Concepts and Calculation in Cryptography

A thesis submitted to
The University of Kent
in the subject of Computer Science
for the degree
of Doctor of Philosophy

By
Dan Grundy

Submitted March 2008

Abstract

This is a study about applying ideas from mathematical methodology to problems in cryptography. It is not a study of cryptography per se, but rather a study of the type of concepts one finds in this area, how they are formulated, and how we reason about them.

The motivation? Cryptography is a notoriously difficult subject to reason about: it is acknowledged within the cryptography community that many of the existing proofs are so complicated that they are near impossible to verify. The question then, is why? What is the source of the difficulty, and what can be done about it?

I claim that a large part of the difficulty arises from the non-avoidance of pitfalls such as over-specific and often ambiguous nomenclature, reliance on unstated domain specific knowledge and assumptions, and poorly structured, informal reasoning. The purpose of this study is to justify this claim, by exploring two fundamental cryptographic concepts (more accurately, two versions of a particular cryptographic concept) , and a proof of a theorem that relates them.

Declaration of originality

This thesis contains no material that has been submitted previously for the award of any other academic degree. Some of the material from Chapter 2 and Chapter 6 (specifically, introductory material on cryptography and reduction) appears in a similar form in the paper "Reduction and Refinement" [6] , cowritten with my PhD supervisor, Eerke Boiten. Material at the end of Chapter 4 is based on the paper "Towards Calculational Asymptotics" [7] , also cowritten with Eerke Boiten. As far as I'm aware, all relevant sources have been acknowledged, and except where otherwise indicated, this thesis is my own work.

Acknowledgements

First and foremost, thank-you to my supervisor, Eerke Boiten, to my wonderful and definitely better half, Anna, to her parents, and of course, to my parents, all of whom believed in me when I didn't. Without their patience, support, guidance, and encouragement I would not have reached this point.

Thanks to Philipp Mohr and Christopher Brown for their friendship and their patience when listening to me rambling on about my views on mathematics; their willingness to listen was of great help to me in formulating much of the first part of this thesis. My thanks also to my good friend Jonny Hughes, who has helped to keep me (moderately) sane over the last few years. Working on a PhD can be a very isolating experience, but it makes you feel lucky to have good friends.

Thanks to Roland Backhouse, Edsger Dijkstra, Wim Feijen, Netty van Gasteren, David Gries, Eric Hehner, and Fred Schneider, for teaching me through their writings how to do mathematics; anyone familiar with their work will have no difficulty observing their influence on my approach to mathematics. In particular, I am extremely grateful to Roland Backhouse for acting as my external thesis examiner, and to Wim Feijen for the opportunity to spend a month in Eindhoven in 2006, which allowed me to meet Jeremy Weissmann and Apurva Mehta, to whom thanks are due for many intellectually stimulating conversations.

Thanks also to John Derrick, Andy King, and Rogerio de Lemos, for many useful and encouraging conversations. These people have helped and taught me more than they realise; I feel privileged to have been a member (both as an undergraduate and as a postgraduate) of a department with so many helpful, knowledgeable people. Also, my thanks to Chris Woodcock for a number of interesting conversations, for acting as internal thesis examiner, and for his help with the proof of a theorem in Chapter 4 .

A special word of thanks is due to Dave Lewis for being an inspiring teacher, and for awakening my interest in mathematics and the more theoretical side of computing science while I was a student at Canterbury Christ Church University.

Finally, a very special thank-you to Kane, for always listening and never questioning, and for never failing to cheer me up when things get me down.

Contents

Chapter 0

Introduction

This is a study about applying ideas from mathematical methodology to problems in cryptography. It is not a study of cryptography per se, though exposition plays an important role, but rather a study of the type of concepts one finds in this area, how they are formulated, and how we reason about them. With that in mind, no prior familiarity with cryptography is assumed or required, though I do assume a working knowledge of predicate calculus and basic algebra. For the record, I see as my target audience mathematicians and computing scientists with an interest in mathematical methodology, non-cryptographers looking to gain insight into the foundations of cryptography, and cryptographers with an interest in improving upon the status quo within their discipline.

The incentive for this study arose out of a long standing personal interest in cryptography, and a growing interest in mathematical methodology, where the latter may be defined as "how to organise a detailed argument so as to keep it manageable" , or more succinctly: "how not to make a mess of things" . Cryptography, being an inherently mathematical topic, provides a rich and —should one be so inclined— practical source of problems that are notoriously difficult to reason about. Indeed, it is acknowledged within the cryptography community that many of the existing proofs are so complicated that they are near impossible to verify. But why is cryptography such a difficult subject to reason about? What is the source of the difficulty, and what can be done about it?

Dijkstra et al showed us the deep connection between program and proof construction, and promoted the use of formal techniques and the necessity of simple, elegant solutions, paving the way by showing how, for example, the use of calculation, the avoidance of unmastered complexity, and the careful introduction of notational conventions allow us to derive such solutions. I claim that a large part of the difficulty in constructing and verifying proofs in cryptography arises from the non-avoidance of these pitfalls, over-specific and often ambiguous nomenclature, reliance on unstated domain specific knowledge and assumptions, and poorly structured, informal reasoning. The purpose of this study is to justify this claim, and to explore to what extent the difficulties can be resolved.

My original, admittedly rather naive intention, was to dip in to cryptography, as it were, selecting a number of proofs from across the board involving a variety of concepts, and to try to

clean them up. Unfortunately, the problems seemed to run so deep that it soon became apparent that that approach was simply not practical. Instead, I take two fundamental cryptographic concepts (more accurately, two versions of a particular cryptographic concept) and explore their formulation and a proof of a non-trivial, important theorem that relates them.

* *

Structure and scope

When building a mathematical theory, the standard tactic is to first introduce definitions that describe the concept under study, and then, using those definitions, to build a library of theorems about that concept. However, in this study the goal is to explore and to identify difficulties with an existing piece of theory, rather than to construct a new theory, so a rather different approach is adopted.

Taking a bird's-eye view, I first explore some of the lessons learnt from mathematical methodology, and the pitfalls to avoid if we are to construct simple, elegant arguments. I then investigate how and why two fundamental concepts from cryptography, and a proof that relates them, fail to avoid these pitfalls.

Since the goal is to explore the type of mathematical concepts and reasoning that arise in cryptography, I first explore rather more generally what is meant by a "mathematical concept", and how we may formulate and reason about them. This includes a discussion of the use of formal versus informal reasoning, and the benefits of "calculation", an algebraic style of reasoning that emphasises the syntactic manipulation of parsed, but otherwise uninterpreted formulae.

Remark. As is perhaps already clear, I'm a proponent of the Dutch, calculational style of reasoning pioneered by Edsger Dijkstra and others. The decision to do mathematics this way is of course a personal choice, though in my discussion of calculational mathematics I try to convey some of what I see as the advantages of this style of reasoning, and hence what led me to choose this approach to mathematics.
End of Remark.

In order to reap the advantages of the calculational style, and to "let the symbols do the work", certain notational pitfalls must be avoided. I describe some notational choices and heuristics that can help achieve this goal and make formalism a pleasure to work with. I also briefly explore the format usually adopted for calculational proofs, heuristics for proof design, and the role "context" plays in theorem proving.

Having explored a little of what mathematics is about, how we may "do" mathematics, and in particular, how I choose to do mathematics, in Chapter 2 I explain —albeit briefly— what cryptography is about, setting the stage for the cryptographic concepts at the centre of this study, namely so-called "weak" and "strong" "one-way functions", and a theorem, along with its proof, that relates them.

Informally, cryptography is the study of constructions where some of the computations involved are deliberately and demonstrably "hard", while others are deliberately "easy". One-way functions are functions that are "easy" to compute but "hard" on average to invert, where strong one-way functions are computationally harder to invert than weak one-way functions. The theorem that relates these concepts asserts that the existence of weak one-way functions equivales the existence of strong one-way functions.

The definitions of one-way functions and the proof of the theorem that relates them are taken from the first volume of Oded Goldreich's two volume work "The Foundations of Cryptography" [27, 28]. I first examine how these definitions are formulated, focusing on the notation used, the concepts involved, and ambiguities or potential sources of confusion. I then explore Goldreich's proof, identifying structural issues and gaps in the reasoning where domain specific knowledge is implicitly appealed to.

I want to make it clear that it is not my intention to "pick on" Goldreich's work: his textbooks are widely acknowledged as both the standard introductory texts, and the standard reference texts on theoretical cryptography, making them the natural choice for my exploration of the concepts and style of theorem proving in this area; that Goldreich has attempted to present results in a manner that is more accessible than in the research literature makes the choice even more compelling; as Goldreich points out:

> I felt that I've based my career on work done in this area, but this work is quite inaccessible (especially to beginners) due to unsatisfactory presentation. I felt that it is my duty to redeem this sour state of affairs, and I now feel great thinking that I've done it! [29]

My goal is not to show that Goldreich has failed to redeem this —indeed "sour"— state of affairs, after all, I applaud his efforts to make the subject more accessible, rather, my goal is to establish what can be done to further improve the situation, by identifying what I see as the remaining difficulties.

The main concepts that underly one-way functions include probability theory, asymptotics, and complexity theory. In chapters 3, 4, and 5, I explore each of these in turn in more depth, with a view to understanding why one-way functions are defined the way they are, identifying further problems, and filling in the gaps in Goldreich's proof.

Having explored the concepts that underly the definitions of one-way functions, in Chapter 6 I explore the structure of Goldreich's proof, and in particular the use and validity of proof by contradiction, and proof by reduction, a technique used to reason about the complexity of one problem relative to another problem, or class of problems. In Chapter 7 I reintroduce the definitions of one-way functions using the alternative notation explored in the previous chapters, and restructure Golreich's proof to avoid the previously identified difficulties.

Finally, in Chapter 8, I reflect on my goals for this work, on what I have achieved with respect to those goals, on the necessity of the various concepts that underly the definitions of one-way functions, how they fit together, and how they affect our ability to reason about one-way functions; I also reflect on how some of the difficulties can be avoided, and how others —it

would seem— cannot. I close with suggestions for future research.

* *

Typesetting conventions

Readers will observe that I break with tradition when it comes to typesetting. In general this makes my job as an author much harder, but I honestly believe it makes life easier for my readers, and that can only be a good thing. Here I describe a few specific cases where I abandon standard conventions, other divergences from the norm, in particular concerning mathematical notation, are dealt with at the appropriate point in the text.

I omit the periods in the abbreviations "i.e." , "e.g." , "et al." , and "etc." , and instead write "ie" , "eg" , "et al" , and "etc" . Here I am following Jeremy Weissmann, who justifies this convention (in a private email) as follows:

> Punctuation is too important to waste periods, so I created a new word "ie" . Same with "eg" and "etc" .

I also write "viz" instead of "viz." .

I double space all punctuation marks except commas. So for example, whereas the convention is to use a single space following the full-stop in

> A sentence. Another sentence. . .

I use two spaces:

> A sentence. Another sentence. . .

The same applies to colons, semicolons, question marks, and exclamation marks. For punctuation symbols that come in pairs, ie quotes and parentheses, I allocate an extra space either side. When using em dashes to set off parenthetic remarks, the convention is to place spaces either side of the dashes, as in

> this — for example — is not very pleasant

or to omit the spaces:

> this—for example—is not very pleasant

Both approaches make it hard for the reader to determine whether an em dash begins or ends a parenthetic remark, particularly if the remark spans multiple lines. To remedy this problem, following Dijkstra et al, I "bind" the dashes to the remark by double spacing on one side of each dash:

this —for example— is rather pleasant

As Gries and Schneider point out:

> Parenthetical remarks delimited by parentheses (like this one) have a space on one side of each parenthesis, so why not parenthetical remarks delimited by em dashes? [32]

I also allocate extra space around mathematical formulae; so, I would typeset

some text $x^2 + y$. some more text

as:

some text $x^2 + y$. some more text

In general I try to avoid mixing text and mathematics; when it comes to typesetting mathematics my motto is "never underestimate the importance of whitespace!" Other issues related to typesetting mathematical formulae are dealt with later in the text.

* * *

Chapter 1

Mathematics and mathematical methodology

For the most part, mathematics is about exploring "concepts" by investigating and "proving" their properties. This usually means starting from a collection of properties, called "postulates", that characterise the concept under study, and then proving additional properties that follow from the base properties (ie, the postulates). These ideas are expanded below.

* *

Concepts and interfaces

A "concept" is an abstract idea, a general notion. Being an abstract idea, a concept is independent of language: the concept "death", for example, can be expressed in —I suspect— every natural language known to man. One way of viewing concepts is as collections of properties, where, rather than reasoning about the concept directly, because usually that's too difficult, we instead name it and list its properties, which are usually described in terms of other, more familiar, concepts, and reason in terms of those properties; in other words, we explore concepts by investigating their properties. Taking this view, to "define" a concept is to give it a name and list its salient properties.

Human concepts, such as love, are usually imprecise: trying to come up with a list of properties that accurately characterise the concept of "love" is likely to be difficult, if not impossible, as people are unlikely to agree not only on the list, but also on how to describe many of the properties. Consequently, when reasoning about human concepts we have to appeal to some common or intuitive understanding; most of the time this works well enough, but often it leads to misunderstanding. The beauty of mathematical concepts is that they tend to comprise properties that exhibit more structure and are easier to articulate. Therefore, a mathematical definition of a concept is a precise description of the properties an object (in the mathematical sense of the word) must possess in order to be called an instance of that concept. These base

properties are called "postulates".

We may view a collection of postulates as a template that describes the "basic shape" of the concept under study: the postulates define the structure of the domains we are interested in, where theorems proved from the postulates hold for any domain that satisfies those postulates. The postulates are in a sense the minimum requirements, since pretty much any domain we care to choose is likely to have additional properties not derivable from the postulates. Postulates, then, can be seen as requirements, and theorems can be seen as observations that follow from those requirements.

When formulating a mathematical definition we may have to decide between various equivalent collections of postulates. Our decision will usually be influenced by purpose —clarification versus manipulation, for example—, but whatever collection of postulates we choose forms an interface between us and the concept under study.

An "interface" is a medium through which two things interact. It is through interfaces that we reason about and communicate concepts. I mentioned above that concepts are independent of language, what I meant was that language is a particular interface we may use to interact with concepts, the use of "natural" language being one example of how we form interfaces; we also use formal languages, sign language, body language, and so on. Artists often explore concepts using alternative interfaces, such as painting, sculpture, photography, and music. Clearly we use interfaces all the time, usually without even realising, but by improving our awareness of how we form and use interfaces, we can question their appropriateness, and where necessary we can refine those interfaces in order to improve our communication and reasoning skills.

We often arrive at a particular collection of postulates, and hence a particular interface, through a process of "abstraction". Abstraction is a method of simplification where we introduce a "new" concept by focusing on certain properties (of some concept) while ignoring others. The resulting concept is said to be "more abstract" than the original concept, and the original concept is said to be "more concrete", or an "instantiation" (or an "instance") of the new, abstracted concept.

We use abstraction all the time when dealing with concepts, usually implicitly. Consider, for example, the concept "car". Cars comprise many properties, such as make, model, number of doors, colour, and so on. But in order to discuss the performance of a car, or cars in general, colour, for example, is of no relevance; hence we —implicitly— perform an abstraction, focusing only on the details relevant to the discussion.

Abstraction, then, is about ignoring differences that can be regarded as irrelevant. By restricting ourselves to a smaller set of properties our domain of discourse is both simplified and made more general. So, returning to the example, a blue car is clearly a car, but not all cars are blue, so the concept "car" is more abstract than the concept "blue car", but by ignoring colour we can reason about a wider range of cars.

The beauty of abstraction is that it allows us to focus on a collection of useful or interesting properties, but in such a way that anything we can prove about the abstracted concept on the basis of those properties, will hold also for any instance of that concept (and hence the original concept), meaning we can study the more abstract concept independently of the original concept.

In mathematics we use abstraction explicitly to discover collections of postulates that characterise new or existing concepts. For example, new concepts may be "discovered" by observing a collection of properties common to a class of objects, and performing an abstraction by extracting those common properties and promoting them to a collection of postulates. The new collection of postulates are explored in their own right, and the resulting theory applies to any object that satisfies those postulates.

Often we have a particular concept in mind that we want to study, in which case we may try to design a useful collection of postulates that characterises that concept. Typically we proceed by selecting an object that deserves to be called an instance of that concept, and then build a library of elementary properties of that object. Once the library is sufficient, we perform an abstraction, making our library of theorems a library of postulates.

When selecting a collection of postulates we should be mindful that the "stronger", or the more specific the postulates, the stronger the theorems we can prove, but at the loss of generality; conversely, the "weaker", or less specific the postulates, the more general the theory. An example of this can be seen in the progression from the naturals to the complex numbers via the integers, the rationals, and the reals: as we gain "solutions" we lose laws.

Having decided on which interface (ie, which collection of postulates) will best serve our requirements, we have many ways of writing down that interface; that is, we have another interface to consider, viz the notation. Consequently, the question of how we should define the concepts we want to study is really a question about interface design: we must decide on both a suitable collection of postulates, and on an appropriate way to write down those postulates, where our choices are likely to have a significant impact on our ability to reason about the concept under study.

So how we should go about discovering and writing down our postulates, and how should we conduct our reasoning in order to communicate our findings with others and to convince them of the validity of our claims? There are essentially two approaches to mathematics: formal and informal. To clarify the distinction between the two, and why we may choose one approach over the other, it is instructive to explore how mathematics has evolved, and in particular, how attitudes have changed over how to formulate mathematical concepts, and what constitutes an acceptable proof.

* *

A —very— brief history of mathematics

This section is based on EWD1277, "Society's role in mathematics" [15], and E.T. Bell's "The Development of Mathematics" [5].

The notion of "proof" has long been central to mathematics, with the first proof (actually a handful of proofs, among them that a circle is bisected by any of its diameters) attributed to Thales of Miletus around 600 BC. However, and perhaps surprisingly, what constitutes a correct proof remains open to debate: by definition a proof should constitute a "convincing argument",

but convincing to whom, and by what standards?

Until the 1800s proofs were conducted following Euclid's approach to geometry, by establishing "logical conclusions" that followed from "self-evident", and hence indisputable, "axioms", where those logical conclusions, and indeed the axioms, were stated primarily in natural language, and based on appeals to intuition rather than on any kind of explicit rules; the study of logic per se was left primarily to the philosophers.

Since proofs appealed to intuition rather than well established rules, it became the role of the mathematical community to decide on the standards by which proofs should be judged; so emerged the so-called "consensus model", where a proof was submitted for peer review and accepted as correct when none of the experts could find anything wrong with it.

As a consequence of Descartes' development of geometry as a branch of algebra (where previously algebra had been considered a branch of geometry), Euclid's "self-evident" axioms had lost some of their exulted status. However, common sense (and tradition) dictated that each of Euclid's postulates were necessary, and obviously true. In 1829, Lobachevsky challenged this view by showing the existence of alternative, "non-Euclidean" geometries, by developing a geometry where Euclid's fifth, "parallel postulate", no longer held.

Lobachevsky's observation was mirrored in developments in algebra following G. Peacock's recognition of algebra as a purely formal mathematical system in his 1830 publication "Treatise on Algebra". So began the shift away from self-evident axioms toward freely invented collections of "postulates". Particularly noteworthy was Hamilton's rejection (in 1843) of commutativity as a postulate when developing the "quarternions", a choice that

> opened the gates to a flood of algebras, in which one after another of the supposedly immutable 'laws' of rational arithmetic and common algebra was either modified or discarded outright as too restrictive. [5] (Page 189)

Historical Aside. Peacock founded what has been called the "philological" or "symbolical" school of mathematicians, to which De Morgan and Boole belonged.
End of Historical Aside.

Although the postulates no longer appealed to intuition, the rules of deduction remained implicit. Consequently, the consensus model was still very much in effect; however, consensus was not always reached: as a famous example, Cantor's 1874 paper "On a Characteristic Property of All Real Algebraic Numbers" [9], which marked the birth of set theory, met with considerable opposition (most notably from Kronecker).

By the late 1800s attempts had begun to place mathematics on sound foundations, and to improve upon the standard, informal arguments, by instead providing "formal" proofs, where not only the assumptions (ie, the postulates) are made explicit, but also the deduction rules, and hence each step of the argument.

George Boole, Augustus de Morgan, and William Jevons are considered to be the initiators of modern logic (see, for example, [49]), but the landmark development came in 1879 with Frege's

"Begriffsschrift" [24] , in which he presented a fully fledged version of the propositional calculus and quantifier theory, marking the birth of so-called "formal mathematics" .

* *

Formal mathematics

The development of formal logic signified a shift away from informal reasoning and appeals to intuition, toward symbolic reasoning where we manipulate strings of uninterpreted formulae according to well-defined rules. In studies of formal logic we distinguish between "proof theory" , the study of syntax, and "model theory" , the study of semantics; that is, we distinguish between form (syntax) and meaning (semantics) . The presentation in this section is based primarily on Chapter 7 of Gries and Schneider's "A logical approach to discrete math" [32] .

*

Proof theory

A "formal system" , or "logic" , is a syntax-oriented deduction system comprising a set of symbols; a set of "well-formed formulae" ; a set of "start" symbols, a subset of the set of well-formed formulae, elements of which are called "axioms" ; and a set of "production" or "inference" rules.

Remark. In view of the above discussion, it's unfortunate that in studies of formal logic, the word "axiom" is generally used instead of the more appropriate "postulate" .
End of Remark.

The purpose of the production rules is to provide a way of producing well-formed formulae from the start symbols. A "theorem" is a formula that can be generated from the axioms by a finite number of productions (applications of the inference rules) . A "proof" is a chain of productions: a witness that a formula can be generated from the axioms using the inference rules. Consequently, reasoning —ie, proving theorems— is a purely syntactic activity, carried out by mechanical application of the rules.

Observe that depending on the choice of axioms and inference rules, the set of theorems may or may not be the same as the set of well-formed formulae. If the set of theorems is a (nonempty) proper subset of the set of well-formed formulae —ie, if at least one formula is a theorem, and at least one is not— the logic is said to be "consistent" .

An axiom is said to be "independent" of the other axioms if it cannot be produced from the other axioms using the inference rules. For example, where Lobachevsky demonstrated the existence of non-Euclidean geometries where the Euclid's parallel postulate no longer holds, in 1868

Eugenio Beltrami demonstrated that in Euclidean Geometry, the parallel postulate is independent of Euclid's other axioms. Independence of axioms is usually more important to the study of logic than to the use of logic.

*

Model theory

In general we want to prove theorems about a particular domain of discourse; that is, we want to establish that statements about a particular concept are "true". However, as mentioned, theorem proving is a purely syntactic activity, where values only arise as the result of an explicitly applied valuation function; in other words, formal theorem proving is independent of the domain of discourse, and hence of the concepts we want to study.

An "interpretation" (alternatively, a "structure") is a function that assigns "meaning" to the symbols of a logic by assigning values to formulae. Interpretations provide the link between the syntactic world of theorem proving, and the domain of discourse we are interested in.

Let I be a set of interpretations for a logic L, where —clearly— a logic may have many possible interpretations. We say that a formula F (of L) is "satisfiable" under I if at least one interpretation in I maps F to **true**, and "valid", or a "tautology", if every interpretation in I maps F to **true**. An interpretation is called a "model" for L if it maps every theorem of L to **true**. L is said to be "decidable" if there exists an algorithm that can decide validity for every formula of L.

We say that L is "sound" if every theorem of L is valid; ie, if every interpretation maps every theorem of L to **true**, alternatively: every interpretation is a model for L. We say that L is "complete" if every valid formula (ie, every tautology) of L is a theorem; ie, if all tautologies are provable from the axioms using the inference rules. Soundness is the converse of completeness and vice versa: if L is sound and complete then every theorem is a tautology and every tautology is a theorem. As Gries and Schneider point out:

> Soundness means that the theorems are true statements about the domain of discourse.
> Completeness means that every valid formula can be proved.

Model theory was used by Gödel and Cohen to prove the independence of the axiom of choice and the continuum hypothesis (there is no set S such that $\#\mathbb{Z} < \#S < \#\mathbb{R}$) by proving that the axiom of choice (and the continuum hypothesis) and its negation are consistent with the Zermelo-Fraenkel axioms of set theory. Specifically, in 1940 Gödel demonstrated the existence of a model of ZFC (the Zermelo-Fraenkel axioms with the axiom of choice) where the continuum hypothesis is **true**, and hence that the continuum hypothesis cannot be disproved from the ZFC axioms [26]; in 1963 Cohen demonstrated (using "forcing") the existence of a model of ZFC where the continuum hypothesis is **false**, and hence that the continuum hypothesis cannot be proved from the ZFC axioms [10]; it follows that the continuum hypothesis must be

independent of ZFC .

* *

Formal versus informal mathematics

We may judge the quality of a mathematical argument, formal or informal, by various criteria, such as correctness, brevity, elegance, ease of verification, and generality. Unfortunately, in practice most proofs fail to meet some or all of these criteria. Broadly speaking, mathematical methodology is the study of how we may design mathematical arguments that meet our quality criteria, in other words, how we "do" mathematics; this includes the study of techniques, tools, and heuristics. So, from a methodological point of view, which should we choose, formal or informal techniques?

In terms of verification, informal proofs tend to place a large burden on the reader, since they are rarely self-contained: they draw on, often without mention, assumptions and previously established results from various branches of mathematics, and usually contain large gaps between steps, where it is left to the reader to fill in those gaps. Similarly, the use of over-specific nomenclature and special-purpose tricks and inventions often renders generalisation impossible, and provides the reader with little or no insight into how to go about constructing similar proofs.

By contrast, formal proofs tend to inspire more confidence than their informal counterparts, since the requirements of formality require us to explicitly state our assumptions, and restrict our freedom to make mistakes: provided we follow the rules, we may only make typographic errors that should be caught by careful checking, where, in principle, such proofs can be checked mechanically using a computer. In other words, formality exposes the inadequacy of the consensus model, it being needed only to overcome the drawbacks of informal reasoning, where the assumptions and the rules of the game are left implicit: clearly the ability to machine check proofs that follow explicit rules renders the need for consensus obsolete.

Although the ability to mechanically check formal proofs suggests that it is in some sense easier to verify formal proofs than informal proofs, it does not a priori imply that formal proofs are easier to find than informal proofs. However, formalism not only allows us to machine check proofs, but also to use "automated theorem provers" to exhaustively search for proofs.

Remark. The question of whether verifying proofs is easier than finding them will be discussed further when we come to explore complexity theory, and in particular the question of whether $\mathbf{P} = \mathbf{NP}$. In the subsequent sections on calculation I explore, albeit briefly, how the use of formalism and attention to syntactic details can help in the discovery of proofs.
End of Remark.

The "Robbins conjecture" is a popular example of a theorem that admitted a simple proof that was only discovered using an automated theorem prover. A "Robbins algebra" is an algebra comprising a binary set and two logical operations, disjunction, $\vee$, and negation, $\neg$, that obey

the following axioms:

- $\vee$ is symmetric and associative
- $\neg(\neg(P \vee Q) \vee \neg(P \vee \neg Q)) \quad \equiv \quad P$ (Robbins' axiom)

Herbert Robbins conjectured that these axioms are equivalent to the boolean algebra axioms. The conjecture was proved in 1996 by William McCune, using the EQP automated theorem prover [40] .

Despite the accepted benefits in precision, formalism has so far failed to sway the mathematical community at large, a common criticism being that formal techniques are cumbersome, tedious, and unnatural. Consequently, for the most part proofs continue to be conducted informally and judged by consensus.

The question then, is whether formal proofs are by necessity verbose, laborious, and so on; in particular, is it possible to strike a pragmatic balance between the use of formalism and readability? Can we reap the benefits of formalism while retaining succinctness? I believe the so-called "calculational" style of mathematics offers just such a balance.

* *

Calculational mathematics

Calculation is a style of reasoning that emerged from efforts to reason about computer programs. The programming challenge, in particular the issue of program correctness, presented a new kind of complexity. A great step forward came with the realisation that programs are mathematical objects, and so can be reasoned about mathematically. However, it soon became apparent that existing approaches to mathematical reasoning, formal and informal, were not appropriate; as Thurston points out:

> The standard of correctness and completeness necessary to get a computer program to work at all is a couple of orders of magnitude higher than the mathematical community's standard of valid proofs. [46]

The next breakthrough came with the realisation that reasoning about correctness becomes a far more attractive proposition if, instead of constructing a program and then trying to verify it, we construct the program and its proof of correctness hand-in-hand. To that end, Dijkstra developed the notion of "predicate transformers" and "weakest preconditions" [12] . The calculational style of reasoning emerged primarily during later efforts by Dijkstra and Scholten to put these ideas on sound theoretical foundations; the "official" reference is [17] , in the sense that this is the first place the calculational style was explicitly presented to the world at large, though the style was in use prior to the publication of this text.

With respect to the advantages of the calculational style over traditional formal methods,

Dijkstra and Scholten point out that:

> The first pleasant —and very encouraging!— experience was the killing of the myth that formal proofs are of necessity long, tedious, laborious, error-prone, and what-have-you. On the contrary, our proofs turned out to be short and simple to check, carried out —as they are— in straightforward manipulations from a modest repertoire. [17] (Page vi)

This is due in part to calculation being algebraic in flavour; as (Rutger) Dijkstra points out:

> Algebras arise as labour saving tools ... when it comes to being short, simple, convincing, and illuminating, algebra and logic are simply not in the same league. [18]

Dijkstra and Scholten reintroduce the familiar predicate logic as a "predicate algebra" by postulating properties of equivalence, negation, and disjunction. In particular, equivalence, denoted by $\equiv$, and pronounced "equivales", is postulated to be symmetric and associative.

Remark. The word "equivale", though not well known, is not new: it dates back to at least the 1600s, where the Oxford English Dictionary defines it as "to be equivalent to".
End of Remark.

The $\equiv$ symbol is used instead of $\iff$ to emphasise that in predicate algebra (boolean) equivalence is a "first class citizen", in the sense that rather than being defined in terms of implication, as is usually the case, its properties are postulated.

Conjunction is defined in terms of equivalence and disjunction by the so-called "Golden rule":

$$P \wedge Q \;\equiv\; P \;\equiv\; Q \;\equiv\; P \vee Q$$

Implication may be defined by any of the following:

$$\neg P \;\vee\; Q \qquad\qquad P \wedge Q \;\equiv\; P \qquad\qquad P \vee Q \;\equiv\; Q$$

(Whichever we choose, we get the other two "for free" as theorems.)

To "calculate" is to transform an input into an output by a sequence of steps performed according to a collection of well defined rules. A "proof" in the calculational setting is a calculation: a chain of value preserving transformations that evaluates a boolean expression to **true**; a "theorem" is a boolean expression that always evaluates to **true**.

*

The calculational style has been used in a number of programming texts aimed at computing science undergraduates. Examples include Backhouse's "Program Construction" [4] , Kaldewaij's "Programming: The Derivation of Algorithms" [38] , and Feijen and van Gasteren's "On a Method of Multiprogramming" [19] .

Gries and Schneider's "A Logical Approach to Discrete Math" [32] differs from conventional treatments of discrete mathematics in its emphasis on the use of logic to prove theorems, rather than on logic as merely a subject for study. Gries and Schneider present an equational logic based on Dijkstra and Scholten's predicate algebra, which they then use to give calculational-like treatments of a variety of topics in discrete mathematics, including set theory, induction, sequences, relations and functions, number theory, combinatorics, algebra, and graph theory.

In her PhD thesis, "On the Shape of Mathematical Arguments" , Netty van Gasteren [48] showed how calculation, along with other lessons learnt from the formal development of programs, can be applied to proving mathematical theorems in general.

* *

On notation

Much of this section is covered by Chapter 16 of Netty van Gasteren's "On The Shape of Mathematical Arguments" [48] , and EWD1300 , "The notational conventions I adopted, and why" [16] , so I'll be brief in my exposition.

When choosing a notation we must exercise caution, as notation can have a profound influence on the way we think, our ability to manipulate our formulae, and consequently the effectiveness of a particular interface. For example, Roman numerals form an interface between us and the positive natural numbers, but there can be little argument that for the purpose of doing arithmetic, the Hindu-Arabic notation offers a far superior interface —the set theoretic representation of the naturals offers an even less appealing interface— . As Dijkstra points out in EWD655 [13] , a good notation must satisfy at least three requirements: it should be unambiguous, short, and geared to our manipulative needs.

The need to be unambiguous should be obvious, particularly if we are to manipulate our formulae rather than interpret them with respect to some model. However, many established notational conventions fail to meet this requirement. For example, many forms of "quantified" expressions fail to make clear which variables are bound and which are free, and the scope of the binding. To overcome this particular problem, following Dijkstra et al, I adopt the following "Eindhoven triple" notation:

$$\langle \oplus i \in T : R.i : P.i \rangle$$

This expression denotes the application of operator $\oplus$ to the values $P.i$ for all i in T where

$R.i$ is **true** . Dissecting the notation,

• i is called a "bound" or "dummy" variable (if there is more than one we separate them by commas) , its scope being delineated by the angle brackets; when the type of the dummy is clear from (or fixed in) the context we omit it and simply write

$$\langle \oplus i : R.i : P.i \rangle$$

• $R.i$ is called the "range" of the quantification, its purpose being to restrict the values of the dummies beyond their basic type information; if the range is omitted, as in

$$\langle \oplus i :: P.i \rangle \qquad ,$$

then the range is understood to be **true**

• $P.i$ is called the "term" of the quantification; the type of the term defines the type of the quantification

Common instantiations of $\oplus$ include $\forall$, $\exists$, Σ , Π , $\uparrow$, $\downarrow$ (respectively, universal and existential quantification, summation, product, maximum, and minimum) . The standard constraint is that the operator forms an abelian monoid, but we often relax this to an abelian semigroup (a fancy way of saying the operator is symmetric and associative) ; ie, we relax the requirement that the operation has an identity element, $\uparrow$ and $\downarrow$ being obvious examples.

The need for brevity is also clear: when manipulating formulae we want to avoid repeating long strings of symbols, since the longer the strings the more likely we are to introduce errors, and the larger the burden on the reader when it comes to verifying our manipulations.

The latter requirement, viz that the notation should be "geared to our manipulative needs" , requires clarification. As far as possible we want to "let the symbols do the work" , meaning we should choose our symbols so they suggest manipulative possibilities; that way, the syntax of our formulae can guide the shape of our proofs. To put it another way, we use symbols to denote concepts, where concepts are collections of properties, and since it's exactly these properties we appeal to when manipulating those symbols, where possible we should choose symbols suggestive of those properties.

For example, an effective —but little practised— heuristic is to choose symmetric symbols for symmetric infix operators, and asymmetric symbols for asymmetric infix operators. Accordingly, the use of $+$ to denote addition is a good choice of symbol, as both it and the operation it denotes are symmetric; however, subtraction is not symmetric, so $-$ is a poor choice of symbol.

Since we manipulate parsed formulae rather than strings, as well as suggesting manipulative possibilities, our symbols should provide a visual aid to parsing. Consider the following definition

taken from an undergraduate text on discrete mathematics:

> Consider two semigroups $(S, *)$ and $(S', *')$. A function $f : S \to S'$ is called a "semigroup homomorphism" or, simply, a "homomorphism" if
>
> $$f(a * b) = f(a) *' f(b) \qquad \text{or, simply} \qquad f(ab) = f(a)f(b)$$

In order to parse an expression like $f(ab) = f(a)f(b)$ we, the reader, have to fill in the two missing, "invisible" operators, which can quickly become a tiresome burden. Consequently it is best to avoid invisible operators. Following Dijkstra et al, I use an infix dot to explicitly denote function application, and so write $f.x$ in contrast to the "standard" $f(x)$ notation; I also write $\cdot$ to explicitly denote multiplication.

Another effective —again, little practised— heuristic is to choose larger symbols for operations with lower binding powers. (Since function application is given highest binding power, it makes sense to choose the smallest practical symbol to denote it, hence the infix dot.) I also dedicate more whitespace to operations with lower binding powers; so for example,

$$P \wedge Q \;\Rightarrow\; R \quad \equiv \quad P \;\Rightarrow\; (Q \Rightarrow R)$$

is considerably easier on the eye than

$$P \wedge Q \Rightarrow R \equiv P \Rightarrow (Q \Rightarrow R) \qquad .$$

As a further aid to parsing, it makes sense to avoid a proliferation of parentheses. In this respect denoting function application by an infix dot is a good choice of notation, since the standard $f(x)$ notation usurps a parenthesis pair. Adopting the fairly standard convention that function application is left-associative, the parentheses are necessary in $f.(g.x)$, but they can be avoided by appealing instead to function composition: $f \circ g.x$. More generally, in view of $\circ$'s associativity, expressions such as

$$f \circ g \circ h.x$$

are semantically unambiguous, and visually far more appealing than the alternative

$$f(g(h(x))) \qquad .$$

As a final remark on notation, and somewhat related to the use of parentheses, I use $\#x$ instead of $|x|$ to denote the cardinality of a set if x is a set, or the "size" (ie, the number of bits) of x if x is a bit-string; I mention this primarily because, as we'll see in the next chapter, Goldreich uses the latter.

* *

On proofs

Adherence to formalism allows us to adopt a strict proof format. The advantages of a uniform, well designed format are considerable, not least that it makes comparison of various proofs of the same theorem far simpler.

Briefly, in the calculational style we adopt the following format:

$$\begin{array}{ll} & A \\ \nabla & \quad \{ \text{ hint why } A \nabla B \} \\ & B \end{array}$$

Where A and B are expressions of the same type (booleans, integers, and reals being common) and ∇ is a transitive relation over that type (common examples being $\equiv$, $\Rightarrow$, $\Leftarrow$, $=$, $<$, $>$, $\leq$, and $\geq$).

Though there is no "official" reference, the credit for this proof format goes to W.H.J. Feijen. For more on the format see, for example, EWD999, "Our proof format" [14], or Chapter 4 of Dijkstra and Scholten's "Predicate Calculus and Program Semantics" [17].

*

The purpose of a hint is primarily to reduce our search space when verifying each step of a proof; in other words, hints are used to close the gaps by supplying the missing links. In algebraic proofs of the form

$$\begin{array}{ll} & a = b \\ & \quad = c \\ & \quad = d \end{array}$$

hints are given either in the surrounding (usually the subsequent) text, or omitted entirely. The former has the disadvantage of forcing the reader to flip back and forth between the proof and the text; the latter relies on the reader's knowledge and ability to fill in the gaps.

Feijen's proof format ensures that hints are a uniform ingredient, and are deliberately positioned so they both signpost and justify —or are at least suggestive of the justification for— the change from one line to the next. For example, in the following

$$\begin{array}{ll} & (A \wedge B) \ \vee \ C \quad \equiv \quad P \ \Rightarrow \ (Q \equiv R) \\ \equiv & \quad \{ \text{ definition of } \Rightarrow \} \\ & (A \wedge B) \ \vee \ C \quad \equiv \quad \neg P \ \vee \ (Q \equiv R) \end{array}$$

the hint signposts that the step focuses on the subexpression containing the implication, allowing us to quickly identify exactly what has changed, and to ignore everything else.

A hint may give the exact manipulation rule used (eg, a previously stated, numbered rule), or may be more general (eg, "arithmetic", or "algebra"). In some cases a hint may give a more detailed explanation or justification for the step, perhaps even outlining the heuristics that motivated the step; when using Feijen's proof format we always allocate at least one full line for a hint, but there is nothing stopping us from using more than one line.

*

Granularity of proof steps is a subjective matter, but the goal is to strike a balance between ease of verification and succinctness. Ideally proof steps should be small enough, and the hints suggestive enough, that the reader does not have to resort to pen and paper to verify them. Although this is a nice goal to aim for, in practice it does not always work out, as it is nearly always necessary to make some basic assumptions.

For example, as stated in the introduction, I assume that anyone reading this thesis has a reasonable working knowledge of predicate calculus, and under that assumption I feel justified in combining a number of simple steps such as

$$
\begin{array}{ll}
 & P \wedge Q \;\equiv\; P \\
\equiv & \quad \{ \text{ golden rule } \} \\
 & P \vee Q \;\equiv\; P \;\equiv\; Q \;\equiv\; P \\
\equiv & \quad \{ \text{ symmetry of } \equiv \} \\
 & P \vee Q \;\equiv\; P \;\equiv\; P \;\equiv\; Q \\
\equiv & \quad \{ \text{ reflexivity of } \equiv \} \\
 & P \vee Q \;\equiv\; \mathbf{true} \;\equiv\; Q \\
\equiv & \quad \{ \text{ identity of } \equiv \} \\
 & P \vee Q \;\equiv\; Q
\end{array}
$$

into a single step with the hint "predicate calculus"; ie:

$$
\begin{array}{ll}
 & P \wedge Q \;\equiv\; P \\
\equiv & \quad \{ \text{ predicate calculus } \} \\
 & P \vee Q \;\equiv\; Q
\end{array}
$$

As Netty van Gasteren points out [48], irrespective of granularity issues we should avoid combining different "types" of steps, such as equality preserving steps, and weakening or strengthening steps. So, for example, we should avoid steps that combine $=$ and $\geq$ or $=$ and $\leq$;

similarly, we should avoid steps that combine $<$ and $\leq$, or $>$ and $\geq$.

*

Proofs, particularly calculational proofs, are "directional" in the sense that given a demonstrandum of the form $P \nabla Q$, we may start from one side or the other, transforming P into Q or vice-versa, by constructing an appropriate chain of value preserving transformations. So for example, if we replace ∇ with $\Rightarrow$, we may either weaken P to Q, or strengthen Q to P. Although in principle we can proceed in either direction, often one direction leads to a "better" proof than the other. For example, in order to prove

$$P \vee Q \quad \equiv \quad (P \wedge \neg Q) \ \vee \ Q \qquad ,$$

we may either transform $P \vee Q$ into $(P \wedge \neg Q) \ \vee \ Q$, or vice-versa, by constructing a chain of equivalences, but which should we choose? We could of course try both possibilities, and simply pick the "best" proof, but it is usually possible to let the shape of the demonstrandum guide us in our proof design. The heuristic is to proceed from the more complex side, so in this case we should start from $(P \wedge \neg Q) \ \vee \ Q$ and transform it into $P \vee Q$. Observe how much nicer the proof is in this direction

$$\begin{array}{ll} & (P \wedge \neg Q) \ \vee \ Q \\ \equiv & \qquad \{ \ \vee \ \text{over} \ \wedge \ \} \\ & (P \vee Q) \ \wedge \ (\neg Q \vee Q) \\ \equiv & \qquad \{ \ \text{excluded middle} \ \} \\ & (P \vee Q) \ \wedge \ \mathbf{true} \\ \equiv & \qquad \{ \ \text{unit of conjunction} \ \} \\ & P \vee Q \end{array}$$

than in the opposite direction:

$$\begin{array}{ll} & P \vee Q \\ \equiv & \qquad \{ \ \text{unit of conjunction} \ \} \\ & (P \vee Q) \ \wedge \ \mathbf{true} \\ \equiv & \qquad \{ \ \text{excluded middle} \ \} \\ & (P \vee Q) \ \wedge \ (\neg Q \vee Q) \\ \equiv & \qquad \{ \ \vee \ \text{over} \ \wedge \ \} \\ & (P \wedge \neg Q) \ \vee \ Q \end{array}$$

In the former each step is essentially forced, to the extent that the proof is almost self-conducting, but in the latter each step requires something of a leap of faith.

The latter proof leads us to the issue of "rabbits" : steps, constructions, and so on with no motivation, that "do the job" , but appear out of nowhere, like a rabbit pulled from the proverbial magician's hat. Wherever possible rabbits are to be avoided, as they provide the reader with little insight into how the proof was constructed, or how to go about constructing similar proofs. Rabbits are usually a sign of poor proof structure; as demonstrated above, the heuristic of proceeding from the more complex side of the demonstrandum can help to avoid the introduction of rabbits.

* *

On contexts

This section is based on personal conclusions and a series of emails from Jeremy Weissmann, summarised in JAW61 , "How I understand context and type information" [50] .

In order to manipulate our formulae, and hence in order to calculate, we need rules. It is the context of a calculation that provides these rules. In the calculational approach we take the view that the context of a calculation constitutes the range of a universal quantification, and that we calculate with the term of the quantification.

So, how do we establish what's in the context? Our formulae contain symbols, and associated with these symbols are the properties we appeal to when we calculate. It is the conjunction of these properties that forms the context.

For example, if the symbol $+$ appears in our formulae, and has the familiar denotation of addition (of reals, naturals, etc) , then we implicitly import into our context properties of addition, such as that it is symmetric, associative, and so on, and we are then free to draw on those properties when manipulating our formulae.

We may work with a single "grand" context, ie a single universal quantification, but by virtue of "nesting" ,

$$\langle \forall x, y : Q \wedge R : P \rangle \quad \equiv \quad \langle \forall x : Q : \langle \forall y : R : P \rangle \rangle$$

(provided y doesn't occur in Q) , we may break that context into pieces, viewing it instead as a number of nested contexts, possibly leaving the outer contexts implicit. When we want to emphasise that a calculation is being carried out within a specific "local" context, we may use the following notation:

$[\![$ Context: C

P

∇ { ?? }

Q

$]\!]$

By virtue of "trading" ,

$$\langle \forall x : Q \wedge R : P \rangle \quad \equiv \quad \langle \forall x : Q : R \Rightarrow P \rangle \qquad ,$$

we may trade context information into our calculation, and vice-versa; this is the basis for "assuming" the antecedent and deriving the consequent when proving theorems involving implication: we simply trade the antecedent into the context and focus on the consequent, drawing on the antecedent and other contextual information as necessary. Observe that if our context implies **false** , then by virtue of trading all our theorems are of the form $\mathbf{false} \Rightarrow P$, and so trivially reduce to **true** . Such contexts are typically deemed "uninteresting" , since we may prove anything.

* *

A small case study: calculating with congruences

The following is an example of how we can use notation to streamline proofs. In "A Logical Approach to Discrete Math" [32] Gries and Schneider use the notation $\stackrel{n}{=}$ to denote congruence modulo n , ie

$$x \stackrel{n}{=} y \quad \equiv \quad x \,\mathbf{mod}\, n \;=\; y \,\mathbf{mod}\, n \qquad ,$$

where congruence may be equivalently defined as

$$(0) \qquad x \stackrel{n}{=} y \quad \equiv \quad n \sqsubseteq (y - x) \qquad ,$$

where $\sqsubseteq$ denotes the "divides" relation, ie

$$a \sqsubseteq b \quad \equiv \quad \langle \exists x :: \; a \cdot x \;=\; b \,\rangle \qquad .$$

Suppose we are asked to prove

$$(1) \qquad b \stackrel{n}{=} c \quad \Rightarrow \quad b^m \stackrel{n}{=} c^m \qquad \text{for } n \geq 0 \qquad ;$$

here's Gries and Schneider's proof (from "The Instructor's Manual to A Logical Approach to

Discrete Math") :

$$
\begin{array}{ll}
 & b^m \stackrel{n}{=} c^m \\
= & \quad \{\ (0)\ \} \\
 & n \ \sqsubseteq\ (c^m - b^m) \\
= & \quad \{\ \text{arithmetic}\ \} \\
 & n \ \sqsubseteq\ (\ (c-b) \cdot \langle \Sigma i : 0 \leq i < m : b^i \cdot c^{n-1-i} \rangle\ ) \\
\Leftarrow & \quad \{\ \text{property of}\ \sqsubseteq\ \} \\
 & n \ \sqsubseteq\ (c-b) \\
= & \quad \{\ (0)\ \} \\
 & b \stackrel{n}{=} c
\end{array}
$$

The proof is very short and for the most part easy to follow, with the second step, viz the introduction of

$$n \ \sqsubseteq\ ((c-b) \cdot \langle \Sigma i : 0 \leq i < m : b^i \cdot c^{n-1-i} \rangle) \qquad ,$$

being the most difficult part of the proof to construct and to verify. An alternative argument comprises a proof by induction on m . The base case is trivial. For the induction step we assume the hypothesis holds for $m-1$ and prove for m :

$$
\begin{array}{ll}
 & b^m \ \mathbf{mod}\ n \\
= & \quad \{\ \text{exponents}\ \} \\
 & b \cdot b^{m-1} \ \mathbf{mod}\ n \\
= & \quad \{\ \text{property of}\ \mathbf{mod}\ \} \\
 & ((b \ \mathbf{mod}\ n) \cdot (b^{m-1} \ \mathbf{mod}\ n)) \ \mathbf{mod}\ n \\
= & \quad \{\ \text{antecedent and induction hypothesis}\ \} \\
 & ((c \ \mathbf{mod}\ n) \cdot (c^{m-1} \ \mathbf{mod}\ n)) \ \mathbf{mod}\ n \\
= & \quad \{\ \text{property of}\ \mathbf{mod}\ \} \\
 & c \cdot c^{m-1} \ \mathbf{mod}\ n \\
= & \quad \{\ \text{exponents}\ \} \\
 & c^m \ \mathbf{mod}\ n
\end{array}
$$

Clearly the latter proof is the longer of the two. However, what follows is essentially the same proof, but here —roughly speaking— we admit the substitution of congruent values for congruent

values, resulting in a much crisper argument:

$$
\begin{array}{ll}
 & b^m \\
= & \quad \{ \text{ exponents } \} \\
 & b \cdot b^{m-1} \\
\stackrel{n}{=} & \quad \{ \text{ antecedent and induction hypothesis } \} \\
 & c \cdot c^{m-1} \\
= & \quad \{ \text{ exponents } \} \\
 & c^m
\end{array}
$$

The symmetry of the argument is very appealing! This form of substitution is valid because multiplication is monotonic with respect to congruence:

$$(2) \qquad x \stackrel{n}{=} y \quad \Rightarrow \quad x \cdot z \stackrel{n}{=} y \cdot z$$

Here's the same proof in more detail:

$$
\begin{array}{ll}
 & b^m \\
= & \quad \{ \text{ exponents } \} \\
 & b \cdot b^{m-1} \\
\stackrel{n}{=} & \quad \{ \text{ antecedent and } (2) \text{ with } x, y, z := b, c, b^{m-1} \} \\
 & c \cdot b^{m-1} \\
\stackrel{n}{=} & \quad \{ \text{ induction hypothesis and } (2) \text{ with } x, y, z := b^{m-1}, c^{m-1}, c \} \\
 & c \cdot c^{m-1} \\
= & \quad \{ \text{ exponents } \} \\
 & c^m
\end{array}
$$

Of course, (1) can be rendered in English as "exponentiation is monotonic with respect to congruence", so we are now justified in using proof steps of the form

$$
\begin{array}{ll}
 & a^d \cdot c \\
\stackrel{n}{=} & \quad \{ \; a \stackrel{n}{=} b \; \} \\
 & b^d \cdot c
\end{array}
$$

where appeals to (1) and (2) are left implicit, much as when we appeal to Leibniz (ie, substitution of equals for equals) we need not mention monotonicity, nor Leibniz in most cases. For example, in the key generation phase of the RSA cryptosystem (discussed in the next

chapter) values are picked for e and d so that

$$(3) \qquad m^{e \cdot d} \stackrel{n}{=} m$$

for any m in the range $0 \leq m < n$. We compute the "encryption", c, of a message m, where $0 \leq m < n$, as $m^e \bmod n$; to "decrypt", ie, to recover m given c, we compute $c^d \bmod n$. We can now easily, and elegantly, prove that decryption "undoes" encryption, as you'd expect:

$$\begin{array}{ll} & c^d \\ \stackrel{n}{=} & \quad \{ \text{ definition of } c \} \\ & (m^e)^d \\ = & \quad \{ \text{ exponents } \} \\ & m^{e \cdot d} \\ \stackrel{n}{=} & \quad \{ (3) \} \\ & m \end{array}$$

Remark. Observe how much nicer the proof is in this direction than the opposite direction, lending further credibility to the heuristic of proceeding from the more complex side of the demonstrandum. **End of Remark.**

* * *

Chapter 2

Cryptography

Some problems are harder to solve than others in the sense that generating a solution requires more resources, such as time, space, energy, and so on; therefore, there exists a measurable "complexity gap" between problems, and it makes sense to distinguish between "easy" problems and "hard" problems. However, where complexity theory is, roughly speaking, about exploring and quantifying this distinction, cryptography is about exploiting it. More specifically, cryptography is the study of constructions where some of the computations involved are deliberately easy, while others are deliberately hard. Having defined the vague terms "easy" and "hard", the goal is to prove that the hard computations are indeed hard.

Ideally we'd like to establish precise lower bounds on hard computations, but complexity theorists have had limited success in establishing lower bounds in general, so instead we reason relatively: we show that the hard computations are at least as hard as solving some problem known or assumed (usually the latter, for reasons to be explained in due course) to be hard. The proof technique for making assertions about the complexity of one problem on the basis of another is called "reduction", where —at this stage very informally— a reduction from a problem P to a problem Q amounts to constructing a program that uses a given or postulated solution to Q to solve P.

For example, the RSA "public-key" cryptosystem [45] is based on the assumption that factoring integers is hard. The RSA algorithm proceeds by generating a pair of "keys" as follows:

- pick two large, distinct prime numbers p and q
- compute $n := p \cdot q$ and $\phi := (p-1) \cdot (q-1)$
- pick an e such that $1 \leq e < \phi$ and $e \perp \phi$
- find d such that $1 \leq d < \phi$ and $e \cdot d \stackrel{\phi}{=} 1$

The pair (n, e) is called a "public-key", and the pair (n, d) is called a "private-key"; we publish (eg, on a website) the public-key, retain (ie, keep secret) the private key, and dispose of p, q, and ϕ.

Remark. The notation $e \perp \phi$ denotes that e and ϕ are "coprime", meaning their greatest common divisor is 1. For readers familiar with number theory, ϕ is shorthand for $\phi.n$, which is used to denote Euler's totient function: the number of positive integers less than n that are coprime to n.
End of Remark.

For our purposes it's not important to understand why the values are picked this way, or how they are computed; what is important, is that they are easy to compute, and that it should be hard to compute d given only n and e, the goal being to show that this is indeed the case.

Clearly we can easily compute d if we know e and ϕ. So, since we know e, our goal is to compute ϕ, which is easy if we can discover p and q; but we know n, and we know, by virtue of how n was constructed and the fundamental theorem of arithmetic, that factoring n would yield p and q as required. Consequently, if factoring is easy then computing d given only n and e must also be easy.

The above argument establishes a reduction from computing d to factoring n. However, this gives an upper bound on the difficulty of computing d: it asserts that computing d given n and e is no harder than factoring n; so, if factoring n is easy, then computing d given only n and e is also easy. According to this argument, even if factoring is hard, it does not a priori follow that computing d must also be hard, as it may be possible to use some other tactic to recover d. What we need to show is that computing d given n and e is at least as hard as factoring n, by showing how an algorithm to compute d could be used to factor n; ie, we need to reduce factoring n to computing d. Unfortunately, things are far less clear in this direction, and it has been suggested that computing d may be easier than factoring [8].

Of course, RSA is a specific cryptographic construction, and the requirement that it is hard to compute d given n and e is specific to that construction; it also happens to be rather a strong requirement: either it is hard to compute d or it is not. For reasons to be explored in due course, assertions in cryptography are usually probabilistic, so rather than establishing claims of the form "x is hard", our proof obligations are instead of the form "x is hard with high probability".

So, what's the problem? Well, primarily that proofs of cryptographic assertions tend to be incredibly complex. In particular, the reductions are often very contrived, making verification and generalisation near impossible. Due to the complex nature of cryptographic constructions, the proofs are usually carried out informally, and rarely at the detailed level of formality that would allow them to be mechanised or machine-checked. Consequently, cryptographic proofs tend to be verified by consensus. However, as pointed out, the difficult nature of these proofs makes them hard to verify, which, along with the tendency toward conference publication where emphasis is on turnaround time rather than scrutiny of correctness, means that many proofs in this area are published essentially unverified: even experts in this field have a low confidence in the full correctness of cryptographic proofs in general, and have suggested a move toward formalisation [33].

Though it remains to decide what is meant by the terms "easy", "hard", and "a high level of probability", deciding which computations should be computationally easy and which should be hard is, as you may expect, a fundamental aspect of cryptography. As mentioned,

RSA is a specific construction, and the requirement that d is hard to compute is specific to that construction, but in general we'd like to build a more abstract theory around a concept that underlies many different cryptographic constructions.

* *

One-way functions

The fundamental concept that underlies many cryptographic constructions is that of a so-called "one-way function". Roughly speaking, a one-way function is a function that is "easy" to compute, but "hard" on average to invert. One-way functions come in two flavours: "weak", and "strong", where strong one-way functions are "harder" to invert than weak one-way functions. The fundamental theorem that relates these concepts asserts that the existence of weak one-way functions equivales the existence of strong one-way functions, ie:

(4) $\exists$ weak one-way functions $\equiv$ $\exists$ strong one-way functions

The following definitions of strong and weak one-way functions are taken verbatim from pages 33 and 35 of Goldreich's "Foundations of Cryptography" [27].

> A function $f : \{0,1\}^* \to \{0,1\}^*$ is called (strongly) one-way if the following two conditions hold:
>
> *1.* Easy to compute: There exists a (deterministic) polynomial-time algorithm A such that on input x algorithm A outputs $f(x)$ (i.e., $A(x) = f(x)$).
>
> *2.* Hard to invert: For every probabilistic polynomial-time algorithm A', every positive polynomial $p(\cdot)$, and all sufficiently large n's,
>
> $$\Pr[A'(f(U_n), 1^n) \in f^{-1}(f(U_n))] < \frac{1}{p(n)}$$

In the subsequent text we are told that:

> U_n denotes a random variable distributed over $\{0,1\}^n$. Hence, the probability in the second condition is taken over all the possible values assigned to U_n and all possible internal coin tosses of A', with uniform probability distribution.

Weak one-way functions are defined as follows:

A function $f : \{0,1\}^* \to \{0,1\}^*$ is called weakly one-way if the following two conditions hold:

1. Easy to compute: There exists a (deterministic) polynomial-time algorithm A such that on input x algorithm A outputs $f(x)$ (i.e., $A(x) = f(x)$).

2. Slightly hard to invert: There exists a polynomial $p(\cdot)$ such that for every probabilistic polynomial-time algorithm A' and all sufficiently large n's,

$$\Pr[A'(f(U_n), 1^n) \notin f^{-1}(f(U_n))] > \frac{1}{p(n)}$$

As in the definition of strong one-way functions, the probability in the second condition is with respect to the possible values assigned to U_n, and all possible "internal coin tosses" of A', with uniform probability distribution.

The goal in the remainder of this chapter is to examine these definitions and Goldreich's proof of (4), focusing on identifying the concepts involved, rabbits and ambiguities, issues with the notation used, and other pitfalls highlighted in the previous chapter; in other words, the goal at this stage is primarily to identify problems rather than to remedy them —I explore what can be done to improve matters in subsequent chapters— .

* *

First observations

As stated above, one-way functions are functions that are easy to compute but hard to invert, where, in both definitions, the first condition captures the former requirement, and the second condition captures the latter requirement. (In the remainder of the thesis I refer to the second condition in both definitions as the "one-wayness requirement", or the "one-wayness condition".) According to both definitions, "easy to compute" means computable in "deterministic polynomial-time". Clearly "hard to invert" is the more complex requirement to quantify, and it is here that the definitions differ.

Observe that neither definition requires that one-way functions cannot be inverted, only that the probability of doing so in "probabilistic polynomial-time" is acceptably small. In the definition of strong one-way functions, "acceptably small" equates to the probability being less than the reciprocal of any polynomial function in n; however, this requirement is "asymptotic", in the sense that it must hold for "large enough" n. In the cryptography jargon a function g is called "negligible" if for every positive polynomial p, and for large enough values of n, $g.n < 1 \,/\, p.n$; formally:

$$\langle \forall p :: \langle \exists N :: \langle \forall n : n > N : g.n < 1 \,/\, p.n \rangle \rangle \rangle$$

In other words, the probability of inverting a strong one-way function using any probabilistic polynomial-time algorithm is negligible. As we will see in Chapter 5 , an important property of negligible functions is that they are closed under polynomial multiplication, so if g is a negligible function, and q is a polynomial, then $g.n \cdot q.n$ is a negligible function.

In the definition of weak one-way functions things are somewhat reversed, as the quantification is in terms of failure to invert f . The bound on the probability is still with respect to the reciprocal of polynomials, but the polynomial is existentially rather than universally quantified. In the cryptography jargon, a function g is called "noticeable" if for some positive polynomial p , and for large enough values of n , $1 / p.n < g.n$; formally:

$$\langle \exists p :: \langle \exists N :: \langle \forall n : n > N : 1 / p.n \ < \ g.n \rangle \rangle \rangle$$

In other words, the probability of failing to invert a weak one-way function using a probabilistic polynomial-time algorithm is noticeable. Goldreich describes noticeability as a "strong negation" of the notion of negligibility, where —clearly— noticeability is not simply the negation of negligibility; unfortunately, no explanation is offered as to why noticeability is defined this way.

Observe that in neither case are we required to find x given $f.x$: any valid inverse will do. If f is injective then clearly x is the only valid inverse, but there is no requirement that one-way functions be injective. However, injectivity, or rather, non-injectivity, can preclude a function from being one-way: a constant function of the form

$$f.x \ = \ c \qquad ,$$

for some constant c , is an extreme example of a non-injective function, where we can simply pick any element of f's domain as a valid inverse; clearly such a function cannot be one-way. Conversely, injectivity is no guarantee of one-wayness: the identity function is injective, but trivially invertible.

*

Both definitions involve quantifications over "positive polynomials" , where a polynomial p in n , over a field $\mathbb{F}$, is an expression of the form

$$(5) \qquad c_0 \ + \ c_1 \cdot n \ + \ c_2 \cdot n^2 \ + \ \ldots \ + \ c_m \cdot n^m \qquad .$$

The values c_0 , c_1 , $\ldots$, c_m , referred to as "coefficients" , are constants of type $\mathbb{F}$; if all of the coefficients are zero, then p is called the "zero polynomial" . m is of type natural, as suggested by its role as a subscript; if p is not the zero polynomial, then m is referred to as its "degree" . If p is either the zero polynomial, or has degree zero, it is referred to as a "constant

polynomial" . n is usually referred to as an "indeterminate" , as its type is essentially left undefined. p is positive if

$$(6) \qquad \langle \forall n : n > 0 : p.n > 0 \rangle \qquad ;$$

clearly the zero polynomial is not positive, but constant polynomials may be positive.

In the definitions of one-way functions we restrict the indeterminate n to being of type natural, since it denotes the length of bit-strings. The type of the coefficients is not specified, though at this stage it's not clear whether this matters.

*

From the definitions alone, the role of 1^n in the one-wayness conditions is unclear. To clarify, 1^n denotes a string of n ones in so-called "unary" notation; so, for example, 1^5 denotes the bit-string `11111` ; 0^n is defined analogously. Following the definition of strong one-way functions Goldreich offers this explanation:

> In addition to an input in the range of f, the inverting algorithm A' is also given the length of the desired output (in unary notation). The main reason for this convention is to rule out the possibility that a function will be considered one-way merely because it drastically shrinks its input, and so the inverting algorithm just does not have enough time to print the desired output (i.e., the corresponding pre-image). ... Note that in the special case of length-preserving functions f (i.e. $|f(x)| = |x|$ for all x's), this auxiliary input is redundant.

Consequently, if we restrict our attention to "length-preserving functions" we can eliminate 1^n from the one-wayness condition. But are we justified in making such a restriction? It turns out that we are, since any non-length-preserving one-way function can be transformed (by way of a rather contrived construction) into a length-preserving one-way function. The proof of this is omitted.

*

Focusing on the one-wayness condition in the definition of strong one-way functions, syntactically,

$$\Pr[A'(f(U_n), 1^n) \in f^{-1}(f(U_n))] < \frac{1}{p(n)}$$

is a mess. By restricting our attention to length-preserving functions we can immediately eliminate 1^n from the definition. In view of the earlier discussion about notation, a further simplification would be to dispense with some of the brackets and make more effective use of white-space:

$$\Pr[\ A'.(f.U_n) \ \in \ f^{-1}.(f.U_n)\ ] \quad < \quad 1\,/\,p.n$$

We could remove the remaining brackets by appealing to function composition:

$$\Pr[\ A' \circ f.U_n \ \in \ f^{-1} \circ f.U_n\ ] \quad < \quad 1\,/\,p.n$$

We could also eliminate the appeal to the inverse function, f^{-1}, and replace the set membership with an equality:

$$\Pr[\ f \circ A' \circ f.U_n \ = \ f.U_n\ ] \quad < \quad 1\,/\,p.n$$

The left hand side of the equality looks a little unwieldy, but from a manipulative point of view equivalence is generally preferred over set membership.

The use of A' to name implementations of f^{-1} in the one-wayness condition is unfortunate, as the name is hardly informative, and the prime unnecessarily adds to the proliferation of symbols; I presume A' is so named following the use of A to name the implementation of f in condition (1). But what should we replace A' with? We could introduce the convention that abstract functions are denoted by lower case letters, and their implementations are denoted by the same letter but in uppercase; so in the above we would replace A by F, and A' by F^{-1}.

After the above simple changes we arrive at,

$$\Pr[\ f \circ F^{-1} \circ f.U_n \ = \ f.U_n\ ] \quad < \quad 1\,/\,p.n$$

which certainly seems to be an improvement over

$$\Pr[A'(f(U_n), 1^n) \in f^{-1}(f(U_n))] < \frac{1}{p(n)} \qquad .$$

Similar syntactic improvements can be made to the one-wayness condition for weak one-way functions to yield:

$$\Pr[\ f \circ F^{-1} \circ f.U_n \ \neq \ f.U_n\ ] \quad > \quad 1\,/\,p.n$$

Observe that the $\Pr[\ \ldots\ ]$ notation used to denote probability is something of an oddity: $\Pr$ is clearly a function, so why the square brackets? Additionally, and potentially more seriously, it can be argued that the notation fails to meet the "unambiguous" requirement, in the sense that it was necessary to explain in the surrounding text that

> the probability in the second condition is taken over all the possible values assigned to U_n and all possible internal coin tosses of A', with uniform probability distribution.

*

Conceptually then, both strong and weak one-way functions involve

- notions of computational complexity; specifically, "deterministic polynomial-time" and "probabilistic polynomial-time" algorithms, where in both cases the one-wayness condition includes a universal quantification over the class of probabilistic polynomial-time algorithms
- probability: in both definitions the one-wayness condition is probabilistic
- a notion of asymptotics: in both cases the one-wayness condition should hold for "large enough n"
- notions of negligible and noticeable functions

Consequently, at the very least our context contains properties of probabilistic polynomial-time algorithms, theorems about probability and random variables, and theorems about asymptotics and polynomials.

* *

Goldreich's proof

Goldreich constructs a "ping-pong" proof of (4), ie a proof by mutual implication, where he first demonstrates that

(ping) $\exists$ weak one-way functions $\Leftarrow$ $\exists$ strong one-way functions

and subsequently that

(pong) $\exists$ weak one-way functions $\Rightarrow$ $\exists$ strong one-way functions

the latter being the more complex result to establish. A transcript of Goldreich's proof appears in Appendix A, where I have tried to preserve type-setting conventions as far as possible; readers are encouraged to read that version before reading the annotated version below. In the following analysis the quoted portions of Goldreich's proof are "framed" to aid readability.

*

Proof of ping

> Consider, for example, a one-way function f (which, without loss of generality, is length-preserving).

Length preserving functions were dealt with above; this restriction presents no problems. Observe that at this stage it is not specified whether f should be a weak or a strong one-way function.

> Modify f into a function g so that $g(p,x) = (p, f(x))$ if p starts with $\log_2 |x|$ zeros, and $g(p,x) = (p,x)$ otherwise, where (in both cases) $|p| = |x|$.

Clearly we are aiming for a constructive proof, but g is a rabbit: it is both contrived and unmotivated. Additionally, p seems a poor choice of name, as so far it has been used to denote a positive polynomial.

> We claim that g is a weak one-way function (because for all but a $\frac{1}{n}$ fraction of the strings of length $2n$ the function g coincides with the identity function).

Though it's not difficult to prove, I don't think it's particularly obvious that $1/n$ of the strings of length $2 \cdot n$ are prefixed with $\log_2 n$ zeros; also, observe that n has crept in as a pseudonym for $|x|$.

As it stands, it appears Goldreich is asserting that g is necessarily a weak one-way function because it corresponds to the identity function for $1/n$ of the possible strings of length $2 \cdot n$, but —and this is rather important— it will transpire that what he actually means is "g cannot be a strong one-way function because it coincides with the identity function for all but a $1/n$ fraction of the strings of length $2 \cdot n$" .

> To prove that g is weakly one-way, we use a "reducibility argument."

The notion of a reduction was explored briefly in the chapter introduction: we use a solution to one problem to solve another problem, in such a way that we can infer something about the difficulty of solving the latter based on the difficulty of solving the former. Since reduction plays a central role in reasoning about cryptographic constructions, it will require careful analysis in due course.

Proposition 2.3.1: *Let f be a one-way function (even in the weak sense). Then g, constructed earlier, is a weakly one-way function.*

Observe then, that f may be either a weak or a strong one-way function.

Intuitively, inverting g on inputs on which it does *not* coincide with the identity transformation is related to inverting f.

Agreed.

Thus, if g is inverted, on inputs of length $2n$, with probability that is noticeably greater than $1 - \frac{1}{n}$, then g must be inverted with noticeable probability on inputs to which g applies f. Therefore, if g is not weakly one-way, then neither is f.

It seems we are heading for a proof by contradiction, the goal being to show that if g can be inverted with probability that precludes it from being weakly one-way, then f cannot be weakly one-way; it follows that if f can be inverted with probability that precludes it from being weakly one-way, then it can't be strongly one-way. However, it's not clear what is meant by g being inverted with "probability that is noticeably greater than $1 - \frac{1}{n}$" , or why the probability should be noticeably greater than $1 - 1/n$.

The full, straightforward, but tedious proof follows.

Not the most encouraging statement. Though I don't intend to pursue the issue, the combination of the three adjectives "full" , "straightforward" , and "tedious" is intriguing: To what extent must full proofs be tedious? Are full proofs necessarily straightforward? Are straightforward proofs tedious? . . .

Given a probabilistic polynomial-time algorithm B' for inverting g, we construct a probabilistic polynomial-time algorithm A' that inverts f with "related" success probability.

The reduction is made rather more explicit here. However, since the demonstrandum has been left implicit, and in particular since the quantifiers have been left implicit, the justification for this step is unclear. Also, as already pointed out, the names A' and B' are hardly informative as they do nothing to suggest that A' is associated with f , and that B' is associated with g .

Following is the description of algorithm A'. On input y, algorithm A' sets $n \stackrel{\text{def}}{=} |y|$ and $l \stackrel{\text{def}}{=} \log_2 n$, selects p' uniformly in $\{0,1\}^{n-l}$, computes $z \stackrel{\text{def}}{=} B'(0^l p', y)$, and halts with output of the n-bit suffix of z.

As already explained, 0^l denotes a bit-string of l zeros. $0^l p'$ denotes the concatenation of 0^l with p'; concatenation of strings is often denoted by juxtaposition, but as remarked in the previous chapter, invisible operators are best avoided. (A small point, but observe how similar the l and the prime look when rendered as superscripts.) On closer inspection, Goldreich uses two different ways of denoting concatenation, namely juxtaposition and pairing: though the input to B' is denoted by the pair

$$(0^l p', y) \qquad ,$$

it is actually a single string, viz

$$0^l \text{ ++ } p' \text{ ++ } y \qquad ,$$

where I use ++ to explicitly denote string concatenation. I can see no advantage to this notational heterogeneity.

Let S_{2n} denote the sets of all $2n$-bit-long strings that start with $\log_2 n$ zeros (i.e., $S_{2n} \stackrel{\text{def}}{=} \{0^{\log_2 n}\alpha : \alpha \in \{0,1\}^{2n-\log_2 n}\}$). Then, by construction of A' and g, we have

$$\begin{aligned}
&\Pr[A'(f(U_n)) \in f^{-1}(f(U_n))] \\
&\quad \geq \Pr[B'(0^l U_{n-l}, f(U_n)) \in (0^l U_{n-l}, f^{-1}(f(U_n)))] \\
&\quad = \Pr[B'(g(U_{2n})) \in g^{-1}(g(U_{2n})) \,|\, U_{2n} \in S_{2n}] \\
&\quad \geq \frac{\Pr[B'(g(U_{2n})) \in g^{-1}(g(U_{2n}))] - \Pr[U_{2n} \notin S_{2n}]}{\Pr[U_{2n} \in S_{2n}]} \\
&\quad = n \cdot \left(\Pr[B'(g(U_{2n})) \in g^{-1}(g(U_{2n}))] - \left(1 - \frac{1}{n}\right)\right) \\
&\quad = 1 - n \cdot (1 - \Pr[B'(g(U_{2n}) \in g^{-1}(g(U_{2n}))])
\end{aligned}$$

(For the second inequality, we used $\Pr[A \,|\, B] = \frac{\Pr[A \cap B]}{\Pr[B]}$ and $\Pr[A \cap B] \geq \Pr[A] - \Pr[\neg B]$.)

The above is essentially a calculation. However, the lack of white space and the proliferation of symbols —a consequence of poor choice of notation— make it extremely hard to parse. In

addition to the syntactic difficulties, the lack of hints make the calculation hard to verify. Let's explore the shape of each step, and try to identify the concepts and properties being appealed to, where, as pointed out in the previous chapter, we proceed by determining what in each step is changed, and try to discover the justification for those changes. Here's the proof again, but with the steps numbered for reference:

$$
\begin{aligned}
& \Pr[A'(f(U_n) \in f^{-1}(f(U_n))] \\
\geq \quad & \{ \text{ Step 0 } \} \\
& \Pr[B'(0^l U_{n-l}, f(U_n)) \in (0^l U_{n-l}, f^{-1}(f(U_n)))] \\
= \quad & \{ \text{ Step 1 } \} \\
& \Pr[B'(g(U_{2n})) \in g^{-1}(g(U_{2n})) \,|\, U_{2n} \in S_{2n}] \\
\geq \quad & \{ \text{ Step 2 } \} \\
& (\Pr[B'(g(U_{2n})) \in g^{-1}(g(U_{2n}))] - \Pr[U_{2n} \notin S_{2n}]) \;/\; \Pr[U_{2n} \in S_{2n}] \\
= \quad & \{ \text{ Step 3 } \} \\
& n \cdot \left(\Pr[B'(g(U_{2n})) \in g^{-1}(g(U_{2n}))] - \left(1 - \tfrac{1}{n}\right)\right) \\
= \quad & \{ \text{ Step 4 } \} \\
& 1 - n \cdot (1 - \Pr[B'(g(U_{2n}) \in g^{-1}(g(U_{2n}))])
\end{aligned}
$$

First, observe that the expression

$$\Pr[B'(g(U_{2n})) \in g^{-1}(g(U_{2n}))]$$

appears on four of the six lines of the proof, but is not manipulated; that's a lot of syntactic baggage that's not contributing anything to the proof, other than to make it hard to identify what is changed in each of the final three steps.

The first step appears to involve manipulating the entire expression within the square brackets, where

$$A'(f(Un))$$

has been replaced by

$$B'(0^l U_{n-l}, f(U_n)) \qquad ,$$

and

$$f^{-1}(f(U_n))$$

has been replaced by

$$(0^l U_{n-l}, f^{-1}(f(U_n))) \qquad .$$

Clearly this involves an appeal (of sorts) to the definition of A', but it's not clear why this is a strengthening step. Observe that the resulting formula has the shape $x \in (a, b)$, where the brackets denote concatenation, as discussed above, but a is a bit-string and b is a set, so this is something of an abuse of notation.

In Step 1,

$$0^l U_{n-l}, f(U_n)$$

is replaced by

$$g(U_{2n}) \qquad ,$$

and

$$(0^l U_{n-l}, f^{-1}(f(U_n)))$$

is replaced by

$$g^{-1}(g(U_{2n})) \qquad ;$$

we additionally have the introduction of $\mid U_{2n} \in S_{2n}$ inside the square brackets. Briefly, the notation $\Pr[A \mid B]$ is used to denote conditional probability, ie the probability that A holds given that B holds, so in this case the condition is that $U_{2n} \in S_{2n}$, and hence that a uniformly selected bit-string of length $2 \cdot n$ is prefixed by $\log_2 n$ zeros. The reason for this seems straight forward: prior to this step, the input to B' is constructed in such a way that it is prefixed by $\log_2 n$ zeros, but a uniformly selected string of length $2 \cdot n$ need not be prefixed by $\log_2 n$ zeros.

Justification is given, albeit in the text following the proof, for Step 2, which apparently involves an appeal to two rules from probability, viz

$$\Pr[A \mid B] \;=\; \frac{\Pr[A \cap B]}{\Pr[B]} \qquad \text{and} \qquad \Pr[A \cap B] \;\geq\; \Pr[A] - \Pr[\neg B] \qquad .$$

Clearly the former is an equality preserving step, and the latter is a strengthening step, but as pointed out in the previous chapter, combining different types of steps in this way is best avoided. Observe that as stated, the rules are misleading: the presence of an intersection in both rules

suggests A and B are sets, though in the latter rule the presence of a $\neg$ suggests that B is a predicate. In the calculation, A and B, respectively

$$B'(g(U_{2n})) \in g^{-1}(g(U_{2n}))$$

and

$$U_{2n} \in S_{2n} \quad ,$$

are predicates, so the theorems appealed to must have the shape

$$(7) \qquad \Pr[A \,|\, B] \quad = \quad \frac{\Pr[A \wedge B]}{\Pr[B]} \quad ,$$

and

$$(8) \qquad \Pr[A \wedge B] \quad \geq \quad \Pr[A] - \Pr[\neg B] \quad .$$

In Step 3 the subexpressions $\Pr[U_{2n} \notin S_{2n}]$ and $\Pr[U_{2n} \in S_{2n}]$ are eliminated, presumably because we can conclude that the probability that a uniformly selected bit-string of length $2 \cdot n$ is not prefixed by $\log_2 n$ zeros, and hence is not in S_{2n}, is $1/n$, from which we know that the probability that a uniformly selected string is in S_{2n} must be $1 - 1/n$, allowing us to replace these expressions with concrete values (this is proved in the next chapter). Clearly there is some further algebraic manipulation going on following this substitution.

The final step involves only algebra.

> It should not come as a surprise that the above expression is meaningful only in case $\Pr[B'(g(U_{2n})) \in g^{-1}(g(U_{2n}))] > 1 - \frac{1}{n}$.

It's not clear which "above expression" is being referred to; Goldreich actually means the calculation, and hence the conclusion that

$$\Pr[A'(f(U_n) \in f^{-1}(f(U_n))] \quad \leq \quad 1 - n \cdot (1 - \Pr[B'(g(U_{2n}) \in g^{-1}(g(U_{2n}))]) \quad .$$

This caveat stems from the appeal to (8), where $A := B'(g(U_{2n}) \in g^{-1}(g(U_{2n}))$, and $B := U_{2n} \in S_{2n}$. Clearly, if $\Pr[A] < \Pr[B]$ then $\Pr[A] - \Pr[B]$ is negative, in which case

$$(\Pr[B'(g(U_{2n})) \in g^{-1}(g(U_{2n}))] - \Pr[U_{2n} \notin S_{2n}]) \;/\; \Pr[U_{2n} \in S_{2n}] \quad ,$$

and hence

$$1 - n \cdot (1 - \Pr[B'(g(U_{2n}) \in g^{-1}(g(U_{2n}))])$$

would be negative, which would not be a very useful result. If the two probabilities are the same, then subtracting the one from the other is clearly zero, which is also not a very useful result. As established above, the probability that $U_{2n} \notin S_n$ is $1 - 1/n$, so $\Pr[A]$, and hence $\Pr[B'(g(U_{2n})) \in g^{-1}(g(U_{2n}))]$, should indeed be greater than $1 - 1/n$.

But is $\Pr[B'(g(U_{2n})) \in g^{-1}(g(U_{2n}))] > 1 - 1/n$? Yes, because by the assumption that B' is not weakly one-way this probability is greater than $1 - 1\,/\,p(2n)$, which approaches 1 faster than $1 - 1/n$ for any non-constant, positive polynomial p. However, the lack of any explicit contradiction hypothesis means this fact remains somewhat hidden.

I think it's fair to say that this would have been far less confusing if the calculation had been rendered with inline hints, and had been carried out in slightly finer grained steps in the presence of an explicit contradiction hypothesis.

> It follows that for every polynomial $p(\cdot)$ and every integer n, if B' inverts g on $g(U_{2n})$ with probability greater than $1 - \frac{1}{p(2n)}$, then A' inverts f on $f(U_n)$ with probability greater than $1 - \frac{n}{p(2n)}$.

Again, since the contradiction hypothesis has been left implicit, it's not clear where $1 - 1\,/\,p(2n)$ comes from, and without extending the above calculation it's not at all clear that A' inverts f on $f(U_n)$ with probability greater than $1 - n\,/\,p(2n)$.

> Hence, if g is not weakly one-way (i.e., for every polynomial $p(\cdot)$ there exist infinitely many m's such that g can be inverted on $g(U_m)$ with probability $\geq 1 - 1/p(m)$), then also f is not weakly one-way (i.e., for every polynomial $q(\cdot)$ there exist infinitely many n's such that f can be inverted on $f(U_n)$ with probability $\geq 1 - 1/q(n)$, where $q(n) = p(2n)/n$). This contradicts our hypothesis (that f is weakly one-way).

Here the contradiction hypothesis, and what has been contradicted, is made explicit, albeit informally. However, it's not clear what is meant by "infinitely many m's", and "infinitely many n's". Also, it's not clear why $q(n)$ is defined as $p(2n)\,/\,n$, particularly as q is universally quantified; it's not even clear that $p(2n)\,/\,n$ is a polynomial.

> To summarize, given a probabilistic polynomial-time algorithm that inverts g on $g(U_{2n})$ with success probability $1-\frac{1}{n}+\alpha(n)$, we obtain a probabilistic polynomial-time algorithm that inverts f on $f(U_n)$ with success probability $n \cdot \alpha(n)$.

Presumably, $\alpha(n)$ corresponds to the noticeable probability mentioned at the start of the proof ("if g is inverted, on inputs of length $2n$, with probability noticeably greater than $1-\frac{1}{n}$ ..."), but it's not clear that A' inverts f on $f(U_n)$ with probability $n \cdot \alpha(n)$.

> Thus, since f is (weakly) one-way, $n \cdot \alpha(n) < 1-(1/q(n))$ must hold for some polynomial q, and so g must be weakly one-way (since each probabilistic polynomial-time algorithm trying to invert g on $g(U_{2n})$ must fail with probability at least $\frac{1}{n}-\alpha(n) > \frac{1}{n \cdot q(n)}$).

The latter part of the claim, viz that "each probabilistic polynomial-time algorithm ... must fail with probability ..." is not at all clear.

*

Proof of pong

> Let f be a weak one-way function, and let p be the polynomial guaranteed by the definition of a weak one-way function. Namely, every probabilistic polynomial-time algorithm fails to invert f on $f(U_n)$ with probability at least $\frac{1}{p(n)}$.

According to the definitions, that f is a weak one-way function means that for some polynomial p, every probabilistic polynomial-time algorithm fails to invert f on $f(U_n)$ with probability greater than $1/p(n)$, rather than at least $1/p(n)$ as suggested; in other words, the inequality is strict.

> We assume for simplicity that f is length-preserving (i.e. $|f(x)| = |x|$ for all x's). This assumption, which is not really essential, is justified by Proposition 2.2.5.

The restriction to length preserving functions was dealt with above; this presents no problems.

We define a function g as follows:

$$g(x_1, \ldots, x_{t(n)}) \stackrel{\text{def}}{=} f(x_1), \ldots, f(x_{t(n)}) \tag{2.5}$$

where $|x_1| = \cdots = |x_{t(n)}| = n$ and $t(n) \stackrel{\text{def}}{=} n \cdot p(n)$. Namely, the $n^2 p(n)$-bit-long input of g is partitioned into $t(n)$ blocks, each of length n, and f is applied to each block.

As with ping, we are again heading for a constructive proof; also as with ping, g counts as a rabbit. In particular, no explanation or motivation is given as to why the input to g comprises $t(n)$ blocks. Also, the name $t(n)$ is superfluous, as it is only used in one other place, where it could easily be replaced by $n \cdot p(n)$.

Suppose that g is not strongly one-way.

Clearly then, we are once again heading for a proof by contradiction.

By definition, it follows that there exists a probabilistic polynomial-time algorithm B' and a polynomial $q(\cdot)$ such that for infinitely many m's,

$$\Pr[B'(g(U_m)) \in g^{-1}(g(U_m))] > \frac{1}{q(m)} \tag{2.6}$$

This is the contradiction hypothesis. However, the negation has been carried out somewhat implicitly, which is quite a jump considering there are four existentially/universally quantified variables in the definition of strong (and indeed weak) one-way functions. Recall that according to the definitions, g is a strong one-way function if

$$\langle \forall B', q :: \langle \exists M :: \langle \forall m : m > M : \ \Pr[B'(g(U_m)) \in g^{-1}(g(U_m))] \ < \ \frac{1}{q(m)} \ \rangle\rangle\rangle$$

and so the negation of this, and hence the assumption that g is not strongly one-way, is, by virtue of the generalised De Morgan's theorem,

$$\langle \exists B', q :: \langle \forall M :: \langle \exists m : m > M : \ \Pr[B'(g(U_m)) \in g^{-1}(g(U_m))] \ \geq \ \frac{1}{q(m)} \ \rangle\rangle\rangle \qquad .$$

Consequently, the inequality in equation (2.6) should be $\geq$ rather than $>$. Observe that by virtue of the inner quantification, viz

$$\ldots \ \langle \ \forall M :: \langle \ \exists m : m > M : \ \ldots$$

equation (2.6) is indeed quantified over infinitely many values of m (more will be said about this in subsequent chapters), though this is not at all clear when the quantifiers are left implicit.

> Let us denote by M' the infinite set of integers for which this holds. Let N' denote the infinite set of n's for which $n^2 \cdot p(n) \in M'$ (note that all m's considered are of the form $n^2 \cdot p(n)$, for some integer n).

The name M' is superfluous as this is the only place it appears in the proof. That aside, the primes in N' and M' seem an unnecessary distraction.

> Using B', we now present a probabilistic polynomial-time algorithm A' for inverting f. On input y (supposedly in the range of f), algorithm A' proceeds by applying the following probabilistic procedure, denoted I, on input y for $a(|y|)$ times, where $a(\cdot)$ is a polynomial that depends on the polynomials p and q (specifically, we set $a(n) \stackrel{\text{def}}{=} 2n^2 \cdot p(n) \cdot q(n^2 p(n))$).

The polynomial $a(n)$ is a rabbit: no explanation or motivation is given in the proof or the surrounding text as to how this polynomial was derived; also, the decision to introduce it in a parenthetic comment is made all the more unfortunate by the closing parenthesis merging with the definition.

> **Procedure** I
>
> *Input*: y (denote $n \stackrel{\text{def}}{=} |y|$).
>
> For $i = 1$ to $t(n)$ do begin
>
> **1.** Select uniformly and independently a sequence of strings $x_1, \ldots, x_{t(n)} \in \{0,1\}^n$.
> **2.** Compute $(z_1, \ldots, z_{t(n)}) \leftarrow B'(f(x_1), \ldots, f(x_{i-1}), y, f(x_{i+1}), \ldots, f(x_{t(n)}))$.
> **3.** If $f(z_i) = y$, then halt and output z_i.
> (This is considered a *success*).
> end

Here we find, as mentioned above, the only other occurrence of $t(n)$.

Observe that Step 3 implicitly relies on f being easy to compute, and hence it being easy verify (in "polynomial-time") candidate solutions returned by B'. However, that this step is stated in terms of f, rather than in terms of f's (guaranteed) implementation, is perhaps a little misleading, since, as we'll see when we explore a little complexity theory, f being polynomial-time computable is integral to the construction of I, and hence A'.

Using Eq. (2.6), we now present a lower bound on the success probability of algorithm A'. To this end we define a set, denoted S_n, that contains all n-bit strings on which the procedure I succeeds with non-negligible probability (specifically, greater than $\frac{n}{a(n)}$). (The probability is taken only over the coin tosses of procedure I.) Namely,

$$S_n \stackrel{\text{def}}{=} \left\{ x : \Pr[I(f(x)) \in f^{-1}(f(x))] > \frac{n}{a(n)} \right\}$$

I have several issues here. The need to introduce S_n is neither explained nor motivated. The fraction $n\,/\,a(n)$ appears with no explanation, and in a parenthetic comment of all places. That S_n contains elements on which I succeeds with "non-negligible" probability requires clarification. And, it's not clear what is meant by, or why the probability is "taken only over the coin tosses of procedure I" .

In the next two claims we shall show that S_n contains all but at most a $\frac{1}{2p(n)}$ fraction of the strings of length $n \in N'$ ...

The fraction $1\,/\,2p(n)$ is yet another unexplained value.

... and that for each string $x \in S_n$ the algorithm A' inverts f on $f(x)$ with probability exponentially close to 1.

What exactly is meant by "exponentially close to 1" ?

It will follow that A' inverts f on $f(U_n)$, for $n \in N'$, with probability greater than $1 - \frac{1}{p(n)}$, in contradiction to our hypothesis.

Since the quantifiers and the contradiction hypothesis have been left implicit, it's not clear what this contradicts.

Claim 2.3.2.1: For every $x \in S_n$,

$$\Pr[A'(f(x)) \in f^{-1}(f(x))] > 1 - \frac{1}{2^n}$$

It seems "exponentially close to 1" means $1 - 1/2^n$, but why 2^n? Is 2 an arbitrary constant?

Proof: By definition of the set S_n, the procedure I inverts $f(x)$ with probability at least $\frac{n}{a(n)}$.

By definition of the set S_n, the procedure I inverts $f(x)$ with probability greater than $n/a(n)$, rather than at least $n/a(n)$ as suggested.

Algorithm A' merely repeats I for $a(n)$ times, and hence

$$\Pr[A'(f(x)) \notin f^{-1}(f(x))] < \left(1 - \frac{n}{a(n)}\right)^{a(n)} < \frac{1}{2^n}$$

The claim follows. □

When I first tried to verify this proof, neither of the steps shown, viz that

$$(9) \qquad \Pr[A'(f(x)) \notin f^{-1}(f(x))] \; < \; \left(1 - \frac{n}{a(n)}\right)^{a(n)}$$

and that

$$(10) \qquad \left(1 - \frac{n}{a(n)}\right)^{a(n)} \; < \; \frac{1}{2^n}$$

made sense to me, and it was unclear how to go about filling in the gaps. It will transpire that the former will make sense once we have investigated a little probability and complexity theory, in particular the notion of "amplification", and that the latter is reasonably straightforward if you are aware of the theorem

$$(11) \qquad (1 + x/n)^n \; \rightarrow \; e^x \qquad\qquad \text{as} \;\; n \rightarrow \infty$$

from analysis, which, unfortunately, is not mentioned in the proof, or indeed the textbook (I was not aware of this theorem until I discussed the proof with Chris Woodcock; my thanks to him for his help with this).

Claim 2.3.2.2: For every $n \in N'$,

$$|S_n| > \left(1 - \frac{1}{2p(n)}\right) \cdot 2^n$$

Proof: We assume to the contrary, that $|S_n| \leq (1 - \frac{1}{2p(n)}) \cdot 2^n$. We shall reach a contradiction to Eq. (2.6) (i.e., our hypothesis concerning the success probability of B').

Since equation (2.6) is essentially the contradiction hypothesis for pong, it's not clear that we are justified in contradicting it to establish Claim 2.3.2.2 ; certainly the structure of the proof is becoming rather entangled.

Recall that by this hypothesis (for $n \in N0$),

$$s(n) \stackrel{\text{def}}{=} \Pr[B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)}))] > \frac{1}{q(n^2p(n))} \qquad (2.7)$$

$N0$ is an obvious typo; unfortunate, but no big deal. More serious, misleading at least: we have a new definition prefixed with the words "recall that" , and as it stands it appears that $s(n)$ is defined as

$$\Pr[B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)}))] > \frac{1}{q(n^2p(n))} \quad ,$$

a boolean expression, rather than

$$\Pr[B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)}))] \quad .$$

Also, as explained above, the inequality in equation (2.6) should be "at least" , meaning we have $s(n) \geq 1 / q(n^2p(n))$.

Let $U_n^{(1)}, \ldots, U_n^{(n \cdot p(n))}$ denote the n-bit-long blocks in the random variable $U_{n^2p(n)}$ (i.e., these $U_n^{(i)}$'s are independent random variables each uniformly distributed in $\{0,1\}^n$).

Aside from the notation $U_n^{(i)}$ being ugly, I can see no reason why the superscripts need to be in parentheses.

> We partition the event considered in Eq. (2.7) into two disjoint events corresponding to whether or not one of the $U_n^{(i)}$'s resides out of S_n.

This is explored below.

> Intuitively, B' cannot perform well in such a case, since this case corresponds to the success probability of I on pre-images out of S_n. On the other hand, the probability that all $U_n^{(i)}$'s reside in S_n is small.

Agreed.

> Specifically, we define
>
> $$s_1(n) \stackrel{\text{def}}{=} \Pr[B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)})) \wedge (\exists i \text{ s.t. } U_n^{(i)} \notin S_n)]$$
>
> and
>
> $$s_2(n) \stackrel{\text{def}}{=} \Pr[B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)})) \wedge (\forall i : U_n^{(i)} \in S_n)]$$

The range of i has been left somewhat implicit here. Also, observe the avoidable notational heterogeneity in the use of s.t. versus : in $\exists i$ s.t. ... and $\forall i$: ... .

> Clearly, $s(n) = s_1(n) + s_2(n)$ (as the events considered in the s_i's are disjoint).

It's questionable just how "clear" this is. Briefly, as we have not yet explored probability theory, two "events" , ie predicates, P and Q are disjoint if

$$\langle \forall x :: P.x \ \wedge \ Q.x \quad \equiv \quad \mathbf{false} \rangle \qquad .$$

The probabilities s_1 and s_2 are indeed disjoint, since the conjunction of

$$\exists i \text{ s.t. } U_n^{(i)} \notin S_n$$

and

$$\forall i : U_n^{(i)} \in S_n$$

is **false** . However, it is not disjointness alone that ensures $s(n) = s_1(n) + s_2(n)$, but rather, it follows from the events in s_1 and s_2 partitioning the event described in s . More specifically, if two events Q and R partition an event P , then Q and R must be disjoint, but the converse need not hold. This is explored in Chapter 3 .

> We derive a contradiction to the lower bound on $s(n)$ (given in Eq. (2.7)) by presenting upper bounds for both $s_1(n)$ and $s_2(n)$ (which sum to less).

In other words, (2.7) (after correction) asserts that

$$s(n) \geq \frac{1}{q(n^2 p(n))} \quad ,$$

and a contradiction is derived by demonstrating that

$$s_1(n) + s_2(n) < \frac{1}{q(n^2 p(n))} \quad ,$$

where $s_1(n) + s_2(n) = s(n)$.

> First, we present an upper bound on $s_1(n)$. The key observation is that algorithm I inverts f on input $f(x)$ with probability that is related to the success of B' to invert g on a sequence of random f-images containing $f(x)$. Specifically, for every $x \in \{0,1\}^n$ and every $1 \leq i \leq n \cdot p(n)$, the probability that I inverts f on $f(x)$ is greater than or equal to the probability that B' inverts g on $g(U_{n^2 p(n)})$ conditioned on $U_n^{(i)} = x$ (since any success of B' to invert g means that f was inverted on the ith block, and thus contributes to the success probability of I). It follows that, for every $x \in \{0,1\}^n$ and every $1 \leq i \leq n \cdot p(n)$,
>
> $$\begin{aligned} &\Pr[I(f(x)) \in f^{-1}(f(x))] \\ &\quad \geq \Pr[B'(g(U_{n^2 p(n)})) \in g^{-1}(g(U_{n^2 p(n)})) \,|\, U_n^{(i)} = x] \end{aligned} \tag{2.8}$$

Agreed, though to clarify, I succeeds with probability at least that of B' since I runs B' a number of times.

> Since for $x \notin S_n$ the left-hand side (l.h.s.) cannot be large, we shall show that (the r.h.s and so) $s_1(n)$ cannot be large.

This claim is a little confusing, since equation (2.8) is in terms of $x \in \{0,1\}^n$, rather than $x \in S_n$.

Specifically, using Eq. (2.8), it follows that

$$\begin{aligned}
s_1(n) &= \Pr[\exists i \text{ s.t. } B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)})) \wedge U_n^{(i)} \notin S_n] \\
&\leq \sum_{i=1}^{n\cdot p(n)} \Pr[B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)})) \wedge U_n^{(i)} \notin S_n] \\
&\leq \sum_{i=1}^{n\cdot p(n)} \sum_{x \notin S_n} \Pr[B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)})) \wedge U_n^{(i)} = x] \\
&= \sum_{i=1}^{n\cdot p(n)} \sum_{x \notin S_n} \Pr[U_n^{(i)} = x] \cdot \Pr[B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)})) \,|\, U_n^{(i)} = x] \\
&\leq \sum_{i=1}^{n\cdot p(n)} \max_{x \notin S_n} \{\Pr[B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)})) \,|\, U_n^{(i)} = x]\} \\
&\leq \sum_{i=1}^{n\cdot p(n)} \max_{x \notin S_n} \{\Pr[I(f(x)) \in f^{-1}(f(x))]\} \\
&\leq n \cdot p(n) \cdot \frac{n}{a(n)} = \frac{n^2 \cdot p(n)}{a(n)}
\end{aligned}$$

(The last inequality uses the definition of S_n, and the one before it uses Eq. (2.8).)

First, observe that the expression

$$B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)})) \tag{12}$$

is repeated on five lines of the proof; that's a total of $30 \cdot 5 = 150$ symbols that are simply being carried around. Contrast this with the calculation in ping, where the expression

$$\Pr[B'(g(U_{2n})) \in g^{-1}(g(U_{2n}))]$$

appeared unmanipulated in four lines of a six line calculation. Abstracting a little, it seems that in both calculations the expression

$$B'(g(U_n)) \in g^{-1}(g(U_n))$$

ie, the core of the one-wayness condition, contributes only symbolic noise, rather than playing an integral role in the proof.

At first glance it appears the final few steps have merged, in the sense that the calculation seems to establish

$$\begin{array}{ll} & \sum_{i=1}^{n\cdot p(n)} \max_{x\notin S_n}\{\Pr[I(f(x)) \in f^{-1}(f(x))]\} \\ \leq & \quad \{\ ??\ \} \\ & n\cdot p(n)\cdot \frac{n}{a(n)} = \frac{n^2\cdot p(n)}{a(n)} \end{array}$$

rather than:

$$\begin{array}{ll} & \sum_{i=1}^{n\cdot p(n)} \max_{x\notin S_n}\{\Pr[I(f(x)) \in f^{-1}(f(x))]\} \\ \leq & \quad \{\ ??\ \} \\ & n\cdot p(n)\cdot \frac{n}{a(n)} \\ = & \quad \{\ ??\ \} \\ & \frac{n^2\cdot p(n)}{a(n)} \end{array}$$

On closer inspection it's clear that the former leads to a type error, but that's only evident once we've parsed both lines and realised our mistake; the latter avoids any possibility of misinterpretation. Here's the proof again, but with the steps numbered for reference, and the last two steps separated for clarity:

$$\begin{array}{ll} & s_1(n) \\ = & \quad \{\ \text{Step 0}\ \} \\ & \Pr[\exists i \text{ s.t. } B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)})) \wedge U_n^{(i)} \notin S_n] \\ \leq & \quad \{\ \text{Step 1}\ \} \\ & \sum_{i=1}^{n\cdot p(n)} \Pr[B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)})) \wedge U_n^{(i)} \notin S_n] \\ \leq & \quad \{\ \text{Step 2}\ \} \\ & \sum_{i=1}^{n\cdot p(n)} \sum_{x\notin S_n} \Pr[B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)})) \wedge U_n^{(i)} = x] \\ = & \quad \{\ \text{Step 3}\ \} \\ & \sum_{i=1}^{n\cdot p(n)} \sum_{x\notin S_n} \Pr[U_n^{(i)} = x] \cdot \Pr[B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)})) \,|\, U_n^{(i)} = x] \\ \leq & \quad \{\ \text{Step 4}\ \} \\ & \sum_{i=1}^{n\cdot p(n)} \max_{x\notin S_n}\{\Pr[B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)})) \,|\, U_n^{(i)} = x]\} \\ \leq & \quad \{\ \text{Step 5}\ \} \\ & \sum_{i=1}^{n\cdot p(n)} \max_{x\notin S_n}\{\Pr[I(f(x)) \in f^{-1}(f(x))]\} \\ \leq & \quad \{\ \text{Step 6}\ \} \end{array}$$

$$
\begin{array}{ll}
 & n \cdot p(n) \cdot \frac{n}{a(n)} \\
= & \quad \{ \text{ Step 7 } \} \\
 & \frac{n^2 \cdot p(n)}{a(n)}
\end{array}
$$

The first step appeals to the definition of $s_1(n)$, but also contains an implicit appeal to the distributivity of conjunction over existential quantification.

In Step 1 the existential quantification is effectively replaced with a summation. Abstracting a little, and using the Eindhoven triple notation, the step has the shape:

$$(13) \qquad \Pr[\langle \exists i : R : \ \ldots \rangle] \quad \leq \quad \langle \Sigma i : R : \Pr[\ \ldots \ ] \rangle$$

Stripping away the irrelevant details, we see that Step 2 has the shape:

$$(14) \qquad \Pr[U_n \notin S_n] \quad \leq \quad \langle \Sigma x : x \notin S_n : \Pr[U_n = x] \rangle$$

Step 3 has the shape:

$$(15) \qquad \Pr[A \wedge B] \quad = \quad \Pr[B] \cdot \Pr[A \,|\, B]$$

And Step 4 has the shape

$$(16) \qquad \langle \Sigma x : x \notin S_n : \Pr[U_n = x] \cdot \Pr[P \,|\, U_n = x] \rangle \quad \leq \quad \langle \uparrow \, x : x \notin S_n : \Pr[P \,|\, U_n = x] \rangle$$

where

$$\langle \uparrow \, x : R : \ \ldots \rangle \quad = \quad \max_R \{ \ \ldots \ \} \qquad .$$

In the text following the calculation, we are told that Step 5 is established using (2.8), which seems clear enough, provided, of course, we accept (2.8).

We are told that Step 6 is established using the definition of S_n, though it's not particularly obvious how. First, observe that

$$\langle \uparrow \, x : x \notin S_n : \Pr[I(f(x)) \in f^{-1}(f(x))] \rangle$$

is a constant, let's denote it c, where $x \in S_n$ if

$$\Pr[I(f(x)) \in f^{-1}(f(x))] \quad > \quad \frac{n}{a(n)} \qquad ,$$

meaning $x \notin S_n$ if

$$\Pr[I(f(x)) \in f^{-1}(f(x))] \quad \leq \quad \frac{n}{a(n)} \qquad ,$$

so $c \leq n \,/\, a(n)$; hence we have:

$\langle \Sigma i : 1 \leq i \leq n \cdot p(n) : c \rangle$
$\leq$ { above }
$\langle \Sigma i : 1 \leq i \leq n \cdot p(n) : \frac{n}{a(n)} \rangle$
$=$ { property of summation }
$n \cdot p(n) \cdot \frac{n}{a(n)}$

The final step is trivial.

We now present an upper bound on $s_2(n)$. Recall that by the contradiction hypothesis, $|S_n| \leq (1 - \frac{1}{2p(n)}) \cdot 2^n$. It follows that

$$\begin{aligned} s_2(n) \;&\leq\; \Pr[\forall i : U_n^{(i)} \in S_n] \\ &\leq\; \left(1 - \frac{1}{2p(n)}\right)^{n \cdot p(n)} \\ &<\; \frac{1}{2^{n/2}} \;<\; \frac{n^2 \cdot p(n)}{a(n)} \end{aligned}$$

(The last inequality holds for sufficiently large n.)

Clearly we have another calculation, but as with the previous calculation, the last two steps seem to have merged. The first step must involve at least an appeal to the definition of $s_2(n)$; however, the expression

$$B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)})) \qquad ,$$

ie, (12) , has been eliminated from the definition by appealing to a rule with the shape

$$(17) \qquad \Pr[A] \quad \geq \quad \Pr[A \wedge B] \qquad ,$$

which, as we'll see in due course, is a standard theorem from probability. Observe that where (12) was carried around unmanipulated in the previous calculation, here it is immediately eliminated.

The remaining steps, which establish that

$$(18) \qquad \left(1 - \frac{1}{2p(n)}\right)^{n \cdot p(n)} \quad < \quad \frac{n^2 \cdot p(n)}{a(n)} \qquad ,$$

are considerably harder to verify, though are similar to the latter part of the argument used to establish Claim 2.3.2.1 , with the penultimate step involving an appeal to (11) .

> Combining the upper bounds on the s_i's, we have $s_1(n) + s_2(n) < \frac{2n^2 \cdot p(n)}{a(n)} = \frac{1}{q(n^2 p(n))}$, where equality is by the definition of $a(n)$.

Agreed.

> Yet on the other hand, $s_1(n) + s_2(n) > \frac{1}{q(n^2 p(n))}$, where the inequality is due to Eq. (2.7). Contradiction is reached, and the claim follows. □

Here we have another erroneous inequality: as explained above, according to the hypothesis we have

$$s_1(n) + s_2(n) \ \geq \ \frac{1}{q(n^2 p(n))} \qquad .$$

Fortunately this does not cause any problem (beyond confusing the reader) since the contradictory bound established on s_1 and s_2 is strict, and so the claim is indeed established.

> Combining Claims 2.3.2.1 and 2.3.2.2, we obtain
>
> $$\begin{aligned} &\Pr[A'(f(U_n)) \in f^{-1}(f(U_n))] \\ &\quad \geq \ \Pr[A'(f(U_n)) \in f^{-1}(f(U_n)) \ \wedge \ U_n \in S_n] \\ &\quad = \ \Pr[U_n \in S_n] \cdot \Pr[A'(f(U_n)) \in f^{-1}(f(U_n)) \,|\, U_n \in S_n] \\ &\quad \geq \ \left(1 - \frac{1}{2p(n)}\right) \cdot (1 - 2^{-n}) > 1 - \frac{1}{p(n)} \end{aligned}$$

Here the pieces are put together to show that A' inverts f with probability that contradicts f being a weak one-way function; once again the final two steps appear on the same line. Here's the proof with the steps numbered (and the last two steps separated) :

$$
\begin{array}{ll}
 & \Pr[A'(f(U_n)) \in f^{-1}(f(U_n))] \\
\geq & \quad \{ \text{ Step 0 } \} \\
 & \Pr[A'(f(U_n)) \in f^{-1}(f(U_n)) \wedge U_n \in S_n] \\
= & \quad \{ \text{ Step 1 } \} \\
 & \Pr[U_n \in S_n] \cdot \Pr[A'(f(U_n)) \in f^{-1}(f(U_n)) \,|\, U_n \in S_n] \\
\geq & \quad \{ \text{ Step 2 } \} \\
 & \left(1 - \frac{1}{2p(n)}\right) \cdot (1 - 2^{-n}) \\
> & \quad \{ \text{ Step 3 } \} \\
 & 1 - \frac{1}{p(n)}
\end{array}
$$

Clearly the first step has the same shape as (17) , and the second step has the same shape as (15) .

Step 2 combines claims 2.3.2.1 and 2.3.2.2 , but it's not obvious how they are combined. Additionally, the inequality appears to be incorrect, since in both claims the inequalities are strict. Observe also that in the resulting formula we have $1/2^n$ written as 2^{-n} , which, although perfectly acceptable, is a little odd since fraction notation is used everywhere else in the proof.

The final step, viz

$$
(19) \qquad \left(1 - \frac{1}{2p(n)}\right) \cdot (1 - 2^{-n}) \quad > \quad 1 - \frac{1}{p(n)} \qquad ,
$$

is rather more difficult to establish, and as we will see, holds only for large enough n ; this step involves a considerably larger jump than the other three, and is far less intuitive.

> It follows that there exists a probabilistic polynomial-time algorithm (i.e., A') that inverts f on $f(U_n)$, for $n \in N'$, with probability greater than $1 - \frac{1}{p(n)}$. This conclusion, which follows from the hypothesis that g is not strongly one-way (i.e., Eq. (2.6)), stands in contradiction to the hypothesis that every probabilistic polynomial-time algorithm fails to invert f with probability at least $\frac{1}{p(n)}$, and the theorem follows.

Agreed, though this would be much clearer if the various hypotheses were made explicit. (As pointed out at the start of the proof, the assumption that f is weakly one-way means that every probabilistic polynomial-time algorithm fails to invert f with probability greater than $1/p(n)$, rather than with probability at least $1/p(n)$; fortunately this does not invalidate the proof, though it is a little careless.)

* *

Where to from here?

Having identified a number of difficulties and ambiguities in the definitions and the proofs, including deficiencies in the choice of notation and naming, structural difficulties, a number of incorrect inequalities in both proofs, and a number of unjustified proof steps, the goal in the remainder of the thesis is to try to resolve some of these difficulties by exploring the concepts involved, and filling in the gaps.

Clearly both proofs draw on theorems from probability. However, as also identified, the standard $\Pr[\ldots]$ notation used to denote probabilistic expressions is something of an oddity. In the next chapter I investigate an alternative notation for probability, which I use to explore elementary probability theory, and to justify some of the manipulations identified in Goldreich's proofs.

In both proofs it is left implicit that we are working in an asymptotic context where some of the properties hold only for large enough n. In Chapter 4 I explore asymptotics, both as a prerequisite to the subsequent chapter on complexity theory, and in order to verify the proof steps that hold only for large enough n.

In order to understand why one-way functions are defined in terms of probabilistic polynomial-time algorithms, and negligible and noticeable probabilities, in Chapter 5 I explore notions of computability and complexity theory.

In Chapter 6 I explore rather more explicitly (in terms of contexts) the structure of the proofs, focusing on the use of reduction and proof by contradiction, in particular the validity of the approach used to establish Claim 2.3.2.2, and exposing exactly what is assumed and what is contradicted.

Having explored the concepts that underly the definitions of one-way functions and Goldreich's proof of (4), and having examined the structure of Goldreich's proof, in Chapter 7 I reintroduce the definitions of one-way functions using the notation explored in the next few chapters, and then reexamine ping and pong.

With respect to ping, I present a slightly cleaner version of Goldreich's proof, and a far simpler, but non-constructive proof, which establishes that strongly one-way functions are by definition weakly one-way, from which it follows that the existence of strong one-way functions implies the existence of weak one-way functions. With respect to pong, I examine the strategy behind the proof, and then restructure the proof to eliminate a number of rabbits.

* * *

Chapter 3

Probability

As identified in the previous chapter, probability plays an important role in cryptography. However, as also identified, the standard $\Pr[\ \ldots\ ]$ notation is something of an oddity. In this chapter I explore the possibility of replacing the standard notation with a new interface, which I use to prove a number of results in elementary probability theory. My primary goal in this chapter is to justify the proof steps in ping and pong that draw on theorems from probability theory, exploring the utility of the new interface being a secondary goal.

Readers familiar with probability theory will observe that I avoid the usual exposition in terms of quantifying the likelihood of physical events occurring, canonical examples being the likelihood that when flipped a so-called "fair" coin lands "heads up", or the likelihood that if we roll a pair of dice they both show six.

* *

Basic concepts in probability theory

A (discrete) "probability space" is a pair comprising a (finite) set of values called a "sample space", and a "distribution": a function that assigns a "weight", a real in the closed interval $[0, 1]$, to each of the elements of the sample space, so that the combined weight equals 1; as a special —but important— case, a distribution is called "uniform" if every element is assigned the same weight.

Predicates over a sample space are referred to as "events". We say that the "probability" of an event is the weight of the subset it characterises; ie, the combined weight of the elements that truthify the event. So, letting d_S denote a distribution over some sample space S, the probability of an event P may be calculated as

$$(20) \qquad \langle \Sigma x \in S : P.x : d_S.x \rangle \qquad .$$

Remark. The terms "sample space" and "event" are used to distinguish between predicates written over sets with an associated distribution function, and predicates written over arbitrary sets, where in the latter case it makes no sense to talk about probabilities. Where no confusion can arise, the terms "predicate" and "event" will be used synonymously.
End of Remark.

A "conditional event" is a pair of predicates: a predicate over a given sample space that "induces" a new probability space (comprising an induced sample space and an induced distribution over that sample space), and a predicate over the induced probability space. More specifically, given a probability space (S, d_S), we say that an event Q is "conditioned on" an event P, a predicate over S, if P induces a new sample space, S^P, where

$$S^P \quad = \quad \langle x \in S : P.x : x \rangle \qquad ,$$

and a distribution over S^P, and Q is an event over S^P. The induced distribution over S^P, and hence the probability of Q is slightly more complex than the induced sample space, since the total weight of a distribution must equal 1 by definition, so unless $S^P = S$, the weights are not simply inherited. In general, denoting the induced distribution d_{S^P}, we have

$$d_{S^P}.x \quad = \quad d_S.x \ / \ \text{probability of } P$$

ie,

$$d_{S^P}.x \quad = \quad d_S.x \ / \ c \qquad ,$$

where $c = \langle \Sigma x : P.x : d_S.x \rangle$. It follows that the probability of Q conditioned on P is

$$\begin{array}{ll} & \langle \Sigma x \in S^P : Q.x : d_{S^P}.x \rangle \\ = & \quad \{ \text{ definition of } d_{S^P} \ \} \\ & \langle \Sigma x \in S^P : Q.x : d_S.x \ / \ c \rangle \\ = & \quad \{ \ x \in S^P \text{ is equivalent to } x \in S \ \wedge \ P.x \ \} \\ & \langle \Sigma x \in S : P.x \ \wedge \ Q.x : d_S.x \ / \ c \rangle \\ = & \quad \{ \ / \ \text{ over } \Sigma \ \} \\ & \langle \Sigma x \in S : P.x \ \wedge \ Q.x : d_S.x \rangle \ / \ c \end{array}$$

Consequently, replacing c with $\langle \Sigma x : P.x : d_S.x \rangle$, the probability of Q conditioned on P may be calculated as

$$(21) \qquad \langle \Sigma x : P.x \ \wedge \ Q.x : d_S.x \rangle \ / \ \langle \Sigma x : P.x : d_S.x \rangle \qquad .$$

If P has probability zero we say that the (conditional) probability is "undefined", since $x \ / \ 0$

is undefined.

* *

Competing notations

Though there are a number of notations in use for probability, the "standard" notation, in the sense of being the most commonly used, looks something like

$$\Pr[P] \quad ,$$

which we usually read as "the probability that P holds", where P is an event. Conditional probability is usually denoted

$$\Pr[P \mid Q] \quad ,$$

which we usually read as "the probability that P holds given that Q holds", or "the probability of P conditioned on Q", where P and Q are events.

As Fokkinga points out [21], the problem with this notation is that it leaves the domain implicit; so for example, in

$$\Pr[P.x \equiv Q.y \equiv R.z]$$

there is no way to identify which variables are "bound", in the sense of being associated with a distribution, and which are "free". In order to remedy this situation, Fokkinga proposes the notation

$$(22) \qquad \langle \mathcal{P} D : Q : P \rangle \quad ,$$

which he suggests should be read as "the probability that P holds for a random draw from D that satisfies Q" [21].

Remark. Actually, Fokkinga proposes the notation

$$(\mathcal{P} D \mid P \bullet Q) \quad ,$$

in keeping with the Z specification notation.
End of Remark.

Fokkinga's notation makes the domain explicit, allowing us to refer in P and Q to the bound variables declared in D. For example, if the sample space is a subset of the naturals, then

$$\langle \mathcal{P}x : x \geq 3 : \text{odd}.(x \cdot y) \rangle$$

denotes the probability that $x \cdot y$ is odd, conditioned on x being at least 3; here x is "bound", but y is "free".

Translating between the standard notation and (my version of) Fokkinga's notation, we have

$$(23) \qquad \langle \mathcal{P}D : Q : P \rangle \quad = \quad \Pr[P \,|\, Q] \qquad ,$$

and:

$$(24) \qquad \langle \mathcal{P}D :: P \rangle \quad = \quad \Pr[P]$$

We may raise several objections to this use of the quantifier notation. First, what kind of operation is $\mathcal{P}$? Certainly it doesn't give rise to a monoid: it doesn't even make sense to talk about associativity of probabilities. Second, the type of the term does not denote the type of the quantification: in (22), P is an event, and hence a boolean expression, but (22) is of type real. Part of the beauty of the quantifier notation is uniformity with respect to the rules we may use to manipulate the range and so on, but by abusing the notation in this way we are obliged to develop a new set of rules. The question then, is can we live with this notational overloading? I propose to try the notation and see how it works out...

* *

A few theorems

Fokkinga observes that for uniform distributions we can take the following interpretation:

$$(25) \qquad \langle \mathcal{P}D : Q : P \rangle \quad = \quad \langle \#D :: P \wedge Q \rangle \;/\; \langle \#D :: Q \rangle$$

However, we would like a more general interpretation that holds also for non-uniform distributions. In keeping with the above discussion of the basic concepts, I reintroduce an explicit distribution function. Specifically, if S is a sample space, then I denote the distribution over S by d_S. Where only a single probability space is involved and no confusion can arise, I omit both the type

information from the quantification, and the subscript from the distribution function. So, by virtue of (21) we have:

$$(26) \qquad \langle \mathcal{P}x : Q.x : P.x \rangle \quad = \quad \langle \Sigma x : Q.x \wedge P.x : d.x \rangle \ / \ \langle \Sigma x : Q.x : d.x \rangle$$

By virtue of (20) , for empty range we have the simpler interpretation

$$(27) \qquad \langle \mathcal{P}x :: P.x \rangle \quad = \quad \langle \Sigma x : P.x : d.x \rangle \qquad\qquad ;$$

alternatively, we can derive (27) using (26) as follows:

$\langle \mathcal{P}x :: P.x \rangle$
$=$ { (26) }
$\langle \Sigma x : P.x : d.x \rangle \ / \ \langle \Sigma x :: d.x \rangle$
$=$ { distributions sum to 1 }
$\langle \Sigma x : P.x : d.x \rangle \ / \ 1$
$=$ { fractions }
$\langle \Sigma x : P.x : d.x \rangle$

As a consequence of (27) we have

$$(28) \qquad \langle \mathcal{P}x : P.x : Q.x \rangle \quad = \quad \langle \mathcal{P}x :: P.x \wedge Q.x \rangle \ / \ \langle \mathcal{P}x :: P.x \rangle \qquad\qquad ,$$

which allows us to rewrite conditional probabilities in terms of non-conditional probabilities, and which we observe, following appeals to (23) and (24) , is equivalent to (7) (Page 38) . Proof of (28) :

$\langle \mathcal{P}x :: P.x \ \wedge \ Q.x \rangle \ / \ \langle \mathcal{P}x :: P.x \rangle$
$=$ { (27) twice }
$\langle \Sigma x : P.x \ \wedge \ Q.x : d.x \rangle \ / \ \langle \Sigma x : P.x : d.x \rangle$
$=$ { (26) }
$\langle \mathcal{P}x : P.x : Q.x \rangle$

We can rewrite (28) as

$$(29) \qquad \langle \mathcal{P}x :: P.x \wedge Q.x \rangle \quad = \quad \langle \mathcal{P}x : P.x : Q.x \rangle \cdot \langle \mathcal{P}x :: P.x \rangle \qquad\qquad ,$$

which is sometimes referred to as the "multiplication theorem for conditional probability" , and which, after translation via (23) and (24) , we observe is equivalent to (15) (Page 50) .

As a consequence of (29) and the symmetry of conjunction, we have what is sometimes called the "product rule for probabilities",

$$(30) \qquad \langle \mathcal{P}x :: P.x \rangle \cdot \langle \mathcal{P}x : P.x : Q.x \rangle \quad = \quad \langle \mathcal{P}x :: Q.x \rangle \cdot \langle \mathcal{P}x : Q.x : P.x \rangle \qquad ,$$

which we can rewrite as

$$(31) \qquad \langle \mathcal{P}x : P.x : Q.x \rangle \quad = \quad \langle \mathcal{P}x : Q.x : P.x \rangle \cdot \langle \mathcal{P}x :: Q.x \rangle \;/\; \langle \mathcal{P}x :: P.x \rangle \qquad ,$$

to arrive at what is known as "Bayes' theorem", which is perhaps an overly grand title as a single application of (29) gets us back to (28).

*

It is common to refer to $\mathbf{true}$ as the "sure event", since

$$(32) \qquad \langle \mathcal{P}x :: \mathbf{true} \rangle \;=\; 1 \qquad ;$$

proof:

$$\begin{array}{ll} & \langle \mathcal{P}x :: \mathbf{true} \rangle \\ = & \quad \{\ (27)\ \} \\ & \langle \Sigma x :: d.x \rangle \\ = & \quad \{\ \text{distributions sum to } 1\ \} \\ & 1 \end{array}$$

Similarly, it is common to refer to $\mathbf{false}$ as the "impossible event", since

$$(33) \qquad \langle \mathcal{P}x :: \mathbf{false} \rangle \;=\; 0 \qquad ;$$

proof:

$$\begin{array}{ll} & \langle \mathcal{P}x :: \mathbf{false} \rangle \\ = & \quad \{\ (27)\ \} \\ & \langle \Sigma x : \mathbf{false} : d.x \rangle \\ = & \quad \{\ \text{empty range}\ \} \\ & 0 \end{array}$$

*

We can use (27) and the properties of summation to establish

$$(34) \qquad \langle \mathcal{P}x :: P.x \ \wedge \ Q.x \rangle \quad \leq \quad \langle \mathcal{P}x :: P.x \rangle$$

and

$$(35) \qquad \langle \mathcal{P}x :: P.x \rangle \quad \leq \quad \langle \mathcal{P}x :: P.x \ \vee \ Q.x \rangle \qquad .$$

Observe that after translation (34) is equivalent to (17) (Page 51). Proof of (34):

$$\begin{array}{ll} & \langle \mathcal{P}x :: P.x \ \wedge \ Q.x \rangle \\ = & \quad \{ \ (27) \ \} \\ & \langle \Sigma x : P.x \ \wedge \ Q.x : d.x \rangle \\ \leq & \quad \{ \text{ range weakening } \} \\ & \langle \Sigma x : P.x : d.x \rangle \\ = & \quad \{ \ (27) \ \} \\ & \langle \mathcal{P}x :: P.x \rangle \end{array}$$

The proof of (35) proceeds in essentially the same way as for (34), except we strengthen the range of the summation instead of weakening it. Observe that by virtue of $\leq$'s transitivity, combining (34) and (35) we have:

$$(36) \qquad \langle \mathcal{P}x :: P.x \ \wedge \ Q.x \rangle \quad \leq \quad \langle \mathcal{P}x :: P.x \ \vee \ Q.x \rangle$$

*

The following is easily proved using (34), (35), or (36):

$$(37) \qquad \langle \forall x :: P.x \ \Rightarrow \ Q.x \rangle \qquad \Rightarrow \qquad \langle \mathcal{P}x :: P.x \rangle \quad \leq \quad \langle \mathcal{P}x :: Q.x \rangle$$

We can start with $\langle \mathcal{P}x :: P.x \rangle$ and appeal to (34):

$$\begin{array}{ll} & \langle \mathcal{P}x :: P.x \rangle \\ = & \quad \{ \text{ antecedent } \} \\ & \langle \mathcal{P}x :: P.x \ \wedge \ Q.x \rangle \\ \leq & \quad \{ \ (34) \ \} \\ & \langle \mathcal{P}x :: Q.x \rangle \end{array}$$

Conversely we can start with $\langle \mathcal{P}x :: Q.x \rangle$ and appeal to (35) :

$$
\begin{array}{ll}
 & \langle \mathcal{P}x :: Q.x \rangle \\
= & \quad \{ \text{ antecedent } \} \\
 & \langle \mathcal{P}x :: P.x \ \vee \ Q.x \rangle \\
\geq & \quad \{ \ (35) \ \} \\
 & \langle \mathcal{P}x :: P.x \rangle
\end{array}
$$

Or we can appeal to (36) :

$$
\begin{array}{ll}
 & \langle \mathcal{P}x :: P.x \rangle \\
= & \quad \{ \text{ antecedent } \} \\
 & \langle \mathcal{P}x :: P.x \ \wedge \ Q.x \rangle \\
\leq & \quad \{ \ (36) \ \} \\
 & \langle \mathcal{P}x :: P.x \ \vee \ Q.x \rangle \\
= & \quad \{ \text{ antecedent } \} \\
 & \langle \mathcal{P}x :: Q.x \rangle
\end{array}
$$

This is my preferred proof as the symmetry is so appealing.

Similarly, we can use the above bounds to prove:

$$(38) \qquad 0 \ \leq \ \langle \mathcal{P}x :: P.x \rangle \ \leq \ 1$$

Here's one possible proof:

$$
\begin{array}{ll}
 & 0 \\
= & \quad \{ \ (33) \ \} \\
 & \langle \mathcal{P}x :: \mathbf{false} \rangle \\
\leq & \quad \{ \ (34) \ \text{ with } \ P \, , Q := \ P \, , \mathbf{false} \ \} \\
 & \langle \mathcal{P}x :: P.x \rangle \\
\leq & \quad \{ \ (34) \ \text{ with } \ P \, , Q := \ P \, , \mathbf{true} \ \} \\
 & \langle \mathcal{P}x :: \mathbf{true} \rangle \\
= & \quad \{ \ (32) \ \} \\
 & 1
\end{array}
$$

Here's Fokkinga's proof (albeit rendered in a more familiar format) based on the interpretation

given in (25), for (admittedly rather unfair) comparison:

$$
\begin{array}{ll}
 & 0 \ \leq \ \langle \mathcal{P} D : Q : P \rangle \ \leq \ 1 \\
= & \quad \{ \ (25) \ \} \\
 & 0 \ \leq \ \langle \# D :: P \wedge Q \rangle \ / \ \langle \# D :: Q \rangle \ \leq \ 1 \\
\Leftarrow & \quad \{ \ \text{number calculus} \ \} \\
 & 0 \ \leq \ \langle \# D :: P \wedge Q \rangle \ \leq \ \langle \# D :: Q \rangle \\
\Leftarrow & \quad \{ \ \text{set calculus} \ \} \\
 & \langle D :: P \wedge Q \rangle \ \subseteq \ \langle D :: Q \rangle \\
\Leftarrow & \quad \{ \ \text{set calculus} \ \} \\
 & (P \wedge Q) \ \Leftarrow \ Q \\
= & \quad \{ \ \text{propositional calculus} \ \} \\
 & \textbf{true}
\end{array}
$$

*

We can use (27) and range splitting over summations to prove the following rather elegant looking theorem:

$$(39) \qquad \langle \mathcal{P} x :: P.x \ \vee \ Q.x \rangle + \langle \mathcal{P} x :: P.x \ \wedge \ Q.x \rangle \quad = \quad \langle \mathcal{P} x :: P.x \rangle + \langle \mathcal{P} x :: Q.x \rangle$$

Proof:

$$
\begin{array}{ll}
 & \langle \mathcal{P} x :: P.x \ \vee \ Q.x \rangle + \langle \mathcal{P} x :: P.x \ \wedge \ Q.x \rangle \\
= & \quad \{ \ (27) \ \text{twice} \ \} \\
 & \langle \Sigma x : P.x \ \vee \ Q.x : d.x \rangle + \langle \Sigma x : P.x \ \wedge \ Q.x : d.x \rangle \\
= & \quad \{ \ \text{range split} \ \} \\
 & \langle \Sigma x : P.x : d.x \rangle + \langle \Sigma x : Q.x : d.x \rangle \\
= & \quad \{ \ (27) \ \text{twice} \ \} \\
 & \langle \mathcal{P} x :: P.x \rangle + \langle \mathcal{P} x :: Q.x \rangle
\end{array}
$$

*

We encountered the notion of "disjoint" events in connection with the definitions of s_1 and s_2, where two events are disjoint, or "mutually exclusive", if

$$\langle \forall x :: P.x \ \wedge \ Q.x \quad \equiv \quad \mathbf{false} \rangle \qquad .$$

Remark. While writing this I searched wikipedia for "disjoint events"; rather bizarrely the list of results included "Tara Palmer-Tomkinson", "Elektra: Assassin", and "The Lovely Bones", a story about "a teenage girl who, after being brutally raped and murdered, watches from heaven as her family and friends go on with their lives, while she herself comes to terms with her own death.".
End of Remark.

For disjoint P and Q we have:

$$(40) \qquad \langle \mathcal{P}x :: P.x \ \vee \ Q.x \rangle \quad = \quad \langle \mathcal{P}x :: P.x \rangle + \langle \mathcal{P}x :: Q.x \rangle$$

Proof:

$$\begin{array}{ll} & \langle \mathcal{P}x :: P.x \rangle + \langle \mathcal{P}x :: Q.x \rangle \\ = & \qquad \{ \ (39) \ \} \\ & \langle \mathcal{P}x :: P.x \ \vee \ Q.x \rangle + \langle \mathcal{P}x :: P.x \ \wedge \ Q.x \rangle \\ = & \qquad \{ \ P \ \text{and} \ Q \ \text{are disjoint} \ \} \\ & \langle \mathcal{P}x :: P.x \ \vee \ Q.x \rangle + \langle \mathcal{P}x :: \mathbf{false} \rangle \\ = & \qquad \{ \ (33) \ ; \ \text{arithmetic} \ \} \\ & \langle \mathcal{P}x :: P.x \ \vee \ Q.x \rangle \end{array}$$

Two disjoint events Q and R, "partition" an event P if they "cover" P, ie, if $P \equiv Q \vee R$, in which case we have:

$$\begin{array}{ll} & \langle \mathcal{P}x :: P.x \rangle \\ = & \qquad \{ \ P \ \equiv \ Q \vee R \ \} \\ & \langle \mathcal{P}x :: Q.x \ \vee \ R.x \rangle \\ = & \qquad \{ \ (40) \ \} \\ & \langle \mathcal{P}x :: Q.x \rangle + \langle \mathcal{P}x :: R.x \rangle \end{array}$$

Which explains why $s = s_1 + s_2$ in the proof of pong.

*

We can use (40) to prove (8) (Page 38), which, after translation, we can restate as:

$$(41) \qquad \langle \mathcal{P}x :: P.x \ \wedge \ Q.x \rangle \quad \geq \quad \langle \mathcal{P}x :: P.x \rangle - \langle \mathcal{P}x :: \neg Q.x \rangle$$

We first observe that (41) can be rewritten as

$$\langle \mathcal{P}x :: P.x \ \wedge \ Q.x \rangle + \langle \mathcal{P}x :: \neg Q.x \rangle \quad \geq \quad \langle \mathcal{P}x :: P.x \rangle$$

and then we calculate:

$$\begin{array}{ll} & \langle \mathcal{P}x :: P.x \ \wedge \ Q.x \rangle + \langle \mathcal{P}x :: \neg Q.x \rangle \\ = & \qquad \{ \ (40) : \ (P \ \wedge \ Q) \ \wedge \ \neg Q \quad \equiv \quad \mathbf{false} \ \} \\ & \langle \mathcal{P}x :: (P.x \ \wedge \ Q.x) \ \vee \ \neg Q.x \rangle \\ = & \qquad \{ \text{ predicate calculus } \} \\ & \langle \mathcal{P}x :: P.x \ \vee \ \neg Q.x \rangle \\ \geq & \qquad \{ \ (35) \ \} \\ & \langle \mathcal{P}x :: P.x \rangle \end{array}$$

Observe that $\langle \mathcal{P}x :: P.x \rangle - \langle \mathcal{P}x :: \neg \, Q.x \rangle$ may be negative, which may or may not be important.

*

Observe that for any three mutually disjoint events we have:

$$\begin{array}{ll} & \langle \mathcal{P}x :: P.x \rangle + \langle \mathcal{P}x :: Q.x \rangle + \langle \mathcal{P}x :: R.x \rangle \\ = & \qquad \{ \ (40) : \ P \wedge Q \ \equiv \ \mathbf{false} \ \} \\ & \langle \mathcal{P}x :: P.x \ \vee \ Q.x \rangle + \langle \mathcal{P}x :: R.x \rangle \\ = & \qquad \{ \ (40) : \ (P \vee Q) \wedge R \ \equiv \ \mathbf{false} \ \} \\ & \langle \mathcal{P}x :: P.x \ \vee \ Q.x \ \vee \ R.x \rangle \end{array}$$

From which it should be clear that we can generalise (40) to arbitrary collections of mutually disjoint events; ie:

$$(42) \qquad \langle \mathcal{P}x :: \langle \exists P :: P.x \rangle \rangle \quad = \quad \langle \Sigma P :: \langle \mathcal{P}x :: P.x \rangle \rangle$$

Similarly, if Q ranges over a collection of events that partition a given event P, then:

$$(43) \qquad \langle \mathcal{P}x :: P \rangle \quad = \quad \langle \Sigma Q :: \langle \mathcal{P}x :: Q.x \rangle \rangle$$

*

We can use (42) to prove the following theorem, known variously as the "law of total probability", or the "law of alternatives". Let P range over a collection of events that partition the sample space, then:

$$(44) \qquad \langle \mathcal{P}x :: Q.x \rangle \quad = \quad \langle \Sigma P :: \langle \mathcal{P}x :: P.x \ \wedge \ Q.x \rangle \rangle$$

Proof:

$$\begin{array}{ll} & \langle \Sigma P :: \langle \mathcal{P}x :: P.x \ \wedge \ Q.x \rangle \rangle \\ = & \qquad \{ \ (42) \ \} \\ & \langle \mathcal{P}x :: \langle \exists P :: P.x \ \wedge \ Q.x \rangle \rangle \\ = & \qquad \{ \ \wedge \ \text{over} \ \exists \ \} \\ & \langle \mathcal{P}x :: Q.x \ \wedge \ \langle \exists P :: P.x \rangle \rangle \\ = & \qquad \{ \ \text{the} \ P \ \text{events partition the sample space} \ \} \\ & \langle \mathcal{P}x :: Q.x \ \wedge \ \mathbf{true} \rangle \\ = & \qquad \{ \ \text{unit of conjunction} \ \} \\ & \langle \mathcal{P}x :: Q.x \rangle \end{array}$$

*

A few additional theorems based on (40):

$$(45) \qquad \langle \mathcal{P}x :: P.x \rangle \quad = \quad \langle \mathcal{P}x :: P.x \ \wedge \ Q.x \rangle + \langle \mathcal{P}x :: P.x \ \wedge \ \neg Q.x \rangle$$
$$(46) \qquad \langle \mathcal{P}x :: P.x \ \vee \ Q.x \rangle \quad = \quad \langle \mathcal{P}x :: P.x \rangle + \langle \mathcal{P}x :: \neg P.x \ \wedge \ Q.x \rangle$$
$$(47) \qquad \langle \mathcal{P}x :: P.x \rangle + \langle \mathcal{P}x :: \neg P.x \rangle \quad = \quad 1$$

Proof of (45):

$$\begin{array}{ll} & \langle \mathcal{P}x :: P.x \ \wedge \ Q.x \rangle + \langle \mathcal{P}x :: P.x \ \wedge \ \neg Q.x \rangle \\ = & \qquad \{ \ (40) : \ (P \wedge Q) \ \wedge \ (P \wedge \neg Q) \quad \equiv \quad \mathbf{false} \ \} \\ & \langle \mathcal{P}x :: (P.x \ \wedge \ Q.x) \ \vee \ (P.x \ \wedge \ \neg Q.x) \rangle \\ = & \qquad \{ \ \text{predicate calculus} \ \} \\ & \langle \mathcal{P}x :: P.x \rangle \end{array}$$

Proof of (46) :

$$
\begin{array}{ll}
 & \langle \mathcal{P}x :: P.x \rangle + \langle \mathcal{P}x :: \neg P.x \ \wedge \ Q.x \rangle \\
= & \quad \{ \ (40) : \ \ P \ \wedge \ (\neg P \wedge Q) \quad \equiv \quad \mathbf{false} \ \} \\
 & \langle \mathcal{P}x : P.x \ \vee \ (\neg P.x \ \wedge \ Q.x) \rangle \\
= & \quad \{ \text{ predicate calculus } \} \\
 & \langle \mathcal{P}x :: P.x \ \vee \ Q.x \rangle
\end{array}
$$

Proof of (47) :

$$
\begin{array}{ll}
 & \langle \mathcal{P}x :: P.x \rangle + \langle \mathcal{P}x :: \neg P.x \rangle \\
= & \quad \{ \ (40) : \ \ P \wedge \neg P \ \equiv \ \mathbf{false} \} \\
 & \langle \mathcal{P}x :: P.x \ \vee \ \neg P.x \rangle \\
= & \quad \{ \text{ excluded middle ; } \ (32) \ \} \\
 & 1
\end{array}
$$

*

Observe that we can rewrite (47) as

$$(48) \qquad \langle \mathcal{P}x :: P.x \rangle \ = \ 1 - \langle \mathcal{P}x :: \neg P.x \rangle \qquad ,$$

which we can use to relate an event to its negation, as in the following useful theorem:

$$(49) \qquad \langle \mathcal{P}x :: P.x \rangle \quad \nabla \quad \epsilon \qquad \equiv \qquad 1 - \epsilon \quad \nabla \quad \langle \mathcal{P}x :: \neg P.x \rangle$$

Where ∇ denotes any of $=$, $<$, $>$, $\leq$, $\geq$. Proof of (49) :

$$
\begin{array}{ll}
 & 1 - \epsilon \quad \nabla \quad \langle \mathcal{P}x :: \neg P.x \rangle \\
\equiv & \quad \{ \ (48) \ \} \\
 & 1 - \epsilon \quad \nabla \quad 1 - \langle \mathcal{P}x :: P.x \rangle \\
\equiv & \quad \{ \text{ arithmetic } \} \\
 & \langle \mathcal{P}x :: P.x \rangle \quad \nabla \quad \epsilon
\end{array}
$$

*

The next theorem is sometimes —ie, in the set theoretic view of probability— referred to as the "union bound", and sometimes as "Boole's inequality":

$$(50) \qquad \langle \mathcal{P}x :: P.x \ \vee \ Q.x \rangle \quad \leq \quad \langle \mathcal{P}x :: P.x \rangle + \langle \mathcal{P}x :: Q.x \rangle$$

Proof:

$$\begin{array}{ll} & \langle \mathcal{P}x :: P.x \rangle + \langle \mathcal{P}x :: Q.x \rangle \\ = & \quad \{ \ (39) \ \} \\ & \langle \mathcal{P}x :: P.x \ \vee \ Q.x \rangle + \langle \mathcal{P}x :: P.x \ \wedge \ Q.x \rangle \\ \geq & \quad \{ \text{ arithmetic (observing that probabilities are non-negative) } \} \\ & \langle \mathcal{P}x :: P.x \ \vee \ Q.x \rangle \end{array}$$

Observe that for any three events we have:

$$\begin{array}{ll} & \langle \mathcal{P}x :: P.x \rangle + \langle \mathcal{P}x :: Q.x \rangle + \langle \mathcal{P}x :: R.x \rangle \\ \geq & \quad \{ \ (50) \ \} \\ & \langle \mathcal{P}x :: P.x \ \vee \ Q.x \rangle + \langle \mathcal{P}x :: R.x \rangle \\ \geq & \quad \{ \ (50) \ \} \\ & \langle \mathcal{P}x :: P.x \ \vee \ Q.x \ \vee \ R.x \rangle \end{array}$$

From which it should be clear that we can generalise (50) to arbitrary collections of events:

$$(51) \qquad \langle \mathcal{P}x :: \langle \exists P :: P.x \rangle \rangle \quad \leq \quad \langle \Sigma P :: \langle \mathcal{P}x :: P.x \rangle \rangle$$

It should also be clear that with a little transformation via (23) and (24), that (51) is equivalent to (13) (Page 50).

* *

On random variables

In both definitions of one-way functions we encounter the random variable U_n. The standard definition of a "random variable" is a function from a sample space to the non-negative reals. Unfortunately, the term "random variable" is misleading, since a random variable is a function, and not a variable in the commonly understood sense of the word. Feller (see Chapter IX of [20]) suggests "random function" would be a more appropriate name, but sticks with the standard "random variable"; I don't think "random function" is much of an improvement over "random variable", so for lack of a better term I also stick with the latter.

Let X be a random variable defined over some sample space S. (Note that where functions tend to be denoted by lowercase letters, random variables are usually denoted by capital letters.) It is common to consider the distribution induced over the range of X; abusing notation somewhat, I denote this distribution d_X. Using the standard notation, d_X is usually defined as

$$d_X.y \quad = \quad \Pr[X = y] \qquad ,$$

which, when rendered in Fokkinga's notation, becomes

$$(52) \qquad d_X.y \quad = \quad \langle \mathcal{P}x \in S :: X.x = y \rangle \qquad ,$$

where the type information is included for clarity.

*

Two events, P and Q, are said to be "independent" if the probability of the one is not affected by the other; ie:

$$(53) \qquad \langle \mathcal{P}x :: P.x \ \wedge \ Q.x \rangle \quad = \quad \langle \mathcal{P}x :: P.x \rangle \cdot \langle \mathcal{P}x :: Q.x \rangle$$

Two random variables, X and Y, defined over the same sample space, are said to be independent if for all $x, y \in \mathbb{R}$,

$$(54) \qquad \langle \mathcal{P}r :: X.r = x \ \wedge \ Y.r = y \rangle \quad = \quad \langle \mathcal{P}r :: X.r = x \rangle \cdot \langle \mathcal{P}r :: Y.r = y \rangle \qquad .$$

It is common to define the "joint distribution" of X and Y as

$$(55) \qquad d_{XY}.(x, y) \ = \ \langle \mathcal{P}r :: X.r = x \ \wedge \ Y.r = y \rangle \qquad .$$

We can define independence of random variables in terms of joint distributions: X and Y are independent equivales

$$(56) \qquad d_{XY}.(x, y) \quad = \quad d_X.x \cdot d_Y.y \qquad .$$

Note that these definitions of independence extend to arbitrary collections of mutually independent random variables.

*

We can "lift" the standard arithmetic operators to operate point-wise over random variables. For example, if c is a constant, then $X \cdot c$ denotes a random variable "identical" to X, except each element of its range is multiplied by c; ie:

$$\langle \forall x :: (X \cdot c).x \ = \ X.x \cdot c \rangle$$

In general we have

$$(57) \qquad \langle \forall x :: (X \oplus c).x \ = \ X.x \oplus c \rangle \qquad ,$$

where $\oplus$ denotes any of the standard arithmetic operations such as addition, subtraction, multiplication, division, exponentiation, and so on, provided the operation preserves the type of X. We can replace the constant c in (57) by a random variable Y, if X and Y are defined over the same sample space, in which case we have:

$$(58) \qquad \langle \forall x :: (X \oplus Y).x \ = \ X.x \oplus Y.x \rangle$$

*

If we drop the requirement that random variables map to the positive reals, then any function with an input bound to some sample space constitutes a random variable. On Page 8 of [27] Goldreich explains that:

> Abusing standard terminology, we allow ourselves to use the term random variable also when referring to functions mapping the sample space into the set of binary strings.

It should be clear from the definitions of one-way functions that U_n is a particularly important random variable in the context of cryptography. We've already seen that a distribution over a sample space S is uniform if all elements in S are assigned the same weight; ie:

$$\langle \forall x \in S :: \ d_S.x \ = \ 1 \,/\, \#S \ \rangle$$

For example, if $S \ = \ \{0,1\}^n$, the distribution over S is uniform if

$$d_S.x \quad = \quad 1 \,/\, 2^n \qquad .$$

On Page 9 of [27] Goldreich explains that:

> U_n denotes a random variable uniformly distributed over the set of strings of length n. Namely, $\Pr[U_n = \alpha]$ equals 2^{-n} if $\alpha \in \{0,1\}^n$, and equals zero otherwise.

Consequently, if $S = \{0,1\}^n$ then U_n denotes a permutation over $\{0,1\}^n$, where

$$(59) \qquad \langle \forall x \in S :: \langle \mathcal{P} i :: U_n.i = x \rangle \ = \ 1 \,/\, 2^n \rangle \qquad .$$

*

Having clarified the definition of U_n, it remains to prove (14) (Page 50) and (16) (Page 50), which after translation we can rewrite as

$$\langle \mathcal{P} i :: U_n.i \notin S_n \rangle \quad \leq \quad \langle \Sigma x : x \notin S_n : \langle \mathcal{P} i :: U_n.i \ = \ x \rangle \rangle$$

and:

$$\langle \Sigma x : x \notin S_n : \langle \mathcal{P} i :: U_n.i = x \rangle \cdot \langle \mathcal{P} i : U_n.i = x : P.i \rangle \rangle$$
$$\leq \quad \langle \uparrow x : x \notin S_n : \langle \mathcal{P} i : U_n.i = x : P.i \rangle \rangle$$

With respect to the former, when we calculate we find

$$\begin{array}{ll} & \langle \Sigma x : x \notin S_n : \langle \mathcal{P} i :: U_n.i \ = \ x \rangle \rangle \\ = & \quad \{ \ (27) \ \} \\ & \langle \Sigma x : x \notin S_n : \langle \Sigma i : U_n.i \ = \ x : d.i \rangle \rangle \\ = & \quad \{ \text{ nesting } \} \\ & \langle \Sigma x, i : x \notin S_n \ \wedge \ U_n.i \ = \ x : d.i \rangle \\ = & \quad \{ \text{ equals for equals } \} \\ & \langle \Sigma x, i : U_n.i \notin S_n \ \wedge \ U_n.i \ = \ x : d.i \rangle \\ = & \quad \{ \text{ nesting } \} \\ & \langle \Sigma i : U_n.i \notin S_n : \langle \Sigma x : U_n.i \ = \ x : d.i \rangle \rangle \\ = & \quad \{ \text{ one-point } \} \\ & \langle \Sigma i : U_n.i \notin S_n : d.i \rangle \\ = & \quad \{ \ (27) \ \} \\ & \langle \mathcal{P} i :: U_n.i \notin S_n \rangle \end{array}$$

Hence, the inequality in (14) is weaker than necessary. (Observe also that the above calculation did not require any properties of U_n.)

With respect to the latter, first observe that for any fixed n,

$$\langle \uparrow x : x \notin S_n : \langle \mathcal{P} i : U_n.i = x : P.i \rangle \rangle$$

is a constant, let's denote it c, where

$$\langle \forall x : x \notin S_n : \langle \mathcal{P} i : U_n.i = x : P.i \rangle \ \leq \ c \rangle \qquad ;$$

we can now establish (16) as follows:

$$\begin{array}{ll}
& \langle \Sigma x : x \notin S_n : \langle \mathcal{P} i :: U_n.i = x \rangle \cdot \langle \mathcal{P} i : U_n.i = x : P.i \rangle \rangle \\
\leq & \quad \{ \text{ above } \} \\
& \langle \Sigma x : x \notin S_n : \langle \mathcal{P} i :: U_n.i = x \rangle \cdot c \rangle \\
= & \quad \{ \ (59) \ \} \\
& \langle \Sigma x : x \notin S_n : \frac{1}{2^n} \cdot c \rangle \\
\leq & \quad \{ \text{ there can be at most } 2^n \text{ elements not in } S_n \ \} \\
& 2^n \cdot \frac{1}{2^n} \cdot c \\
= & \quad \{ \text{ algebra } \} \\
& c
\end{array}$$

* *

An aside: expectation and variance

The "expected value" or "expectation" of a random variable,

$$(60) \qquad \text{exp}.X \quad = \quad \langle \Sigma i \in S :: d_S.i \cdot X.i \rangle \qquad ,$$

gives a single value representative of the variable's average value. The "variance" of a random variable,

$$(61) \qquad \text{var}.X \ = \ \text{exp}.X^2 - (\text{exp}.X)^2 \qquad ,$$

indicates how far its values are from its expectation: the smaller the variance, the closer the values are to the expectation. Clearly it's only meaningful to take the expectation or the variance of random variables that map to (a subset of) the reals.

Expectation can be equivalently defined as

$$(62) \qquad \text{exp}.X \quad = \quad \langle \Sigma i \in \text{ran}.X :: d_X.i \cdot i \rangle \qquad ,$$

where $\text{ran}.X$ denotes the range of X. Variance can be defined equivalently as

$$(63) \qquad \text{var}.X \;=\; \text{exp}.(X - \text{exp}.X)^2 \qquad .$$

In the following proof of (62), i ranges over $\text{ran}.X$, and j ranges over S:

$$\begin{array}{ll} & \langle \Sigma i :: d_X.i \cdot i \rangle \\ = & \qquad \{\ (52)\ \} \\ & \langle \Sigma i :: \langle \mathcal{P} j :: X.j = i \rangle \cdot i \rangle \\ = & \qquad \{\ (27)\ \} \\ & \langle \Sigma i :: \langle \Sigma j : X.j = i : d_S.j \rangle \cdot i \rangle \\ = & \qquad \{\ \cdot \ \text{over}\ \Sigma\ \} \\ & \langle \Sigma i :: \langle \Sigma j : X.j = i : d_S.j \cdot i \rangle \rangle \\ = & \qquad \{\ \text{nesting, twice}\ \} \\ & \langle \Sigma j :: \langle \Sigma i : X.j = i : d_S.j \cdot i \rangle \rangle \\ = & \qquad \{\ \text{one-point rule}\ \} \\ & \langle \Sigma j :: d_S.j \cdot X.j \rangle \rangle \end{array}$$

Proof of (63):

$$\begin{array}{ll} & \text{exp}.(X - \text{exp}.X)^2 \\ = & \qquad \{\ \text{algebra}\ \} \\ & \text{exp}.(X^2 - 2 \cdot X \cdot \text{exp}.X + (\text{exp}.X)^2) \\ = & \qquad \{\ \text{properties of exp proved below, ie } (64), (66), \text{ and } (68)\ \} \\ & \text{exp}.X^2 - 2 \cdot (\text{exp}.X)^2 + (\text{exp}.X)^2 \\ = & \qquad \{\ \text{arithmetic}\ \} \\ & \text{exp}.X^2 - (\text{exp}.X)^2 \end{array}$$

*

A few additional theorems about expectation and variance follow. For any constant c we have

$$(64) \qquad \text{exp}.(X \cdot c) \;=\; \text{exp}.X \cdot c$$

$$(65) \qquad \text{var}.(X \cdot c) \;=\; \text{var}.X \cdot c^2$$

$$(66) \qquad \text{exp}.(X + c) \;=\; \text{exp}.X + c$$

$$(67) \qquad \mathsf{var}.(X + c) \;=\; \mathsf{var}.X$$

As a corollary of (66) we have

$$(68) \qquad \mathsf{exp}.(X - c) \;=\; \mathsf{exp}.X - c \qquad\qquad ,$$

and as corollaries of (64) we have

$$(69) \qquad \mathsf{exp}.X \cdot \mathsf{exp}.Y \;=\; \mathsf{exp}.(X \cdot \mathsf{exp}.Y) \qquad\qquad ,$$

and

$$(70) \qquad (\mathsf{exp}.X)^2 \;=\; \mathsf{exp}.(X \cdot \mathsf{exp}.X) \qquad\qquad .$$

Proof of (64) :

$$\begin{array}{ll} & \mathsf{exp}.X \cdot c \\ = & \qquad \{\ (60)\ \} \\ & \langle \Sigma i :: d.i \cdot X.i \rangle \cdot c \\ = & \qquad \{\ \cdot \ \text{ over } \ \Sigma\ \} \\ & \langle \Sigma i :: d.i \cdot X.i \cdot c \rangle \\ = & \qquad \{\ (57)\ \} \\ & \langle \Sigma i :: d.i \cdot (X \cdot c).i \rangle \\ = & \qquad \{\ (60)\ \} \\ & \mathsf{exp}.(X \cdot c) \end{array}$$

Proof of (65) :

$$\begin{array}{ll} & \mathsf{var}.(X \cdot c) \\ = & \qquad \{\ (61)\ \} \\ & \mathsf{exp}.(X \cdot c)^2 - (\mathsf{exp}.(X \cdot c))^2 \\ = & \qquad \{\ \text{arithmetic}\ \} \\ & \mathsf{exp}.(X^2 \cdot c^2) - \mathsf{exp}.(X \cdot c) \cdot \mathsf{exp}.(X \cdot c) \\ = & \qquad \{\ (64)\ \ \text{thrice}\ \} \\ & \mathsf{exp}.X^2 \cdot c^2 - \mathsf{exp}.X \cdot c \cdot \mathsf{exp}.X \cdot c \\ = & \qquad \{\ \text{arithmetic}\ \} \end{array}$$

$$
\begin{array}{ll}
 & (\mathsf{exp}.X^2 - (\mathsf{exp}.X)^2) \cdot c^2 \\
= & \quad \{\ (61)\ \} \\
 & \mathsf{var}.X \cdot c^2
\end{array}
$$

Proof of (66) :

$$
\begin{array}{ll}
 & \mathsf{exp}.(X + c) \\
= & \quad \{\ (60)\ \} \\
 & \langle \Sigma i :: d.i \cdot (X + c).i \rangle \\
= & \quad \{\ (57)\ \} \\
 & \langle \Sigma i :: d.i \cdot (X.i + c) \rangle \\
= & \quad \{\ \cdot \ \text{ over } \ +\ \} \\
 & \langle \Sigma i :: d.i \cdot X.i + d.i \cdot c \rangle \\
= & \quad \{\ \text{distributivity}\ \} \\
 & \langle \Sigma i :: d.i \cdot X.i \rangle + \langle \Sigma i :: d.i \cdot c \rangle \\
= & \quad \{\ (60)\ \text{ and }\ \cdot \text{ over }\ \Sigma\ \} \\
 & \mathsf{exp}.X\ +\ \langle \Sigma i :: d.i \rangle \cdot c \\
= & \quad \{\ \text{distributions sum to }\ 1\ ;\ \text{ arithmetic}\ \} \\
 & \mathsf{exp}.X + c
\end{array}
$$

Proof of (67) :

$$
\begin{array}{ll}
 & \mathsf{var}.(X + c) \\
= & \quad \{\ (63)\ \} \\
 & \mathsf{exp}.(X + c - \mathsf{exp}.(X + c))^2 \\
= & \quad \{\ (69)\ \} \\
 & \mathsf{exp}.(X + c - \mathsf{exp}.X - c)^2 \\
= & \quad \{\ \text{arithmetic}\ \} \\
 & \mathsf{exp}.(X - \mathsf{exp}.X)^2 \\
= & \quad \{\ (63)\ \} \\
 & \mathsf{var}.X
\end{array}
$$

* *

A few thoughts on Fokkinga's notation

So, having now used Fokkinga's notation to develop a number of simple proofs, how did it fare?

Explicitly identifying dummies certainly leads to gains in clarity, but it also makes it easy to translate between probability notation and the summation interpretation, and easy to identify functions that may be considered as random variables, something that is not always clear with the standard $\Pr[\ldots]$ notation.

Looking, for example, at (23) and (24), it's clear that with Fokkinga's notation we gain a few symbols over the standard notation, but in view of the gains in clarity, I think the tradeoff is worth it —indeed, I will continue to use this notation in the next chapter when I come to discuss probabilistic algorithms— .

With respect to calculation, I think it is fair to say that all of the above proofs are easy to verify, since they are all pretty much self-conducting; observe also that each calculation —with the exception of the derivation of (27) using (26), on Page 59— adheres to the heuristic of proceeding from the more complex side of the demonstrandum.

* * *

Chapter 4

A brief excursion into asymptotics

As identified in Chapter 2 , one-way functions are set in an "asymptotic" context, where the one-wayness condition is required only to hold for "large enough values of n" , consequently some of Goldreich's proof steps hold only for large values of n .

In this chapter I first explore —albeit very briefly— what asymptotics is about, based on the first part of Chapter 9 of "Concrete Mathematics" [31] by Graham, Knuth, and Patashnik. I then fill in the gaps in the previously identified proof steps (in the proof of pong) that hold only for large enough values of n . This chapter can also be seen as a precursor to the next on computability and complexity theory, where the latter is the study of the sort of problems we can solve with bounded resources, where the bounds are described as functions of the input size.

* *

Asymptotic ranking

Asymptotics is the study of growth rates. Given two functions f and g , we say that f grows "asymptotically slower" than g , or f is "asymptotically less than" g , if $f.n < g.n$ as n approaches infinity. For example, if $f.n = n^2$, and $g.n = n^3$, then clearly for $n > 1$ we have $f.n < g.n$. However, if we let $f.n = n^3$, and $g.n = 2^n$, then $f.n$ grows faster than $g.n$ for $n \in [2..9]$, but $g.n$ grows (much) faster than $f.n$ for $n \geq 10$.

We write $f \prec g$ (equivalently: $g \succ f$) to denote that f grows asymptotically slower than g , where

$$f \prec g \quad \equiv \quad \frac{f.n}{g.n} = 0 \quad \text{as} \quad n \to \infty \quad .$$

Note that $\to$ should be read as "tends to" , or "approaches" . The $\prec$ relation is transitive,

which allows us to asymptotically rank functions of n ; so for example (taken from [31]) ,

$$1 \prec \log\log n \prec \log n \prec n^{\epsilon} \prec n^{c} \prec n^{\log n} \prec c^{n} \prec n^{n} \prec c^{c^{n}} \ \ldots$$

where ϵ and c are arbitrary constants with $0 < \epsilon < 1 < c$.

* *

Order notation

A function $f.n$ is said to have "order" $O.(g.n)$, pronounced "big-oh of $g.n$" , if

$$\langle \exists c, N :: \langle \forall n : n > N : \mathsf{abs}.(f.n) \ \leq \ c \cdot \mathsf{abs}.(g.n) \rangle \rangle \qquad ,$$

where abs denotes the "absolute value" function. Saying that $f.n$ has order $O.(g.n)$ asserts that f is asymptotically at most g ; in other words, big-oh upper bounds f's rate of growth.

Historical Aside. The symbol originally used for order notation, first introduced by Paul Bachmann in his 1892 book "Analytische Zahlentheorie" , was capital omicron; it was subsequently popularised by Edmund Landau, and is sometimes called the "Landau symbol" in recognition of his work. For further details see [39] .
End of Historical Aside.

For example, if $f.n \ = \ 2 \cdot n^2 + 3 \cdot n - 5$, then we may say that $f.n$ has order $O.n^2$, since

$$\begin{array}{ll} & \mathsf{abs}.(2 \cdot n^2 + 3 \cdot n - 5) \\ \leq & \quad \{ \text{ arithmetic } \} \\ & \mathsf{abs}.(2 \cdot n^2 + 3 \cdot n^2 + 5 \cdot n^2) \\ = & \quad \{ \text{ arithmetic ; absolute value } \} \\ & 10 \cdot \mathsf{abs}.n^2 \end{array}$$

Generalising, any polynomial $p.n$ of degree k has order $O.n^k$. The significance of this is that when working (asymptotically) with polynomials, we can restrict our attention to the term with the highest degree; so, in view of the above asymptotic ranking, for any polynomial function p , we have

$$p.n \prec n^{\log n} \prec c^{n} \prec n^{n} \prec c^{c^{n}} \ \ldots$$

Consequently, polynomial functions grow asymptotically slower than exponential functions.

* *

A few theorems

It remains to prove (10) , (18) , and (19) , viz (rendered using my preferred notation)

$$\left(1 - \frac{n}{a.n}\right)^{a.n} \quad < \quad \frac{1}{2^n} \qquad ,$$

$$\left(1 - \frac{1}{2 \cdot p.n}\right)^{n \cdot p.n} \quad < \quad \frac{n^2 \cdot p.n}{a.n} \qquad ,$$

and

$$1 - \frac{1}{p.n} \quad < \quad \left(1 - \frac{1}{2 \cdot p.n}\right) \cdot \left(1 - \frac{1}{2^n}\right) \qquad ,$$

which hold only for large enough n .

Remark. In the remainder of the chapter I drop the n associated with the polynomials a and p , in the hope that this simplifies rather than confuses.
End of Remark.

*

(10) can be established by appealing to the identity

$$\lim_{m \to \infty} (1 + x \,/\, m)^m \quad = \quad e^x$$

which can be restated as

$$(71) \qquad (1 + x \,/\, m)^m \;\to\; e^x \qquad \text{as} \qquad m \to \infty \qquad .$$

This is sometimes taken as the defining property of the exp function $\exp.x = e^x$, though it can be proved using l'Hôpital's rule. Importantly, $(1 + x \,/\, m)^m$ increases steadily with m , and so approaches e^x from below as m tends to infinity; conversely, $(1 + x \,/\, m)^m$ steadily decreases as m tends to $-\infty$, and so approaches e^x from above; for further details see, for example, G.H. Hardy's "A Course of Pure Mathematics" [34] .

Proof of (10) :

$$
\begin{array}{ll}
 & (1 - n/a)^a \\
= & \quad \{ \text{ fractions } \} \\
 & \left(1 + \frac{(-1)}{a/n}\right)^a \\
= & \quad \{ \text{ exponents } \} \\
 & \left(\left(1 + \frac{(-1)}{a/n}\right)^{a/n}\right)^n \\
\to & \quad \{ (71) \text{ as } a/n \to \infty \} \\
 & \left(e^{-1}\right)^n \\
= & \quad \{ \text{ algebra } \} \\
 & 1/e^n \\
< & \quad \{ e \approx 2.7 \} \\
 & 1/2^n
\end{array}
$$

Just to clarify the use of (71) , a is defined as $2 \cdot n^2 \cdot p \cdot q.(n^2 \cdot p)$, where p and q are positive polynomials, consequently a tends to infinity as n tends to infinity, but more importantly, $a/n = 2 \cdot n \cdot p \cdot q.(n^2 \cdot p)$, and so tends to infinity as n tends to infinity.

*

Goldreich establishes (18) in two steps, namely,

$$(72) \qquad \left(1 - \frac{1}{2 \cdot p}\right)^{n \cdot p} < \frac{1}{2^{n/2}} \quad ,$$

and

$$(73) \qquad \frac{1}{2^{n/2}} < \frac{n^2 \cdot p}{a} \quad .$$

To prove (72) I reasoned as follows:

$$
\begin{array}{ll}
 & \left(1 - \frac{1}{2 \cdot p}\right)^{n \cdot p} \\
= & \quad \{ \text{ exponents, heading for } (71) \} \\
 & \left(\left(1 - \frac{1}{2 \cdot p}\right)^{2 \cdot p/2}\right)^n \\
= & \quad \{ \text{ exponents again } \}
\end{array}
$$

$$
\begin{array}{ll}
 & \Big(\text{sqrt.}\Big(1 - \frac{1}{2 \cdot p}\Big)^{2 \cdot p}\Big)^{n} \\
\to & \qquad \{ \ (71) \ \text{ as } \ 2 \cdot p \to \infty \ \} \\
 & \big(\text{sqrt.} e^{-1}\big)^{n} \\
= & \qquad \{ \ \text{exponents ; fractions} \ \} \\
 & 1 \,/\, e^{n/2} \\
< & \qquad \{ \ e \approx 2.7 \ \} \\
 & 1 \,/\, 2^{n/2}
\end{array}
$$

As identified in the hint, the third step depends on p tending to infinity as n tends to infinity. However, this is a stronger requirement than the only given restraint on p , viz that it is a positive polynomial, since that does not rule out zero degree (ie, constant) polynomials, which would invalidate the above.

The latter inequality is easier to establish:

$$
\begin{array}{ll}
 & \frac{n^2 \cdot p}{a} \\
= & \qquad \{ \ \text{definition of } \ a \ \} \\
 & \frac{n^2 \cdot p}{2 \cdot n^2 \cdot p \cdot q.(n^2 \cdot p)} \\
= & \qquad \{ \ \text{fractions} \ \} \\
 & \frac{1}{2 \cdot q.(n^2 \cdot p)} \\
> & \qquad \{ \ 2 \cdot q.(n^2 \cdot p) \ < \ 2^{n/2} \ \text{ for large enough } \ n \ \} \\
 & 1 \,/\, 2^{n/2}
\end{array}
$$

In view of the above calculation, it can be argued that the value $1 \,/\, 2^{n/2}$ is superfluous, since it's clear that

$$\frac{1}{e^{n/2}} \quad < \quad \frac{n^2 \cdot p}{a} \qquad .$$

However, we can go a step further and avoid (73) altogether by showing that

$$\frac{1}{2 \cdot q} \ + \ \frac{1}{e^{n/2}} \quad < \quad 1 \,/\, q \qquad .$$

Trivially, the demonstrandum simplifies to

$$\frac{1}{e^{n/2}} \quad < \quad \frac{1}{2 \cdot q} \qquad ,$$

which holds for large enough n as required.

*

To prove (19) I make use of the lemma that for positive x and y

$$(74) \qquad x < y \quad \Rightarrow \quad \frac{x-1}{x} < \frac{y-1}{y} \qquad .$$

First, we rewrite the demonstrandum as

$$\frac{(p-1)/p}{(2\cdot p-1)/2\cdot p} \quad < \quad \frac{2^n-1}{2^n}$$

and then calculate:

$$\begin{array}{ll} & \frac{(p-1)/p}{(2\cdot p-1)/2\cdot p} \\ = & \qquad \{ \text{ fractions } \} \\ & \frac{p-1}{p} \cdot \frac{2\cdot p}{2\cdot p-1} \\ = & \qquad \{ \text{ fractions } \} \\ & \frac{(p-1)\cdot 2\cdot p}{p\cdot(2\cdot p-1)} \\ = & \qquad \{ \text{ cancel } p \text{ and expand the numerator } \} \\ & \frac{2\cdot p-2}{2\cdot p-1} \\ < & \qquad \{ (74) \text{ with } x, y := (2\cdot p-1), 2\cdot p \} \\ & \frac{2\cdot p-1}{2\cdot p} \\ < & \qquad \{ (74) \text{ where } 2\cdot p < 2^n \text{ for large enough } n \} \\ & \frac{2^n-1}{2^n} \end{array}$$

Observe that the 2^n in the above comes from (10), but much as with (72), it is again superfluous, since from the above it should be clear that

$$\frac{2\cdot p-1}{2\cdot p} \quad < \quad \frac{e^n-1}{e^n}$$

and hence that (10) could be weakened to

$$(1-n/a)^a \quad < \quad 1/e^n \qquad ,$$

and that (19) could be weakened to:

$$1 - 1/p \quad < \quad \left(1 - \frac{1}{2 \cdot p}\right) \cdot \left(1 - \frac{1}{e^n}\right)$$

* *

Towards calculational asymptotics

Although I use the $\rightarrow$ symbol as a connective in the above proofs of (10) and (72), its status as a relation is unclear; what, for example, are its manipulative rules? Also, if we consider, for example, the proof of (10), ignoring that the final step can be avoided, the inequality does not obviously follow from the preceding steps, given that the appeal to (71) introduces an approximation, which (in principle) could be from either side. In essence, the problem is that there is no general theory of calculational asymptotics.

Eerke Boiten and I address these concerns in [7], where we use a variation of the above proof of (10) as a running example. Towards a theory of calculational asymptotics we introduce and explore properties of a "for large enough" quantifier, which we denote $\langle\diamondsuit\rangle$, where

$$\langle\langle\diamondsuit\rangle x :: P.x\rangle \quad \equiv \quad \langle \exists X :: \langle \forall x : x > X : P.x\rangle\rangle \qquad .$$

We take a modal view of this quantifier, referring to it as "eventually always". We also introduce its dual, denoted $[\diamondsuit]$, and referred to as "always eventually", where

$$\langle [\diamondsuit] x :: P.x\rangle \quad \equiv \quad \neg\langle\langle\diamondsuit\rangle x :: \neg P.x\rangle \qquad ,$$

and hence

$$\langle [\diamondsuit] x :: P.x\rangle \quad \equiv \quad \langle \forall X :: \langle \exists x : x > X : P.x\rangle\rangle \qquad .$$

Observe that "always eventually" can be interpreted as "infinitely often". These quantifiers are used —albeit briefly— in Chapter 7 when I revisit the definitions of one-way functions.

In particular, we introduce and explore properties of a relation that we denote $\sim$, where

$$f \sim g \quad \equiv \quad \frac{f.n}{g.n} = 1 \qquad \text{as} \qquad n \rightarrow \infty \qquad .$$

This allows us to prove (10) as follows:

$$
\begin{array}{ll}
 & (1 - n/a)^{-a} \\
= & \quad \{ \text{ arithmetic } \} \\
 & \left(\left(1 + \frac{(-1)}{a/n} \right)^{a/n} \right)^{n} \\
\sim & \quad \{ \ (71) \ \} \\
 & \left(e^{-1} \right)^{-n} \\
= & \quad \{ \text{ algebra } \} \\
 & e^{n} \\
\succ & \quad \{ \ c^n \prec d^n \quad \text{for} \quad 1 < c < d \ \} \\
 & 2^{n}
\end{array}
$$

From which it follows that

$$2^n \quad < \quad (1 - n/a)^{-a}$$

and hence that

$$(1 - n/a)^{a} \quad < \quad 2^{-n}$$

as required. The proof of (72) can be similarly adapted.

* * *

Chapter 5

Computability and complexity

In order to make precise the concept of computational difficulty, and to assign meaning to the terms "easy" and "hard", we need a sufficiently abstract model of computation that allows us to reason about the difficulty of computing functions, but without having to worry about implementation specific details. In 1936 four "universal" models of computation were proposed:

- Turing machines, proposed by and named after Alan Turing, are based on a mechanistic model of problem solving
- the Lambda calculus, proposed by Alonzo Church, is based on constrained inductive definitions
- partial recursive functions, proposed by Kurt Gödel and Stephen Kleene, are based on an inductive mechanism for the definition of functions
- general recursive functions, also proposed by Gödel and Kleene, along with the French logician Jacques Herbrand, are defined using an equational mechanism

What is remarkable about these models, aside from their near simultaneous presentation, is that they all define the same class of computable functions: whatever can be computed in one model can be computed in all of the others.

Turing machines and recursive functions are respectively the standard tools for studying computational complexity and computability, where —roughly speaking— complexity theory is concerned with quantifying the amount of resources required to carry out a computation, and computability theory asks whether a problem can be solved at all, regardless of the resources required.

* *

Problems and instances

Before delving into computability and complexity theory, I need to clarify some important terminology. A "problem" is a general question with a fixed number of well defined input parameters, along with constraints on what constitutes a valid answer; an "instance" of a problem is a specific set of input parameters for a particular problem. We distinguish between different types of problem, and in particular, between "decision problems" and "function problems".

*

Decision problems are predicates: they require only a **true** or **false** answer. However, decision problems tend to be viewed and formulated in terms of "language" problems.

An "alphabet", Σ, is simply a finite set of symbols. The "Kleene closure" of an alphabet, denoted Σ^*, is the set containing the empty string and all finite strings of symbols that can be constructed from the symbols of the alphabet. In algebraic terms, the Kleene closure of an alphabet forms a monoid, sometimes referred to as the "free monoid over Σ", with elements (called words) generated from the symbols of the alphabet under the operation of concatenation (of words), and the empty string as its identity element. Any subset of Σ^* is called a "language" over Σ^*. For example, the Kleene closure of the "binary" alphabet, ie $\{0,1\}^*$, is the set of all (finite) bit-strings plus the empty string, and the set $\{0,1,00,01,10\}$ is a language over $\{0,1\}^*$.

We can define logics in terms of languages: the set of symbols is an alphabet, and the set of well formed formulae is a subset of the Kleene closure of the alphabet (a subset since the presence of a grammar means that not all strings in the Kleene closure are considered well formed; so, where the Kleene closure consists of all strings that can be generated from the alphabet, the grammar defines the subset considered to be well formed), and hence a language; similarly, the set of theorems also forms a language.

In the primarily Turing machine oriented world of complexity theory, decision problems are usually phrased in terms of language "recognition": testing whether a given string is a member of a particular language, which is equivalent to deciding membership of a set. For simplicity, complexity theorists restrict their attention to languages over the Kleene closure of the binary alphabet, and hence to sets of bit-strings.

For example, we may interpret any bit-string as a natural number, in which case the set of all bit-strings, ie the Kleene closure of the binary alphabet, minus the empty string, forms the language `NATURAL`. The language `PRIMES` is the subset of bit-strings that, when interpreted as natural numbers, are prime; primality testing amounts to deciding membership of the language `PRIMES`.

This view of algorithms as language recognisers tends to complicate assertions about correctness. For example, on Page 19 of their text "Randomized Algorithms", Motwani and Raghavan [43] define the complexity class $\mathbf{P}$ (discussed in due course) as the class of all languages L that have a polynomial-time algorithm A, such that for any input $x \in \Sigma^*$,

- $x \in L \Rightarrow A(x)$ accepts
- $x \notin L \Rightarrow A(x)$ rejects

A common alternative is to write $A(x) = 1$ and $A(x) = 0$ instead of "$A(x)$ accepts" and "$A(x)$ rejects".

As mentioned, language recognition amounts to deciding set membership, but sets can be viewed as predicates and vice-versa, so language recognition amounts to decidability of predicates. Taking this view, algorithms are simply implementations of predicates, so we could instead write

$$\langle \forall x :: Q.x \Rightarrow A.x \quad \wedge \quad \neg Q.x \Rightarrow \neg A.x \rangle \qquad ,$$

where Q is a predicate and A is a polynomial-time (as per the first part of the definition) implementation of Q. Trivially, we can rewrite the above as

$$Q.x \quad \equiv \quad A.x \qquad ,$$

but we know that $Q.x \equiv x \in L$, so equivalently we have

$$x \in L \quad \equiv \quad A.x \qquad .$$

*

In general, a function problem is any kind of problem other than a decision problem. Of particular relevance to this study are "search problems", problems of the form "given x find a y such that..." . For reasons of simplicity, studies of complexity are usually limited to decision problems, but problems in cryptography, and in particular the problem of inverting one-way functions, are usually search problems. Unfortunately this presents something of a conceptual gap in terms of relating cryptography to complexity theory.

We can formalise search problems in terms of relations: Let R be a binary relation on $\{0,1\}^*$; given an $x \in \{0,1\}^*$, the search problem associated with R is to find a $y \in \{0,1\}^*$ such that $x\ R\ y$. So, an algorithm A solves the search problem associated with R if for every $x \in \{0,1\}^*$, if x is related to some element in $\{0,1\}^*$ then

$$x\ R\ A.x \qquad ,$$

otherwise $A.x = \perp$, where $\perp$ denotes that no solution exists. For example, given a function f, if R relates elements in f's range to their inverses, then solving the search problem associated with R is equivalent to inverting f.

* *

Computability

Questions about computability arose out of questions about proof and decidability. In 1920 Hilbert proposed a research project in "meta-mathematics", otherwise known as "Hilbert's program", to show that all of mathematics follows from a provably consistent, finite collection of postulates. In 1928 Hilbert proposed the "Entscheidungsproblem", German for "decision problem": does there exist an algorithm that, when given a description of a formal system, can decide validity of any formula of that system? Before the Entscheidungsproblem could be resolved, the notion of an "algorithm" had to be formally defined.

Remark. The term "algorithm" is derived from al-Khwarizimi's "Liber algorism", "the book of al-Khwarizimi", written circa 825 AD.
End of Remark.

Briefly, an algorithm is a procedure, or finite list of well defined instructions, for solving any instance of a problem. More specifically, models of computation, such as recursive functions or Turing machines, both of which are discussed below, are used to formalise our notion of an algorithm, so for example, a Turing machine's list of actions constitutes an algorithm.

*

Recursive functions

The class of "primitive recursive functions" contains the functions that can be constructed from a collection of basic functions using substitution (often referred to as "composition") and "primitive" recursion. The basic functions comprise the zero function, the successor function, and a family of generalised identity (or "projection") functions; formally:

- $\mathrm{zero}.x \;=\; 0$
- $\mathrm{succ}.x \;=\; x+1$
- $\mathrm{id}^n_i.(x_1\,,\;\ldots\;,\;x_n) \;=\; x_i$

Given two functions known to be primitive recursive, substitution allows us to produce a new primitive recursive function by applying one of those functions to the result of the other: if f and g are known to be primitive recursive, we can construct a new function h as $f.(g.x)$. In general, given a function f that takes m arguments, and given m functions g_1 to g_m, each of which takes n arguments, we can construct an n argument function h, where

$$h.(x_1\,,\;\ldots\;,\;x_n) \quad=\quad f.(g_1.(x_1\,,\;\ldots\;,\;x_n)\,,\;\ldots\;,\;g_m.(x_1\,,\;\ldots\;,\;x_n)) \qquad .$$

Primitive recursion, much like substitution, allows us to define a new function, h, in terms of two given primitive recursive functions, f and g, where h takes n arguments if f takes

$n-1$ arguments and g takes $n+1$ arguments. Primitive recursion can be expressed formally in terms of the following schema:

$$\begin{cases} h.(0\,,\ x_2,\ \ldots\ ,\ x_n) & = & f.(x_2\,,\ \ldots\ ,\ x_n) \\ h.(x_1+1\,,\ x_2,\ \ldots\ ,\ x_n) & = & g.(x_1\,,\ h.(x_1\,,x_2\,,\ \ldots\ ,x_n),\ x_2\,,\ \ldots\ ,\ x_n) \end{cases}$$

Examples of primitive recursive functions include addition, subtraction, exponentiation, and the factorial function.

We can extend the class of primitive recursive functions to the class of "partial recursive functions" by admitting an additional production rule: a "minimisation" or "unbounded search" operator, sometimes referred to as "μ-recursion" . The minimisation operator searches for the least natural number with a given property; more specifically, it introduces for each given total function $f.(y\,,\ x_1\,,\ x_2,\ \ldots\ ,\ x_n)$ a new function $h.(x_1\,,\ x_2\,,\ \ldots\ ,\ x_n)$ that returns the least y such that

$$(75) \qquad f.(y\,,\ x_1\,,\ \ldots\ ,\ x_n)\ =\ 0 \qquad\qquad .$$

All primitive recursive functions are total by definition, since the three primitive functions are total, and applications of substitution and recursion to total functions preserve totality. However, minimisation need not preserve totality, since in general there may not be a value of y that satisfies (75) .

In addition to the primitive recursive functions, the class of partial recursive functions contains total functions that are not primitive recursive, and functions that cannot be primitive recursive because they are not total. In practice it is hard to find partial recursive functions that are total, but are not primitive recursive, Ackermann's function [1] being a well known example. The subset of partial recursive functions that are total are referred to as "general recursive functions" , or simply "recursive functions" .

In terms of programming languages, substitution corresponds to support for subroutines (ie, the ability to replace an argument with the result of another computation) , while minimisation corresponds to being able to write loops with a guard. Primitive recursion is not quite as powerful as minimisation: it corresponds to being able to write loops, but only where the number of passes through the loop is predetermined. In other words, minimisation corresponds to being able to write a `while` loop, and primitive recursion corresponds to being able to write a `for` loop.

*

Turing machines

The notion of a "Turing machine" is based on human problem solving using pencil and (an unlimited supply of) paper. There are various equivalent ways of defining Turing machines, but the standard, informal interpretation, is a machine that comprises an infinite linear tape divided

into "cells", each of which stores a single symbol, and a finite state control unit that reads and writes symbols to and from the tape.

The symbols on the Turing machine's tape are drawn from some finite alphabet, Σ, which includes a "blank symbol" to denote empty cells, and a "halting symbol" to denote successful termination. The tape initially contains a finite input string derived from a subset of Σ that does not include the blank symbol or the halting symbol; all other cells are empty. The machine starts with the control unit in some specified initial state, and the read/write head positioned over the leftmost symbol of the input string.

Turing machines "compute" by reading a symbol from the tape, and performing an "action" that consists of printing a symbol from Σ onto the scanned cell, moving the read/write head left or right by one cell, and assuming a new state, where actions are defined by the current state and the symbol read from the tape. The machine halts when it outputs the halting symbol and enters a special halting state; the output is defined as the contents of the cells to the left of the halting symbol.

Remark. Turing machines are not real machines, so defining them in terms of physical objects such as tapes and heads is perhaps unfortunate. It is possible to avoid this physical description by giving a purely formal definition in terms of languages, but that's not necessary here.
End of Remark.

Each Turing machine defines a single mathematical function on the data provided on its input tape; ie, a Turing machine is a computer with a fixed program. However, we can describe any Turing machine by encoding its list of actions as a string. Briefly, a "universal Turing machine" receives as part of its input a description of a Turing machine, and "simulates" that machine. A programming language is said to be "Turing complete" if it is equivalent in expressiveness to a universal Turing machine.

*

The Church-Turing thesis

Church's thesis asserts that the class of computable functions can be identified with the class of recursive functions. Turing's thesis asserts that the class of computable functions can be identified with the class of functions computable by Turing machines. Hence, Church's thesis and Turing's thesis assert that the class of computable functions, the class of recursive functions, and the class of functions computable by Turing machines are isomorphic.

The Church-Turing thesis, proposed in 1943 by Stephen C. Kleene, generalises Church's thesis and Turing's thesis by asserting that the class of computable functions can be identified with any "reasonable and general" model of computation, and hence that all such models can compute the same class of functions. The beauty of the Church-Turing thesis is that it allows us to reason about computability in whichever model is most appropriate.

Consider, for example, the "RAM" model of computation. A RAM ("random access machine") consists of a —usually infinite— number of memory cells, each capable of holding an integer, a finite number of registers, one designated as a program counter, and a program constructed from some finite set of instructions that usually includes operations for memory and register transfers, branching, and simple arithmetic. The input is usually defined as the contents of the first n memory cells, where n is placed in a special input register. The program counter points to the next instruction to be executed, and is incremented after each instruction is executed; the program halts when the program counter exceeds the program's length.

The RAM model is clearly a more "realistic" model of computation than, say, the Turing machine model, in the sense that it corresponds closely to the sort of machines we use in practice. However, for developing a theory about computation, the RAM model is a more cumbersome formulation. In support of the Church-Turing thesis, it is, for example, fairly easy to show that any RAM machine can be simulated by a Turing machine and vice-versa.

*

Uncomputable functions

Church and Turing independently showed that there exist problems that are not computable, by establishing that it is impossible to decide algorithmically whether statements in arithmetic are true or false, and hence that a general solution to the Entscheidungsproblem is impossible. More specifically, Church proved that there is no computable function that can decide whether an arbitrary pair of lambda calculus expressions are equivalent, while Turing reduced the "halting problem" (given a program decide whether it halts) to the Entscheidungsproblem. (The notion of "reduction" is discussed in more depth in the next chapter.)

Rice's theorem generalises these negative results by asserting that no algorithm exists that can determine any "non-trivial" property of the function computed by a given program, where a property of a partial function is called "trivial" if it holds for all partial recursive functions or for none.

It is intriguing to observe that the class of computable functions is denumerable, since any recursively enumerable set (a set is called "recursively enumerable", or "semidecidable", if it denotes the range of some recursive function; equivalently: a set is recursively enumerable if there exists an algorithm that enumerates its elements) is denumerable, but the class of all functions is non-denumerable (even the subset of boolean-valued functions is non-denumerable); consequently the class of uncomputable functions is vastly larger than the class of computable functions. (This can be shown formally using a Cantor-style diagonalisation argument.)

*

Computability and one-way functions

Given that there exist both computable and uncomputable functions, why in the definitions of one-way functions, do we not associate "easy" with being computable, and "hard" with being uncomputable?

First, computable means computable in some finite amount of time, but from a practical point of view, since we humans are finite beings, and since we only have limited computing power at our disposal, finite may as well be infinite if it means computable in, say, a thousand years. So although computability is interesting from a theoretical point of view, it's often less interesting from a practical point of view, other than as a step toward determining whether a problem can be solved in any reasonable amount of time.

Second, and more fundamentally, if we associate "easy" with computable, it is not possible to associate "hard" with uncomputable: For simplicity, let's restrict our attention to a length preserving one-way function, f. Given an n-bit element of f's range, since f is computable we can try it on every n-bit string, at least one of which must be an inverse. Clearly there are only a finite number of n-bit strings, so this approach is guaranteed to terminate, and hence is computable.

* *

Complexity

Having defined various models of computation, we can define within those models cost measures on the complexity of the computations involved. In practice we tend to restrict our attention to two specific measures: time complexity and space complexity. In the Turing machine model, time complexity refers to the number of steps the machine takes from start to finish, where each step is considered to take the same amount of time, while space complexity is the number of tape cells required; for obvious reasons, only time complexity is considered here.

Historical Aside. Juris Hartmanis and Richard Stearns are credited with introducing the field of computational complexity and giving it its name in their 1965 paper "On the computational complexity of algorithms" [35] .
End of Historical Aside.

We can describe the performance of an algorithm in terms of time-complexity by introducing a function mapping each input to the number of steps the algorithm takes on that input; I use the notation $T_A.x$ to denote the number of steps algorithm A takes on input x. For inputs of a fixed size, the "easiest" input gives a lower bound on performance, ie

$$\langle \downarrow x \in \{0,1\}^n :: T_A.x \rangle \quad ,$$

and the "hardest" input gives an upper bound on performance:

$$\langle \uparrow x \in \{0,1\}^n :: T_A.x \rangle$$

When analysing the complexity of an algorithm, the easiest case is usually not very interesting. Intuitively, the average running time, which amounts to taking the expectation over T_A, would seem a natural candidate, but deriving such an analysis is often difficult and perhaps a little artificial, since we are forced to assume a distribution over the input domain. Instead, studies of complexity tend to focus on looking for lower bounds on worst case behaviour.

Since we are not given T_A, an important goal in algorithm design and analysis is to find functions that describe or bound T_A as a function of the size of the input; ie, we look for functions f where

$$\langle \forall x \in \{0,1\}^* :: T_A.x \ \leq \ f.\#x \rangle \qquad .$$

In essence, complexity theory asks which problems we can solve using algorithms where T_A is bounded by some given function; so for example, we may ask which problems are solvable by algorithms where

$$\langle \forall x \in \{0,1\}^* :: T_A.x \ \leq \ (\#x)^2 \rangle \qquad ,$$

ie, which problems we can solve where T_A is at most quadratic in the size of the input.

By bounding resources we can partition the space of computable functions into "complexity classes": collections of problems that can be solved with limited computational resources. More specifically, a complexity class comprises a specification of the type of problems within the class, the computational model, a bound on the available resources, and a notion of correctness.

The "time hierarchy" theorems, proved in the mid 1960s by Hartmanis and Stearns [35], ensure the existence of problems that cannot be solved in a given number of steps. These theorems are the fundamental tool for constructing a hierarchy of complexity classes (as opposed to deciding where problems fit into the hierarchy), since they ensure that for any time-bounded complexity class, there exists a larger time-bounded complexity class. For example, the theorems ensure that the class of problems that can be solved in n^3 steps is larger than the class of problems that can be solved in n^2 steps, for inputs of length n.

Remark. The time hierarchy theorems use the notion of a "time-constructible function", where a function $f : \mathbb{N} \to \mathbb{N}$ is time-constructible if there exists a Turing machine that for all natural n halts on inputs of length n after precisely $f.n$ steps.
End of Remark.

In principle we could define classes for every mathematical function, but there are objections to such an approach. First, as established earlier, the class of computable functions is countable, but the class of all functions is uncountable, so it would make little sense to define an uncountable hierarchy of classes, the vast majority of which would be empty. Second, and more importantly, we want to set up our classes so they are invariant under a change of computational model: the Church-Turing thesis allows us to choose the most appropriate model to reason about computability, we'd like to be able to do the same when reasoning about complexity.

The "extended Church-Turing thesis", or "Cobham-Edmonds thesis", a strengthening of the Church-Turing thesis, asserts that time complexity between models of computation is polynomially related, meaning a problem has time complexity t in some "reasonable and general" model of computation equivales it has time complexity polynomial in t in the Turing machine model. (In support of the extended Church-Turing thesis, it is, for example, fairly straightforward to show that any RAM machine can be simulated by a Turing machine with at most a quadratic time penalty.) Consequently, in order to remain invariant under a change of computational model, the classes we define should be invariant under polynomial transformations.

*

Deterministic polynomial-time

A Turing machine is said to be "deterministic" if its actions are determined uniquely by the current state and the symbol read from the tape; ie, if for each state/symbol pair there is only one possible action that can be performed.

The complexity class "deterministic polynomial-time", denoted $\mathbf{P}$, is the class of predicates decidable by deterministic algorithms with worst case running time polynomial in the size of the input; such algorithms are are called "polynomial-time" algorithms. Specifically, a predicate P is a member of $\mathbf{P}$ if there exists a deterministic Turing machine A, and a polynomial p such that

$$\langle \forall x \in \{0,1\}^* :: A.x \ \equiv \ P.x \quad \wedge \quad T_A.x \ \leq \ p.\#x \rangle \qquad .$$

For example, for any integer n, it is possible to decide in polynomial-time whether or not n is prime, a result that was only established as recently as 2002 [2] .

The complexity class "function $\mathbf{P}$", denoted $\mathbf{FP}$, generalises the definition of $\mathbf{P}$ from predicates decidable in polynomial-time, to the class of all functions computable in polynomially many steps (in the size of the input) .

Though a little misleading, the term "polynomial-time" is commonly overloaded to mean any problem that can be solved by a deterministic Turing machine where the number of steps the machine takes is bounded by some polynomial, and hence is usually associated with $\mathbf{FP}$ rather than with $\mathbf{P}$.

*

Nondeterministic polynomial-time

Experience suggests that for many predicates, finding a proof of $P.x$ is harder than verifying a given proof. For example, consider the problem of deciding whether a graph is k-colourable : given a k-colouring we can easily check its validity, but finding or refuting the existence of a k-colouring is, it would seem, computationally hard.

The class "nondeterministic polynomial-time" , denoted $\mathbf{NP}$, is defined as the class of predicates, P , where it is possible to verify a proof of $P.x$ in polynomial-time. $\mathbf{NP}$ is usually defined in terms of nondeterministic Turing machines (hence the name of the class) as the class of predicates where we can use a nondeterministic Turing machine to compute a proof of $P.x$ that is verifiable in polynomial-time using a deterministic Turing machine.

A Turing machine is said to be "nondeterministic" if for each state/symbol pair there is at least one action possible, from which the control unit makes an "arbitrary" choice that will lead to a correct output; we may think of such machines either as being capable of making extremely lucky "guesses" , or as trying all possible paths in parallel and outputting any correct answer, or $\perp$ if no answer exists.

Historical Aside. Michael Rabin and Dana Scott introduced the concept of nondeterministic machines in their 1959 paper "Finite Automata and Their Decision Problem" [44] .
End of Historical Aside.

The class "function $\mathbf{NP}$" , denoted $\mathbf{FNP}$, generalises $\mathbf{NP}$ from decision problems to function problems. Specifically, $\mathbf{FNP}$ is defined as the class of function problems of the form: given an input a and a polynomial-time computable predicate $P.(a, b)$, if there exists a value of b satisfying $P.(a, b)$, then (using nondeterminism) output any such b , otherwise output $\perp$.

Deciding whether $\mathbf{P} = \mathbf{NP}$ is considered to be a fundamental question in mathematics and computing science, with potentially profound consequences if the two classes are equivalent [11] , though it is conjectured, and generally believed, that $\mathbf{P}$ is a proper subset of $\mathbf{NP}$ [25] . Observe that if $\mathbf{P} = \mathbf{NP}$, then finding proofs is no harder than verifying them, which, based on our experience of theorem proving, would be a surprising result indeed. It can be shown that

$$\mathbf{P} = \mathbf{NP} \quad \equiv \quad \mathbf{FP} = \mathbf{FNP} \qquad .$$

*

Exponential-time

An algorithm is said to run in "exponential-time" if there exists a constant c greater than 1 , and a polynomial p such that

$$\langle \forall x \in \{0,1\}^* :: T_A.x \ \leq \ c^{p.\#x} \rangle \qquad .$$

For example, finding the best strategy in a game of Chess on an $n \times n$ board is known to be exponential in n [22] .

The subset of exponential-time algorithms where p is more than constant but less than linear are called "superpolynomial" algorithms. The fastest known algorithms for integer factorisation have superpolynomial complexity.

As described in Chapter 4 , superpolynomial and exponential functions grow asymptotically faster than polynomial functions, so for large enough inputs superpolynomial and exponential problems are considered harder to solve than problems that can be solved in polynomial-time.

*

Probabilistic complexity

Experience suggests that many problems can be solved more efficiently using randomisation, primality testing being the canonical example. We can further refine our notion of a Turing machine in order to reason about randomisation: a Turing machine is said to be "probabilistic" if for each state/symbol pair there is at least one action possible, from which the control unit chooses an action according to some probability distribution. Studies of probabilistic complexity restrict the number of possible actions to two, and usually view probabilistic Turing machines as having an extra tape containing bits that constitute random choices, which are referred to rather unfortunately as "coin tosses" .

We can take two views of probabilistic Turing machines: in the "online model" the random choices are computed "online" by the machine at each step, and so are viewed as being internal to the machine; in the "offline model" the choices are a priori computed "offline" and provided to the machine as part of its input. The two views are equivalent with respect to time complexity.

The introduction of a probabilistic element allows us to weaken our notion of correctness. Specifically, if we adopt the online view, then depending on the internal random choices, $A.x$ may sometimes return a correct answer, and may sometimes return an incorrect answer; if we adopt the offline view, $A.(x, r)$ returns a correct answer dependent on the externally provided random choices, r .

If we allow the machine to make mistakes, we must quantify the probability that it will return a correct answer. Restricting our attention —for now— to predicates, a probabilistic implementation of a predicate is said to have

- "zero-sided error" if it always returns the correct answer or "unknown" , and so never returns an incorrect answer

- "one-sided error" if it may err in only one direction: it may incorrectly output **true** when the answer should be **false** , but will never incorrectly output **false** if the answer should indeed be **false** , or vice-versa

- "two-sided error" if it may err in either direction

An important concept when dealing with probabilistic algorithms is that of "amplification", the notion that we can repeat a probabilistic program a number of times with independent random choices to reduce the probability of an incorrect answer, or rather, to "amplify" the probability of a correct answer.

A zero-sided algorithm may return "unknown" for some inputs, which isn't a very satisfactory answer to a $\mathbf{true}$ or $\mathbf{false}$ question, but we can repeat the algorithm until it outputs something other than unknown, at which point the answer must (by definition) be correct.

If a one-sided algorithm always correctly outputs $\mathbf{false}$, then we only need repeat the algorithm when it outputs $\mathbf{true}$. If after repeating some appropriate number of times it never outputs $\mathbf{false}$, then with high probability the answer is $\mathbf{true}$, where the actual probability depends on the number of times the algorithm was repeated, and the probability that it correctly outputs $\mathbf{true}$.

Since a two-sided algorithm may err in either direction, in general we must repeat it a number of times whatever the answer and "rule by majority verdict".

*

Probabilistic polynomial-time

A probabilistic Turing machine A is said to be an "online probabilistic polynomial-time Turing machine" if there exists a polynomial p such that regardless of the (internal) random choices

$$\langle \forall x \in \{0,1\}^* :: \ T_A.x \ \leq \ p.\#x \ \rangle \qquad ,$$

or an "offline probabilistic polynomial-time Turing machine" if there exists a polynomial p such that

$$\langle \forall x, r :: \ T_A.(x,r) \ \leq \ p.\#x \ \rangle \qquad ,$$

where $x \in \{0,1\}^*$ and $r \in \{0,1\}^{p.\#x}$.

Recall that the quantification in the one-wayness condition in the definitions of one-way functions is over all (online) probabilistic polynomial-time algorithms. However, the class of probabilistic polynomial-time algorithms does not constitute a complexity class per se, as we have not taken into account correctness.

The complexity class "bounded-error probabilistic polynomial-time", or $\mathbf{BPP}$, is defined as the class of predicates, P, for which there exists a probabilistic polynomial-time Turing machine, A, with two-sided error where

$$\langle \forall x :: \langle \mathcal{P} r :: A.x.r \ \equiv \ P.x \rangle \ \geq \ 2/3 \rangle \qquad .$$

This definition is usually followed by the claim that the only thing important about the value $2/3$ is that it is a constant greater than a half. Clearly if we were to replace $\geq 2/3$ by $= 1/2$, the algorithm would be equivalent to simply guessing, and we would not be able to use amplification to improve the situation.

If we instead require that the probability of success is bounded away from a half, but not necessarily by a constant, we admit the possibility of, for example, a probabilistic polynomial-time Turing machine that outputs **true** with probability $1/2 + 1/2^{\#x}$ if the correct answer should be **true**, and probability $1/2 - 1/2^{\#x}$ if the correct answer should be **false**. It is possible to show that since the probability is so close to a half, we would have to repeat the algorithm an exponential number of times in order to rule by majority, but of course, the polynomial bound on running time does not allow us to repeat the algorithm an exponential number of times.

By contrast, if we lower bound the probability of success by any constant greater than a half, we can reduce the error bound to a negligible quantity by repeating the algorithm polynomially many times. That said, the need for a constant is too strong: provided we bound away from a half by any function noticeable in the size of the input, it is possible to repeat the algorithm polynomially many times to reduce the error bound to a negligible value; more specifically, so the error probability vanishes exponentially.

Consequently, $\mathbf{BPP}$ is the class of predicates decidable by probabilistic polynomial-time Turing machines that may err in either direction, provided the error probability is negligible. Bounding the probability of success away from a half by any noticeable function ensures that $\mathbf{BPP}$ is the most liberal notion of probabilistic polynomial-time that remains useful.

It is possible to generalise the definition of $\mathbf{BPP}$ to search problems, though it is not common; indeed, the only place I have seen this discussed, albeit briefly, is in Goldreich's textbook "Computational Complexity: A Conceptual Perspective" [30], which is due to be published in April 2008. Goldreich defines the search variant of $\mathbf{BPP}$ in terms of probabilistic polynomial-time algorithms that output a correct answer with only a negligible probability of error.

In terms of relations, a probabilistic polynomial-time algorithm A solves the search problem associated with a relation R, if for every $x \in \{0,1\}^*$, if x is related to some element in $\{0,1\}^*$ then

$$\langle \mathcal{P}r :: x \; R \; A.(x,r) \rangle \; > \; 1 - \mu.\#x \qquad ,$$

otherwise

$$\langle \mathcal{P}r :: A.(x,r) \; = \perp \rangle \; > \; 1 - \mu.\#x \qquad ,$$

where μ is a negligible function. This definition is two-sided in the sense that any such algorithm can output an incorrect answer or $\perp$ when a correct answer exists, or can output a bit-string when the answer should be $\perp$.

*

An aside: intermediate notions of probabilistic polynomial-time

As you would expect, it is possible to define complexity classes for probabilistic polynomial-time machines with one-sided and zero-sided error. The complexity class $\mathbf{RP}$, or "randomised polynomial-time", is the class of predicates, P, for which there exists a probabilistic polynomial-time Turing machine, A, with one-sided error where

$$\langle \forall x : P.x : \langle \mathcal{P} r :: A.x.r \rangle > 1/2 \rangle \quad \wedge \quad \langle \forall x, r : \neg P.x : \neg A.x.r \rangle \quad ,$$

or, equivalently, by virtue of trading and the contrapositive rule:

$$\langle \forall x : P.x : \langle \mathcal{P} r :: A.x.r \rangle > 1/2 \rangle \quad \wedge \quad \langle \forall x, r : A.x.r : P.x \rangle$$

Hence, if an $\mathbf{RP}$ algorithm outputs **true** even once, then the answer must be **true**; conversely, if it consistently outputs **false** then we can be confident that the answer is indeed **false**. $\mathbf{RP}$ has an obvious complement, called $\mathbf{coRP}$: the class of predicates for which there exists a probabilistic polynomial-time Turing machine with one-sided error where

$$\langle \forall x : \neg P.x : \langle \mathcal{P} r :: \neg A.x.r \rangle > 1/2 \rangle \quad \wedge \quad \langle \forall x, r : P.x : A.x.r \rangle \quad ,$$

or, equivalently, again by virtue of trading and the contrapositive:

$$\langle \forall x : \langle \mathcal{P} r :: A.x.r \rangle < 1/2 : P.x \rangle \quad \wedge \quad \langle \forall x, r : P.x : A.x.r \rangle$$

The class $\mathbf{ZPP}$, "zero-sided probabilistic polynomial-time", may be defined as the intersection of $\mathbf{RP}$ and $\mathbf{coRP}$, or, equivalently, as the class of predicates decidable in "expected polynomial-time" by a probabilistic Turing machine with zero-sided error.

Observe that a probabilistic Turing machine that runs in expected polynomial-time is not the same as a probabilistic polynomial-time Turing machine: the latter always terminates after a polynomial number steps for any input, whereas the former need only terminate after a polynomial number of steps on average, meaning the number of steps for some inputs may not be polynomially bounded by the size of the input. It is possible to show that although $\mathbf{ZPP}$ algorithms need not halt in polynomial-time for all inputs, as a consequence of the zero-sided correctness condition, $\mathbf{ZPP}$ is a subset of $\mathbf{BPP}$, and so is indeed an intermediate class.

*

An aside: Las Vegas and Monte Carlo algorithms

So-called "Monte Carlo" algorithms are probabilistic algorithms that can have either one-sided or two-sided error; "Las Vegas" algorithms are probabilistic algorithms that have zero-sided error.

Clearly any Las Vegas algorithm can be converted to a Monte Carlo algorithm by simply halting after an appropriate number of steps and outputting a solution, but the converse is not known to be true.

The term "Monte Carlo" is attributed to Stanislaw Ulam who, during a period of recovery from a bout of encephalitis, played a lot of solitaire and wondered if a more practical method existed than resorting to pure combinatorial analysis to estimate the odds of particular card combinations. The name was in honour of Ulam's uncle, a gambler, who was "always sneaking off to the roulette wheels at Monte Carlo" [37, 47] . The term "Las Vegas" was introduced by László Babai in his 1979 technical report, "Monte-Carlo algorithms in graph isomorphism testing" [3] .

With respect to the randomised complexity classes, $\mathbf{BPP}$ can be defined as the class of predicates decidable in worst-case polynomial-time by a two-sided Monte Carlo algorithm, $\mathbf{RP}$ and $\mathbf{coRP}$ as the classes of predicates decidable in worst-case polynomial-time by a one-sided Monte Carlo algorithm, and $\mathbf{ZPP}$ as the class of predicates decidable by Las Vegas algorithms that run in expected polynomial-time.

*

More on probabilistic search

Amplification is not quite as straight forward for search problems, since in general majority rule no longer makes sense. However, if we can efficiently check candidate solutions, as we can in the case of one-way functions, we can use amplification as in the zero-sided case: we repeat the algorithm a number of times and check each answer returned, stopping as soon as we find a correct answer; if no correct answer is found we output "unknown" .

Recall that in the proof of pong, in (9) (Page 44) Goldreich claimed that for every $x \in S_n$

$$\Pr[A'(f(x)) \notin f^{-1}(f(x))] \;<\; \left(1 - \frac{n}{a(n)}\right)^{a(n)} \qquad ,$$

where, as Goldreich points out, A' merely repeats I at most $a(n)$ times.

Since for any $x \in S_n$, I succeeds in any single repetition with probability greater than $n \,/\, a(n)$, it follows —by virtue of (49) (Page 67)— that I fails with probability less than $1 - n \,/\, a(n)$. Next, observe that the $a(n)$ repetitions of I constitute $a(n)$ independent events, since each repetition of I does not depend on any previous repetitions; therefore, generalising (53) (Page 69) , the probability that A' fails is indeed less than

$$\left(1 - \frac{n}{a(n)}\right)^{a(n)} \qquad .$$

The above, and the theorems and observations from the previous chapter, are enough to justify the claim that provided a probabilistic algorithm has at least a negligible probability of success,

then we can ensure the probability of failure vanishes exponentially by repeating the algorithm polynomially many times: We know that if an algorithm succeeds with probability at least $1/a$, where a is some positive polynomial in n, then it fails with probability at most $1 - 1/a$. So, if we repeat the algorithm $n \cdot a$ times (which, of course, is polynomial in n), provided a is not a constant polynomial, it follows from the above that its probability of failure is at most

$$(76) \qquad (1 - 1/a)^{n \cdot a} \qquad .$$

But it should be clear from the previous chapter, that (76) tends to $1/e^n$ as a tends to infinity, so the probability of failure vanishes exponentially in n.

*

Complexity and one-way functions

Complexity theorists tend to associate "easy", or "feasible" with problems that can be solved in polynomial-time, and "hard" or "infeasible" with problems that can't. Observe that under the assumption that $\mathbf{P}$ is a proper subset of $\mathbf{NP}$, problems in $\mathbf{NP} - \mathbf{P}$ and $\mathbf{FNP} - \mathbf{FP}$ are considered to be hard (or infeasible), since although it is easy to verify candidate solutions, it is believed to be hard to compute those solutions.

Assuming randomisation allows us to solve a wider class of problems in polynomial-time (it is not known whether $\mathbf{P} = \mathbf{BPP}$), and provided we accept a negligible margin of error, we associate "easy" with $\mathbf{BPP}$ and its search variant. In the definitions of one-way functions "easy" is associated with the apparently stricter notion of being computable in deterministic polynomial-time; this makes sense because, without going into details, for the purposes of cryptography "easy to compute" should mean "always easy to compute a correct answer". However, the definition of "hard" requires a little more explanation, since it means something subtly different to cryptographers than it does to complexity theorists.

Since for (hopefully) obvious reasons, strong one-way functions are more important to cryptography than weak one-way functions, and since any weak one-way function can be amplified to produce a strong one-way function, the following discussion focuses on the definition of "hard" associated with strong one-way functions.

First, observe that any function computable in polynomial-time can be inverted in exponential-time: consider for simplicity a length preserving function, f; given an n-bit element of f's range, we can find a valid inverse in at most 2^n steps, each of which is computable in polynomial-time, by simply applying f to every bit-string of length n.

However, we can make a stronger assertion, viz that computing the inverse of any polynomial-time computable function is in $\mathbf{FNP}$: let a denote an element in f's range, and let b denote a candidate inverse of a, then P corresponds to the predicate

$$f.b = a \qquad ;$$

clearly P is decidable in polynomial-time, since by definition f has a polynomial-time implementation.

It follows then, that a necessary condition for the existence of one-way functions as defined, is that $\mathbf{P} \neq \mathbf{NP}$, and hence that $\mathbf{FP} \neq \mathbf{FNP}$, since if $\mathbf{P} = \mathbf{NP}$ then functions computable in polynomial-time must also be invertible in polynomial-time.

However, that $\mathbf{P} \neq \mathbf{NP}$ is not a sufficient condition for cryptography: the requirement for strong one-way functions is that they are hard to invert on average, but complexity classes are defined in terms of worst case behaviour, where algorithms exist that are efficient on average for many problems in $\mathbf{NP}$ and $\mathbf{FNP}$.

A further necessary condition is that $\mathbf{NP}$ is not contained in $\mathbf{BPP}$, since it is not known whether $\mathbf{NP}$ and $\mathbf{BPP}$ are disjoint, or whether one is a subset of the other, and a proof that $\mathbf{P} \neq \mathbf{NP}$ need not assert anything about the relationship between $\mathbf{BPP}$ and $\mathbf{NP}$.

Having associated "easy" with those problems computable in probabilistic polynomial-time with a negligible probability of error, and having associated "hard" with those that cannot, "hard on average" must mean that for some instances it may be possible to compute a correct solution in (probabilistic) polynomial-time, but these instances should be sufficiently rare that on average polynomially bounded computations lead to an incorrect answer or "unknown".

Negligibility captures the notion of "sufficiently rare". In the case of $\mathbf{BPP}$, for example, a negligible probability of error means that incorrect outputs are sufficiently rare; in the case of strong one-way functions, negligibility refers instead to correct outputs (ie, valid inverses) computable in probabilistic polynomial-time being sufficiently rare, and hence that strong one-way functions are hard to invert on average.

I explained in Chapter 2 that negligible functions remain negligible when multiplied by any polynomial, the significance of this is that for any probabilistic polynomial-time algorithm A, if the probability that A produces a correct answer is negligible, even if we repeat A a polynomial number of times, the probability of producing a correct answer remains negligible; ie, we can't use polynomial amplification to improve our chances of producing a correct answer.

Let me try to illustrate the above explanation by way of an example. Let $S_n = \{0,1\}^n$, with which we'll associate the uniform distribution, and let f be a function that maps S_n to $\{0,1\}$.

Suppose that f maps a single element in S_n to 0, and the other $2^n - 1$ elements to 1, then we have

$$d_f.0 = \frac{1}{2^n} \quad \text{and} \quad d_f.1 = 1 - \frac{1}{2^n} .$$

Consequently, given 1, by simply picking an n-bit string uniformly at random, we have a $1 - 1/2^n$ probability of selecting a valid inverse, meaning the error probability is negligible. However, given 0, if we pick an n-bit string at random, we have only a $1/2^n$ probability of success, meaning the probability of success is negligible. Therefore, for a simple random guessing algorithm, 0 represents a hard instance, and 1 represents an easy instance.

We can model the average probability of success (ie, the probability that selecting an n-bit string uniformly at random is a valid inverse for f applied to a uniformly selected n-bit string) as

$$\langle \mathcal{P} x, y \in S_n :: f.x = f.y \rangle \quad ,$$

which "simplifies" to

$$(77) \qquad \frac{2^{2 \cdot n} - 2^{n+1} + 2}{2^{2 \cdot n}} \quad ,$$

where

$$1 - \frac{2}{2^n} \quad < \quad \frac{2^{2 \cdot n} - 2^{n+1} + 2}{2^{2 \cdot n}} \quad < \quad 1 - \frac{1}{2^n} \quad ,$$

for $n \geq 2$. To establish the lower bound, first observe that

$$\begin{array}{ll} & 1 - 2/2^n \\ = & \quad \{ \text{ fractions } \} \\ & \frac{2^n \cdot (2^n - 2)}{2^n \cdot 2^n} \\ = & \quad \{ \text{ arithmetic } \} \\ & \frac{2^{2 \cdot n} - 2^{n+1}}{2^{2 \cdot n}} \end{array}$$

and subsequently that

$$2^{2 \cdot n} - 2^{n+1} \quad < \quad 2^{2 \cdot n} - 2^{n+1} + 2 \quad .$$

The upper bound can be established similarly, though it's somewhat obvious from the definition and the above discussion, and arguably moot since $1 - 2 \,/\, 2^n$ provides a good approximation of (77).

In other words, though a hard instance exists, there are enough easy instances that the probability of success in the average case approaches 1 as n approaches infinity, and hence the probability of failure is negligible. With (strong) one-way functions we require the opposite: easy cases may exist, but there should be enough hard instances that the probability of success is negligible on average, and so the probability of failure approaches 1 as n approaches infinity, but not just for a random guess algorithm, rather, for all probabilistic polynomial-time algorithms.

* * *

Chapter 6

Reduction and proof by contradiction

In this chapter I explore rather more explicitly the structure of Goldreich's proofs, and in particular the use and validity of proof by contradiction, and the use of reduction, a technique used to reason about the complexity of one problem relative to another problem, where —presumably— we have a better understanding of the complexity of the latter. I first explore proof by contradiction and reduction, and then look at the structure of the two proofs.

* *

Proof by contradiction

Contexts provide a nice way of explaining proof by contradiction. Suppose our demonstrandum is of the form

$$(78) \qquad A \quad \Rightarrow \quad B \qquad .$$

As is a standard step in many proofs involving implication, we may absorb the antecedent, A, into the context, leaving us to prove the consequent, B, drawing on A and other elements of the context as necessary. Suppose we don't know how to prove B directly, and so instead opt to prove it by contradiction; that is, we prove

$$\neg B \quad \Rightarrow \quad \mathbf{false} \qquad ,$$

from which we can conclude $B \equiv \mathbf{true}$ since

$$\begin{array}{ll} & \neg B \quad \Rightarrow \quad \mathbf{false} \\ \equiv & \qquad \{ \text{ definition of } \Rightarrow \} \end{array}$$

$\neg\,\neg B \quad \vee \quad \mathbf{false}$

$\equiv$ { $\neg\,\neg$ and identity of disjunction }

B

But how do we reach a contradiction? In principle we are free to contradict anything in the context, but in view of the shape of (78), the obvious choice is to contradict A by deriving $\neg\, A$, in which case the proof takes the shape

$[\![$ Context: A

$\neg B$

$\Rightarrow$ { ?? }

C

$\Rightarrow$ { ?? }

$\neg A$

$\equiv$ { context: $A \equiv \mathbf{true}$ so $\neg A \equiv \mathbf{false}$ }

$\mathbf{false}$

$]\!]$

where C is an intermediate step, or chain of steps, in the calculation.

Suppose we decide to establish C also by contradiction; that is, we demonstrate

$$\neg B \quad \Rightarrow \quad (\neg\, C \;\Rightarrow\; \mathbf{false}) \qquad ,$$

in order to conclude $\neg\, B \;\Rightarrow\; C$. We can absorb $\neg\, B$ into the context and prove

$[\![$ Context: $A, \neg B$

$\neg C$

$\Rightarrow$ { ?? }

$\neg A \vee B$

$\equiv$ { context: $A \equiv \neg B \equiv \mathbf{true}$ so $\neg A \equiv B \equiv \mathbf{false}$ }

$\mathbf{false} \vee \mathbf{false}$

$\equiv$ { idempotence }

$\mathbf{false}$

$]\!]$

The disjunction is merely to emphasise that we are free to contradict A or $\neg B$, so if we were to contradict $\neg B$ the proof could be rendered more simply as

$$
\begin{array}{ll}
[\![& \text{Context: } A, \neg B \\
 & \\
 & \neg C \\
\Rightarrow & \quad \{ \text{ ?? } \} \\
 & B \\
\equiv & \quad \{ \text{ context: } \neg B \equiv \mathbf{true} \text{ so } B \equiv \mathbf{false} \} \\
 & \mathbf{false} \\
 & \\
]\!] &
\end{array}
$$

*

On the use of proof by contradiction

Although proof by contradiction is a sound and well established strategy, that does not mean that it is a good strategy: in many situations it results in a contorted argument, if only because of the potential for confusion arising from negating the hypothesis, and establishing the negation of something in the context in order to derive a contradiction. Consequently, proponents of the calculational style tend to prefer to avoid proof by contradiction where possible. As van Gasteren points out:

> [An] advantage of the calculational style is that the validity of the steps is independent of the truth value of the expressions massaged: a step $P = \{\ldots\}\ Q$ holds irrespective of the value of, for instance, P. As a result, proof by contradiction loses much of its special status: a calculation of the form $\neg P \Rightarrow \{\}\ldots\{\}\ \mathbf{false}$ is not more special than a calculation of the form $P \Leftarrow \{\}\ldots\{\}\ \mathbf{true}$. In fact, the one can be transformed into the other, by transforming steps $R \Rightarrow Q$ into $\neg R \Leftarrow \neg Q$, and $R \equiv Q$ into $\neg R \equiv \neg Q$. [48] (Page 162)

However, where proof by contradiction cannot be avoided, or where it is genuinely the most appropriate tactic, confusion can be avoided by following a few simple guidelines:

- state up front that the proof is going to be by contradiction
- be explicit about the contradiction hypothesis
- explicitly state, and if necessary justify, what you are going to contradict

* *

Reduction

Reduction proofs are the main tool used to establish statements about relative complexity, and in particular, statements of the form "P is a member of complexity class C if Q is a member of C" . Informally, a problem P is said to "reduce" to a problem Q if we can use any solution to Q to solve any instance of P .

An "oracle" is a hypothetical black-box that computes a function —possibly an uncomputable function— in a single step. An "oracle machine" is a Turing machine connected to an oracle. Such a machine can be viewed as having two tapes, one of which is reserved for oracle inputs and outputs. The Turing machine writes a query to the oracle's input tape, then calls the oracle to compute its function for that query; the oracle computes its function and replaces the input query with the result.

A "Turing reduction" from P to Q is an oracle machine that solves P using an oracle for Q . A "many-one reduction" , or "transformation" , is a restricted form of Turing reduction that converts instances of one decision problem into instances of another decision problem, and makes as its final step a single call to the oracle.

Remark. Many-one reductions are so-named since the transformation need not be injective; the term "one-reduction" is sometimes used to describe injective (many-one) reductions.
End of Remark.

The "reduces to" relation is a preorder: it is transitive (if P reduces to Q and Q reduces to R , then P reduces to R , so a solution to R yields a solution to P and a solution to Q) , and reflexive (since any problem trivially reduces to itself) ; the former property is particularly useful for establishing membership of complexity classes.

As it stands, the best the definitions allow us to conclude in terms of relative complexity, is whether or not a problem is solvable. For example, suppose we want to show that P is computable. We pick for Q a problem known to be computable, and reduce P to Q by showing how we can use an oracle for Q to solve P ; it follows from Q being computable that P must also be computable. Conversely, suppose we want to show that P is uncomputable. We pick for Q some problem known to be uncomputable, and reduce Q to P ; in so doing we would have demonstrated that any solution to P could be used to solve Q , but since Q is uncomputable, by contradiction P must also be uncomputable.

In order to use reduction to make finer grained comparisons of the form "P can be solved with complexity x if Q can be solved with complexity y" , we need to consider the complexity of the reduction algorithm and the complexity of instantiations of the oracle. To see why, it is, for example, easy to show how we could use a solution to a polynomial-time problem to solve an $\mathbf{NP}$ problem, if we allow the reduction program to run in exponential-time, since all $\mathbf{NP}$ problems can be solved in exponential-time. However, such a reduction is of little interest, since the reduction is at least as hard as the original problem, and so tells us nothing about the complexity relationship between the two problems. Consequently, in order to make useful assertions about the complexity of one problem on the basis of another problem, we need to bound the complexity

of instantiations of the oracle, and the complexity of the reduction algorithm that uses the oracle.

The bounds used will depend on the granularity of the intended comparison. For example, if we want to establish that P is computable in polynomial-time if Q is computable in polynomial-time, we require only that the reduction algorithm runs in overall polynomial-time when the oracle for Q is replaced by a polynomial-time implementation. However, if we want to make a finer grained comparison by showing that P is quadratic if Q is quadratic, then the reduction may only make a constant number of calls to the oracle, since the algorithm must run in overall quadratic time when the oracle is replaced with a quadratic implementation. In general, the tighter the comparison we want to make, the tighter the bound must be on the reduction.

Remark. A reduction computable in polynomial-time is called a "Karp reduction" if it is a many-one reduction, or a "Cook reduction" if it is a Turing reduction but not a many-one reduction. This terminology is unfortunate if only because these names are not in the least bit suggestive of the concepts they denote.
End of Remark.

The notions of reduction described so far are deterministic in the sense that the oracle, and consequently the reduction algorithm, always returns a correct answer. If we admit the possibility that the oracle is correct only with a certain probability, then we must take into account not only the complexity of the reduction, but also the distribution induced by our reduction program. In particular, when reducing one problem to another, the mapping between instances need not be injective, in which case the probability of solving the one is not simply the probability of solving the other. Consequently, our reduction algorithm may need to use some form of amplification.

For example, to show that P is a member of $\mathbf{BPP}$ if Q is a member of $\mathbf{BPP}$, we must show how a probabilistic polynomial-time implementation of Q that outputs an answer that is correct with a negligible probability of error, can be used to construct an answer to P that is also correct with only a negligible probability of error, in polynomial-time.

*

An example reduction

The following is based on an example given on Page 172 of Moret's "The Theory of Computation" [41]. Let S be a set of natural numbers, and let the "weight" of S, which we'll denote w, be the sum of the elements of S; ie:

$$w = \langle \Sigma i \in S :: i \rangle$$

The "partition" problem is to decide whether S can be partitioned into two subsets with the same weight. Given a (natural) bound b, the "smallest subsets" problem is to decide how many subsets of S have weight at most b.

We can reduce the partition problem to the smallest subsets problem; that is, given an algorithm, or rather, an oracle, to solve the smallest subsets problem, we can use it to solve the partition problem. First, an obvious, but important special case: for a solution to the partition problem to exist, w must be even, so if w is odd the answer is "no".

Assuming that w is even, we first call the oracle with

$$b := w/2 \quad ,$$

and then with

$$b := w/2 - 1 \quad .$$

The difference between the results is the number of subsets of S with weight exactly $w/2$; hence, if the difference is zero the answer to the partition problem is "no", otherwise the answer is "yes". (To clarify the latter assertion, whenever there exists a subset with weight $w/2$, the subset containing the "leftover" elements must also have weight $w/2$, since their union is S, thus guaranteeing the existence of a valid partitioning.)

The above is clearly an example of a polynomial-time Turing reduction (ie, a Cook reduction). It is easy to see how the partition problem and smallest subsets problem, and the above reduction between them, generalise to arbitrary sets of objects with a corresponding function that assigns a weight to each object (the weight of the set being the sum of the weights of the elements).

* *

Structural analysis of ping

The demonstrandum has the shape

$$(79) \qquad f \text{ is a one-way function} \quad \Rightarrow \quad g \text{ is a weak one-way function} \quad .$$

First, observe that it's not clear whether f should be a weak or a strong one-way function. Goldreich in essence makes the assumption that f is a weak one-way function; I'll explore the consequences of this in due course.

The antecedent, viz the assumption that f is weakly one-way, is absorbed into the context, leaving us to prove that g is weakly one-way. The proof is by contradiction, so we establish:

⟦ Context: f is weakly one-way

g is not weakly one-way

⇒ { ?? }

false

⟧

But how exactly is the contradiction derived, and in particular, what is contradicted?

*

Being explicit about quantifiers, recall that f is weakly one-way if

$$\langle \exists q :: \langle \forall A' :: \langle \exists N :: \langle \forall n : n > N : \Pr[\ \ldots\ \notin\ \ldots\ ]\ >\ 1 \,/\, q(n) \rangle \rangle \rangle \rangle$$

which, by an appeal to (49) (Page 67), can be rewritten as

$$(80) \qquad \langle \exists q :: \langle \forall A' :: \langle \exists N :: \langle \forall n : n > N : \Pr[P]\ <\ 1 - 1 \,/\, q(n) \rangle \rangle \rangle \rangle \qquad ,$$

where

$$P \quad \equiv \quad A' \circ f.U_n\ \in\ f^{-1} \circ f.U_n \qquad .$$

Similarly, g is weakly one-way if

$$\langle \exists p :: \langle \forall B' :: \langle \exists M :: \langle \forall m : m > M : \Pr[Q]\ <\ 1\ -\ 1 \,/\, p(m) \rangle \rangle \rangle \rangle \qquad ,$$

where

$$Q \quad \equiv \quad B' \circ g.U_m\ \in\ g^{-1} \circ g.U_m \qquad .$$

The negation of this, and hence the assumption that g is not weakly one-way, means we have

$$(81) \qquad \langle \forall p :: \langle \exists B' :: \langle \forall M :: \langle \exists m : m > M : \Pr[Q]\ \geq\ 1\ -\ 1 \,/\, p(m) \rangle \rangle \rangle \rangle \qquad .$$

Remark. In view of the definitions of weak and strong one-way functions, it is perhaps natural to expect the polynomials associated with f and g to be named p and q respectively, but unfortunately Goldreich adopts the converse as above.
End of Remark.

Goldreich contradicts (80) by establishing its negation, ie:

$$(82) \qquad \langle \forall q :: \langle \exists A' :: \langle \forall N :: \langle \exists n : n > N : \Pr[P] \ \geq \ 1 \ - \ 1 \,/\, q(n) \rangle \rangle \rangle \rangle$$

So, the proof of (79) has the shape:

$[\![$ Context: (80)

(81)
$\Rightarrow$ $\{$?? $\}$
(82)
$\equiv$ $\{$ $(82) \ \equiv \ \neg(80)$ $\}$
false

$]\!]$

The goal then, is to establish

$$(83) \qquad (81) \ \Rightarrow \ (82)$$

in the context of (80) .

*

In words, (81) asserts that for every polynomial p , there exists a probabilistic polynomial-time algorithm B' , that can invert g with probability at least $1 - 1 \,/\, p(m)$ for "infinitely many" values of m , where each m is of the form $2 \cdot n$ for some n , by virtue of how g was constructed.

Since our goal is to establish (82) , we demonstrate how, for any polynomial q , we can construct an algorithm A' that inverts f with probability at least $1 - 1 \,/\, q(n)$ for infinitely many values of n . Observe that each polynomial q may be associated with a different algorithm A' , and likewise for the antecedent of our demonstrandum: each polynomial p may be associated with a different algorithm B' .

The idea underlying the proof is to show how, for an arbitrary polynomial q , we can construct an appropriate algorithm A' , using an appropriate B' , leading to what is essentially a family of reductions. Goldreich describes a generic A' , and then analyses its success probability; it is during this analysis, or rather, as a consequence of this analysis, that we discover how to relate the polynomials p and q .

Specifically, Goldreich establishes that, in contradiction to (80) , for any polynomial $q(n)$, we can construct an algorithm A' that inverts f with unallowable probability of success, by

using an algorithm B' associated with the polynomial $p(2n)$, where $p(2n) = q(n) \cdot n$. An appropriate B' is guaranteed to exist by virtue of (81).

As suggested above, the proof actually involves a family of reductions, in the sense that although a generic A' is described, it may be necessary to use a different B' for each polynomial q. Observe that the bound on the reduction is defined by (82): since the requirement is to show that for each polynomial q we can construct an appropriate A' that runs in probabilistic polynomial-time, our reduction must run in overall probabilistic polynomial-time. Of course, this presents no problems: in addition to constructing an appropriate input, A' makes only a single call to B', but since B' is a probabilistic polynomial-time algorithm, so is A'.

Something that is left rather implicit in the proof is the business of "infinitely many" values of n and m. The goal was to show that A' inverts f with unallowable success probability for infinitely many values of n. However, since (each) B' works for infinitely many values of m, each of which is of the form $2 \cdot n$ for some n, it follows that A' must also work for infinitely many values of n.

*

Following his proof of ping, Goldreich points out that:

> We have just shown that unless no one-way functions exist, there exist weak one-way functions that are not strong ones. This rules out the possibility that all one-way functions are strong ones.

However, since the proof involved assuming that f was weakly one-way in order to show that g was weakly one-way, it's not clear that it establishes that the existence of strong one-way functions implies the existence of weak one-way functions. In particular, two questions need answering: First, does it matter whether f is weak one-way function or a strong one-way function? And second, can g be a strong one-way function?

With respect to the former, provided f is one-way, it doesn't matter whether it's weakly or strongly one-way: as pointed out in Chapter 2, the proof proceeds by showing that if g is not weakly one-way then f cannot be weakly one-way; hence, if f is weakly one-way we reach a contradiction (as explored above), but if f is strongly one-way we still reach a contradiction, since by showing that f can't be weakly one-way, it follows that f certainly can't be strongly one-way.

With respect to the latter, the probability of inverting g on a uniformly selected input of length $2 \cdot n$ is at least $1 - 1/n$ by definition; "at least" because the probability that g acts as the identity function is $1 - 1/n$, but with at least a small probability we can invert elements where g does not act as the identity function, even by just guessing. This precludes g from being a strong one-way function, since in order to be strongly one-way the probability would have to be less than $1 / p.2n$ for all positive polynomials p, but $1 - 1/n$ tends to 1 as n tends to infinity, whereas $1 / p.2n$ tends to zero as n tends to infinity, provided p isn't constant.

It follows then, that if f is a strong one-way function, then g is a weak one-way function, and so weak one-way functions must exist if strong one-way functions exist. In other words, this construction gives us a way of weakening strong one-way functions.

* *

Structural analysis of pong

The demonstrandum has the shape

(84) f is a weak one-way function $\Rightarrow$ g is a strong one-way function .

As with the proof of ping, the antecedent, viz that f is weakly one-way, is absorbed into the context, leaving us to prove that g is strongly one-way, drawing on the context as necessary. As with ping, the proof is by contradiction.

First, f being weakly one-way means it satisfies the one-wayness condition

$$(85) \qquad \langle \exists p :: \langle \forall A' :: \langle \exists N :: \langle \forall n : n > N : \Pr[P] \ < \ 1 \ - \ 1 \,/\, p(n) \rangle \rangle \rangle \rangle \qquad ,$$

where P is defined as above. (I've introduced (85) because, whereas in the proof of ping, p was associated with f, and q with g, here we have the opposite.) Since (85) is a given, we may discharge the outer existential quantifier, which guarantees the existence of an appropriate polynomial p (in the jargon this is expressed by saying that we may replace a dummy variable bound to an existential quantifier by a "witness"); therefore,

$$(86) \qquad \langle \forall A' :: \langle \exists N :: \langle \forall n : n > N : \Pr[P] \ < \ 1 \ - \ 1 \,/\, p(n) \rangle \rangle \rangle$$

is absorbed into the context rather than (85).

Next, recall that g is strongly one-way if

$$(87) \qquad \langle \forall q, B' :: \langle \exists M :: \langle \forall m : m > M : \Pr[Q] \ < \ 1 \,/\, q(n) \rangle \rangle \rangle \qquad ,$$

where Q is defined as above. To prove (87) we demonstrate that

$$\neg(87) \quad \Rightarrow \quad \textbf{false} \qquad ,$$

in the context of (86), where the contradiction is reached by deriving $\neg(86)$; hence, the proof

of (84) has the shape:

$[\![$ Context: (86)

$\neg(87)$
$\Rightarrow$ { ?? }
$\neg(86)$
$\equiv$ { context: $(86) \equiv \mathbf{true}$ }
$\mathbf{false}$

$]\!]$

Clearly, $\neg(87)$ is equivalent to

$$\langle \exists q, B' :: \langle \forall M :: \langle \exists m : m > M : \Pr[Q] \ \geq \ 1 \,/\, q(n) \rangle \rangle \rangle \quad ,$$

where, as above, we can discharge the outer (two) existential quantifiers, replacing the dummies with witnesses q and B' such that

$$(88) \qquad \langle \forall M :: \langle \exists m : m > M : \Pr[Q] \ \geq \ 1 \,/\, q(n) \rangle \rangle \quad .$$

The negation of (86) is

$$(89) \qquad \langle \exists A' :: \langle \forall N :: \langle \exists n : n > N : \Pr[P] \ \geq \ 1 \, - \, 1 \,/\, p(n) \rangle \rangle \rangle \quad ,$$

and so our goal is to show that $(88) \Rightarrow (89)$, which we do by reduction. Specifically, we use the witness B' to construct an algorithm A' where

$$\langle \forall N :: \langle \exists n : n > N : \Pr[P] \ \geq \ 1 \, - \, 1 \,/\, p(n) \rangle \rangle \quad .$$

As we have seen, this involves constructing an algorithm I that, given an element y in f's range, constructs an element in g's range that contains y as a "sub-block" , and calls B' on that element. (Actually, algorithm I repeats this process a number of times by varying the position of y in the constructed input to B' .) A' repeats algorithm I a polynomial number of times to amplify the probability of a correct answer.

Observe that by definition A' must be a probabilistic polynomial-time algorithm, and so the reduction must run in overall probabilistic polynomial-time; this is guaranteed by the given construction of A' in terms of I and B' , since algorithm I calls B' a polynomial number

of times, and B' runs in probabilistic polynomial-time by definition; so, by repeating I a polynomial number of times, A' must also run in probabilistic polynomial-time, as required.

The probabilistic analysis of A' is split into two pieces that correspond to Goldreich's Claim 2.3.2.1 , viz

$$(90) \qquad \langle \forall x \in S_n :: \Pr[A'(f(x)) \in f^{-1}(f(x))] \ > \ 1 \ - \ 1 \,/\, 2^n \rangle \qquad ,$$

and Claim 2.3.2.2 , viz

$$(91) \qquad \langle \forall n \in N' :: |S_n| \ > \ (1 \ - \ (2p(n))^{-1}) \cdot 2^n \rangle \qquad ,$$

which are established in the local context of (88) , where

$$(92) \qquad (90) \ \wedge \ (91) \quad \Rightarrow \quad (89) \qquad ;$$

ie, (92) corresponds to the final part of Goldreich's proof of pong, where he "combines" Claim 2.3.2.1 and Claim 2.3.2.2 , the validity of which has already been explored. Consequently, the proof of (84) has the shape:

$[\![$ Context: (86)

(88)
$\Rightarrow$ { explored below }
$(90) \ \wedge \ (91)$
$\Rightarrow$ { (92) }
(89)
$\equiv$ { context: $(89) \ \equiv \ \neg(86) \ \equiv$ **false** }
false

$]\!]$

(90) is proved directly. The steps have been explored in Chapter 4 and Chapter 5 , but the claim amounts to demonstrating that for those elements we can invert with non-negligible probability, we can use amplification to ensure that the probability of error vanishes exponentially.

(91) is proved by contradiction, in two parts. First, Goldreich defines $s(n)$ as

$$\Pr[B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)}))] \qquad ,$$

where, according to (88) ,

$$s(n) \quad \geq \quad \frac{1}{q(n^2 p(n)))} \qquad .$$

He then partitions $s(n)$ into two events, $s_1(n)$ and $s_2(n)$, so that

$$s(n) \quad = \quad s_1(n) + s_2(n) \qquad ,$$

and establishes that $\neg(91)$ implies

$$(93) \qquad s_1(n) \quad \leq \quad \frac{n^2 \cdot p(n)}{a(n)} \qquad ,$$

and

$$(94) \qquad s_2(n) \quad < \quad \frac{n^2 \cdot p(n)}{a(n)} \qquad ,$$

in contradiction to (88) . Consequently, the proof of (91) has the shape:

$[\![$ Context: (88)

$\neg(91)$
$\Rightarrow$ { step 1 }
$(93) \wedge (94)$
$\Rightarrow$ { step 2 }
$\neg(88)$
$\equiv$ { context: $(88) \equiv \mathbf{true}$ }
$\mathbf{false}$

$]\!]$

The first step is established by two calculations, which have been explored and justified in previous chapters. The second step amounts to the following

$\Pr[B'(g(U_{n^2 p(n)})) \in g^{-1}(g(U_{n^2 p(n)}))]$
$=$ { definition of $s(n)$ above }
$s_1(n) + s_2(n)$

$$
\begin{aligned}
< \quad & \{\ (93)\ \text{and}\ (94)\ \} \\
& \frac{n^2 \cdot p(n)}{a(n)} + \frac{n^2 \cdot p(n)}{a(n)} \\
= \quad & \{\ \text{fractions}\ \} \\
& \frac{2 \cdot n^2 \cdot p(n)}{a(n)} \\
= \quad & \{\ \text{definition of}\ a(n)\ \} \\
& \frac{2 \cdot n^2 \cdot p(n)}{2 \cdot n^2 \cdot p(n) \cdot q(n^2 p(n))} \\
= \quad & \{\ \text{fractions}\ \} \\
& 1 \,/\, q(n^2 p(n))
\end{aligned}
$$

which contradicts (88) , and so the theorem follows.

As a final comment, it's worth observing that of s_1 and s_2 , it's only the proof of the bound on the latter that uses the contradiction hypothesis, and so "forces" the proof of (91) to be by contradiction —this will be revisited in the next chapter— .

* * *

Chapter 7

Cryptography revisited

Having explored the concepts that underly the definitions of one-way functions, and having examined the structure of Goldreich's proof, in this chapter I reintroduce the definitions of one-way functions using the alternative notation explored in the previous chapters, and then reexamine ping and pong.

With respect to ping, I present a slightly cleaner version of Goldreich's proof, and a far simpler, but non-constructive proof, which establishes that strongly one-way functions are by definition weakly one-way, from which it follows that the existence of strong one-way functions implies the existence of weak one-way functions. With respect to pong, I explore why the proof is structured the way it is, and restructure the proof to eliminate a number of rabbits and the "nested" proof by contradiction.

It is fairly straightforward to compare my revised version of ping to Goldreich's version in Appendix A . However, while restructuring Goldreich's proof of pong I make numerous references to his proof, so for the sake of completeness, and so the reader can compare my version to Goldreich's version in Appendix A , a rather more self-contained version of my proof appears in Appendix B .

* *

One-way functions revisited

The first part of each definition asserts that one-way functions are computable in deterministic polynomial-time, which —as we now know— means that, given a one-way function f , the problem of computing $f.x$ is a member of the complexity class $\mathbf{FP}$.

With respect to the one-wayness part of the definitions, adopting the alternative notation explored earlier in the thesis, a function f is weakly one-way if

$$(95) \qquad \langle \exists p :: \langle \forall F :: \langle \diamondsuit n :: \langle \mathcal{P}x :: f \circ F \circ f.x \ = \ f.x \rangle \ < \ 1 \ - \ 1 \,/\, p.n \rangle \rangle \rangle$$

or strongly one-way if

$$(96) \qquad \langle \forall p, F :: \langle \langle \diamond \rangle \, n :: \langle \mathcal{P}x :: f \circ F \circ f.x \;=\; f.x \rangle \;<\; 1 \,/\, p.n \rangle \rangle$$

In both cases p ranges over positive polynomials with at least degree 1 ; F ranges over the class of probabilistic polynomial-time algorithms; n is of type natural; and x is a bit-string selected uniformly at random from $\{0,1\}^n$.

These reformulations follow Goldreich's definitions in the sense that I've adopted the online view of randomised algorithms. It's tempting to adopt the offline view, but as we've seen in the proof of (4) , the random choices do not contribute anything to the analysis, so there seems little point in referring to them explicitly.

* *

Ping revisited

In this section I first present a modified version of Goldreich's proof, with emphasis on readability, and hence verifiability; I then present a simpler, non-constructive proof, that strong one-way functions are weak by definition.

*

Let f be a weak one-way function, and let g be a function defined in terms of f , where, for $\#x_0 \;=\; \#x_1 \;=\; n$,

$$g.(x_0 \;\texttt{++}\; x_1) \quad = \quad \begin{cases} x_0 \;\texttt{++}\; f.x_1 & \text{if } x_0 \text{ is prefixed by } \log_2 n \text{ zeros} \\ x_0 \;\texttt{++}\; x_1 & \text{otherwise} \end{cases}$$

Since f is weakly one-way it satisfies (95) . In the following, let $Q.x$ denote the predicate

$$(97) \qquad g \circ G \circ g.x \quad = \quad g.x \qquad .$$

The goal is to show that g is weakly one-way, and hence that

$$\langle \exists q :: \langle \forall G :: \langle \langle \diamond \rangle \, m :: \langle \mathcal{P}x :: Q.x \rangle \;<\; 1 \;-\; 1 \,/\, q.m \rangle \rangle \rangle \qquad ,$$

where, by construction, every m is of the form $2 \cdot n$ for some n .

Since what follows is essentially a restatement of Goldreich's proof, we proceed by contradiction, by demonstrating that if g is not weakly one-way, and hence that

$$\langle \forall q :: \langle \exists G :: \langle [\diamond] \, m :: \langle \mathcal{P}x :: Q.x \rangle \;\geq\; 1 \;-\; 1 \,/\, q.m \rangle \rangle \rangle \qquad ,$$

then f cannot be weakly one-way.

*

Let F, our algorithm to invert f, be constructed as previously described by Goldreich; ie, let F be the same as the algorithm Goldreich denoted A'. Before calculating F's probability of success, let's first deal with the claim that g coincides with the identify function for all but a $1/n$ fraction of strings of length $2 \cdot n$ —ie, even length strings— since

$1/n$ of bit-strings of length $2 \cdot n$ are prefixed with at least $\log_2 n$ zeros .

Our state space comprises $2^{2 \cdot n}$ strings in total, of which $2^{2 \cdot n \, - \, \log_2 n}$ are prefixed by at least $\log_2 n$ zeros, which it's claimed is $1/n$ of the total state space, leading to the demonstrandum

$$2^{2 \cdot n \, - \, \log_2 n} \quad = \quad 2^{2 \cdot n} \, / \, n \qquad ,$$

and the simple proof:

$$\begin{array}{ll} & 2^{2 \cdot n - \log_2 n} \\ = & \quad \{ \text{ exponents } \} \\ & 2^{2 \cdot n} \, / \, 2^{\log_2 n} \\ = & \quad \{ \text{ exponents } \} \\ & 2^{2 \cdot n} \, / \, n \end{array}$$

As explained in the previous chapter (Page 112), the above ensures that g cannot be a strong one-way function.

*

Taking as our sample space the set of bit-strings of length $2 \cdot n$, with which we associate the uniform distribution, and defining $P.x$ as the predicate "x is prefixed by at least $\log_2 n$ zeros", we have

$$(98) \qquad \langle \mathcal{P} x :: P.x \rangle \quad = \quad 1/n \qquad ,$$

and its complement

$$(99) \qquad \langle \mathcal{P} x :: \neg P.x \rangle \quad = \quad 1 \, - \, 1/n \qquad .$$

To derive a contradiction we assume that g is not weakly one-way, and hence that

$$(100) \qquad \langle \mathcal{P}x :: Q.x \rangle \quad \geq \quad 1 \; - \; 1 \,/\, q.(2 \cdot n) \qquad .$$

Next, since F calls G with an input known to satisfy P, it follows that F's probability of success is at least the conditional probability

$$(101) \qquad \langle \mathcal{P}x : P.x : Q.x \rangle \qquad .$$

Since we know the respective probabilities of P and Q, our goal is to evaluate (101) by isolating P and Q, leading to the following calculation:

$$\begin{array}{ll}
 & \langle \mathcal{P}x : P.x : Q.x \rangle \\
= & \quad \{ \; (28) \; \} \\
 & \langle \mathcal{P}x :: P.x \; \wedge \; Q.x \rangle \; / \; \langle \mathcal{P}x :: P.x \rangle \\
\geq & \quad \{ \; (41) \; \} \\
 & (\langle \mathcal{P}x :: Q.x \rangle \; - \; \langle \mathcal{P}x :: \neg P.x \rangle) \; / \; \langle \mathcal{P}x :: P.x \rangle \\
= & \quad \{ \; (98) \text{ and } (99) \; \} \\
 & (\langle \mathcal{P}x :: Q.x \rangle \; - \; (1 \; - \; 1/n)) \; / \; \frac{1}{n} \\
= & \quad \{ \text{ fractions } \} \\
 & n \cdot (\langle \mathcal{P}x :: Q.x \rangle \; - \; (1 \; - \; 1/n)) \\
= & \quad \{ \text{ distributivity } \} \\
 & n \cdot \langle \mathcal{P}x :: Q.x \rangle \; - \; n \cdot (1 \; - \; 1/n) \\
\geq & \quad \{ \; (100) \; \} \\
 & n \cdot (1 \; - \; 1 \,/\, q.(2 \cdot n)) \; - \; n \cdot (1 \; - \; 1/n) \\
= & \quad \{ \text{ distributivity } \} \\
 & n \; - \; n \,/\, q.(2 \cdot n) \; - \; n \; + \; 1 \\
= & \quad \{ \text{ algebra } \} \\
 & 1 \; - \; n \,/\, q.(2 \cdot n)
\end{array}$$

It follows that, if

$$q.(2 \cdot n) \quad = \quad p.n \cdot n \qquad ,$$

then $\langle \mathcal{P}x : P.x : Q.x \rangle$, and hence F's probability of success, is at least $1 \; - \; 1 \,/\, p.n$, establishing the required contradiction, and hence the demonstrandum.

*

Clearly the above proof, and in particular the calculation, is longer than Goldreich's, but the steps are finer grained in order to aid readability. It can be argued that Goldreich's construction of g is not very useful. We can avoid the construction and instead demonstrate that any strong one-way function is by definition a weak one-way function, ie

$$\text{strong}.f \quad \Rightarrow \quad \text{weak}.f \qquad ,$$

from which it follows that if strong one-way functions exist, so too must weak one-way functions.

The key observation is that for $p.n > 2$ we have $1/p.n < 1 - 1/p.n$. So, if f is a strong one-way function, and P is defined as

$$f \circ F \circ f.x \quad = \quad f.x \qquad ,$$

then we have

$$\begin{array}{ll} & \langle \forall p, F :: \langle \diamondsuit n :: \langle \mathcal{P}x :: P \rangle \ < \ 1/p.n \rangle \rangle \\ \Rightarrow & \quad \{\ 1/p.n \ < \ 1 - 1/p.n \ \text{ for } \ p.n \ > \ 2\ \} \\ & \langle \forall p, F :: \langle \diamondsuit n :: \langle \mathcal{P}x :: P \rangle \ < \ 1 - 1/p.n \rangle \rangle \\ \Rightarrow & \quad \{\ \text{quantifiers}\ \} \\ & \langle \exists p :: \langle \forall F :: \langle \diamondsuit n :: \langle \mathcal{P}x :: P \rangle \ < \ 1 - 1/p.n \rangle \rangle \rangle \end{array}$$

* *

Pong revisited

In the remainder of the chapter I explore why the proof of pong is structured the way it is, and the motivation behind the mysterious values, $a.n$, $n \cdot p.n$, $n/a.n$, and $1/(2 \cdot p.n)$; in particular, I explore whether it is possible to restructure the proof so that either these values or a different set of values arise a little more naturally.

For simplicity, let it be understood that in the remainder of the chapter, p and a denote polynomials in n, while q denotes a polynomial in m, where m denotes the size of inputs to g as a function of n.

*

Observe that in the proof of ping it was possible to establish the probability of (101) directly, in the sense that we were able to isolate P and Q, and replace them with concrete values.

So why wasn't it possible to reason this way in the proof of pong? To see why, let's consider a simpler scenario. Let f be a one-way function, and define g as

$$g.(x_0 \mathbin{++} x_1) \quad = \quad f.x_0 \mathbin{++} f.x_1 \qquad .$$

So, where f is applied to strings of length n, g is applied to strings of length $2 \cdot n$ by applying f to the left and right n bits. Next, suppose our goal is to show that if f is one-way, then g is one-way (under the same notion of one-wayness), which seems a reasonable enough supposition.

Aiming for a proof by contradiction, we assume that g is not one-way, and hence that there exists an algorithm G that can invert g with unallowable success probability. Our first task is to use G to construct an algorithm F to invert $f.x$. For simplicity, define F as

$$F.(f.x) \quad = \quad G.(f.x \mathbin{++} U_n) \qquad .$$

Let's now attempt to establish F's success probability using essentially the same line of reasoning as in the proof of ping. First, let $P.x$ denote the predicate "$f.x$ is a prefix of $U_{2 \cdot n}$". For a given $f.x$, there are 2^n bit-strings of length $2 \cdot n$ prefixed by $f.x$; consequently we have:

$$(102) \qquad \langle \mathcal{P} x :: P.x \rangle \quad = \quad 1 \,/\, 2^n$$

Next, let Q be defined as per (97). Since F calls G with a string that satisfies P, it follows that F's probability of success must be at least the conditional probability

$$(103) \qquad \langle \mathcal{P} x : P.x : Q.x \rangle \qquad .$$

Since we know the respective probabilities of P and Q, we again try to isolate them, just as in the proof of ping; however, when we calculate we hit an unfortunate snag:

$$\begin{array}{ll}
 & \langle \mathcal{P} x : P.x : Q.x \rangle \\
= & \quad \{ \ (28) \ \} \\
 & \langle \mathcal{P} x :: P.x \ \wedge \ Q.x \rangle \ / \ \langle \mathcal{P} x :: P.x \rangle \\
\geq & \quad \{ \ (41) \ \} \\
 & (\langle \mathcal{P} x :: Q.x \rangle \ - \ \langle \mathcal{P} x :: \neg P.x \rangle) \ / \ \langle \mathcal{P} x :: P.x \rangle \\
= & \quad \{ \ (102) \ \} \\
 & (\langle \mathcal{P} x :: Q.x \rangle \ - \ (1 \ - \ 1 \,/\, 2^n)) \ / \ \frac{1}{2^n} \\
= & \quad \{ \text{ fractions ; arithmetic } \} \\
 & 2^n \cdot \langle \mathcal{P} x :: Q.x \rangle \ - \ 2^n \cdot (1 \ - \ 1 \,/\, 2^n)
\end{array}$$

The problem: $\langle \mathcal{P}x :: Q.x \rangle$ does not approach 1 exponentially fast, so the last line of the calculation is negative for large n, which is not much use. Of course, in the second step we could try swapping P and Q, to derive

$$(\langle \mathcal{P}x :: P.x \rangle \ - \ \langle \mathcal{P}x :: \neg Q.x \rangle) \ / \ \langle \mathcal{P}x :: P.x \rangle \qquad ,$$

but alas, that doesn't get us anywhere either.

Briefly, since the goal is to isolate P and Q, the first step of the calculation eliminates the conditional probability, while the second step eliminates the conjunction, this being the problematic step. Unfortunately, looking over the theorems in Chapter 3, it seems we are forced to appeal to (41) (Page 65) as we have no other way of splitting a conjunction without introducing an equally awkward term, such as $\langle \mathcal{P}x :: P.x \ \vee \ Q.x \rangle$ —at least, not with the inequality in the right direction— .

*

Since it's not clear how to reason directly about the probability of I succeeding on a single repetition, Goldreich instead reasons indirectly by introducing the set S_n, of "easy elements" that algorithm I can invert with "non-negligible probability". He then demonstrates that there are sufficiently many elements in S_n, and that they are sufficiently easy that the probability of inverting $f.U_n$, and hence the "average" probability of success, exceeds $1 - 1/p$, which contradicts the assumption that f is weakly one-way.

Let's accept Goldreich's construction of g as a function that "expands" a weak one-way function f, by applying f to a number of n-bit blocks. However, rather than requiring that inputs to g comprise $n \cdot p$ blocks, for now let's require only that inputs comprise some polynomial, call it t, number of blocks. Let's also accept the construction of I and A', but let's leave the polynomial a undefined. Finally, let's also leave undefined the lower bound on I's probability of success on elements in S_n, and the bound on the size of S_n. The goal is to see if we can restructure the proof in such a way that these values either arise naturally, or we are provided with sufficient motivation to intelligently guess values.

Note that in the following discussion I return to denoting the inverting algorithms by A' and B' rather than by F and G, in the hope that this makes it easier to relate my comments to Goldreich's proof.

*

If we jump to the end of Goldreich's proof, the final step involves combining claims 2.3.2.1 and 2.3.2.2, to establish that there are sufficiently many, sufficiently easy elements, such that

$$(104) \qquad \left(1 - \frac{1}{2 \cdot p}\right) \cdot \left(1 - \frac{1}{2^n}\right) \quad > \quad 1 - 1/p \qquad ,$$

where, as established in Chapter 4 , we can replace 2^n by e^n . The value $1 - 1/(2 \cdot p)$ denotes Goldreich's characterisation of "sufficiently many" , while the value $1 - 1/2^n$ characterises "sufficiently easy" . Let's reason backwards, exploring other possibilities for these values.

Let's deal with "sufficiently easy" first. In Chapter 5 , in the discussion of probabilistic search (Page 100) , I showed that if we have an algorithm that succeeds with non-negligible probability, then, by repeating that algorithm a polynomial number of times, we can ensure that the error probability vanishes exponentially, and hence is negligible. So, "easy" elements are those that we can invert with non-negligible probability, and hence using polynomial amplification can invert with a negligible probability of error.

Clearly then, our lower bound on "sufficiently easy" elements should be non-negligible, but what bound should we use? A cursory glance at (76) (Page 101) reveals the answer: $1/a$, where a is some non-constant, positive polynomial in n . If I succeeds with probability greater than $1/a$, then I fails with probability at most $1 - 1/a$, so if we repeat I at least $n \cdot a$ times, the probability of failure is less than $1/e^n$, as required. So, for completeness, S_n is defined as the set of n-bit strings that can be inverted with probability greater than $1/a$ using algorithm I ; if A' repeats I at least $n \cdot a$ times, it follows that A' inverts elements in S_n with only a negligible probability of error. Observe that we did not have to define a .

*

Next, let's deal with "sufficiently many" . Rewriting (104) , we need to find a value of σ such that

$$\sigma \cdot \left(1 - \frac{1}{e^n}\right) \quad > \quad 1 - 1/p \qquad ,$$

which we can rewrite as:

$$(105) \qquad 1 - \frac{1}{e^n} \quad > \quad \frac{1 - 1/p}{\sigma}$$

We can immediately rule out a couple of values of σ . First, σ cannot be $1 - 1/p$, as that would falsify the inequality. Second, σ cannot be 1 , because although that would satisfy the inequality, σ denotes the proportion of easy instances, so if $\sigma = 1$ then every instance must be easy, and hence $S_n = \{0,1\}^n$, which is not a reasonable supposition. It follows then, that σ must be a proper fraction, let's denote it σ_0/σ_1 .

Substituting σ_0/σ_1 for σ , we can rewrite (105) as

$$(106) \qquad \frac{e^n - 1}{e^n} \quad > \quad \frac{p-1}{p} \cdot \frac{\sigma_1}{\sigma_0} \qquad .$$

Since $\sigma_0 < \sigma_1$, it follows that $\sigma_1 / \sigma_0 > 1$, and hence that

$$\frac{p-1}{p} \cdot \frac{\sigma_1}{\sigma_0} > \frac{p-1}{p} \quad ,$$

so we must be careful to pick values that preserve (106) ; in particular, σ_1 / σ_0 can be only slightly larger than 1 .

Given the shape of (106) , ideally we'd like to be able to simplify the right side of the inequality to an expression of the form $(b-1)/b$, where b is some polynomial function of p . As mentioned above, setting $\sigma := 1 - 1/p$, and hence setting $\sigma_0 := p-1$ and $\sigma_1 := p$, falsifies (106) . However, it's clear that $p/(p-1)$ meets the requirement of being only slightly greater than 1 , and both the numerator and denominator are polynomial functions of p , so it seems a reasonable starting point for our investigation.

It should be clear that $(p-1)/p$ is strictly monotonic. In particular, as p approaches infinity, $(p-1)/p$ approaches 1 from below, and so $p/(p-1)$ approaches 1 from above. It follows that if we increase p , then $p/(p-1)$ approaches 1 faster than if we decrease p ; so, since we want this ratio to be as close to 1 as possible, it's clear that we should increase p .

Arguably the simplest approach is to add 1 to $p-1$ and p , and so set $\sigma_0 := p$, and $\sigma_1 := p+1$, in which case the right side of (106) becomes

$$\begin{aligned} & \frac{p-1}{p} \cdot \frac{p+1}{p} \\ = & \quad \{ \text{ algebra } \} \\ & \frac{p^2-1}{p^2} \end{aligned}$$

which is less than $(e^n - 1)/e^n$, and so satisfies (106) as required. So, (106) becomes

$$\frac{e^n - 1}{e^n} > \frac{p-1}{p} \cdot \frac{p+1}{p} \quad ,$$

which, with a little manipulation, we can rewrite as

$$(107) \qquad \left(1 - \frac{1}{p+1}\right) \cdot \left(1 - \frac{1}{e^n}\right) > 1 - 1/p \quad ;$$

therefore, $\sigma := 1 - 1/(p+1)$.

*

So, we've derived a bound that satisfies (106), and a lower bound on "sufficiently easy" instances, but it remains to show that the former is valid; ie, we need to establish that

$$(108) \qquad \#S_n \quad > \quad \left(1 - \frac{1}{p+1}\right) \cdot 2^n \qquad .$$

Intriguingly, we haven't yet had to define t or a.

We can prove (108) in essentially the same way that Goldreich proves Claim 2.3.2.2, namely, by defining s_1 and s_2 so that their sum equals the success probability of B', and by showing that if

$$\#S_n \quad \leq \quad \left(1 - \frac{1}{p+1}\right) \cdot 2^n \qquad ,$$

then s_1 and s_2 sum to less than the success probability of B'.

Recall that by assumption the success probability of B' is $1/q$. Goldreich shows that $s_1 \leq 1/(2 \cdot q)$, and that $s_2 < 1/(2 \cdot q)$, where

$$(109) \qquad \frac{1}{2 \cdot q} + \frac{1}{2 \cdot q} \quad = \quad 1/q \qquad ,$$

and so $s_1 + s_2 < 1/q$ as required.

To establish the bound on s_1, we can reuse Goldreich's calculation (Page 49). If we replace the occurrences of $n \cdot p$ in Goldreich's calculation with t, and we replace n/a in the penultimate step with $1/a$, then following the final step of the calculation we end up with $s_1 \leq t/a$. Assuming we can show that $s_2 < t/a$, we need to define t and a so that $t/a = 1/(2 \cdot q)$, which is easily achieved by defining a as $2 \cdot t \cdot q$.

Much as with the bound on s_1, we can adapt Goldreich's proof that $s_2 < 1/(2 \cdot q)$. (As pointed out in the previous chapter, it is this part of the proof that draws on the contradiction hypothesis.) Replacing $2 \cdot p$ in Goldreich's calculation by $p+1$, and replacing $n \cdot p$ by t, following the first step we have

$$s_2 \quad \leq \quad \left(1 - \frac{1}{p+1}\right)^t \qquad .$$

We want to establish that

$$\left(1 - \frac{1}{p+1}\right)^t \quad < \quad \frac{1}{2 \cdot q} \qquad ,$$

which follows if

$$\left(1 - \frac{1}{p+1}\right)^t \quad \sim \quad \frac{1}{e^n} \qquad ,$$

since $1 / e^n$ is less than $1 / (2 \cdot q)$ for large enough n (since exponential functions grow faster than polynomial functions) . Finally then, we're "forced" to define t , but much as with the "sufficiently easy" lower bound, the shape of the demonstrandum presents an obvious definition: we set $t := n \cdot (p+1)$, and theorem follows.

*

The above discussion demonstrates that it is possible to reorder Goldreich's proof so that the values, or rather, a slightly different (but valid) set of values, arise rather more naturally. However, it retains the same approach to establishing the bound on the size of S_n by way of a "nested" proof by contradiction. In his forthcoming textbook, "Computational Complexity, A Conceptual Perspective" [30] , Goldreich gives a "proof sketch" of pong, where he avoids mentioning s_1 and s_2 , and so avoids the need for a nested proof by contradiction. The proof is discussed below. (To clarify, I refer to the proof explored in Chapter 2 that involves s_1 and s_2 as Goldreich's "original" proof of pong, and the version discussed below as Goldreich's "new" proof of pong, even though the discussion below focuses on a specific part of this proof.)

First, let P denote the probability that a uniformly selected n-bit string is an element of S_n . Asserting that

$$\#S_n \quad > \quad \left(1 - \frac{1}{2 \cdot p}\right) \cdot 2^n$$

is equivalent to asserting that P is greater than $1 - 1 / (2 \cdot p)$.

The proof again proceeds from the observation that the probability I inverts $f.x$ is related to the success probability of B' inverting a sequence of uniformly random f images containing $f.x$. Goldreich defines ξ as s_1 , so, adopting a convenient mixture of notation,

$$\xi \quad = \quad \Pr[B' \circ g.U_{t \cdot n} \in g^{-1} \circ g.U_{t \cdot n} \quad \wedge \quad \langle \exists i : 1 \leq i \leq t : U_n^i \notin S_n \rangle] \qquad ,$$

where t is used (by Goldreich) to denote $n \cdot p$. He then presents a shortened version of the calculation of the bound on s_1 , to conclude that

$$(110) \qquad \xi \quad \leq \quad \frac{1}{2 \cdot q} \qquad .$$

So far then, the new proof remains essentially the same as Goldreich's original proof, but in the next step Goldreich avoids introducing and reasoning about s_2 by instead showing that

$$
\begin{array}{ll}
 & \xi \\
\geq & \quad \{\ (41)\ \} \\
 & \Pr[B' \circ g.U_{t \cdot n} \in g^{-1} \circ g.U_{t \cdot n}] \ - \ \Pr[\langle \forall i : 1 \leq i \leq t : U_n^i \in S_n \rangle] \\
\geq & \quad \{\ \text{assumption, independent events}\ \} \\
 & 1 / q \ - \ P^t
\end{array}
$$

Having shown that

$$
(111) \qquad 1 / q \ - \ P^t \quad \leq \quad \xi \quad \leq \quad 1 / (2 \cdot q) \qquad ,
$$

it follows that

$$
\begin{array}{ll}
 & 1 / q \ - \ P^t \quad \leq \quad 1 / (2 \cdot q) \\
\equiv & \quad \{\ \text{arithmetic}\ \} \\
 & 1 / q \ - \ 1 / (2 \cdot q) \quad \leq \quad P^t \\
\equiv & \quad \{\ \text{fractions}\ \} \\
 & 1 / (2 \cdot q) \quad \leq \quad P^t \\
\equiv & \quad \{\ \text{exponents}\ \} \\
 & \left(\frac{1}{2 \cdot q} \right)^{1 / t} \quad \leq \quad P
\end{array}
$$

And so we arrive at a lower bound on P. Goldreich then claims that

$$
P \quad \geq \quad \left(\frac{1}{2 \cdot q} \right)^{1 / t} \qquad \Rightarrow \qquad P \quad > \quad 1 - \frac{1}{2 \cdot p}
$$

for sufficiently large n. He does not justify this claim, but it is easily established by demonstrating that

$$
\frac{1}{2 \cdot q} \quad > \quad \left(1 - \frac{1}{2 \cdot p} \right)^t \qquad .
$$

Since t is defined as $n \cdot p$, we know from the earlier calculation of the bound on s_2, that $(1 \ - \ 1 / (2 \cdot p))^{n \cdot p}$ tends to $1 / e^{n / 2}$, and so is indeed less than $1 / (2 \cdot q)$ for large enough n. Hence,

$$
P \quad > \quad 1 \ - \ \frac{1}{2 \cdot p}
$$

as required.

*

In my restructured version of Goldreich's original proof, I arrived at the definition of a as $2 \cdot t \cdot q$ by virtue of the structure of the argument, and in particular by virtue of (109). Similarly, I defined t as $n \cdot (p+1)$ as the obvious choice to complete the proof of the bound on s_2. Here I explore how we can adopt Goldreich's new proof strategy in such a way that values for a and t again arise naturally.

In terms of the overall strategy, all that's changed is how we establish the bound on the size of S_n, so we can retain my alternative bound on "sufficiently easy" instances as $1 - 1/(p+1)$, and try to calculate values for a and t. We define ξ as above, but rather than establishing (110), we can instead establish that

$$\xi \quad \leq \quad t/a \qquad .$$

That $\xi \geq 1/q - P^t$ remains valid, and so instead of (111), we have

$$1/q - P^t \quad \leq \quad \xi \quad \leq \quad t/a \qquad .$$

Using the same line of reasoning as in Goldreich's new proof, it follows that

$$(1/q - t/a)^{1/t} \quad \leq \quad P \qquad .$$

So, our goal is to show that

$$P \quad \geq \quad (1/q - t/a)^{1/t} \qquad \Rightarrow \qquad P \quad > \quad 1 - \frac{1}{p+1}$$

by establishing that

$$1/q - t/a \quad > \quad \left(1 - \frac{1}{p+1}\right)^t \qquad .$$

If we define t as $n \cdot (p+1)$, then we can rewrite the demonstrandum as

$$1/q - \frac{n \cdot (p+1)}{a} \quad > \quad 1/e^n \qquad ,$$

and so our goal is to find an appropriate definition of a. A sensible choice of a would allow us to simplify the left hand side of the inequality to an expression of the form $1/b$, where b is some polynomial function of n. Hence, the obvious choice is to define a as $2 \cdot n \cdot (p+1) \cdot q$,

so the left side of the inequality reduces to $1 / 2 \cdot q$, which is greater than $1 / e^n$, as required.

* * *

Chapter 8

Conclusions and future work

As stated in the introduction, my goal with this work was to explore why concepts in cryptography are so hard to reason about. I suggested that a large part of the difficulty arises from the non-avoidance of pitfalls such as over-specific and often ambiguous nomenclature, reliance on unstated domain specific knowledge and assumptions, and poorly structured, informal reasoning. The purpose of this work has primarily been to justify these claims, but also to explore whether the difficulties can be resolved, or whether they are inherent to the topic.

In order to justify these claims, in Chapter 2 I introduced and explored two fundamental cryptographic concepts, namely weak and strong one-way functions, and a proof that the existence of weak one-way functions equivales the existence of strong one-way functions, where both the definitions and the proof where taken from [27] .

We saw that the definitions drew on a number of concepts, but contained ambiguities and were syntactically awkward. The proof was by a ping-pong argument, where both parts, but particularly the latter, viz that the existence of weak one-way functions implies the existence of strong one-way functions, relied on domain specific knowledge and assumptions, and were poorly structured with many gaps and ambiguities in the reasoning.

In the subsequent chapters I explored the underlying concepts and the structure of the proof, filled in many of the gaps in the reasoning, and identified a number of underlying assumptions, particularly with respect to complexity theory. I also gave a much simpler proof that the existence of strong one-way functions implies the existence of weak one-way functions, and a much cleaner, restructured version of Goldreich's proof of the converse.

In the next two sections I summarise what has been learnt and what has been achieved in the previous chapters, fill in a few gaps, and identify and question to what extent the remaining difficulties are inherent to the subject and the concepts involved. I first comment on the necessity of the concepts that underly one-way functions, and then on the difficulties identified in the proof that relates them, the improvements made, and how the concepts involved affect the reasoning. In the final section I close with suggestions for future research.

* *

Concepts in cryptography

We have seen that one-way functions draw on concepts from probability theory, asymptotics, and complexity theory. Each of these concepts introduces its own inherent difficulties, and so complicates the reasoning about one-way functions. Therefore, it seems natural to question the necessity of these concepts. That is, we may ask whether the difficulty in reasoning about one-way functions is due to the combination of the underlying concepts: perhaps it is the choice of concepts, or perhaps it is the way the concepts are formulated that affects our ability to reason about one-way functions.

Remark. As stated in the introduction, my original intention was to select a number of proofs from across the board and to try to clean them up. I realise now that that approach was doomed to failure because I simply did not understand the underlying concepts, nor even what the underlying concepts were.
End of Remark.

In the remainder of this section I summarise how and why the concepts used to define one-way functions are in some sense inherent to cryptography; in the next section I reiterate what has been achieved in terms of streamlining Goldreich's proofs, and comment on how the underlying concepts affect our ability to reason about one-way functions.

*

As explained in Chapter 2 , where complexity theory is about quantifying the gap between "easy" and "hard" problems, cryptography is about exploiting it. More specifically, cryptography is about the design of constructions where some of the computations are provably "easy" , but others are provably "hard" . Hence, this view of cryptography is by definition about exploiting (primarily negative) results from complexity theory.

In studies of complexity we introduce abstract models of computation, such as the Turing machine model, that allow us to reason about problem solving without having to worry about irrelevant implementation specific details. The Church-Turing thesis asserts that all "reasonable and general" models of computation are isomorphic, meaning whatever can be computed in one model of computation can be computed in any other model. This means that we can reason about computability using the most appropriate model of computation, and our results will hold in all other models. This can be summarised by saying that the various different models of computation give us alternative, but equivalent interfaces to the notion of "computability" .

Having defined various equivalent models of computation, and hence the notion of computability, we saw that there exist problems (such as the halting problem) that are uncomputable; indeed, we saw that the class of uncomputable problems is vastly larger than the class of computable problems. With that in mind, it's tempting to associate "easy" with being computable, and "hard" with being uncomputable, but as we saw, this makes no sense in the context of one-way functions, since it followed that if a function is computable, then given an element in its range, finding a valid inverse must also be computable: we simply compute the function over

all possible inputs until we find a valid inverse; the algorithm is guaranteed to terminate since an inverse is guaranteed to exist. Consequently, we need quantitative notions of "easy" and "hard" rather than all or nothing assertions; that is, we need a way of measuring the complexity of computations.

*

Within the various models of computation we can introduce cost measures on the complexity of the computations involved. So for example, in the Turing machine model we define "time complexity" as the number of steps the machine takes from start to finish, where each step is assumed to take the same amount of time. We can of course introduce other cost measures, such as "space complexity", but for our purposes time complexity remains the most important cost measure.

To reason about time complexity I introduced the notation $T_A.x$ to denote the number of steps algorithm A takes on input x. A goal in algorithm design is to find functions in $\#x$ that approximate or bound T_A for a given algorithm A. The goal in complexity theory is —roughly speaking— to decide which problems can be solved with bounded resources; so as a simple example, given a bounding function $f.\#x$ and a problem P, we may ask if there exists an algorithm A to solve P, where $T_A.x \leq f.\#x$ for all x.

Having introduced cost measures on the complexity of computations, we can partition the class of computable functions into so-called "complexity classes", where a complexity class comprises a specification of the type of problems in the class, a description of the computational model, a bound on computational resources, and a notion of correctness. So for example, a predicate P is a member of the complexity class $\mathbf{P}$ ("deterministic polynomial-time") if there exists a polynomial p and a deterministic Turing machine A such that

$$\langle \forall x \in \{0,1\}^* :: A.x \equiv P.x \quad \wedge \quad T_A.x \leq p.\#x \rangle \qquad .$$

Although in principle we can define classes with arbitrary bounds on computational resources, to ensure that we can reason about complexity using the most appropriate computational model, much as we can reason about computability using the most appropriate model, we restrict ourselves to bounds that ensure the classes remain invariant under a change of computational model.

The extended Church-Turing thesis asserts that in terms of time complexity, all "reasonable and general" models of computation are polynomially related. Consequently, to ensure complexity classes remain invariant under a change of computational model, we must choose bounds that remain invariant under polynomial transformations. It follows then, that the "simplest" complexity class we can define, contains problems that can be solved in some number of steps polynomial in the size of the input.

We can compare the relative complexity of problems, and in particular we can compare the complexity of groups of problems, by comparing the bounds on complexity classes; that is, we can assert that one class of problems is "easier" or "harder" than another class of problems. However, given arbitrary bound functions $f.n$ and $g.n$, where n denotes the size of the input,

it is often not practical to make assertions about the relationship between f and g for "all" values of n (and hence for all instances of problems) . In particular, in Chapter 4 we saw that many functions behave differently for small values than they do for large values.

Therefore, when comparing bounds, a property might not hold for all instances, but if it holds for "large enough" instances, then in a sense it holds for "almost all" instances, since it holds for infinitely many instances, but fails to hold for only a finite number of instances. Consequently, we use asymptotics to compare the complexity of problems, and hence to distinguish between "easy" problems and "hard" problems. Clearly then, it follows that asymptotics also is inherent to cryptography.

As a simple but significant example, we saw in Chapter 4 that exponential functions grow faster than polynomial functions, so for large enough inputs, polynomial-time problems are "easier" to solve than exponential-time problems; alternatively: for "almost all" instances, exponential-time problems are "harder" to solve than polynomial-time problems; consequently, we consider exponential-time problems to be harder to solve in general than polynomial-time problems.

*

Unfortunately there is a conceptual gap between complexity theory and cryptography. This is primarily because the goals of complexity theory differ from cryptography: the former is about quantifying what can be computed with bounded resources, while the latter is primarily about designing systems where certain computations are provably easy, but others are provably hard, consequently the two topics are studied from different perspectives. In particular, in studies of complexity theory we are usually interested in worst-case behaviour, where attention tends to be restricted to decision problems, but in cryptography we are usually interested in average-case behaviour (with respect to the hard computations) , and we tend to deal with search problems rather than predicates.

More specifically, in studies of complexity theory we tend to regard a problem as being "easy" if it can be solved in polynomial-time, or "hard" if it cannot, but in cryptography "hard" is usually associated with problems that are not solvable in polynomial-time "on average" . Under this definition a problem may have "easy instances" that are solvable in polynomial-time, and "hard instances" that are not solvable in polynomial-time, but there are enough hard instances, or the input distribution is sufficiently biased, that the problem is not solvable in polynomial-time on average.

So why is average-case complexity so important in cryptography? Well, given that applications of cryptography primarily amount to keeping things secret, it would make no sense to use a cryptosystem that was hard only in the worst case, since most of the time we would be able to discover the secret.

To reason about computations being hard on average we must take into account the input distribution, which means that assertions in cryptography are inherently probabilistic. Observe that in the case of one-way functions, the input distribution is not simply assumed to be the uniform distribution, but rather, the distribution induced by the one-way function in question.

That is, given a one-way function f, we consider the input distribution to some inverting algorithm A, where the distribution is induced by applying f to a uniformly selected n-bit string (ie, the distribution is over f's range). Of course, we don't know the specific details of f, so we don't know the actual distribution induced, only the bound on our ability to invert f (in probabilistic polynomial-time) guaranteed by virtue of f being one-way.

*

As mentioned above, in complexity theory "easy" is usually associated with being computable in polynomial-time, but more specifically, depending on our notion of correctness, "easy" tends to be associated either with being computable in deterministic polynomial-time if correctness is a necessity, or being computable in probabilistic polynomial-time if we are willing to accept a small probability of error. In general, we assume that probabilistic algorithms are more efficient than deterministic algorithms; that is, we assume that a wider range of problems can be solved in polynomial-time using probabilistic machines than can be solved in polynomial-time using deterministic machines.

In cryptography, and in particular in the definitions of one-way functions, "easy" is associated with being computable in deterministic polynomial-time when referring to those computations that should be easy. So for example, if f is a one-way function, it should always be easy to correctly compute $f.x$.

However, when it comes to defining "hard", since hard essentially means "not easy", we adopt the more liberal notion of "easy" associated with probabilistic polynomial-time. In particular, we need to take into account the possibility of guessing a valid inverse, so we must weaken our notion of correctness, which —of course— requires us to consider probabilistic algorithms. And so we see again, albeit from a different perspective, that probability is inherent to cryptography.

In Chapter 5 we saw that if we have an algorithm that outputs a correct answer with a non-negligible probability of success, then we can repeat the algorithm a polynomial number of times to ensure the error probability vanishes exponentially, and hence is negligible; that is, we can "amplify" the probability that the algorithm produces a correct answer. However, if we have an algorithm that is correct with only a negligible probability, then we may have to repeat the algorithm an exponential number of times to significantly improve our chances of success (ie, to reduce the error probability to a negligible value).

Consequently, if we associate "easy" with being computable in probabilistic polynomial-time with only a negligible probability of error, then it's natural to associate "hard" with not being computable in probabilistic polynomial-time with negligible probability of error; in other words, computations are "hard" if the probability of success is negligible with respect to probabilistic polynomial-time computations.

Strong one-way functions are indeed defined using this notion of "hard", where we assert that if f is a strongly one-way function, then the probability that f can be inverted on a uniformly selected input is negligible with respect to probabilistic polynomial-time computations. Since weak one-way functions are only "slightly hard to invert", we adopt a weaker definition

of "hard" (viz, "slightly hard") where the probability of failure is non-negligible for some polynomial.

*

So, in the context of cryptography "hard" means the probability of success is negligible with respect to probabilistic polynomial-time computations, but what does this mean in terms of the established complexity classes?

We saw in Chapter 5 that all one-way functions can be inverted in exponential-time by brute-force search, but we were able to make the stronger assertion that one-way functions can be inverted in nondeterministic polynomial-time, by virtue of their being computable in (deterministic) polynomial-time. Since the class of problems solvable in nondeterministic polynomial-time is known to be a proper subset of the class of problems solvable in exponential-time, it follows that in the context of one-way functions we cannot simply associate "hard" with being computable in exponential-time.

As already mentioned, in complexity theory "hard" is usually associated with problems in $\mathbf{FNP} - \mathbf{FP}$, but this also is no good for the purposes of cryptography, since $\mathbf{FNP}$ is defined in terms of worst case behaviour, where many problems in $\mathbf{FNP}$ are known to be easy on average. Consequently, the definition of "hard" used in cryptography corresponds to the subset of problems in $\mathbf{FNP} - \mathbf{FP}$ that are hard to compute on average.

With respect to the probabilistic complexity classes, we associate "easy" with the search variant of $\mathbf{BPP}$, the class of problems that can be solved in probabilistic polynomial-time with a negligible probability of error. So, by associating "hard" with problems where the probability of success is negligible on average, and hence the probability of an error is non-negligible on average, "hard" problems lie outside this variant of $\mathbf{BPP}$ on average. Hence the significance of deciding whether $\mathbf{NP} \subseteq \mathbf{BPP}$, and hence whether $\mathbf{FNP}$ is a subset of the search variant of $\mathbf{BPP}$.

With the above in mind, it seems a little unfair to simply assert (as Goldreich does) that "we associate efficient computations with $\mathbf{BPP}$", and to expect the reader to figure out how this relates to one-way functions in terms of "easy" and "hard" instances.

*

Intriguingly, it is not known whether any one-way functions exist. It should be clear that if $\mathbf{P} = \mathbf{NP}$, and hence $\mathbf{FP} = \mathbf{FNP}$, then one-way functions cannot exist as defined; hence the significance of the $\mathbf{P} = \mathbf{NP}$ question to cryptography. However, a proof that $\mathbf{P} \neq \mathbf{NP}$ need not imply the existence of problems in $\mathbf{NP} - \mathbf{P}$ or $\mathbf{FNP} - \mathbf{FP}$ that are hard on average, and so need not guarantee the existence of one-way functions.

Consequently, if $\mathbf{FP} = \mathbf{FNP}$, then any theory concerning or based on the properties of one-way functions collapses, since our context reduces to $\mathbf{false}$; more specifically, our theorems would still hold, but the theory may be deemed "uninteresting", as described in Chapter 1 (Page 20). Even if $\mathbf{FP} \neq \mathbf{FNP}$, the theory could still collapse if none of the problems in

$\mathbf{FNP} - \mathbf{FP}$ are hard on average. It would seem then, that the requirements for cryptography are quite strong, and based on a number of unproved assumptions about the complexity classes, and the complexity of problem solving in general.

However, it has been suggested that if one-way functions exist, it may be easier to find weak one-way functions than to find strong one-way functions, hence the importance of (4) (Page 27) , and in particular, that we can construct strong one-way functions from weak one-way functions.

* *

Calculation in cryptography

We've seen that there were problems with both of Goldreich's proofs, including unexplained steps, mysterious values, appeals to unstated theorems and domain specific knowledge, incorrect inequalities, and so on, and that both proofs, but particularly the proof of pong, were poorly structured. We also saw that both proofs had calculational elements, but the calculations were hard to follow because of poor notation, lack of white space, and the near complete absence of hints. Particularly disappointing was that no explanation or motivation was given for the strategy used in the proofs.

Although it was possible to overcome many of these problems, a number of difficulties remain. In this section I summarise the various improvements made to Goldreich's proofs, and comment on the remaining difficulties, and to what extent they arise as a consequence of the underlying concepts.

*

In terms of notation, both proofs highlighted that although the predicate

$$A'(f(U_n)) \in f^{-1}(f(U_n))$$

in the one-wayness condition could be improved syntactically (eg, by reducing the number of brackets and judicious use of whitespace) , it is something of a red herring when it comes to calculating, since it is not manipulated; indeed, it simply contributes syntactic noise and so makes proof steps harder to verify.

We also saw how replacing the "standard" $\Pr[Q \mid P]$ notation with a variation of Fokkinga's suggested notation, viz $\langle \mathcal{P}x : P.x : Q.x \rangle$, can help bring improvements in clarity by explicitly identifying which variables are "bound" , in the sense of being associated with a probability distribution, and which are "free" . We saw in Chapter 3 that despite some concerns over the

abuse of quantifier notation, this variant of Fokkinga's notation is a pleasure to calculate with.

*

With respect to ping, I showed that it is possible to rearrange Goldreich's proof to improve readability, and that Feijen's proof format, and in particular the presence of hints combined with better use of notation and white space makes the calculational part of the proof far easier to verify.

The asymptotic element of the definitions of one-way functions introduces two additional quantifiers of the form

$$\langle \exists N :: \langle \forall n : n > N : \ \ldots \ \rangle \rangle \qquad ,$$

but in Goldreich's proofs these quantifiers were left implicit. Although there were no steps in the proof of ping that held only for "large enough" values, the asymptotic element of the definitions caused confusion due to the use of proof by contradiction. In particular, assuming towards a contradiction that g was not weakly one-way had the effect of "flipping" the "for large enough" quantifiers so that they became "for infinitely many" . The "eventually always" and "always eventually" quantifiers can help provide clarity here, as the one is the dual of the other.

Even having restructured the proof and adopted different notation, the resulting proof was still not particularly satisfactory, not least because the construction of g seems contrived, and because of the use of proof by contradiction. But as we've seen, it was possible to avoid these concerns by instead showing that any strong one-way function is by definition a weak one-way function, from which it follows that the existence of strong one-way functions implies the existence of weak one-way functions. However, where Goldreich's "constructive" proof demonstrates not only that weak one-way functions exist if strong one-way functions exist, but also that weak one-way functions (if they exist) need not be strongly one-way, my "non-constructive" proof only establishes the former.

*

Goldreich's proof of pong is clearly more complex than his proof of ping. So for example, there were no mysterious values introduced in the proof of ping (though I've argued that the construction of g was rather mysterious) , the reduction did not involve any notion of amplification, and none of the proof steps held only for "large enough" values. Consequently, the proof of pong was much harder to verify, and indeed to understand, than the proof of ping.

As in the proof of ping, the asymptotic element of the definitions caused difficulties. Again, the combination of leaving the quantifiers implicit and the use of proof by contradiction caused confusion. As mentioned, in this proof there were steps that held only for large enough values. Particularly disappointing was that no justification was given for these proof steps, and that they relied on unstated knowledge, in particular the following theorem:

$$(1 + x \,/\, n)^n \ \sim \ e^x$$

As mentioned in Chapter 4, the lack of a calculational theory of asymptotics inspired the investigation of the $\diamondsuit\!\!\!\!\!\!\square$ and $\boxed{\diamond}$ quantifiers, as described in [7], to try and improve this part of the reasoning. Work is ongoing in this area.

Having explored the underlying concepts, and having verified the various intermediate steps, at the start of Chapter 7 my biggest remaining concerns with the proof of pong were with its general structure, the mysterious values, and —to a lesser extent— the use of "nested" proof by contradiction to establish the bound on S_n.

With respect to the general proof strategy, we saw that despite the lack of explanation, it is well motivated, and actually reasonably intuitive: we cannot reason directly about I's success probability, so we instead show that if g is not strongly one-way, then there are sufficiently many, sufficiently easy elements in f's range, so that by repeating I a polynomial number of times, the average probability of success exceeds $1 - 1/p$, in contradiction to the hypothesis that f is weakly one-way.

With the general strategy understood, it was possible to reorder the proof so that appropriate values (though not necessarily Goldreich's values) for the lower bound on "sufficiently easy", the bound on "sufficiently many", the number of times we need to repeat I, and the number of blocks required as input to g, arise rather more naturally.

In particular, having explored amplification, establishing a bound on "sufficiently easy" presented no problem: for any element we can invert with a non-negligible probability of success, we can repeat the algorithm a polynomial number of times to ensure the error probability vanishes exponentially. The main difficulty was in determining a lower bound on "sufficiently many", but as we saw, the shape of the demonstrandum gave us strong heuristic guidance on how to pick an appropriate bound. Having established bounds on "sufficiently easy" and "sufficiently many", appropriate choices for the remaining values were obvious.

As we have also seen, courtesy of Goldreich, it was possible to adopt a different, arguably simpler strategy, to proving the lower bound on the size of S_n, which avoids the nested proof by contradiction, but where values for t, the number of n-bit blocks used as input to g, and a, the number of times we repeat algorithm I, still arise naturally.

Though considerably improved, the final (ie, restructured) proof of pong is still by contradiction. As it stands, it seems that we are forced to resort to proof by contradiction because we have no other way of reasoning about the class of "all probabilistic polynomial-time algorithms". In particular, the given definitions of the probabilistic complexity classes do not allow us to perform induction over those classes. It remains to be seen whether there is a way out of this dilemma, perhaps by adopting a different interface to the probabilistic complexity classes.

* *

Future work

This work has focused on one-way functions, so an obvious extension would be to investigate related cryptographic concepts in a similar way, looking for common issues, generalisations, and patterns of reasoning. The next natural concepts to explore would be so-called "hardcore predicates", and "pseudorandom generators".

A hardcore predicate of a function f is a polynomial-time computable predicate of x that is hard to compute given $f.x$. A pseudorandom generator is an efficient (deterministic) algorithm that "stretches" a short random input, called a "seed", into a longer sequence that is "computationally indistinguishable" from a uniformly selected (and hence truly random) sequence.

Briefly, the relationship between these concepts is that one-way functions exist if and only if pseudorandom generators exist; if a bijective function has a hardcore predicate then it must be one-way; and, hardcore predicates can be used to construct a pseudorandom generator from any one-way function. The existence of one-way functions follows from the existence of pseudorandom generators since it can be shown that any pseudorandom generator is a one-way function by definition. The converse is considerably harder to prove, primarily because the output of a one-way function need not appear to be random, but in 1999 Håstad, Impagliazzo, Levin, and Luby showed how to construct a pseudorandom generator from any one-way function [36].

I mentioned at the end of Chapter 4 that in our paper "Towards Calculational Asymptotics" [7], Eerke Boiten and I explore the "eventually always" and "always eventually" quantifiers in order to reason calculationally about asymptotics. We anticipate that these quantifiers and their theory will prove useful in the investigation and streamlining of other concepts in cryptography.

*

As we've seen in the proofs of ping and pong, and as explored in Chapter 5, reduction involves a constructive element, namely the construction of an algorithm that uses a (given or postulated) solution to one problem to solve another problem. Unfortunately, these constructions tend to appear as rabbits, where an appropriate construction appears with little or no motivation, and its success probability is subsequently analysed. (In the proof of ping the reduction algorithm was reasonably intuitive, but in the proof of pong the reduction was considerably more complex.)

It would be interesting to explore to what extent we can calculate reduction algorithms with appropriate probabilities; that is, can we construct an appropriate reduction program and its proof of probabilistic correctness hand-in-hand? It may, for example, be possible to incorporate ideas from McIver and Morgan's probabilistic guarded command language [42] to aid this part of the reasoning.

In our paper "Reduction and Refinement" [6], Eerke Boiten and I explore —as the title suggests— the relationship between reduction and refinement, with a view to exploiting the well established and well understood theories of program refinement in order to construct reductions in proofs of assertions about cryptographic constructs. In particular, we show how refinement is

a special case of reduction, and how reduction is an instance of a novel generalisation that we refer to as "refinement with context". This work is ongoing.

*

As a final point, in this study I have concentrated on "human" theorem proving rather than "mechanical" verification, though issues to do with mechanical verification and automated theorem proving were discussed (albeit briefly). However, it should be clear that the desire to clarify the concepts and theorems appealed to, and to move toward syntactic manipulation, can be seen as a step toward mechanical verification, though for the calculationalist, the ultimate future goal must surely lie with derivation rather than verification...

* * *

Appendix A

Transcript of Goldreich's proof

As the chapter title suggests, a transcript of Goldreich's proof of (4) (taken from [27]) follows. The version here differs from the original only in the sense that I have used a different font, page size, and margins; however, as far as possible I have retained the original spacing in mathematical formulae.

* *

$\exists$ weak one-way functions $\Leftarrow$ $\exists$ strong one-way functions

Consider, for example, a one-way function f (which, without loss of generality, is length-preserving). Modify f into a function g so that $g(p,x) = (p, f(x))$ if p starts with $\log_2 |x|$ zeros, and $g(p,x) = (p,x)$ otherwise, where (in both cases) $|p| = |x|$. We claim that g is a weak one-way function (because for all but a $\frac{1}{n}$ fraction of the strings of length $2n$ the function g coincides with the identity function). To prove that g is weakly one-way, we use a "reducibility argument."

Proposition 2.3.1: Let f be a one-way function (even in the weak sense). Then g, constructed earlier, is a weakly one-way function.

Proof: Intuitively, inverting g on inputs on which it does *not* coincide with the identity transformation is related to inverting f. Thus, if g is inverted, on inputs of length $2n$, with probability that is noticeably greater than $1-\frac{1}{n}$, then g must be inverted with noticeable probability on inputs to which g applies f. Therefore, if g is not weakly one-way, then neither is f. The full, straightforward, but tedious proof follows.

Given a probabilistic polynomial-time algorithm B' for inverting g, we construct a probabilistic polynomial-time algorithm A' that inverts f with "related" success probability. Following is the description of algorithm A'. On input y, algorithm A' sets $n \stackrel{\text{def}}{=} |y|$ and $l \stackrel{\text{def}}{=} \log_2 n$, selects p' uniformly in $\{0,1\}^{n-l}$, computes $z \stackrel{\text{def}}{=} B'(0^l, p', y)$, and halts with output of the n-bit suffix of

z. Let S_{2n} denote the sets of all $2n$-bit-long strings that start with $\log_2 n$ zeros (i.e., $S_{2n} \stackrel{\text{def}}{=} \{0^{\log_2 n}\alpha : \alpha \in \{0,1\}^{2n-\log_2 n}\}$). Then, by construction of A' and g, we have

$$\begin{aligned}
&\Pr[A'(f(U_n)) \in f^{-1}(f(U_n))] \\
&\quad \geq \Pr[B'(0^l U_{n-l}, f(U_n)) \in (0^l U_{n-l}, f^{-1}(f(U_n)))] \\
&\quad = \Pr[B'(g(U_{2n})) \in g^{-1}(g(U_{2n})) \,|\, U_{2n} \in S_{2n}] \\
&\quad \geq \frac{\Pr[B'(g(U_{2n})) \in g^{-1}(g(U_{2n}))] - \Pr[U_{2n} \notin S_{2n}]}{\Pr[U_{2n} \in S_{2n}]} \\
&\quad = n \cdot \left(\Pr[B'(g(U_{2n})) \in g^{-1}(g(U_{2n}))] - \left(1 - \frac{1}{n}\right)\right) \\
&\quad = 1 - n \cdot (1 - \Pr[B'(g(U_{2n}) \in g^{-1}(g(U_{2n}))])
\end{aligned}$$

(For the second inequality, we used $\Pr[A \,|\, B] = \frac{\Pr[A \cap B]}{\Pr[B]}$ and $\Pr[A \cap B] \geq \Pr[A] - \Pr[\neg B]$.) It should not come as a surprise that the above expression is meaningful only in case $\Pr[B'(g(U_{2n})) \in g^{-1}(g(U_{2n}))] > 1 - \frac{1}{n}$.

It follows that for every polynomial $p(\cdot)$ and every integer n, if B' inverts g on $g(U_{2n})$ with probability greater than $1 - \frac{1}{p(2n)}$, then A' inverts f on $f(U_n)$ with probability greater than $1 - \frac{n}{p(2n)}$. Hence, if g is not weakly one-way (i.e., for every polynomial $p(\cdot)$ there exist infinitely many m's such that g can be inverted on $g(U_m)$ with probability $\geq 1 - 1/p(m)$), then also f is not weakly one-way (i.e., for every polynomial $q(\cdot)$ there exist infinitely many n's such that f can be inverted on $f(Un)$ with probability $\geq 1 - 1/q(n)$, where $q(n) = p(2n)/n$). This contradicts our hypothesis (that f is weakly one-way).

To summarize, given a probabilistic polynomial-time algorithm that inverts g on $g(U_{2n})$ with success probability $1 - \frac{1}{n} + \alpha(n)$, we obtain a probabilistic polynomial-time algorithm that inverts f on $f(U_n)$ with success probability $n \cdot \alpha(n)$. Thus, since f is (weakly) one-way, $n \cdot \alpha(n) < 1 - (1/q(n))$ must hold for some polynomial q, and so g must be weakly one-way (since each probabilistic polynomial-time algorithm trying to invert g on $g(U_{2n})$ must fail with probability at least $\frac{1}{n} - \alpha(n) > \frac{1}{n \cdot q(n)}$). ■

* *

∃ weak one-way functions ⇒ ∃ strong one-way functions

Let f be a weak one-way function, and let p by the polynomial guaranteed by the definition of a weak one-way function. Namely, every probabilistic polynomial-time algorithm fails to invert f on $f(U_n)$ with probability at least $\frac{1}{p(n)}$. We assume for simplicity that f is length-preserving (i.e. $|f(x)| = |x|$ for all x's). This assumption, which is not really essential, is justified by Proposition 2.2.5. We define a function g as follows:

$$g(x_1, \ldots, x_{t(n)}) \stackrel{\text{def}}{=} f(x_1), \ldots, f(x_{t(n)}) \tag{2.5}$$

where $|x_1| = \cdots = |x_{t(n)}| = n$ and $t(n) \stackrel{\text{def}}{=} n \cdot p(n)$. Namely, the $n^2p(n)$-bit-long input of g is partitioned into $t(n)$ blocks, each of length n, and f is applied to each block.

Clearly, g can be computed in polynomial-time (by an algorithm that breaks the input into blocks and applies f to each block). Furthermore, it is easy to see that inverting g on $g(x_1, \ldots, x_{t(n)})$ requires finding a pre-image to each $f(x_i)$. One may be tempted to deduce that it is also clear that g is a strongly one-way function. A naive argument might proceed by assuming implicitly (with no justification) that the inverting algorithm worked separately on each $f(x_i)$. If that were indeed the case, then the probability that an inverting algorithm could successfully invert all $f(x_i)$ would be at most $(1 - \frac{1}{p(n)})^{n \cdot p(n)} < 2^{-n}$ (which is negligible also as a function of $n^2p(n)$). However, the assumption that an algorithm trying to invert g works independently on each $f(x_i)$ cannot be justified. Hence, a more complex argument is required.

Following is an outline of our proof. The proof that g is strongly one-way proceeds by a contradiction argument. We assume, on the contrary, that g is not strongly one-way; namely, we assume that there exists a polynomial-time algorithm that inverts g with probability that is not negligible. We derive a contradiction by presenting a polynomial-time algorithm that, for infinitely many n's, inverts f on $f(U_n)$ with probability greater than $1 - \frac{1}{p(n)}$ (in contradiction to our hypothesis). The inverting algorithm for f uses the inverting algorithm for g as a subroutine (without assuming anything about the manner in which the latter algorithm operates). (We stress that we do not assume that the g-inverter works in a particular way, but rather use any g-inverter to construct, in a generic way, an f-inverter.) Details follow.

Suppose that g is not strongly one-way. By definition, it follows that there exists a probabilistic polynomial-time algorithm B' and a polynomial $q(\cdot)$ such that for infinitely many m's,

$$\Pr[B'(g(U_m)) \in g^{-1}(g(U_m))] > \frac{1}{q(m)} \tag{2.6}$$

Let us denote by M' the infinite set of integers for which this holds. Let N' denote the infinite set of n's for which $n^2 \cdot p(n) \in M'$ (note that all m's considered are of the form $n^2 \cdot p(n)$, for some integer n).

Using B', we now present a probabilistic polynomial-time algorithm A' for inverting f. On input y (supposedly in the range of f), algorithm A' proceeds by applying the following probabilistic procedure, denoted I, on input y for $a(|y|)$ times, where $a(\cdot)$ is a polynomial that depends on the polynomials p and q (specifically, we set $a(n) \stackrel{\text{def}}{=} 2n^2 \cdot p(n) \cdot q(n^2p(n))$).

Procedure I

Input: y (denote $n \stackrel{\text{def}}{=} |y|$).

For $i = 1$ to $t(n)$ do begin

1. Select uniformly and independently a sequence of strings $x_1, \ldots, x_{t(n)} \in \{0,1\}^n$.
2. Compute $(z_1, \ldots, z_{t(n)}) \leftarrow B'(f(x_1), \ldots, f(x_{i-1}), y, f(x_{i+1}), \ldots, f(x_{t(n)}))$.
3. If $f(z_i) = y$, then halt and output z_i.
(This is considered a *success*).
end

Using Eq. (2.6), we now present a lower bound on the success probability of algorithm A'. To this end we define a set, denoted S_n, that contains all n-bit strings on which the procedure I succeeds with non-negligible probability (specifically, greater than $\frac{n}{a(n)}$). (The probability is taken only over the coin tosses of procedure I.) Namely,

$$S_n \stackrel{\text{def}}{=} \left\{ x : \Pr[I(f(x)) \in f^{-1}(f(x))] > \frac{n}{a(n)} \right\}$$

In the next two claims we shall show that S_n contains all but at most a $\frac{1}{2p(n)}$ fraction of the strings of length $n \in N'$ and that for each string $x \in S_n$ the algorithm A' inverts f on $f(x)$ with probability exponentially close to 1. It will follow that A' inverts f on $f(U_n)$, for $n \in N'$, with probability greater than $1 - \frac{1}{p(n)}$, in contradiction to our hypothesis.

Claim 2.3.2.1: For every $x \in S_n$,

$$\Pr[A'(f(x)) \in f^{-1}(f(x))] > 1 - \frac{1}{2^n}$$

Proof: By definition of the set S_n, the procedure I inverts $f(x)$ with probability at least $\frac{n}{a(n)}$. Algorithm A' merely repeats I for $a(n)$ times, and hence

$$\Pr[A'(f(x)) \notin f^{-1}(f(x))] < \left(1 - \frac{n}{a(n)}\right)^{a(n)} < \frac{1}{2^n}$$

The claim follows. □

Claim 2.3.2.2: For every $n \in N'$,

$$|S_n| > \left(1 - \frac{1}{2p(n)}\right) \cdot 2^n$$

Proof: We assume to the contrary, that $|S_n| \leq (1 - \frac{1}{2p(n)}) \cdot 2^n$. We shall reach a contradiction to Eq. (2.6) (i.e., our hypothesis concerning the success probability of B'). Recall that by this hypothesis (for $n \in N0$),

$$s(n) \stackrel{\text{def}}{=} \Pr[B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)}))] > \frac{1}{q(n^2p(n))} \tag{2.7}$$

Let $U_n^{(1)}, \ldots, U_n^{(n \cdot p(n))}$ denote the n-bit-long blocks in the random variable $U_{n^2p(n)}$ (i.e., these $U_n^{(i)}$'s are independent random variables each uniformly distributed in $\{0,1\}^n$). We partition the event considered in Eq. (2.7) into two disjoint events corresponding to whether or not one of the $U_n^{(i)}$'s resides out of S_n. Intuitively, B' cannot perform well in such a case, since this case corresponds to the success probability of I on pre-images out of S_n. On the other hand, the probability that all $U_n^{(i)}$'s reside in S_n is small. Specifically, we define

$$s_1(n) \stackrel{\text{def}}{=} \Pr[B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)})) \wedge (\exists i \text{ s.t. } U_n^{(i)} \notin S_n)]$$

and

$$s_2(n) \stackrel{\text{def}}{=} \Pr[B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)})) \wedge (\forall i : U_n^{(i)} \in S_n)]$$

Clearly, $s(n) = s_1(n) + s_2(n)$ (as the events considered in the s_i's are disjoint). We derive a contradiction to the lower bound on $s(n)$ (given in Eq. (2.7)) by presenting upper bounds for both $s_1(n)$ and $s_2(n)$ (which sum to less).

First, we present an upper bound on $s_1(n)$. The key observation is that algorithm I inverts f on input $f(x)$ with probability that is related to the success of B' to invert g on a sequence of random f-images containing $f(x)$. Specifically, for every $x \in \{0,1\}^n$ and every $1 \leq i \leq n \cdot p(n)$, the probability that I inverts f on $f(x)$ is greater than or equal to the probability that B' inverts g on $g(U_{n^2p(n)})$ conditioned on $U_n^{(i)} = x$ (since any success of B' to invert g means that f was inverted on the ith block, and thus contributes to the success probability of I). It follows that, for every $x \in \{0,1\}^n$ and every $1 \leq i \leq n \cdot p(n)$,

$$\begin{aligned}&\Pr[I(f(x)) \in f^{-1}(f(x))] \\ &\quad \geq \Pr[B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)})) \,|\, U_n^{(i)} = x] \end{aligned} \tag{2.8}$$

Since for $x \notin S_n$ the left-hand side (l.h.s.) cannot be large, we shall show that (the r.h.s and so) $s_1(n)$ cannot be large. Specifically, using Eq. (2.8), it follows that

$$\begin{aligned}
s_1(n) &= \Pr[\exists i \text{ s.t. } B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)})) \wedge U_n^{(i)} \notin S_n] \\
&\leq \sum_{i=1}^{n\cdot p(n)} \Pr[B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)})) \wedge U_n^{(i)} \notin S_n] \\
&\leq \sum_{i=1}^{n\cdot p(n)} \sum_{x \notin S_n} \Pr[B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)})) \wedge U_n^{(i)} = x] \\
&= \sum_{i=1}^{n\cdot p(n)} \sum_{x \notin S_n} \Pr[U_n^{(i)} = x] \cdot \Pr[B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)})) \,|\, U_n^{(i)} = x] \\
&\leq \sum_{i=1}^{n\cdot p(n)} \max_{x \notin S_n}\{\Pr[B'(g(U_{n^2p(n)})) \in g^{-1}(g(U_{n^2p(n)})) \,|\, U_n^{(i)} = x]\} \\
&\leq \sum_{i=1}^{n\cdot p(n)} \max_{x \notin S_n}\{\Pr[I(f(x)) \in f^{-1}(f(x))]\} \\
&\leq n \cdot p(n) \cdot \frac{n}{a(n)} = \frac{n^2 \cdot p(n)}{a(n)}
\end{aligned}$$

(The last inequality uses the definition of S_n, and the one before it uses Eq. (2.8).)

We now present an upper bound on $s_2(n)$. Recall that by the contradiction hypothesis, $|S_n| \leq (1 - \frac{1}{2p(n)}) \cdot 2^n$. It follows that

$$\begin{aligned}
s_2(n) &\leq \Pr[\forall i : U_n^{(i)} \in S_n] \\
&\leq \left(1 - \frac{1}{2p(n)}\right)^{n\cdot p(n)}
\end{aligned}$$

$$< \frac{1}{2^{n/2}} < \frac{n^2 \cdot p(n)}{a(n)}$$

(The last inequality holds for sufficiently large n.)

Combining the upper bounds on the s_i's, we have $s_1(n)+s_2(n) < \frac{2n^2 \cdot p(n)}{a(n)} = \frac{1}{q(n^2 p(n))}$, where equality is by the definition of $a(n)$. Yet on the other hand, $s_1(n) + s_2(n) > \frac{1}{q(n^2 p(n))}$, where the inequality is due to Eq. (2.7). Contradiction is reached, and the claim follows. □

Combining Claims 2.3.2.1 and 2.3.2.2, we obtain

$$\begin{aligned}
&\Pr[A'(f(U_n)) \in f^{-1}(f(U_n))] \\
&\quad \geq \Pr[A'(f(U_n)) \in f^{-1}(f(U_n)) \wedge U_n \in S_n] \\
&\quad = \Pr[U_n \in S_n] \cdot \Pr[A'(f(U_n)) \in f^{-1}(f(U_n)) \,|\, U_n \in S_n] \\
&\quad \geq \left(1 - \frac{1}{2p(n)}\right) \cdot (1 - 2^{-n}) > 1 - \frac{1}{p(n)}
\end{aligned}$$

It follows that there exists a probabilistic polynomial-time algorithm (i.e., A') that inverts f on $f(U_n)$, for $n \in N'$, with probability greater than $1 - \frac{1}{p(n)}$. This conclusion, which follows from the hypothesis that g is not strongly one-way (i.e., Eq. (2.6)), stands in contradiction to the hypothesis that every probabilistic polynomial-time algorithm fails to invert f with probability at least $\frac{1}{p(n)}$, and the theorem follows. ■

* * *

Appendix B

My version of Goldreich's proof

The chapter title is slightly misleading, as this appendix contains only my restructured version of Goldreich's proof of pong (it does not contain either of my versions of ping). As explained in Chapter 7, while presenting my restructured version of pong, I made numerous references to Goldreich's proof, so for the sake of completeness, and for ease of comparison, a rather more (though not entirely) self-contained version of the restructured proof follows; the astute reader will observe that I have made a few subtle changes, which includes weakening a few inequalities.

* *

Let f be a (length preserving) weak one-way function that operates on n-bit inputs, and let g be a function that is constructed by applying f to the concatenation of a polynomial number of n-bit blocks; specifically,

$$(112) \qquad g.(x_1 \text{ ++ } \ldots \text{ ++ } x_t) \quad = \quad f.x_1 \text{ ++ } \ldots \text{ ++ } f.x_t \qquad ,$$

where t is a polynomial in n, and each block (ie, each x_i) is an n-bit string. Denote by m the size of the input to g as a function of n, so $m := t \cdot n$.

Remark. Since f is guaranteed to have a polynomial-time implementation, we can apply f to a polynomial number of n-bit blocks and the resulting function will also run in polynomial-time. **End of Remark.**

Our goal is to show that g is a strong one-way function for a suitable value of t. That is, if we concatenate enough blocks, we can "amplify" the difficulty of inverting a weak one-way function, so that the resulting function is strongly one-way.

*

Since f is a weak one-way function it satisfies the condition

$$\langle \exists p :: \langle \forall F :: \langle \Diamond\!\!\!\!\!\!\;\;\scriptstyle\square\; n :: \langle \mathcal{P} x \in \{0,1\}^n :: P.x \rangle \ < \ 1 - 1 \,/\, p.n \rangle\rangle\rangle \qquad ,$$

where $P.x$ denotes the predicate "F inverts $f.x$" . (Note that in this chapter all distributions are assumed to be uniform, so in the above x is a uniformly selected n-bit string.) Our goal is to show that g is strongly one-way, and so satisfies the condition

$$\langle \forall q, G :: \langle \Diamond\!\!\!\!\!\!\;\;\scriptstyle\square\; m :: \langle \mathcal{P} x \in \{0,1\}^m :: Q.x \rangle \ < \ 1 \,/\, q.m \rangle\rangle \qquad ,$$

where $Q.x$ denotes the predicate "G inverts $g.x$" .

Aiming for a proof by contradiction, we assume that g is not strongly one-way, and hence that there exists an algorithm G that can invert g with unallowable success probability for some polynomial q ; formally:

$$\langle \exists q, G :: \langle \boxed{\diamond}\, n :: \langle \mathcal{P} x \in \{0,1\}^m :: Q.x \rangle \ \geq \ 1 \,/\, q.m \rangle\rangle$$

Consequently, our goal is to show how we can use q and G to construct an algorithm that inverts f with probability that contradicts f being weakly one-way; ie:

$$\langle \boxed{\diamond}\, n :: \langle \mathcal{P} x \in \{0,1\}^n :: P.x \rangle \ \geq \ 1 - 1 \,/\, p.n \rangle \qquad .$$

Remark. In the remainder of the chapter I drop the n and the m associated with the polynomials p and q in the hope that it is clear that p is a positive polynomial in n and that q is a positive polynomial in m .
End of Remark.

*

As suggested above, our goal is to use G to construct an algorithm, denoted F , to invert f on a uniformly selected n-bit string, but G operates on m-bit inputs, so given an n-bit input we must "pad" it to m-bits by concatenating $t-1$ blocks of n-bits . However, there is no reason to favour concatenating the blocks in any particular order, nor of "fixing" the content of the blocks in advance. With that in mind, let y denote the result of $f.x$ for some uniformly selected n-bit value x , and denote by I the following probabilistic procedure:

- select uniformly and independently a sequence of n-bit strings $x_1 , \ \ldots \ , \ x_t$
- select i uniformly at random so that $1 \leq i \leq t$
- compute $y_1 \text{ ++ } \ldots \text{ ++ } y_t \ := \ G.(f.x_1 \text{ ++ } \ldots \text{ ++ } f.x_{i-1} \text{ ++ } y \text{ ++ } f.x_{i+1} \text{ ++ } \ldots \text{ ++ } f.x_t)$
- if $f.y_i \ = \ y$ then halt and output y_i

Algorithm F simply repeats I some polynomial (to be derived in due course) number of times.

*

It should be clear that F's success probability is inherently tied to I's success probability, but unfortunately it's not clear how to reason directly about the probability of I succeeding on a single repetition, so we instead reason indirectly, by introducing the set of "easy instances", denoted S_n, where algorithm I inverts f with high probability; formally,

$$S_n \quad = \quad \langle x \in \{0,1\}^n :: R.x \geq \alpha \rangle \qquad ,$$

where $R.x$ denotes the probability that I inverts $f.x$, and α characterises our notion of "easy" (or, if you prefer, "high probability"). We then demonstrate that there are sufficiently many easy instances, ie,

$$\#S_n \quad \geq \quad \sigma \cdot 2^n \qquad ,$$

where σ characterises our notion of "sufficiently many", that the "average" probability of success, and hence the probability that F inverts f on a uniformly selected n-bit input, is at least $1 - 1/p$, which contradicts the assumption that f is weakly one-way.

*

Let's deal with α, and hence with "sufficiently easy" first. If an algorithm succeeds with non-negligible probability, repeating the algorithm a polynomial number of times ensures the error probability vanishes exponentially. Consequently, we define "easy" instances as those that we can invert with a non-negligible probability of success, and so can invert with only a negligible probability of error using polynomial amplification.

Having decided that our lower bound on "sufficiently easy" should be non-negligible, what bound should we use? Well, a function is called negligible if for large enough values of n, it grows slower than $1/a.n$ for every positive polynomial a, so a function is non-negligible if it grows at least as fast as $1/a.n$ for some positive polynomial a. So with that in mind, we define α as $1/a$, where a is some positive polynomial in n.

It follows that for those elements where I succeeds with probability at least $1/a$, then I fails with probability at most $1 - 1/a$, so if F repeats I at least $n \cdot a$ times, then provided a is not a constant polynomial, the probability that F fails is at most $1/e^n$, which is negligible; conversely, the probability that F succeeds is at least $1 - 1/e^n$.

So, for completeness, S_n is defined as the set of n-bit strings where algorithm I can invert f with probability at least $1/a$, from which it follows that if F repeats I at least $n \cdot a$ times, then F inverts f on elements in S_n with only a negligible probability of error.

*

Next, let's deal with "sufficiently many". Having established that the probability that F inverts f on elements in S_n is at least $1 - 1/e^n$, and hence that

$$\langle \mathcal{P}x : x \in S_n : P.x \rangle \quad \geq \quad 1 - 1/e^n \qquad ,$$

we need to show that there are sufficiently many easy instances, that the probability of inverting f on a uniformly selected n-bit string is at least $1 - 1/p$, in contradiction to the assumption that f is weakly one-way. First, observe that

$$\#S_n \ \geq \ \sigma \cdot 2^n \qquad \equiv \qquad \langle \mathcal{P}x :: x \in S_n \rangle \ \geq \ \sigma \qquad .$$

Next, observe that we have:

$$\begin{array}{ll} & \langle \mathcal{P}x :: P.x \rangle \\ \geq & \quad \{ \text{ probability } \} \\ & \langle \mathcal{P}x :: P.x \ \wedge \ x \in S_n \rangle \\ = & \quad \{ \text{ probability } \} \\ & \langle \mathcal{P}x :: x \in S_n \rangle \cdot \langle \mathcal{P}x : x \in S_n : P.x \rangle \\ \geq & \quad \{ \text{ above } \} \\ & \sigma \cdot (1 - 1/e^n) \end{array}$$

Consequently, our goal is to find a value of σ such that

$$(113) \qquad \sigma \cdot \left(1 - \frac{1}{e^n}\right) \quad \geq \quad 1 - 1/p \qquad ,$$

and to show that σ is a valid bound on the size of S_n.

We can rewrite (113) as

$$(114) \qquad 1 - \frac{1}{e^n} \quad \geq \quad \frac{1 - 1/p}{\sigma} \qquad ,$$

and we can immediately rule out a couple of values of σ. First, σ cannot be $1 - 1/p$, as that would falsify the inequality. Second, σ cannot be 1, because although that would satisfy the inequality, since σ denotes the proportion of easy instances, if $\sigma = 1$ then every instance must be easy, and hence $S_n = \{0,1\}^n$, which is not a reasonable supposition. It follows then, that σ must be a proper fraction, let's denote it σ_0 / σ_1.

Substituting σ_0 / σ_1 for σ, we can rewrite (114) as

$$(115) \qquad \frac{e^n - 1}{e^n} \quad \geq \quad \frac{p - 1}{p} \cdot \frac{\sigma_1}{\sigma_0} \qquad .$$

Since $\sigma_0 < \sigma_1$, it follows that $\sigma_1 / \sigma_0 > 1$, and hence that

$$\frac{p-1}{p} \cdot \frac{\sigma_1}{\sigma_0} \quad > \quad \frac{p-1}{p} \qquad ,$$

so we must be careful to pick values that preserve (115) ; in particular, σ_1 / σ_0 can be only slightly larger than 1 .

Given the shape of (115) , ideally we'd like to be able to simplify the right side of the inequality to an expression of the form $(b-1)/b$, where b is some polynomial function of p . As mentioned above, setting $\sigma := 1 - 1/p$, and hence setting $\sigma_0 := p-1$ and $\sigma_1 := p$, falsifies (115) . However, it's clear that $p/(p-1)$ meets the requirement of being only slightly greater than 1 , and both the numerator and denominator are polynomial functions of p , so it seems a reasonable starting point for our investigation.

It should be clear that $(p-1)/p$ is strictly monotonic. In particular, as p approaches infinity, $(p-1)/p$ approaches 1 from below, and so $p/(p-1)$ approaches 1 from above. It follows that if we increase p , then $p/(p-1)$ approaches 1 faster than if we decrease p ; so, since we want this ratio to be as close to 1 as possible, it's clear that we should increase p .

Arguably the simplest approach is to add 1 to $p-1$ and p , and so set $\sigma_0 := p$, and $\sigma_1 := p+1$, in which case the right side of (115) becomes

$$\begin{array}{ll} & \frac{p-1}{p} \cdot \frac{p+1}{p} \\ = & \quad \{ \text{ algebra } \} \\ & \frac{p^2-1}{p^2} \end{array}$$

which is less than $(e^n - 1)/e^n$, and so satisfies (115) as required. So, (115) becomes

$$\frac{e^n - 1}{e^n} \quad > \quad \frac{p-1}{p} \cdot \frac{p+1}{p} \qquad ,$$

which is a little stronger than necessary, but that doesn't matter. With a little manipulation we can rewrite the above as

$$\left(1 - \frac{1}{p+1}\right) \cdot \left(1 - \frac{1}{e^n}\right) \quad > \quad 1 - 1/p \qquad ;$$

therefore, $\sigma := 1 - 1/(p+1)$.

*

Having established a candidate bound on "sufficiently many", we need to show that it is valid, by establishing that

$$\#S_n \quad \geq \quad \left(1 - \frac{1}{p+1}\right) \cdot 2^n \qquad .$$

If we let ω denote the probability that a uniformly selected n-bit string is an element of S_n, then our goal is to show that ω is at least $1 - 1/(p+1)$. Observe that we haven't yet had to define t or a.

First, observe that selecting t strings uniformly and independently at random from $\{0,1\}^n$, is equivalent to selecting a single string uniformly at random from $\{0,1\}^{t \cdot n}$. Consequently, in the predicate Q —viz, "G inverts $g.x$", where x is a uniformly selected m-bit string— algorithm G is applied to an input of the form

$$g.(x_1 \texttt{ ++ } \ldots \texttt{ ++ } x_t) \qquad ,$$

as per (112), but where each x_i is selected uniformly and independently at random from $\{0,1\}^n$.

Remark. Let it be understood that in the remainder of the proof, the dummy i is associated with the range $1 \leq i \leq t$, the dummies $\hat{x}$ and x are respectively n-bit and m-bit strings, and that x_i denotes the ith n-bit block of x.
End of Remark.

Next, observe that G succeeds with high probability if each x_i is an element of S_n, and with low probability if any x_i lies outside S_n, since this corresponds to the success probability of algorithm I on instances outside S_n. Focusing on the latter case, we introduce β defined as follows:

$$\beta \quad = \quad \langle \mathcal{P}x :: Q.x \ \wedge \ \langle \exists i :: x_i \notin S_n \rangle \rangle \qquad ,$$

First, we derive a lower bound on β. The calculation proceeds from the observation that the probability I inverts $f.\hat{x}$ is related to the success probability of G inverting a sequence of uniformly random f images containing $f.\hat{x}$; in particular, we have:

$$(116) \qquad R.\hat{x} \quad \geq \quad \langle \mathcal{P}x : x_i = \hat{x} : Q.x \rangle$$

The calculation follows.

$$
\begin{array}{ll}
 & \beta \\
= & \quad \{ \text{ definition of } \beta \text{ and } \wedge \text{ over } \exists \} \\
 & \langle \mathcal{P}x :: \langle \exists i :: Q.x \ \wedge \ x_i \notin S_n \rangle \rangle \\
= & \quad \{ \text{ probability } \} \\
 & \langle \Sigma i :: \langle \mathcal{P}x :: Q.x \ \wedge \ x_i \notin S_n \rangle \rangle \\
\leq & \quad \{ \text{ proved elsewhere } \} \\
 & \langle \Sigma i :: \langle \Sigma \hat{x} \notin S_n :: \langle \mathcal{P}x :: Q.x \ \wedge \ x_i = \hat{x} \rangle \rangle \rangle \\
= & \quad \{ \text{ probability } \} \\
 & \langle \Sigma i :: \langle \Sigma \hat{x} \notin S_n :: \langle \mathcal{P}x :: x_i = \hat{x} \rangle \cdot \langle \mathcal{P}x : x_i = \hat{x} : Q.x \rangle \rangle \rangle \\
\leq & \quad \{ \text{ proved elsewhere } \} \\
 & \langle \Sigma i :: \langle \uparrow \hat{x} \notin S_n :: \langle \mathcal{P}x : x_i = \hat{x} : Q.x \rangle \rangle \rangle \\
\leq & \quad \{ (116) \} \\
 & \langle \Sigma i :: \langle \uparrow \hat{x} \notin S_n :: R.\hat{x} \rangle \rangle \\
\leq & \quad \{ \text{ definition of } S_n \text{ and arithmetic } \} \\
 & t \cdot (1 \,/\, a)
\end{array}
$$

Hence we have $\beta \ \leq \ t \,/\, a$. Next, we derive an upper bound on β as follows:

$$
\begin{array}{ll}
 & \beta \\
\geq & \quad \{ \text{ definition of } \beta \text{ and probability } \} \\
 & \langle \mathcal{P}x :: Q.x \rangle \ - \ \langle \mathcal{P}x :: \langle \forall i :: x_i \in S_n \rangle \rangle \\
\geq & \quad \{ \text{ assumption, independent events } \} \\
 & 1 \,/\, q \ - \ \omega^t
\end{array}
$$

Consequently we have

$$
1 \,/\, q \ - \ \omega^t \quad \leq \quad \beta \quad \leq \quad t \,/\, a \qquad ,
$$

from which it follows that:

$$
\begin{array}{ll}
 & 1 \,/\, q \ - \ \omega^t \quad \leq \quad t \,/\, a \\
\equiv & \quad \{ \text{ arithmetic } \} \\
 & 1 \,/\, q \ - \ t \,/\, a \quad \leq \quad \omega^t \\
\equiv & \quad \{ \text{ exponents } \} \\
 & (1 \,/\, q \ - \ t \,/\, a)^{1 \,/\, t} \quad \leq \quad \omega
\end{array}
$$

And so we arrive at a lower bound on ω, which we can use to show that

$$\omega \quad \geq \quad (1/q - t/a)^{1/t} \qquad \Rightarrow \qquad \omega \quad \geq \quad 1 - \frac{1}{p+1}$$

by establishing that

$$1/q - t/a \quad \geq \quad \left(1 - \frac{1}{p+1}\right)^t \qquad .$$

If we define t as $n \cdot (p+1)$, then we can rewrite the demonstrandum as

$$1/q - \frac{n \cdot (p+1)}{a} \quad \geq \quad 1/e^n \qquad ,$$

and so our goal is to find an appropriate definition of a. A sensible choice of a would allow us to simplify the left hand side of the inequality to an expression of the form $1/b$, where b is some polynomial function of n. With that in mind, the obvious choice is to define a as $2 \cdot n \cdot (p+1) \cdot q$, so the left side of the inequality reduces to $1/2 \cdot q$, which is greater than $1/e^n$ for large enough n, and so satisfies the inequality; hence, we have

$$\omega \quad > \quad 1 - \frac{1}{p+1} \qquad ,$$

which, again, is a little stronger than is strictly necessary.

Since ω, and hence the probability that a uniformly selected n-bit string is an element of S_n, is greater than $1 - 1/(p+1)$, and as F inverts f on instances in S_n with probability at least $1 - 1/e^n$, it follows that F inverts f on a uniformly selected n-bit string with probability greater than $1/p.n$, in contradiction to f being weakly one-way. So by contradiction, g must be strongly one-way.

* * *

References

[0] Wilhelm Ackermann. On Hilbert's Construction of the Real Numbers. Appears in [49]; original (in German) appears as [1].

[1] Wilhelm Ackermann. Zum Hilbertschen Aufbau der reellen Zahlen. *Mathematische Annalen*, 99:118–133, 1928. English translation by Stefan Bauer-Mengelberg appears as [0] in [49].

[2] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004.

[3] Laszlo Babai. Monte-Carlo algorithms in graph isomorphism testing. Technical Report D.M.S 79-10, Université de Montreal, 1979.

[4] Roland Backhouse. *Program Construction: Calculating Implementations from Specifications*. John Wiley & Sons Inc., 2003.

[5] E.T. Bell. *The Development of Mathematics*. McGraw-Hill, 1945.

[6] E.A. Boiten and D.C. Grundy. Reduction and refinement. In E.A. Boiten, J. Derrick, and G. Smith, editors, *REFINE 2007: the 11th BCS-FACS Refinement Workshop*, 2007. Appears in *Electronic Notes in Theoretical Computer Science (ENTCS)*, 201(C):31–44, 2008.

[7] Eerke Boiten and Dan Grundy. Towards calculational asymptotics, 2008. Submitted for publication.

[8] D. Boneh and R. Venkatesan. Breaking RSA may not be equivalent to factoring. In *Advances in Cryptology — Eurocrypt 1998 Proceedings*, volume 1233 of *Lecture Notes in Computer Science (LNCS)*, pages 59–71. Springer-Verlag, 1998.

[9] Georg Cantor. Über eine Eigenschaft des Inbegriffes aller reelen algebraischen Zahlen. *Journal für die Reine und Angewandte Mathematik*, 77:258–262, 1874.

[10] Paul J. Cohen. The independence of the continuum hypothesis. *Proceedings of the National Academy of Sciences of the United States of America*, 50(6):1143–1148, 1963.

[11] Stephen Cook. The importance of the P versus NP question. *Journal of the ACM*, 50(1):27–29, 2003. 50th Anniversary Issue.

[12] Edsger W. Dijkstra. *A Discipline of Programming*. Prentice Hall, 1976.

[13] Edsger W. Dijkstra. EWD655 Essays on the nature and role of mathematical elegance (3): On notation (a sequel to EWD619), 1978.
Available from `http://www.cs.utexas.edu/users/EWD/ewd06xx/EWD655.PDF`
(last checked February 2008).

[14] Edsger W. Dijkstra. EWD999 Our proof format, 1987. Early draft of Chapter 4 of [17].
Available from `http://www.cs.utexas.edu/users/EWD/ewd09xx/EWD999.PDF`
(last checked February 2008).

[15] Edsger W. Dijkstra. EWD1277 Society's role in mathematics, 1998.
Available from `http://www.cs.utexas.edu/users/EWD/ewd12xx/EWD1277.PDF`
(last checked February 2008).

[16] Edsger W. Dijkstra. EWD1300 The notational conventions I adopted, and why, 2000.
Available from `http://www.cs.utexas.edu/users/EWD/ewd13xx/EWD1300.PDF`
(last checked February 2008).

[17] Edsger W. Dijkstra and Carel S. Scholten. *Predicate Calculus and Program Semantics*. Springer-Verlag, 1990.

[18] Rutger M. Dijkstra. "Everywhere" in predicate algebra and modal logic. *Information Processing Letters*, 58(5):237–243, 1996.

[19] W.H.J. Feijen and A.J.M. van Gasteren. *On a Method of Multiprogramming*. Springer-Verlag, 1999.

[20] William Feller. *An Introduction to Probability Theory and Its Applications, Volume I, Third Edition*. John Wiley & Sons Inc., 1968.

[21] Maarten M. Fokkinga. Z-style notation for Probabilities, January 2004. Unregistered Technical Note. University of Twente, Enschede, Netherlands.

[22] Aviezri S. Fraenkel and David Lichtenstein. Computing a Perfect Strategy for $n \times n$ Chess Requires Time Exponential in n. *J. Comb. Theory, Ser. A*, 31(2):199–214, 1981.

[23] Gottlob Frege. Begriffsschrift, a formula language, modeled upon that of arithmetic, for pure thought. Appears in [49]; original (in German) appears as [24].

[24] Gottlob Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Halle, 1879. English translation by Stefan Bauer-Mengelberg appears as [23] in [49].

[25] William I. Gasarch. Guest column: The P=?NP poll. *ACM SIGACT News Complexity Theory Column 36*, 33(2):34–47, 2002.

[26] Kurt Gödel. *The Consistency of the Continuum Hypothesis*. Princeton University Press, 1940.

[27] Oded Goldreich. *Foundations of Cryptography: Volume I Basic Tools*. Cambridge University Press, 2001.

[28] Oded Goldreich. *Foundations of Cryptography: Volume II Basic Applications*. Cambridge University Press, 2004.

[29] Oded Goldreich. (Planned) Speech for a Book Party at Radcliffe Institute, 2004. http://www.wisdom.weizmann.ac.il/~oded/PIC/bookparty.html (last checked October 2007).

[30] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008. Details and drafts available online from http://www.wisdom.weizmann.ac.il/~oded/cc-book.html (last checked November 2007).

[31] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, 1989.

[32] David Gries and Fred B. Schneider. *A Logical Approach to Discrete Math*. Springer-Verlag, 1994.

[33] Shai Halevi. A plausible approach to computer-aided cryptographic proofs. Cryptology ePrint Archive, Report 2005/181, 2005. http://eprint.iacr.org/.

[34] G.H. Hardy. *A Course of Pure Mathematics (Tenth Edition)*. Cambridge University Press, 1952.

[35] Juris Hartmanis and Richard Stearns. On the computational complexity of algorithms. *Trans. Amer. Math. Soc*, 117(5):285–306, 1965.

[36] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[37] Paul Hoffman. *The man who loved only numbers*. Fourth Estate, 1999.

[38] Anne Kaldewaij. *Programming: The Derivation of Algorithms*. Prentice Hall, 1990.

[39] Donald E. Knuth. Big Omicron and Big Omega and Big Theta. *ACM SIGACT News*, 8(2):18–23, 1976.

[40] W. McCune. Solution of the Robbins Problem. *Journal of Automated Reasoning*, 19(3):263–276, 1997.

[41] Bernard M. Moret. *The Theory of Computation*. Addison-Wesley, 1997.

[42] Annabelle McIver and Carroll Morgan. *Abstraction, refinement and proof for probabilistic systems*. Springer-Verlag, 2005.

[43] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[44] Michael O. Rabin and Dana S. Scott. Finite automata and their decision problem. *IBM Journal of Research and Development*, 3(2):114–125, 1959.

[45] R.L. Rivest, A. Shamir, and L.M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[46] William P. Thurston. On proof and progress in mathematics. *Bulletin of the AMS*, 30(2):161–177, 1994.

[47] S.M. Ulam. *Adventures of a Mathematician*. Scribner, 1976.

[48] A.J.M. van Gasteren. *On the Shape of Mathematical Arguments*. LNCS 445. Springer-Verlag, 1990. Originally appeared as a PhD thesis.

[49] Jean van Heijenoort. *From Frege to Gödel*. Harvard University Press, 1977.

[50] Jeremy Weissmann. JAW61 How I understand context and type information, 2006. Available from `http://www.mathmeth.com/jaw/main/jaws/jaw61.pdf` (last checked February 2008).

* * *