntopp. Anonymity, unobservability, and pseudonymity — a proposal for terminology. In Hannes Federrath, editor, *Proceedings of the International Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *Lecture Notes in Computer Science*, pages 1–9, Berkeley, California, USA, July 2000. Spinger, Berlin.

[121] Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky. Approximate non-interference. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pages 3–17, Cape Breton, Nova Scotia, Canada, June 2002. IEEE Computer Society, Los Alamitos, California, USA.

[122] Mark Semenovich Pinsker. *Information and information stability of random variables and processes.* Holden-Day, San Francisco, 1964.

[123] Josyula R. Rao and Pankaj Rohatgi. Can pseudonymity really guarantee privacy? In *Proceedings of the 9th USENIX Security Symposium*, pages 85–96, Denver, Colorado, USA, August 2000. USENIX Association, Berkeley, California, USA.

[124] Jean-François Raymond. Traffic analysis: Protocols, attacks, design issues, and open problems. In Hannes Federrath, editor, *Proceedings of the International Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *Lecture Notes in Computer Science*, pages 10–29, Berkeley, California, USA, July 2000. Spinger, Berlin.

[125] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, November 1998.

[126] Marc Rennhard and Bernhard Plattner. Introducing MorphMix: Peer-to-peer based anonymous internet usage with collusion detection. In *Proceedings of the 2002 ACM Workshop on Privacy in the Electronic Society*, pages 91–102, Washington, DC, USA, November 2002. ACM Press, New York, New York, USA.

[127] Alfréd Rényi. On measures of entropy and information. In *Proceedings of the Fourth Berkeley Symposium on Mathematics, Statistics, and Probability*, volume 1, pages 547–561, Berkeley, California, USA, 1961. University of California Press, California, USA.

[128] Stefan Richter. Formalizing integration theory, with an application to probabilistic algorithms. Master's thesis, Technische Universität München, Munich, Germany, 2003.

[129] A. W. Roscoe. CSP and determinism in security modelling. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy*, pages 114–127, Oakland, California, USA, May 1995. IEEE Computer Society, Los Alamitos, California, USA.

[130] A. W. Roscoe, J. C. P. Woodcock, and L. Wulf. Non-interference through determinism. In Dieter Gollmann, editor, *Proceedings of the 3rd European Symposium on Research in Computer Security*, volume 875 of *Lecture Notes in Computer Science*, pages 31–53, Brighton, United Kingdom, November 1994. Spinger, Berlin.

[131] P Y A Ryan and S A Schneider. Process algebra and non-interference. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, pages 214–227, Mordano, Italy, June 1999. IEEE Computer Society, Los Alamitos, California, USA.

[132] Andrei Sabelfeld and David Sands. Probabilistic noninterference for multi-threaded programs. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, pages 200–214, Cambridge, United Kingdom, July 2000. IEEE Computer Society, Los Alamitos, California, USA.

[133] Steve Schneider and Abraham Sidiropoulos. CSP and anonymity. In Elisa Bertino, Helmut Kurth, Giancarlo Martella, and Emilio Montolivo, editors, *Proceedings of the 4th European Symposium on Research in Computer Security*, volume 1146 of *Lecture Notes in Computer Science*, pages 198–218, Rome, Italy, September 1996. Spinger, Berlin.

[134] Andrei Serjantov. *On the Anonymity of Anonymity Systems.* PhD thesis, University of Cambridge, Cambridge, United Kingdom, June 2004.

[135] Andrei Serjantov. A fresh look at the generalised mix framework. In Nikita Borisov and Philippe Golle, editors, *Proceedings of the 7th International Symposium on Privacy Enhancing Technologies*, volume 4776 of *Lecture Notes in Computer Science*, pages 17–29, Ottawa, Canada, June 2007. Springer, Berlin.

[136] Andrei Serjantov and George Danezis. Towards an information theoretic metric for anonymity. In Roger Dingledine and Paul Syverson, editors, *Proceedings of Privacy Enhancing Technologies Workshop (PET 2002)*, volume 2482 of *Lecture Notes in Computer Science*, pages 259–263, San Francisco, California, USA, April 2002. Spinger, Berlin.

[137] Andrei Serjantov, Roger Dingledine, and Paul Syverson. From a trickle to a flood: Active attacks on several mix types. In Fabien A. P. Petitcolas, editor, *Proceedings of the 5th International Workshop on Information Hiding*, volume 2578 of *Lecture Notes in Computer Science*, pages 36–52, Noordwijkerhout, The Netherlands, October 2002. Spinger, Berlin.

[138] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948.

[139] Claude E. Shannon and Warren Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, 1949.

[140] Vitaly Shmatikov. Probabilistic analysis of anonymity. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pages 119–128, Cape Breton, Nova Scotia, Canada, June 2002. IEEE Computer Society, Los Alamitos, California, USA.

[141] Vitaly Shmatikov. Probabilistic model checking of an anonymity system. In Steve Schneider, editor, *Journal of Computer Security*, volume 12, pages 355–377, 2004.

[142] Vitaly Shmatikov and Ming-Hsui Wang. Measuring relationship anonymity in mix networks. In *Proceedings of the 5th ACM Workshop on Privacy in Electronic Society*, pages 59–62, Alexandria, Virginia, USA, October 2006. ACM Press, New York, New York, USA.

[143] Geoffrey Smith. On the foundations of quantitative information flow. In Luca de Alfaro, editor, *Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures*, volume 5504 of *Lecture Notes in Computer Science*, pages 288–302, York, United Kingdom, March 2009. Spinger, Berlin.

[144] Sandra Steinbrecher and Stefan Köpsell. Modelling unlinkability. In Roger Dingledine, editor, *Proceedings of the 3rd International Workshop on Privacy Enhancing Technologies*, volume 2760 of *Lecture Notes in Computer Science*, pages 32–47, Dresden, Germany, March 2003. Spinger, Berlin.

[145] David Sutherland. A model of information. In *Proceedings of the 9th National Security Conference*, pages 175–183, Gaithersburg, Maryland, USA, September 1986.

[146] J. M. Swart. A conditional product measure theorem. *Statistics and Probability Letters*, 28:131–135, June 1996.

[147] Paul F. Syverson, David M. Goldschlag, and Michael G. Reed. Anonymous connections and onion routing. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 44–54, Oakland, California, USA, May 1997. IEEE Computer Society Press, Los Alamitos, California, USA.

[148] Paul F. Syverson and Stuart G. Stubblebine. Group principals and the formalization of anonymity. In *Proceedings of the World Congress on Formal Methods*, volume 1708 of *Lecture Notes in Computer Science*, pages 814–833, Toulouse, France, September 1999. Spinger, Berlin.

[149] Parisa Tabriz and Nikita Borisov. Breaking the collusion detection mechanism of MorphMix. In George Danezis and Philippe Golle, editors, *Proceedings of the 6th International Workshop on Privacy Enhancing Technologies*, volume 4258 of *Lecture Notes in Computer Science*, pages 368–383, Cambridge, United Kingdom, June 2006. Spinger, Berlin.

[150] Gergely Tóth and Zoltán Hornák. Measuring anonymity in a non-adaptive, real-time system. In David Martin and Andrei Serjantov, editors, *Proceedings of the 4th International Workshop on Privacy Enhancing Technologies*, volume 3424 of *Lecture Notes in Computer Science*, pages 226–241, Toronto, Canada, May 2004. Spinger, Berlin.

[151] Gergely Tóth, Zoltán Hornák, and Ferenc Vajda. Measuring anonymity revisited. In Sanna Liimatainen and Teemupekka Virtanen, editors, *Proceedings of the Ninth Nordic Workshop on Secure IT Systems*, pages 85–90, Espoo, Finland, November 2004.

[152] Ron van der Meyden and Kaile Su. Symbolic model checking the knowledge of the dining cryptographers. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop*, pages 280–291, Pacific Grove, California, USA, June 2004. IEEE Computer Society, Los Alamitos, California, USA.

[153] Dennis Volpano and Geoffrey Smith. Probabilistic noninterference in a concurrent language. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop*, pages 34–43, Rockport, Massachusetts, June 1998. IEEE Computer Society, Los Alamitos, California, USA.

[154] Stan Wagon. *The Banach-Tarski Paradox*. Cambridge University Press, Cambridge, United Kingdom, September 1993.

[155] David Williams. *Probability with Martingales*. Cambridge Mathematical Textbooks. Cambridge University Press, Cambridge, United Kingdom, February 1991.

[156] J. Todd Wittbold and Dale M. Johnson. Information flow in nondeterministic systems. In *Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 144–161, Oakland, California, USA, May 1990. IEEE Computer Society Press, Los Alamitos, California, USA.

[157] Aris Zakinthinos and E. S. Lee. A general theory of security properties. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 94–102, Oakland, California, USA, May 1997. IEEE Computer Society Press, Los Alamitos, California, USA.