Primality Testing is Polynomial-time: A Mechanised Verification of the AKS Algorithm

Hing Lun Chan

A thesis submitted for the degree of
Doctor of Philosophy in Computer Science at
The Australian National University

October 2019

This document was produced using TEX, LATEXand BIBTEX

I declare that this thesis is entirely my own work. Although it contains materials from my published work with my supervisor, to the best of my knowledge it does not contain any materials previously published or written by another person except where otherwise indicated.

Hing Lun Chan
19 October 2019

To my beloved sweetheart, Jantsen.

Acknowledgments

The completion of this PhD project fulfills my personal dream. It is an achievement built upon extensive help from people all around. Looking back, it took about eight years to finish, and I have been offered every assistance along the way. The list to say thanks is long, but I shall start with my family.

First and foremost, I express my heartfelt thanks to my wife Jantsen. As a mature student, I have my fair share of health issues, family issues, and other age-related issues. She is always the one showing her support with care and patience. I feel confident and comfortable in her presence. She is my angel, and this thesis is dedicated to her.

Without the patient guidance of Michael Norrish, my chief supervisor, this work could not have seen the light at the end of the tunnel. He offers critical advice, accurate judgements, and friendly support. He can tolerate my mistakes, always revealing a way forward when I get stuck. We met regularly on a fortnightly basis to discuss the work progress. The fond memories of each session will last for a lifetime.

Besides being a friend, Michael is also the maintainer of the HOL4 theorem prover. He can guide my skills to take full advantage of the system, and answer all my technical queries with ease. The continual improvement of the HOL4 system is much appreciated, as new features streamline the conversion of proof ideas into scripts. Such convenience in a theorem proving environment deserves many thanks.

I would like to express my gratitude to Peter Baumgartner and Jeremy Dawson, my faithful supervisors, and my superiors Rajeev Gore, Dirk Pattinson, Ranald Clouston and Alwen Tiu. They all show great interest in my work. Peter and Jeremy were keen to enquire about my progress. Rajeev enjoyed my talks and invited me to give guest lectures in his course. Dirk provided me with valuable feedback, Ranald offered opportunities for me to share my work to a wide audience, and Alwen gave me encouragement.

The Australian National University is an excellent institution for intellectual work. I have been given unconditional help from the higher degree research unit, with both administrative support and research training. Many thanks to Elspeth Davies for assisting with my overseas travels, Inger Mewburn for giving valuable advice on my Visualise Your Thesis competition, Candida Spence for kindly providing feedback about my presentation content, and Marie-Claire Miličević for resolving many issues related to my higher degree study.

My thanks extend to Yiming Xu, an ANU undergraduate student doing her research project using my libraries on groups and subgroups. Her excellent work, completed within the short allocated time, shows the general applicability of the libraries I have developed.

Thanks to the many editors of the Journal of Automated Reasoning (JAR), including Jasmin

Blanchette, Stephan Merz, Jeremy Avigad, and Lawrence Paulson. Larry showed great interest in my early PhD work, and encouraged me to complete the milestone.

Also thanks to Candida Spence of the Information Literacy team at the ANU Library for checking the formatting of this thesis. Numerous errors had been pointed out by anonymous examiners in detailed comments; they have been corrected. The readability of this thesis has been greatly improved by various people, including my wife, Michael Norrish, Jeremy Dawson, and other anonymous reviewers. I am responsible for any remaining mistakes.

Throughout my work, I learn from the masters. I would like to thank those who pointed me in the right direction. John Harrison showed me how to formulate the AKS Main Theorem. Terence Tao clarified by email a difficult point in his AKS webpage. Laurent Théry was fascinated by one of my conference talks that, by the next day, he showed me his scripts reproducing what I had talked about the day before.

Thanks to the AKS team for providing the topic of my research, and bringing to the world such ground-breaking insight into primality testing. I am deeply impressed with the Gödel Prize lecture by Manindra Agrawal, and his many talks sketching the history of the AKS algorithm.

Thanks also to the various authors who offer me well-written textbooks or articles on algebra, especially finite fields, some with lucid expositions of the AKS algorithm. The ideas for the proofs in this thesis are taken from various sources. They are acknowledged in a footnote at the start of the proof, to enhance readability.

Lastly, I would like to thank the world of mathematics, populated by many highly gifted mathematicians. I read broadly, and learn many excellent ideas and techniques through their work. That world is a world of dreams, with wonderful colors one cannot see, and delightful music one cannot hear. The experience is aptly expressed in the following quote:

In the broad light of day
mathematicians check their equations and their proofs,
leaving no stone unturned in their search for rigour.
But, at night, under the full moon, they dream,
they float among the stars
and wonder at the miracle of the heavens.
They are inspired.
Without dreams there is no art,
no mathematics, no life.
— Michael Atiyah[1]

[1] In “Dreams”, from “The Unravelers”, a book produced by the IHES.

Abstract

We present a formalisation of the Agrawal-Kayal-Saxena (AKS) algorithm, a deterministic polynomial-time primality test. This algorithm was first announced by the AKS team in 2002, later improved in 2004. Our work is based on the improved version, with Parts 1 and 2 aim at a formal proof of the correctness of the algorithm, and Part 3 aims at a formal analysis of the complexity of the algorithm. The entire work is carried out in the HOL4 theorem prover.

The correctness of the AKS algorithm relies on a main theorem developed by the AKS team, based on the theory of finite fields. To achieve the goal for Parts 1 and 2, we start by building up a hierarchy of HOL4 libraries for algebraic structures: from monoids, to groups, then rings and fields. Equipped with this foundation, we develop an abstract algebra library covering subgroups, quotient groups, ideals, and vector spaces. We extend the algebra library with polynomials, quotient rings, quotient fields, and finite fields. With all these we can formulate the AKS main theorem, which gives the correctness of the algorithm. For the formal proof, we need to dive into several advanced topics in finite field, in particular the existence and uniqueness of finite fields, and properties of cyclotomic polynomials.

Although algebraic structures, including finite fields, have been formalised in other theorem provers, our work is the first such comprehensive library in HOL4, covering also the uniqueness of finite fields up to isomorphism. Furthermore, by casting the AKS main theorem in the context of finite fields, we can see clearly the inter-relationship of various parts of the proof. As a result, we can make slight adjustments to the published version of the AKS algorithm. These slight adjustments are minor in terms of the significance of the AKS achievement, answering the challenge "Is Primes in P?" in the affirmative, but they simplify the implementation and analysis of the AKS algorithm.

The AKS algorithm consists of several loops: loops for checking if a condition still holds, and loops for searching if a condition will hold. Thus for the goal of Part 3, we embark on an analysis of such loops: formalising their behaviour, in particular the bound on the number of iterations. The AKS algorithm mostly involves modular computations, using numbers or manipulating polynomials. We develop tools and techniques to formally assert the recurrence properties of loop computations, with emphasis on the analysis of the time complexity behaviour. As far as we know, this approach to complexity analysis has not been done in other theorem provers.

Many offshoots from this work are interesting, even new to published proofs of the AKS algorithm. We have an elegant proof to a key result that enables us to slightly improve the bound on the AKS parameter. We present the relationship between the AKS algorithm and the AKS main theorem. We distill a picture to visualise the logic behind the proof of the AKS main theorem. We show in detail an implementation of the AKS algorithm that is suitable for loop analysis of complexity. We introduce an approach to study the time complexity of simple loops.

Contents

List of Figures

List of Tables

List of Algorithms

Publications

Parts of this thesis have been published as the papers listed below. Some ideas in Chapter 3 and Chapter 5, on the correctness of the AKS algorithm, first appear in the second paper. Some parts of Chapter 2 on basic algebra come from the first and third papers. Chapter 4 on advanced algebra is based on the fourth paper.

- ◇ Hing Lun Chan and Michael Norrish. A String of Pearls: Proofs of Fermat's Little Theorem. In Chris Hawblitzel and Dale Miller, editors, *Proceedings of Certified Programs and Proofs, 2012*. LNCS number 7679, pages 188—207. Springer, December 2012. Print ISBN: 978-3-642-35307-9, doi: 10.1007/978-3-642-35308-6_16. Also published in Andrea Asperti, editor, *Journal of Formalized Reasoning*. Volume 6, number 1, pages 63—87. December 2013. ISSN 1972-5787, doi: 10.6092/issn.1972-5787/3728.

- ◇ Hing Lun Chan and Michael Norrish. Mechanisation of AKS Algorithm: Part 1 — the Main Theorem. In Christian Urban and Xingyuan Zhang, editors, *Interactive Theorem Proving, ITP 2015. 6th International Conference, Nanjing, China, August 24-27, 2015, Proceedings.* LNCS number 9236, pages 117—136. Springer, August 2015. First Online 19 August 2015, doi: 10.1007/978-3-319-22102-1_8.

- ◇ Hing Lun Chan and Michael Norrish. Proof Pearl: Bounding Least Common Multiples with Triangles. In Jasmin Christian Blanchette and Stephan Merz, editors, *Interactive Theorem Proving, ITP 2016. 7th International Conference, Nancy, France, August 22-25, 2016, Proceedings.* LNCS number 9807, pages 140—150. Springer, August 2016. First Online 07 August 2016, doi: 10.1007/978-3-319-43144-4_9. Also published in *Journal of Automated Reasoning*, Springer Netherlands. First online 14 October 2017, doi: 10.1007/s10817-017-9438-0. Printed in February 2019, Volume 62, Issue 2, pages 171—192.

- ◇ Hing Lun Chan and Michael Norrish. Classification of Finite Fields with Applications. In *Journal of Automated Reasoning*, Springer Netherlands. First online 25 October 2018, doi: 10.1007/s10817-018-9485-1. Printed in October 2019, Volume 63, Issue 3, pages 667—693.

Chapter 1

Introduction

This thesis is about a formal proof of the Agrawal-Kayal-Saxena (AKS) algorithm in the theorem-prover HOL4. This is based on a formal definition of the AKS algorithm, with a proof of its correctness. This is followed by a formal implementation of the AKS algorithm, with a proof of its computational complexity based on a machine model. We identify 3 *phases for the AKS algorithm, highlighting the role of a critical parameter* k*, and its effect on performance. We touch on the impact of the AKS algorithm, and why its formalisation is significant. We discuss what has been done, what we have achieved, and the layout of this thesis.*

> If you can't explain your mathematics to a machine,
> it is an illusion to think you can explain it to a student.
> — Nicolaas Govert de Bruijn (2003)[1]

1.1 Formalisation

To formalise is to understand, in detail: explain the logic to a machine, as de Bruijn proclaims.

There are many levels to understand a mathematical proof. Take the example of the AKS algorithm, the theme topic of this thesis. The AKS team presented their proof in nine pages (Agrawal et al. [2002, 2004]). Expositions of various lengths and depths have been written (Bernstein [2002]; Aaronson [2003]; Saptharishi [2007]), whole chapters have been devoted to this topic (Crandall and Pomerance [2005]; Shoup [2008]; Rempe-Gillen and Waldecker [2014]), even a whole book (Dietzfelbinger [2004]) has been published. Nonetheless, to the Fields medalist Terence Tao [2009], the essence of the proof can be understood within a single webpage.

A formalisation, with proof scripts to be compiled by a theorem-prover, provides yet another level of understanding, one that is machine-checkable. The formalisation process reveals the dependency of various concepts, identifies their intricate relationships, and ultimately unfolds the logical threads that lead to the validity of the result.

For the formalisation of the AKS algorithm, we are keen to understand:

[1] From his invited lecture titled *Memories of the Automath Project*. For a delightful discussion of this quote, see *Zen and the art of formalization* by Asperti and Avigad [2011].

1. What is the AKS algorithm?
2. Is the AKS algorithm correct?
3. How to implement the AKS algorithm?
4. What is the run-time behaviour of the AKS algorithm?

The AKS algorithm is about primes, a major topic in number theory. The first involves some concepts in modular arithmetic, using numbers and polynomials. The second involves some knowledge of abstract algebra, in particular the theory of finite fields. The third involves an appreciation of machine execution, with some understanding of subroutines. The fourth involves a model to execute an algorithm, and techniques to solve recurrence relations.

Our formalisation therefore touches on a potpourri of topics in mathematics. However, all topics lead back to the AKS algorithm. For a peek at these topics, see Figure 1.1.

1.2 PRIMES is in P

Given a number n greater than 1, a primality test is a method to determine if n is *prime*, *i.e.*, whether n has only the trivial factors 1 and itself.[2]

The primality test takes the form of an algorithm: a step-by-step method with input n and output the verdict: n is prime or not. The run-time behaviour of an algorithm is an estimate of the number of steps from input to output. Intuitively, the larger the input number, the more the number of steps. To formalise this idea, we need at least a notion of the input size.

It is customary to measure the size of input n by $(\log\ n)$, the base 2 logarithm of n, which is indicative of its number of binary digits. Throughout this work, we shall use instead $\lceil \log\ n \rceil$, the round-up value, which is defined[3] for all values of n. An algorithm with input n and its number of steps bounded by a polynomial function of $\lceil \log\ n \rceil$ belongs to class P, the class of polynomial-time algorithms. Compare to the class of exponential-time algorithms, such class P algorithms are considered efficient, at least in theory.

The name PRIMES refers to the class of primality test algorithms. Since internet security protocols make use of primes with many digits to generate keys, an efficient primality test is keenly sought after. For a long time, only probabilistic primality tests in class P are known, but deterministic primality tests are theoretically more desirable. A deterministic and efficient primality test remains elusive, and the challenge to find one is known as "Is PRIMES in P?".

On August 4, 2002, the AKS team announced their algorithm in a paper (Agrawal et al. [2002]) with the title "PRIMES is in P". This immediately caused a sensation throughout the computer science community, even making news headlines in the popular press.[4] After careful analysis by experts, the AKS algorithm was substantially refined in Agrawal et al. [2004]. The

[2] Note that 0 is never a prime, and 1 is a unit, neither prime nor composite.

[3] We define $\lceil \log\ 0 \rceil = 0$, while $\log\ 0$ is undefined.

[4] As reported in The New York Times [Robinson, 2002, August 8, 2002], and also in Notices of The American Mathematical Society [Bornemann, 2003, May 2003].

breakthrough was officially recognized when the AKS team was awarded the Gödel Prize by the European Association for Theoretical Computer Science EATCS [2006]:

> *The result obtained by Agrawal, Kayal, and Saxena can be seen as a crowning achievement of a long algorithmic and mathematical quest.*
>
> *A remarkable aspect of the article is that the final exposition itself turns out to be rather simple. The text as published in Annals of Mathematics is a masterpiece in mathematical reasoning. It has a high density of tricks and techniques, but the arguments come in a brilliantly simple manner; they remain completely elementary. The contents of the paper are therefore accessible to a broad audience.*

The revised version shows that the AKS algorithm can determine whether a number n is prime with the number of steps bounded by some polynomial function of $\lceil \log n \rceil^{\frac{15}{2}}$. The rather high polynomial exponent means the AKS algorithm cannot compete with the known probablilistic primality tests.

Nevertheless, the AKS algorithm remains the only known unconditional, deterministic and polynomial-time primality test. We shall refer to the first 2002 paper as the *original* AKS paper, and the later 2004 paper as the *revised* AKS paper, or simply the AKS paper.

1.3 AKS Phases

The AKS algorithm consists of 3 phases:

	AKS Algorithm *Input:* a number n *Output:* decide whether n is `PRIME`	
Phase 1	Power Free Test	the number n is not a square, not a cube, *etc.* if n is not power free, n is not `PRIME`
Phase 2	Parameter Search	the parameter k is chosen with some criteria based on n if k is a factor of n, n is `PRIME` only when $k = n$
Phase 3	Polynomial Checks	identities to hold under two moduli involving n and k. if n passes all polynomial identity checks, n is `PRIME`

Table 1.1: Phases of the AKS algorithm.

Phase 1 is a preliminary step. This is because the theory behind the AKS algorithm, to be presented in Parts I and II, shall declare that, if the input n passes the checks of Phase 3 from a good parameter k of Phase 2, then $n = p^e$ for some prime p and exponent e. With Phase 1 ensuring that n is power free, the exponent $e = 1$. Therefore, n is prime.

For Phases 2 and 3, Table 1.1 shows the appearance of a parameter k. As we shall see, this AKS parameter k is central to the AKS algorithm. Its existence through a search comprises the whole of Phase 2. The search, denoted by aks_param n, is sequential. The search can give 3 possible results:

1. a nice k, where k is a factor of n,

2. a good k, where k is not a factor of n, and satisfies a special condition,

3. a bad situation, where the search, up to some cut-off, fails to find a nice k or a good k.

We shall spell out details of the search (Section 3.3.1), and prove that the search never fails.

If Phase 2 returns a parameter of the type nice k, k is a factor of n. Therefore n is prime if and only if this factor is itself, *i.e.*, $n = k$. On the other hand, if a parameter of the type good k is returned, this parameter k is used to drive Phase 3, which consists of a series of polynomial identity checks of the form:

$$(\mathsf{X} + \boldsymbol{c})^n \equiv \mathsf{X}^n + \boldsymbol{c} \pmod{n,\ \mathsf{X}^k - \mathbf{1}} \tag{1.1}$$

for a range of constants c, *i.e.*, $0 < c \leq s$. The limit s is specified by an expression based on n and k. Details of such polynomial identity checks will be explained later, but the form of the polynomial identity is so striking that the AKS team suggested a special term for this form: an *introspective* check. We shall eventually see how such introspective polynomial checks characterise the AKS algorithm.

With these notions, we can get a glimpse of our formal definition of the AKS algorithm:

Definition 1. *The AKS algorithm with* 3 *phases.*

$$\begin{array}{l}
\mathsf{aks}\ n \stackrel{\mathsf{def}}{=} \\
\quad \mathsf{power_free}\ n \wedge \\
\quad \textbf{case}\ \mathsf{aks_param}\ n\ \textbf{of} \\
\quad \mid\ \mathsf{nice}\ k \triangleright\ k = n \\
\quad \mid\ \mathsf{good}\ k \triangleright \\
\quad\quad \forall c.\ 0 < c \wedge c \leq (\mathsf{SQRT}\ \varphi(k))\lceil \log n \rceil \Rightarrow (\mathsf{X} + \boldsymbol{c})^n \equiv \mathsf{X}^n + \boldsymbol{c} \pmod{n,\ \mathsf{X}^k - \mathbf{1}} \\
\quad \mid\ \mathsf{bad} \triangleright\ \mathsf{F}
\end{array}$$

Note how our formal definition of aks n takes into account the 3 phases of the AKS algorithm.

We shall formally establish that the AKS algorithm, backed up by some beautiful results from the theory finite fields, stands up to its claim:

Theorem 2. *The AKS algorithm is a primality test.*

$$\vdash \mathsf{prime}\ n \iff \mathsf{aks}\ n$$

We shall explain computation in monadic style in Chapter 6, Section 6.1. Using monadic style primitives (*e.g.*eqM n k for equality check) and subroutines: power_freeM n for power free test, paramM n for parameter search, and poly_intro_rangeM n k k for polynomial introspective checks, we put each phase of the AKS algorithm together by composing them in monadic style:

Definition 3. *The AKS algorithm implemented in monadic style.*

```
aksM n ≝
 do
  b ← power_freeM n;
  if b then
   do
    c ← paramM n;
    case c of
    | nice k ▷ eqM k n
    | good k ▷ poly_intro_rangeM n k k
    | bad ▷ return F
   od
  else return F
 od
```

and prove these important results, where valueOf (aksM n) is the final value of the monadic computation with input n, and stepsOf ∘ aksM is the number of steps of monadic computation expressed as a function of n:

Theorem 4. *The AKS implementation is correct, and belongs to the polynomial class.*

$$\vdash \mathsf{valueOf}\ (\mathsf{aksM}\ n) \iff \mathsf{aks}\ n$$
$$\vdash \mathsf{stepsOf} \circ \mathsf{aksM} \in \mathcal{O}(\lceil \log n \rceil^{21})$$

These results constitute our formal proof of the AKS algorithm, *i.e.*, "PRIMES is in P". Given that our implementation of the AKS algorithm is not intended for efficiency, and our model of elementary computations is quite conservative (see Section 6.3), together with the fact that our recurrence theory vastly over-estimates for the sake of simplicity, we are happy with our $\mathcal{O}(\lceil \log n \rceil^{21})$. The original AKS paper estimated the algorithm[5] in the order $\widetilde{\mathcal{O}}(\lceil \log n \rceil^{12})$, and the revised AKS paper lowered this to $\widetilde{\mathcal{O}}(\lceil \log n \rceil^{\frac{15}{2}})$.

1.4 AKS Formalisation

Because the AKS algorithm is a major milestone in primality testing, we would like to formally verify the authors' claim, to be 100% confident of the breakthrough.

The AKS algorithm depends critically on a parameter k derived from the input number n. In the original AKS paper, the parameter k needs to be a prime, and its bound depends on the Prime Number Theorem for the distribution of primes, and the Brun-Titchmarsh Theorem for an upper bound on the distribution of primes in an arithmetic progression. Using these results

[5] The $\widetilde{\mathcal{O}}$ notation ignores logarithmic powers of the variable, *e.g.* $\mathcal{O}(f(n)\lceil \log n \rceil^3)$ is shortened to $\widetilde{\mathcal{O}}(f(n))$.

from analytic number theory, the AKS team obtained an upper bound for parameter k, showing that the AKS algorithm is in P, the class of polynomial-time algorithms.

The Prime Number Theorem aims at the limiting behaviour of the distribution of primes. However, if only the bounds on the distribution is required, such a result had been obtained by Chebyshev. This result is known as a weak form of the Prime Number Theorem.

When experts examined the AKS proof, they realized that while the best bound for the parameter k depends on those theorems of analytic number theory, the weak form of prime distribution can already produce an acceptable bound, still keeping "PRIMES is in P". Therefore, initial attempts to formalise the AKS algorithm concentrated on establishing prime distributions of some form in theorem provers. As early as 19 August 2002, John Harrison [2002] mentioned during his talk on formalisation of real numbers that:

> *By the way, some deep results about the distribution of primes are used in the recent polynomial-time primality-testing algorithm . . .*

As reported in Wiedijk [2003], John Harrison formalised, in HOL Light, a weak form of the Prime Number Theorem and a theory of cyclotomic polynomials for this purpose.

Laurent Théry [2003] discussed the formalisation of algorithms in number theory: those related to primes and primality tests. He reported a formal proof of the Miller-Rabin algorithm, a probabilistic primality test, by Joe Hurd [2003]. He predicted that the next candidate for formalisation will be the AKS algorithm.

Théry observed that there is no relationship between the description of an algorithm and the difficulty of its formal proof. As an example in the report, he found that a correctness proof in Coq of the simple Knuth's algorithm to list the first n primes required a variety of techniques, including a formal proof of Bertrand's postulate.[6] This was prior to the revised AKS paper, thus he was concerned that a formal proof of the AKS algorithm will involve much deeper properties about primes than Bertrand's postulate.

With the role of the Prime Number Theorem in determining the bound on the parameter k clarified, the proof in the revised AKS paper of 2004 is considered "simple and elementary", as expressed in the Gödel Prize 2006 citation (see Section 1.2). The improvements concern the parameter k:

- The parameter k is no longer required to be prime. Instead, the correctness proof of the AKS algorithm is obtained by invoking properties of cyclotomic polynomials, which is a standard topic in finite field theory.

- Since the condition on k is relaxed, the bound on its size no longer depends explicitly on Prime Number Theorem. Instead, the bound can be derived from a lemma about a lower bound on the least common multiple of consecutive numbers. The lemma has a standard proof by elementary calculus in the literature.

These improvements cleaned up the presentation of the AKS algorithm.

[6] This is first proved by Chebyshev in 1852, that there is always a prime between n and $2n$ for $n > 1$.

Due to these simplications, there was renewed interest to formalise the AKS algorithm. Indeed, Campos et al. [2004] tried in ACL2, and de Moura and Tadeu [2008] attempted in Coq. Both only showed that a prime will pass all the AKS tests. This is the easy part. The hard part is show that a non-prime cannot pass all the AKS tests.

1.5 Our Contribution

Our goal is to achieve a formal proof of "PRIMES is in P", based on the AKS algorithm. This includes a proof of its correctness, both the easy and hard parts, and a proof of its complexity, establishing that the AKS algorithm is indeed in the polynomial-time class.

We aim for a formalisation in HOL4 that is:[7]

- self-contained, not relying on external libraries of previous work,
- elementary, not involving concepts from analysis of real-valued functions, and
- algebraic, following the improved version of the algorithm by the AKS team.

Our work in the formalisation of the AKS algorithm is built up by layers. We develop theories for various topics, many of these are interesting by themselves.

In a previous effort, we formalised the correctness of the AKS algorithm, both the easy and hard parts, in Chan and Norrish [2015]. In that work, we followed the original AKS paper, taking the parameter k to be a prime. We also established k's existence on general grounds, but without giving it a bound.

In this thesis, we follow the revised AKS paper, and achieve the following:

1. We prove a lemma about a lower bound on the least common multiple of consecutive numbers without any use of calculus. Instead, the lemma is established by an ingenious use of the Leibniz's triangle, a variant of the Pascal's triangle. The lower bound given by the lemma provides a bound on the parameter k of the AKS algorithm (Definition 1).

2. The correctness of the AKS algorithm (Theorem 2) is now proved with parameter k no longer required to be prime. This is a result of another effort, the formalisation of finite field theory including cyclotomic polynomials.

3. We have a formal implementation of the AKS algorithm (Definition 3), involving details of each phase. The implementation is not optimized for performance, but it simplifies the subsequent complexity analysis.

4. We reach the ultimate goal: a proven correct implementation of the AKS algorithm, with verification that its computational complexity is indeed bounded by a polynomial of the input size (Theorem 4).

[7] See Section 8.2 (page 133) for reflections on how well we meet our aims.

5. Our work is backed up by a thorough formalisation of finite fields, from the hierarchy of algebraic structures, to their existence and uniqueness up to isomorphism by cardinality. The collection of abstract algebra libraries is developed with generic types. It has proved its usefulness in several key areas in the formalisation of AKS algorithm.

6. We introduce a simple monadic framework to analyse the computational complexity of algorithms. This proves to be adequate to show that the AKS algorithm runs in polynomial time.

Thus we achieve a complete formal elementary proof of the correctness of the AKS algorithm, and provide an elementary complexity analysis to verify "PRIMES is in P".

Our theorem prover is HOL4, and our techniques are all *elementary*: basic number theory, and standard abstract algebra. We build a fairly complete library for finite fields, including quotient fields, cyclotomic polynomials, culminating in their existence and uniqueness for the classification of finite fields.

1.6 Thesis Structure

In order to present a coherent account of the formalisation of the AKS algorithm, with minimal distraction from other issues, this thesis is structured into chapters, as follow:

1. Introduction — an overview of the AKS formalisation work (Chapter 1).

2. Part 1: Foundation

   - Basic Algebra — number theory and algebra theorems (Chapter 2).
   - AKS Algorithm — a study of its phases and parameters (Chapter 3).

3. Part 2: Correctness

   - Advanced Algebra — finite field and cyclotomic factors (Chapter 4).
   - AKS Main Theorem — a formal proof of the correctness of algorithm (Chapter 5).

4. Part 3: Complexity

   - Complexity Models — toolkit for computational complexity analysis (Chapter 6).
   - AKS Complexity — implemetation and analysis of the algorithm (Chapter 7).

5. Conclusion — summary and preview of future work (Chapter 8).

Refer to Figure 1.1 for the relationships between chapters and topics. An Appendix is included with A.1 providing Script References to all definitions, lemma, theorems and corollaries in this thesis, and A.2 describing the Script Library organisation in our source code repository.

HOL4 Sources Our entire respository of HOL4 proof scripts can be found at this location: http://bitbucket.org/jhlchan/hol/src/. The respository is tagged with `phd-thesis-02`, and consists of sub-folders `aks/`, `/algebra` and `/algorithm`. Each folder or subfolder has a description of its content in a file named `README.md`. Scripts in the repository are provided in two formats:

- those with suffix .hol are intended for use in interactive HOL4 sessions,
- those with suffix Script.sml are intended for HOL4 compilation using `Holmakefile`.[8]

We shall omit the suffix when referring to proof scripts. Each proof script has documentation in block comments at the beginning. The full library of proof scripts is described in Appendix (Section A.2, page 152). Hyperlinks have been set up for each Definition, Theorem, Lemma, and Corollary to associate with the actual location of the item in our repository.

1.7 Summary

We introduce the AKS algorithm through its 3 phases, and formulate our goals: to prove that it is a primality test (Theorem 2), and to show that an implementation is correct and has a run-time bounded by a polynomial function of the input size (Theorem 4). Together they verify formally that the AKS algorithm exemplifies "PRIMES is in P". The revised AKS paper, complete with traditional proofs of both the correctness and complexity of the AKS algorithm, consists of only nine pages. Although the paper is short, there is a substantial amount of coding to be done in order to formalise the AKS algorithm from the ground up. Most of the work relates to the proper development of the supporting libraries. Our discussion shall be divided into 3 Parts: Foundation, Correctness, and Complexity. We start with Part 1, to develop a blueprint for the formalisation work.

1.8 Remarks

For informative accounts of formalisation of mathematics and automated theorem proving, see John Harrison [1996] and his talks over the years (Harrison [2013, 2015]). A comprehensive survey of iteractive theorem proving was given by Andrea Asperti [2009]. An article titled *Formally Verified Mathematics* was contributed by Jeremy Avigad and Harrison [2014]. Another recent talk on advances of formal mathematics was given by Josef Urban [2016].

During December 2008, the Notices of the American Mathematical Society (Magid [2008]) published *A Special Issue on Formal Proof*. In August 2011, the journal Mathematical Structures in Computer Science (Curien [2011]) had a special issue on *Interactive Theorem Proving and the Formalisation of Mathematics*. In February 2013, the Journal of Automated Reasoning had *Special Issue: Formal Mathematics for Mathematicians* (Trybulec et al. [2013]).

[8] Compile using HOL4 version with commit tag `6711e409`.

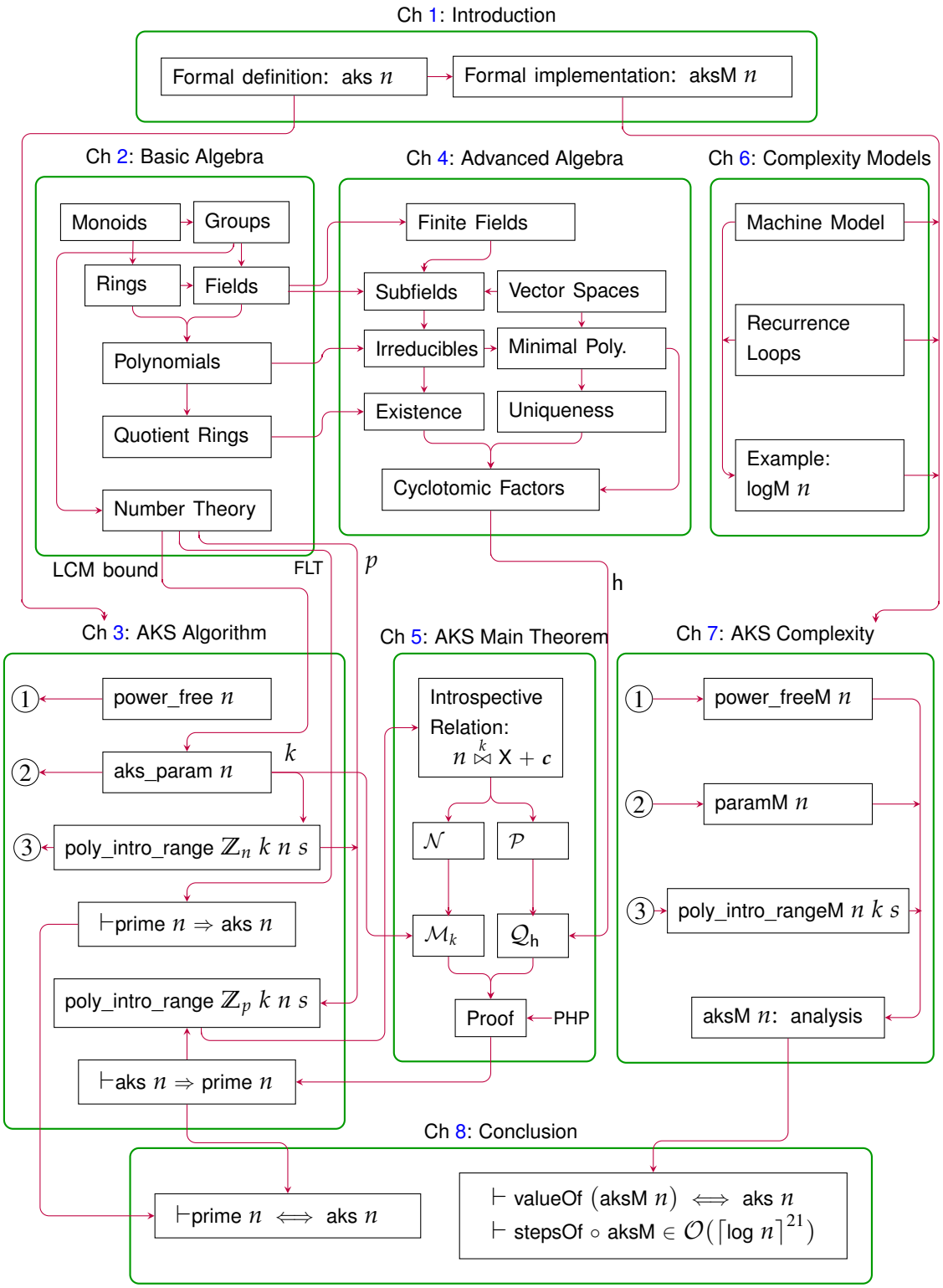


Figure 1.1: Dependency Diagram of thesis topics. Terms and symbols are defined in the relevant chapters. Very briefly, here FLT is Fermat's Little Theorem, and PHP is the Pigeonhole Principle.

1.9 Notation

All statements starting with a turnstile ($\vdash$) are HOL4 theorems, automatically pretty-printed to LaTeX from the relevant theory in the HOL4 development. Generally, our notation allows an appealing combination of quantifiers ($\forall$, $\exists$), logical connectives ($\wedge$ for "and", $\vee$ for "or", $\neg$ for "not", $\Rightarrow$ for "implies", and $\iff$ for "if and only if"), with logical constants T for true, and F for false. For functional notation, we use λ for function abstraction, and juxtaposition for function application.

Set-theoretic Notation We use standard set notations: $\in$ for set membership, $\cup$ for set union, $\cap$ for set intersection, and set comprehensions such as $\{ x \mid x < 7 \}$. The universal set of type α is denoted by $\mathcal{U}(:\alpha)$. For a finite set S, the cardinality is written $|\mathsf{S}|$, the sum over its elements is $\sum \mathsf{S}$, and the product over its elements is $\prod \mathsf{S}$.

List Notation Lists are enclosed in square-brackets [], with list members separated by semi-colon (;), using infix operators :: for "cons", $\frown$ for append, and . . for inclusive range. When $\ell = h::t$, h is the head of ℓ and t is the tail of ℓ. The empty list is []. For a nonempty list ℓ, LAST ℓ picks the element at the end, and FRONT ℓ takes all elements except the end. Other common list operators are: LENGTH, SUM, REVERSE, and MEM for list member. We use λ for function abstraction, and juxtaposition for function application. The application of a single argument function f to every element x in a list xs is denoted by MAP f xs. Extending this to a function of two arguments, MAP2 f xs ys denotes the application of f taking the first argument from xs and the second argument from ys.

Relation and Maps Given a binary relation $\mathcal{R}$, its reflexive and transitive closure is marked by an asterisk ($*$), *i.e.*, $\mathcal{R}^*$. We write $f : s \hookrightarrow t$ to mean that function f is injective from set s to set t, and write $f : s \leftrightarrow t$ to mean that function f is bijective from set s to set t.

Multiplication Except for explicit products like $7 \times 13 = 91$, we shall use juxtaposition for the product of two numbers x and y, as xy. When a function f is applied to a product, it is shown as $f\ xy$; whereas $(f\ x)y$ shows the product of $f\ x$ and y. This convention closely resembles the traditional presentation in mathematics. In other domains, *e.g.*, algebraic structures and polynomials, we shall use $x \times y$ to represent the product of two elements x, y.

Number-theoretic Notation For natural number division of n by m, the quotient is denoted by n div m, and the remainder is denoted by n mod m. We write $n \mid m$ when n divides m, or, equivalently, $n \in$ divisors m. The notation $x \equiv y \pmod{n}$ means that both x and y give the same remainder after division by n. We use $\varphi(n)$ to denote the Euler φ-function of n, the count of numbers up to n that are coprime to n. All numeric functions are integer functions, rounding down unless enclosed by up-brackets $\lceil\ \rceil$, the rounding-up or ceiling function. For example,

ROOT k n is the integer k-th root of n, SQRT n is the integer square root of n, n div 2 is the integer half of n, and $\lceil \log n \rceil$ is the round-up of the logarithm of n in base 2.

HOL4 types The HOL4 theorem prover uses a simple type system. Our formalisation of the AKS algorithm uses the basic number type, `num`, and sets and lists derived from number type. In order for abstract algebra to be applicable to objects of various types, the algebraic structures are defined over a generic type α.

These are the basic notations. More notation will be introduced as the need arises.

Part I

Foundations

Part 1 is about the AKS algorithm, from pseudocode to primality test. For each phase of the algorithm, we define concepts to formalise the actions, and develop theories with background from basic algebra. Showing a prime can go through the AKS algorithm is straight-forward. Proving a number going through the AKS algorithm must be a prime needs a strategy: get a prime, build a finite field, and formulate the AKS Main Theorem in finite field. The AKS Main Theorem is the focus of Part 2, but all preparatory work are provided in Part 1.

Chapter 2

Basic Algebra

This chapter walks through various topics in algebra that are basic for an understanding of the AKS algorithm. We describe our formalisation of abstract algebra, from monoids and groups, to rings and fields. The correctness proof of the AKS algorithm draws ideas from the theory of finite fields. Modular systems, based on numbers and polynomials, provide examples of these algebraic structures. Their properties and terminology are frequently used in discussions of the AKS algorithm. We begin with fairly obvious definitions and results, but finish with some remarkable formulae, including a novel approach to the first lemma in the AKS paper.

> Mathematics is the work of the human mind,
> which is destined to study rather than to know,
> to seek the truth rather than to find it.
> — Évariste Galois (1832)[1]

2.1 Algebraic Structures

It was the work of Galois' mind where the concept of a group was first conceived and studied. This transformed the study of algebra: from concrete numbers to abstract structures. Moreover, a short paper by Galois [1830] laid the foundation for the study of finite fields — they are also called Galois fields in his honour.

The theory behind the AKS algorithm is based on finite fields. Our formalisation of finite fields is at the top of a hierarchy of theories defining a family of algebraic structures:

- A Monoid $\mathcal{M}$ with a binary operation $\times$ that is closed, associative, and has an identity $\#e$. An abelian monoid, AbelianMonoid $\mathcal{M}$, has the operation also commutative.
- A Group $\mathcal{G}$ is a monoid with an inverse x^{-1} for each element x such that $x \times x^{-1} = \#e$. An abelian group, AbelianGroup $\mathcal{G}$, is a commutative group.
- A commutative Ring $\mathcal{R}$ has two components, a group $\mathcal{R}$.sum with operation $+$ and a monoid $\mathcal{R}$.prod with operation $\times$. Both share the same carrier, and:

[1] From his writing *On the progress in pure analysis*, as translated in Neumann [2013].

- $\mathcal{R}$.sum is an abelian group with identity $\mathbb{0}$;
- $\mathcal{R}$.prod is an abelian monoid with identity $\mathbb{1}$;
- Multiplication is distributive over addition.

- A Field $\mathcal{F}$ is a commutative ring, with all nonzero elements having multiplicative inverses; *i.e.*, they form a group $\mathcal{F}^*$.

Our formalisation takes only the commutative ring, since our target is the finite field. Hereafter, rings are understood to be commutative rings. This hierarchy exemplifies the naïve approach to algebra in HOL4, which is sufficient for the purpose of this thesis.

The traditional mathematical presentation and notations undergo some drastic modifications when rendered in HOL. We first determine our representation for these algebraic structures. At our base, we have the α `monoid` type:

$$\alpha\ \texttt{monoid} = \langle\!\![\ \mathsf{carrier} : \alpha \to \texttt{bool};\ \mathsf{op} : \alpha \to \alpha \to \alpha;\ \mathsf{id} : \alpha\ ]\!\!\rangle$$

A monoid $(\mathcal{M} : \alpha\ \texttt{monoid})$ is a value of the α `monoid` type, effectively a triple of three different fields. Using the HOL record machinery, we can refer to these fields with a pleasant "dot" notation; *e.g.*, $\mathcal{M}$.op. The $\mathcal{M}$.carrier is a subset of all possible values of type α; the operation $\mathcal{M}$.op is a binary operation on those values and the identity $\mathcal{M}$.id is one of those values.

It is then straightforward to define the predicate Monoid over such values that carves out those that satisfy the axioms of a monoid. Here we annotate the definition's variables with their types, and remove the special overloading used in the rest of this thesis so that uses of $\mathcal{M}$.op are explicit.

Definition 5. *Axioms of a monoid: closure, associativity and identity.*

$$\begin{array}{l}
\mathsf{Monoid}\ (\mathcal{M} : \alpha\ \texttt{monoid}) \stackrel{\mathsf{def}}{=} \\
\quad (\forall (x : \alpha)\ (y : \alpha).\ x \in \mathcal{M}.\mathsf{carrier} \wedge y \in \mathcal{M}.\mathsf{carrier} \Rightarrow \mathcal{M}.\mathsf{op}\ x\ y \in \mathcal{M}.\mathsf{carrier}) \wedge \\
\quad (\forall (x : \alpha)\ (y : \alpha)\ (z : \alpha). \\
\qquad x \in \mathcal{M}.\mathsf{carrier} \wedge y \in \mathcal{M}.\mathsf{carrier} \wedge z \in \mathcal{M}.\mathsf{carrier} \Rightarrow \\
\qquad\quad \mathcal{M}.\mathsf{op}\ (\mathcal{M}.\mathsf{op}\ x\ y)\ z = \mathcal{M}.\mathsf{op}\ x\ (\mathcal{M}.\mathsf{op}\ y\ z)) \wedge \mathcal{M}.\mathsf{id} \in \mathcal{M}.\mathsf{carrier} \wedge \\
\quad \forall (x : \alpha).\ x \in \mathcal{M}.\mathsf{carrier} \Rightarrow \mathcal{M}.\mathsf{op}\ \mathcal{M}.\mathsf{id}\ x = x \wedge \mathcal{M}.\mathsf{op}\ x\ \mathcal{M}.\mathsf{id} = x
\end{array}$$

A group $\mathcal{G}$ is a monoid with all its elements invertible. By hiding the underlying types, and overloading $\mathcal{G}$.carrier by G, $\mathcal{G}$.op $x\ y$ by $x \times y$, and $\mathcal{G}$.id by $\#e$, the result is better for readability:

Definition 6. *A group is a monoid with every element having an inverse.*

$$\mathsf{Group}\ \mathcal{G} \stackrel{\mathsf{def}}{=} \mathsf{Monoid}\ \mathcal{G} \wedge \forall x.\ x \in \mathsf{G} \Rightarrow \exists y.\ y \in \mathsf{G} \wedge y \times x = \#e$$

We can prove that such inverses are unique and are inverses on the left and right. Skolemizing, we define the inverse function, and characterise it thus

$$\vdash \mathsf{Group}\ \mathcal{G} \Rightarrow \forall x.\ x \in \mathsf{G} \Rightarrow x \times x^{-1} = \#e \wedge x^{-1} \times x = \#e$$

The ring type consists of a combination of two monoid values:

$$\alpha\ \texttt{ring} = \langle\!\![\ \mathsf{carrier} : \alpha \to \texttt{bool};\ \mathsf{sum} : \alpha\ \texttt{monoid};\ \mathsf{prod} : \alpha\ \texttt{monoid}\ ]\!\!\rangle$$

For a ring $\mathcal{R}$, its additive monoid is denoted by $\mathcal{R}$.sum and its multiplicative monoid is denoted by $\mathcal{R}$.prod. When we come to define the ring axioms, we can use prettier syntax in the description of the distributive law. In other words, $x + y$ is really $\mathcal{R}$.sum.op $x\ y$, and $x \times y$ is really $\mathcal{R}$.prod.op $x\ y$. Denoting the carrier of ring $\mathcal{R}$ by R, this is the definition of a ring in HOL4:

Definition 7. *Axioms of a ring: additive group, multiplicative monoid, and distribution law.*

$$\begin{aligned}&\mathsf{Ring}\ \mathcal{R} \stackrel{\mathsf{def}}{=}\\ &\quad \mathsf{AbelianGroup}\ \mathcal{R}.\mathsf{sum} \wedge \mathsf{AbelianMonoid}\ \mathcal{R}.\mathsf{prod} \wedge \mathcal{R}.\mathsf{sum.carrier} = \mathsf{R} \wedge \mathcal{R}.\mathsf{prod.carrier} = \mathsf{R} \wedge\\ &\quad \forall x\ y\ z.\ x \in \mathsf{R} \wedge y \in \mathsf{R} \wedge z \in \mathsf{R} \Rightarrow x \times (y + z) = x \times y + x \times z\end{aligned}$$

Finally, the definition of what it is to be a field $\mathcal{F}$ can be quite terse:

Definition 8. *A field is a ring with nonzeros forming a multiplicative group.*

$$\mathsf{Field}\ \mathcal{F} \stackrel{\mathsf{def}}{=} \mathsf{Ring}\ \mathcal{F} \wedge \mathsf{Group}\ \mathcal{F}^*$$

HOL4 Conventions In HOL4, definitions such as these subsequently require that all theorem statements to be qualified with assumptions such as $\mathsf{Field}\ \mathcal{F}$. This is because the value $\mathcal{F}$ is just a pair of record values, and is not known to satisfy the field axioms without that explicit assumption. Though tedious, these qualifications of our theorem statements do not significantly impact the theorem-proving task. Rather, the burden lies mostly in the initial writing of the goal.

Although overloading can be used to pretty-print for readability, we keep the types barely visible through the use of different fonts. For example, a field $\mathcal{F}$ has elements in the carrier F, with the nonzero elements F^* forming a group $\mathcal{F}^*$.

In the rest of this thesis, overloading is used extensively to hide complicated terms such as $\mathcal{R}$.sum.op $x\ y$, but the logical assumptions (such as $\mathsf{Field}\ \mathcal{F}$) always appear. Occasionally, this can make the assumption seem vacuous as the syntax for the operators no longer appears to refer back to the value (*e.g.*, $\mathcal{F}$) at all. For example, this apparent vacuousness can be seen in the last ring axiom above.

We shall look at the properties of these algebraic structures, especially those that will be used later in the AKS algorithm.

2.2 Monoids and Groups

Both monoid and group have only one operation, generally called multiplication. Repeated multiplication of an element x by itself n times is exponentiation, with the usual notation x^n. By definition, $x^0 = \#e$. It may happen that $x^n = \#e$ for some $n > 0$. The smallest such exponent n is the multiplicative order of x, or simply the *order* of x, denoted by $\mathsf{order}_{\mathcal{M}}(x)$:

Definition 9. *The multiplicative* $\mathsf{order}_{\mathcal{M}}(x)$ *is the least nonzero exponent to give the identity, or zero is no such exponent exists.*

$$0 < n \Rightarrow (\mathsf{order}_{\mathcal{M}}(x) = n \iff x^n = \#e \wedge \forall m.\ 0 < m \wedge m < n \Rightarrow x^m \neq \#e)$$
$$\mathsf{order}_{\mathcal{M}}(x) = 0 \iff \forall n.\ 0 < n \Rightarrow x^n \neq \#e$$

Although finding $x^n = \#e$ does not imply that n is the order, we do have:

Theorem 10. *The order of a monoid element divides an exponent that can give the identity.*

$$\vdash \mathsf{Monoid}\ \mathcal{M} \Rightarrow \forall x\ n.\ x \in \mathsf{M} \Rightarrow (x^n = \#e \iff \mathsf{order}_{\mathcal{M}}(x) \mid n)$$

Proof. Let $k = \mathsf{order}_{\mathcal{M}}(x)$. Dividing n by k gives a quotient q and a remainder r, such that $n = qk + r$ and $r < k$. Thus,

$$\#e = x^n = x^{qk+r} = x^k \times x^r = \#e^k \times x^r = x^r$$

Therefore $x^r = \#e$. But $r < k$ and order k is minimal, so $r = 0$. In other words, $k \mid n$. □

This enables us to deduce the order of x^n from the order of x:

Theorem 11. *For a monoid element, the order of its power is reduced by the greatest common divisor of the power and its order.*

$$\vdash \mathsf{Monoid}\ \mathcal{M} \Rightarrow \forall x\ n.\ x \in \mathsf{M} \wedge 0 < n \Rightarrow \mathsf{order}_{\mathcal{M}}(x^n) = \mathsf{order}_{\mathcal{M}}(x)\ \mathsf{div}\ \mathsf{gcd}(n, \mathsf{order}_{\mathcal{M}}(x))$$

Proof. Let $k = \mathsf{order}_{\mathcal{M}}(x)$, $y = x^n$, $j = \mathsf{order}_{\mathcal{M}}(y)$, and $d = \mathsf{gcd}(n,k)$. Since $d \mid n$ and $d \mid k$,,

$$y^{k\ \mathsf{div}\ d} = (x^n)^{k\ \mathsf{div}\ d} = (x^k)^{n\ \mathsf{div}\ d} = \#e^{n\ \mathsf{div}\ d} = \#e$$

Thus $j \mid k\ \mathsf{div}\ d$ by order divides exponent (Theorem 10).

Now $x^{nj} = (x^n)^j = y^j = \#e$, so $k \mid nj$, again by Theorem 10. By Bézout's identity[2], there exists s, t such that $ns = kt + d$. Multiply by j to give $nsj = ktj + dj$. Because $k \mid nj$, $k \mid dj$. Since $d \mid k$, so $k\ \mathsf{div}\ d \mid j$.

With $j \mid k\ \mathsf{div}\ d$ and $k\ \mathsf{div}\ d \mid j$, $j = k\ \mathsf{div}\ d$. □

We write $\mathcal{H} \leq \mathcal{G}$ to mean that $\mathcal{H}$ is a subgroup of $\mathcal{G}$:

[2]This form of Bézout's identity allows us to remain within the natural numbers.

Definition 12. *A subgroup $\mathcal{H}$ of group $\mathcal{G}$ is a group with subset carrier and the same operation.*

$$\mathcal{H} \leq \mathcal{G} \stackrel{\text{def}}{=} \mathsf{Group}\ \mathcal{H} \wedge \mathsf{Group}\ \mathcal{G} \wedge \mathsf{H} \subseteq \mathsf{G} \wedge \mathcal{H}.\mathsf{op} = \mathcal{G}.\mathsf{op}$$

Similar ideas apply to submonoids. A subgroup can also be identified by:

Theorem 13. *Let $\mathcal{H}$ be a nonempty subset of the group $\mathcal{G}$ equipped with the group operation. If for all elements $x, y \in \mathcal{H}$, $x \times y^{-1} \in \mathcal{H}$, then $\mathcal{H}$ is a subgroup of $\mathcal{G}$.*

$$\begin{aligned}\vdash \mathcal{H} \leq \mathcal{G} \iff\\ \mathsf{Group}\ \mathcal{G} \wedge \mathcal{H}.\mathsf{op} = \mathcal{G}.\mathsf{op} \wedge \mathcal{H}.\mathsf{id} = \mathcal{G}.\mathsf{id} \wedge \mathsf{H} \neq \emptyset \wedge \mathsf{H} \subseteq \mathsf{G} \wedge\\ \forall x\ y.\ x \in \mathsf{H} \wedge y \in \mathsf{H} \Rightarrow \mathcal{G}.\mathsf{op}\ x\ (y^{-1}) \in \mathsf{H}\end{aligned}$$

Since a subgroup $\mathcal{H}$ induces an equal-size partition *via* cosets, we have:

Theorem 14. ***(Lagrange's Theorem).*** *For a finite group $\mathcal{G}$, the cardinality of any subgroup is a divisor of $|\mathsf{G}|$.*

$$\vdash \mathcal{H} \leq \mathcal{G} \wedge \mathsf{FINITE\ G} \Rightarrow |\mathsf{H}| \mid |\mathsf{G}|$$

For a finite group $\mathcal{G}$, there are only a finite number of elements. Let $a \in \mathsf{G}$. Then the set $\{a, a^2, a^3, \dots\}$ is finite. Together with the group operation, this forms a subgroup, called the group generated by element a, denoted by $\langle a \rangle$. Its cardinality is easily derived:

Theorem 15. *For an element a with nonzero order in a finite group $\mathcal{G}$, the subgroup $\langle a \rangle$ has cardinality* $\mathsf{order}_{\mathcal{G}}(a)$.

$$\vdash \mathsf{Group}\ \mathcal{G} \wedge a \in \mathsf{G} \wedge 0 < \mathsf{order}_{\mathcal{G}}(a) \Rightarrow |\langle a \rangle.\mathsf{carrier}| = \mathsf{order}_{\mathcal{G}}(a)$$

Applying Lagrange's Theorem (Theorem 14), we have:

Theorem 16. *In a finite group $\mathcal{G}$, the order of an element divides $|\mathsf{G}|$.*

$$\vdash \mathsf{FiniteGroup}\ \mathcal{G} \Rightarrow \forall a.\ a \in \mathsf{G} \Rightarrow \mathsf{order}_{\mathcal{G}}(a) \mid |\mathsf{G}|$$

We can also derive this important result:

Theorem 17. *In a finite group, an element raised to cardinality exponent gives the identity.*

$$\vdash \mathsf{FiniteGroup}\ \mathcal{G} \wedge a \in \mathsf{G} \Rightarrow a^{|\mathsf{G}|} = \#e$$

Proof. Let element a in a finite group $\mathcal{G}$ has order k. Then $a^k = \#e$. Now $|\mathsf{G}|$ is a multiple of k, *i.e.*, $|\mathsf{G}| = qk$ for some q. Thus $a^{|\mathsf{G}|} = (a^k)^q = \#e^q = \#e$. □

Order Notation Although the order of an element x in a monoid $\mathcal{M}$ should be denoted by $\mathsf{order}_{\mathcal{M}}(x)$, the monoid subscript is usually abbreviated. For example, we write $\mathsf{order}_n(k)$ for

order of k in the monoid $\mathbb{Z}_n^*$.[3] Later in Section 4.2, we shall introduce quotient rings over rings $\mathcal{R}$ or fields $\mathcal{F}$, and write $\mathsf{order}_{\mathsf{h}}(\mathsf{X})$ for the order of polynomial X in the monoid $\mathcal{R}_{\mathsf{h}}^*[\mathsf{X}]$ or $\mathcal{F}_{\mathsf{h}}^*[\mathsf{X}]$ derived from a modulus polynomial h.

2.3 Rings and Fields

A ring with $\mathbb{1} = \mathbb{0}$ is called a trivial ring. A field is a non-trivial ring, since $\mathbb{1}$ is in $\mathcal{F}^*$ with $\mathbb{0}$ excluded. As noted in Section 2.1, our rings are more commonly known as commutative rings with identity. We did not go all the way to non-commutative rings since our goal is about finite fields. By the same token, we do not consider skew fields. However, we do consider other algebraic structures related to fields, *e.g.*, integral domains in Section 2.4.

Let a be a nonzero element in a ring. Due to the distributive law, and the fact that $\mathbb{1} \times a = a$, all nonzero a have the same additive order as $\mathbb{1}$. The unique additive order of $\mathbb{1}$ in a ring $\mathcal{R}$ is called its *characteristic*, denoted by $\mathsf{char}(\mathcal{R})$:

Definition 18. *The characteristic of a ring is the additive order of its multiplicative identity.*

$$\mathsf{char}(\mathcal{R}) \stackrel{\mathsf{def}}{=} \mathsf{order}_{\mathcal{R}.\mathsf{sum}}(\mathbb{1})$$

The characteristic of a finite ring is positive: the smallest number of repeated additions of $\mathbb{1}$ giving $\mathbb{0}$.

The multiplicative invertibles of a ring $\mathcal{R}$ are called units, denoted by $\mathcal{R}^*$. Since each unit has a multiplicative inverse, we have:

Theorem 19. *The units of a ring form a group.*

$$\vdash \mathsf{Ring}\ \mathcal{R} \Rightarrow \mathsf{Group}\ \mathcal{R}^*$$

For a field $\mathcal{F}$, all nonzero elements are units, by definition, and the group is $\mathcal{F}^*$.

2.4 Integral Domains

Since a field $\mathcal{F}$ has a multiplicative group $\mathcal{F}^*$, it means that the product of two nonzero elements cannot be $\mathbb{0}$, by multiplicative closure. Hence a field has no zero divisors, an instance of a special kind of ring:

Definition 20. *An integral domain is a non-trivial ring with no zero divisors.*

$$\begin{array}{l}\mathsf{IntegralDomain}\ \mathcal{D} \stackrel{\mathsf{def}}{=} \\ \quad \mathsf{Ring}\ \mathcal{D} \wedge \mathbb{1} \neq \mathbb{0} \wedge \forall x\ y.\ x \in \mathsf{D} \wedge y \in \mathsf{D} \Rightarrow (x \times y = \mathbb{0} \iff x = \mathbb{0} \vee y = \mathbb{0})\end{array}$$

[3] For k not in the monoid $\mathbb{Z}_n^*$, $\mathsf{order}_n(k) = \mathsf{order}_n(k \bmod n)$ for $n > 0$.

For an integral domain $\mathcal{D}$, let a be a nonzero element, *i.e.*, $a \neq \mathbb{0}$. If the integral domain is finite, the sequence $a, a^2, a^3, \ldots$ must repeat. Suppose $a^j = a^{j+k}$ for some j and k with $k \neq 0$. Then ring subtraction and distribution give $a^j \times (a^k - \mathbb{1}) = \mathbb{0}$. The absence of zero divisors in an integral domain implies that $a^k = \mathbb{1}$. Note that $a^k = \mathbb{1}$ and $k \neq 0$ means that a has a multiplicative inverse: a^{k-1}. Therefore:

Theorem 21. *A finite integral domain is always a field, hence a finite field.*

$$\vdash \mathsf{FiniteIntegralDomain}\ \mathcal{F} \Rightarrow \mathsf{Field}\ \mathcal{F}$$
$$\vdash \mathsf{FiniteIntegralDomain}\ \mathcal{F} \Rightarrow \mathsf{FiniteField}\ \mathcal{F}$$

This allows us to conclude that a finite non-trivial ring is a finite field simply by checking that there are no zero divisors (see Section 4.2, page 61).

2.5 Polynomials

Polynomials are expressions of the form:

$$c_n \mathsf{X}^n + c_{n-1}\mathsf{X}^{n-1} + \cdots + c_1 \mathsf{X} + c_0 \tag{2.1}$$

where the coefficients c_j come from a ring $\mathcal{R}$. These polynomials, equipped with the usual polynomial addition and multiplication, form a polynomial ring, denoted by $\mathcal{R}[\mathsf{X}]$. In these polynomial rings, the additive identity is $\mathbf{0}$, the zero polynomial, and the multiplicative identity is $\mathbf{1}$, the constant polynomial one. When the ring $\mathcal{R}$ has further properties, *e.g.*, without zero divisors so that it is an integral domain $\mathcal{D}$, or all nonzero elements are invertible so that it is a field $\mathcal{F}$, we shall denote the polynomial ring by $\mathcal{D}[\mathsf{X}]$ or $\mathcal{F}[\mathsf{X}]$, respectively.

Polynomial Implementation For simplicity, polynomials are implemented as finite lists. Indeed, our type $(\alpha\ \texttt{poly})$ is a synonym for $(\alpha\ \texttt{list})$ in HOL4. Internally, a polynomial of type α is a list of cofficients taken from $\mathcal{U}(:\alpha)$, with the first element the constant term, and the last element the leading coefficient. The zero polynomial $\mathbf{0}$ is taken as the empty list, its degree is defined as 0, and the degree of a nonzero polynomial is defined as one less than its length. The results of polynomial operations are normalised, so that a nonzero polynomial always has a nonzero leading coefficient. Polynomial addition is componentwise list addition, and polynomial scalar multiplication is componentwise list scaling. Polynomial multiplication is defined by list recursion, using addition and scalar multiplication. Polynomial division requires proper notions of a polynomial divisor and a polynomial remainder, see Section 2.5.1. Further discussion of our implementation is deferred to Section 8.2.2, *Polynomial as lists* (page 135).

Polynomial Notation Denote by h the polynomial as given in (2.1). For this polynomial h, its degree is $\mathsf{deg}\ \mathsf{h} = n$ and its leading coefficient is $\mathsf{lead}\ \mathsf{h} = c_n$. If all coefficients are zero, then $\mathsf{h} = \mathbf{0}$, otherwise $\mathsf{lead}\ \mathsf{h} \neq \mathbb{0}$ after normalisation. We write $\mathsf{poly}\ \mathsf{h}$ to assert that h is a normalised polynomial, and write $\mathsf{monic}\ \mathsf{h}$ to indicate that h is a monic polynomial, with $\mathsf{lead}\ h = \mathbb{1}$.

A polynomial h with coefficients from a ring $\mathcal{R}$, as in (2.1), can be viewed as a function in X. The valuation $\mathsf{h}(a)$ is the value of h when X is substituted by an element $a \in \mathsf{R}$. We extend the notion of substitution to polynomials: substituting the X in polynomial h by another polynomial g gives a polynomial $\mathsf{h}[\![\mathsf{g}]\!]$.[4] We also extend the modulus notation for polynomials, *e.g.*, $\mathsf{f} \equiv \mathsf{g} \pmod{\mathsf{h}}$ means that both f and g give the same remainder after division by h. In other words, $\mathsf{h} \mid (\mathsf{f} - \mathsf{g})$.

2.5.1 Polynomial Modulus

For polynomial division, the polynomial divisor h is called the modulus. Besides requiring the modulus to be nonzero, $\mathsf{h} \neq \mathbf{0}$, there are other restrictions for the modulus h. We first consider the general case, where the polynomials have coefficients from a ring $\mathcal{R}$.

Clearly, a monic modulus can always perform the long division for any polynomial dividend — since a suitable multiple of the modulus can cancel the leading term of the dividend. Thus the degree of the dividend p is reduced at each step, leading to termination of the division process with this result:

$$\mathsf{p} = \mathsf{p} \operatorname{div} \mathsf{h} \times \mathsf{h} + \mathsf{p} \bmod \mathsf{h} \tag{2.2}$$

where $(\mathsf{p} \operatorname{div} \mathsf{h})$ is the quotient, and $(\mathsf{p} \bmod \mathsf{h})$ is the remainder. When the modulus is not monic, this division process is possible only when the leading coefficient of the modulus is invertible, *i.e.*, a unit element in a ring, denoted by $\mathsf{unit}\ (\mathsf{lead}\ \mathsf{h})$. In this case, both the dividend and the modulus can be multiplied by the inverse of the leading coefficient of the modulus, reducing the polynomial division to the monic situation.[5]

For polynomial division to terminate with $(\mathsf{p} \operatorname{div} \mathsf{h})$ and $(\mathsf{p} \bmod \mathsf{h})$ satisfying Equation 2.2, the degree of the remainder has to be strictly less than the degree of the modulus. This means $0 < \deg \mathsf{h}$, *i.e.*, the modulus is not a constant polynomial. In this case, polynomial division by modulus h satisfies:

$$\begin{aligned} \vdash\ & \mathsf{Ring}\ \mathcal{R} \wedge \mathsf{poly}\ \mathsf{p} \wedge \mathsf{poly}\ \mathsf{h} \wedge \mathsf{unit}\ (\mathsf{lead}\ \mathsf{h}) \wedge 0 < \deg \mathsf{h} \Rightarrow \\ & \mathsf{p} = \mathsf{p} \operatorname{div} \mathsf{h} \times \mathsf{h} + \mathsf{p} \bmod \mathsf{h} \wedge \deg\ (\mathsf{p} \bmod \mathsf{h}) < \deg \mathsf{h} \end{aligned}$$

When the modulus h is a constant polynomial, we have:

$$\begin{aligned} &\vdash \mathsf{Ring}\ \mathcal{R} \wedge \mathsf{poly}\ \mathsf{p} \wedge \mathsf{poly}\ \mathsf{h} \wedge \mathsf{unit}\ (\mathsf{lead}\ \mathsf{h}) \wedge \deg \mathsf{h} = 0 \Rightarrow \mathsf{p} \operatorname{div} \mathsf{h} = (\mathsf{lead}\ \mathsf{h})^{-1} \times \mathsf{p} \\ &\vdash \mathsf{Ring}\ \mathcal{R} \wedge \mathsf{poly}\ \mathsf{p} \wedge \mathsf{poly}\ \mathsf{h} \wedge \mathsf{unit}\ (\mathsf{lead}\ \mathsf{h}) \wedge \deg \mathsf{h} = 0 \Rightarrow \mathsf{p} \bmod \mathsf{h} = \mathbf{0} \end{aligned}$$

We lose the fact that the degree of the remainder decreases, but retain the Euclidean Equation 2.2.

Thus in our formulation, division of polynomials with coefficients from a ring $\mathcal{R}$ is defined only for certain types of modulus: those polynomials with leading coefficient invertible. Since nonzero field elements are invertible, any nonzero polynomial with coefficients from a field $\mathcal{F}$ can be a modulus.

[4] Viewing polynomials as functions, this is their function composition.

[5] A variation of this idea is to use *pseudo division*: the dividend polynomial is first multiplied by enough powers of c_n, the leading coefficient of the modulus, to allow for leading term cancellation in the division process.

In this thesis, many theorems concerning polynomial division with coefficients from a ring are presented with the version of a monic modulus, though a more general version of the theorem may also have been proved. This is adequate because, when applied to the AKS algorithm, the modulus is $\mathsf{X}^k - \mathbf{1}$ with $k > 0$, which is monic, or one of its monic factors h.

2.5.2 Polynomial Divisibility

An irreducible polynomial, denoted by ipoly p, has only trivial factors. Let z be an irreducible polynomial. If the irreducible z divides a power, say $\mathsf{z} \mid \mathsf{p}^n$, it must divide the base, *i.e.*, $\mathsf{z} \mid \mathsf{p}$. Extending this result to a set S of monic irreducible polynomials, denoted by miset S, and write $\prod S$ for the product of all polynomials in the set S, we have:

Theorem 22. *For polynomials with coefficients from a field, let $q = \prod S$ be a product of monic irreducibles. If q divides a polynomial power, then q divides the polynomial base.*

$$\vdash \mathsf{Field}\ \mathcal{F} \wedge \mathsf{FINITE}\ S \wedge \mathsf{miset}\ S \wedge \mathsf{poly}\ \mathsf{p} \wedge \mathsf{p}^n \equiv \mathbf{0}\ (\mathrm{mod} \prod S) \Rightarrow \mathsf{p} \equiv \mathbf{0}\ (\mathrm{mod} \prod S)$$

This result is used in the proof of Theorem 102, regarding an exponent for the introspective relation.

2.5.3 Polynomial Roots

An element a is a root of polynomial h if $\mathsf{h}(a) = \mathbb{0}$. The roots of polynomial h is the set roots h, or $\mathsf{roots}_{\mathcal{R}}$ h if the underlying ring $\mathcal{R}$ needs to be specified. Each root a gives a factor $(\mathsf{X} - \boldsymbol{a})$:

Theorem 23. ***(Root Factor Theorem).*** *Each polynomial root corresponds to a linear factor dividing the polynomial.*

$$\vdash \mathsf{Ring}\ \mathcal{R} \Rightarrow \forall \mathsf{h}\ a.\ \mathsf{poly}\ \mathsf{h} \wedge a \in \mathsf{R} \Rightarrow (\mathsf{h}(a) = \mathbb{0} \iff (\mathsf{X} - \boldsymbol{a}) \mid \mathsf{h})$$

If the coefficients of a polynomial are from a field, the leading coefficient can always be reduced to $\mathbb{1}$ by multiplying with its inverse. This shows that such a polynomial will have the same roots as the resulting monic polynomial.

Although a polynomial with coefficients from a ring may have more roots than its degree [6], this cannot happen when its coefficients come from a ring without zero divisors, *i.e.*,

Lemma 24. *A nonzero polynomial with coefficients from an integral domain has no more roots than its degree.*

$$\vdash \mathsf{IntegralDomain}\ \mathcal{F} \Rightarrow \forall \mathsf{h}.\ \mathsf{poly}\ \mathsf{h} \wedge \mathsf{h} \neq \mathbf{0} \Rightarrow |\mathsf{roots}\ \mathsf{h}| \leq \deg \mathsf{h}$$

Since a field is also an integral domain, this result holds for polynomials with coefficients from a field:

[6] For example, in $\mathbb{Z}_6$, $2 * 3 = 0$. Hence in $\mathbb{Z}_6[\mathsf{X}]$, $(X-2)(X-3) = X^2 - 5X = X(X-5)$, which is an example of a degree 2 polynomial with more than 2 roots.

Theorem 25. *A nonzero polynomial with coefficients from a field cannot have more roots than its degree.*

$$\vdash \mathsf{Field}\ \mathcal{F} \Rightarrow \forall \mathsf{p}.\ \mathsf{poly}\ \mathsf{p} \wedge \mathsf{p} \neq \mathbf{0} \Rightarrow |\mathsf{roots}\ \mathsf{p}| \leq \deg \mathsf{p}$$

Thus a polynomial with coefficients from a field with positive degree n has at most n factors.

2.6 Finite Fields

The characteristic of a finite field is particularly interesting:

Theorem 26. *A finite field has prime characteristic.*

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \Rightarrow \mathsf{prime}\ \mathsf{char}(\mathcal{F})$$

Proof. A field has no zero divisors, thus its characteristic has no proper factor. □

2.6.1 Field Orders

A finite field $\mathcal{F}$ is an integral domain, with all nonzero elements in F^* having nonzero orders. The set of elements with order equal to n is denoted by $(\mathsf{orders}\ \mathcal{F}^*\ n)$. Its cardinality is related to the Euler's φ-function, $\varphi(n)$, counting the number of coprimes from 1 to n:

Theorem 27. *In a finite field $\mathcal{F}$, the number of elements with order n is $\varphi(n)$ if n divides $|\mathsf{F}^*|$, otherwise 0.*

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \Rightarrow \forall n.\ |\mathsf{orders}\ \mathcal{F}^*\ n| = \mathbf{if}\ n \mid |\mathsf{F}^*|\ \mathbf{then}\ \varphi(n)\ \mathbf{else}\ 0$$

Proof. [7] In a finite field $\mathcal{F}$, the multiplicative group $\mathcal{F}^*$ is finite. Let $q = |\mathsf{F}^*|$, then every element has a nonzero order that divides q (Theorem 16). Thus $\mathsf{orders}\ \mathcal{F}^*\ 0 = \emptyset$, and $|\mathsf{orders}\ \mathcal{F}^*\ 0| = 0$. For $n > 0$, first observe that field elements of the same order are in power form of each other:

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \Rightarrow \forall x\ y.\ x \in \mathsf{F} \wedge y \in \mathsf{F} \wedge \mathsf{order}_{\mathcal{F}^*}(x) = \mathsf{order}_{\mathcal{F}^*}(y) \Rightarrow \exists j.\ y = x^j$$

This is because both elements are roots of $\mathsf{X}^k - \mathbf{1}$, where k is their common order. Let z be an field element of order k, then $z, z^2, \ldots, z^k$ are all roots of $\mathsf{X}^k - \mathbf{1}$, which can have at most k roots (Theorem 25). Thus z is a generator of the multiplicative cyclic group of the roots of $\mathsf{X}^k - \mathbf{1}$, denoted by $(\mathsf{Generated}\ \mathcal{F}^*\ z)$:

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \Rightarrow \forall z.\ z \in \mathsf{F} \wedge z \neq \mathbb{0} \Rightarrow \mathsf{roots}\ \left(\mathsf{X}^{\mathsf{order}_{\mathcal{F}^*}(z)} - \mathbf{1}\right) = (\mathsf{Generated}\ \mathcal{F}^*\ z).\mathsf{carrier}$$

Moreover, the order of a power (Theorem 11) for a field element is:

$$\vdash \mathsf{Field}\ \mathcal{F} \Rightarrow \forall x.\ x \in \mathsf{F} \Rightarrow \forall n.\ 0 < n \Rightarrow \mathsf{order}_{\mathcal{F}^*}(x^n) = \mathsf{order}_{\mathcal{F}^*}(x)\ \mathsf{div}\ \gcd(n, \mathsf{order}_{\mathcal{F}^*}(x))$$

[7] This proof, based on Gauss' sum, is adapted from [McEliece, 1987, Theorem 5.7].

This means that if orders $\mathcal{F}^*\ n$ is nonempty with an element x, the elements in the set are precisely those x^k where k is coprime to n. Thus $|\text{orders } \mathcal{F}^*\ n| = \varphi(n)$, the count of coprimes to n not exceeding n.

We claim that orders $\mathcal{F}^*\ n$ is nonempty whenever $n \mid q$. Suppose the contrary, that $n \mid q$ but orders $\mathcal{F}^*\ n = \emptyset$. Let $u =$ divisors q, $v = \{d \in \text{divisors } q \mid \text{orders } \mathcal{F}^*\ d \neq \emptyset\}$. Then $n \notin v$, so that v is a proper subset of u. Therefore, by $v \subset u$,

$$\sum_{d \in v} \varphi(d) < \sum_{d \in u} \varphi(d)$$

Note that every field element has an order that divides q, and the sets orders $\mathcal{F}^*\ d$ for various divisor d of q form a partition of F^*. Since each value $d \in v$ has $|\text{orders } \mathcal{F}^*\ d| = \varphi(d)$, the left-hand side is:

$$\sum_{d \in v} \varphi(d) = \sum_{d \in v} |\text{orders } \mathcal{F}^*\ d| = \left| \bigcup_{d \in v} \text{orders } \mathcal{F}^*\ d \right| = |\mathsf{F}^*| = q$$

and using an identity for Euler's φ-function, the right-hand side is:

$$\sum_{d \in u} \varphi(d) = \sum_{d \mid q} \varphi(d) = q \quad \text{by Theorem 58 near the end.}$$

Combining both sides, we have $q < q$, a contradiction. Hence orders $\mathcal{F}^*\ n \neq \emptyset$. □

2.6.2 Cyclic Multiplicative Group

This is a fundamental feature about finite fields:

Theorem 28. *The multiplicative group of a finite field is cyclic.*

$$\vdash \text{FiniteField } \mathcal{F} \Rightarrow \text{cyclic } \mathcal{F}^*$$

Proof. [8] By Theorem 27, the set orders $\mathcal{F}^*\ |\mathsf{F}^*| \neq \emptyset$. Thus there exists a field element of order $|\mathsf{F}^*|$, *i.e.*, a generator of $\mathcal{F}^*$, making it cyclic. □

2.6.3 Primitives

A generator of the cyclic group $\mathcal{F}^*$ of the finite field $\mathcal{F}$ is called a *primitive* of the finite field:

Definition 29. *A primitive of a finite field $\mathcal{F}$ is an element z with order $|\mathsf{F}^*|$.*

$$z \in \text{orders } \mathcal{F}^*\ |\mathsf{F}^*| \stackrel{\text{def}}{=} z \in \mathsf{F}^* \wedge \text{order}_{\mathcal{F}^*}(z) = |\mathsf{F}^*|$$

By Theorem 27, there are $\varphi(|\mathsf{F}^*|)$ primitives for a finite field $\mathcal{F}$. We shall see the role played by primitives in Section 4.3.2 about isomorphic fields.

[8] This proof, based on order of field elements, is adapted from [McEliece, 1987, Corollary of Theorem 5.7].

2.7 Number Theory

The abstract concepts of algebraic structures have some concrete applications, one of which concerns the modular systems in our next topic. This is relevant to the AKS algorithm because the introspective checks (see Section 1.3, Equation 1.1) are polynomial computations with double moduli $(\mathrm{mod}\ n, \mathsf{X}^k - \mathbf{1})$. We also take this opportunity to walk through some topics in number theory. Some results are well-known, while some are not. They all have a place in understanding the nature of the AKS algorithm.

2.7.1 Modular Systems

Given a modulus n, the remainders under $(\mathrm{mod}\ n)$ form the ring $\mathbb{Z}_n$:

Theorem 30. *For positive n, the ring $\mathbb{Z}_n$ has cardinality n and characteristic n.*

$$\begin{aligned}
&\vdash 0 < n \Rightarrow \mathsf{Ring}\ \mathbb{Z}_n \\
&\vdash |\mathbb{Z}_n.\mathsf{carrier}| = n \\
&\vdash 0 < n \Rightarrow \mathsf{char}(\mathbb{Z}_n) = n
\end{aligned}$$

Homomorphism is a structure-preserving map between two algebraic structures of the same type: between monoids, groups, rings, or fields. Here is a homomorphism between two rings, indicated by $(\mapsto_r)$:

Theorem 31. *If m is a factor of n, then $(\mathrm{mod}\ m)$ is a homomorphism between the rings $\mathbb{Z}_n$ and $\mathbb{Z}_m$.*

$$\vdash 0 < n \wedge m \mid n \Rightarrow (\lambda x.\ x\ \mathsf{mod}\ m) : \mathbb{Z}_n \mapsto_r \mathbb{Z}_m$$

Proof. Since n is a multiple of m, taking $(\mathrm{mod}\ n)$ then $(\mathrm{mod}\ m)$ is the same as taking $(\mathrm{mod}\ m)$ once. Therefore we have, for two x, y less than n:

$$\begin{aligned}
&(x + y)\ \mathsf{mod}\ n\ \mathsf{mod}\ m = (x + y)\ \mathsf{mod}\ m = (x\ \mathsf{mod}\ m + y\ \mathsf{mod}\ m)\ \mathsf{mod}\ m \\
&xy\ \mathsf{mod}\ n\ \mathsf{mod}\ m = xy\ \mathsf{mod}\ m = (x\ \mathsf{mod}\ m)(y\ \mathsf{mod}\ m)\ \mathsf{mod}\ m
\end{aligned}$$

Thus the map preserves both modular addition and multiplication, hence a homomorphism. □

This result is used later in the proof of introspective shifting (Theorem 71, page 52).

Given two numbers m, n, their greatest common divisor $d = \gcd(m, n)$ can be expressed as a linear combination of m and n, by Bézout's identity:

$$\vdash 0 < m \Rightarrow \exists p\ q.\ pm = qn + \gcd(m, n)$$

If $\gcd(m, n) = 1$, then m has a multiplicative inverse p in $(\mathrm{mod}\ n)$. Therefore,

Definition 32. *The units in the ring $\mathbb{Z}_n$ are the elements coprime to n.*

$$\begin{aligned}
&\mathbb{Z}_n^* \stackrel{\mathsf{def}}{=} \\
&\quad \langle\!|\mathsf{carrier} := \{\ x \mid 0 < x \wedge x < n \wedge \gcd(n, x) = 1\ \};\ \mathsf{id} := 1;\ \mathsf{op} := (\lambda x\ y.\ xy\ \mathsf{mod}\ n)|\!\rangle
\end{aligned}$$

Since $\varphi(n)$ is the count of coprimes up to n, we have:

Theorem 33. *The group $\mathbb{Z}_n^*$ has cardinality $\varphi(n)$. Each coprime to n has an order that divides $\varphi(n)$.*

$$\vdash 1 < n \Rightarrow \mathsf{Group}\ \mathbb{Z}_n^* \qquad \text{by Theorem 19}$$
$$\vdash 1 < n \Rightarrow |\mathbb{Z}_n^*.\mathsf{carrier}| = \varphi(n) \qquad \text{by definitions}$$
$$\vdash 0 < k \wedge \gcd(k,n) = 1 \Rightarrow \mathsf{order}_k(n) \mid \varphi(k) \qquad \text{by Theorem 16}$$

The last one shows the order divides the cardinality, giving the analogue of Theorem 17:

Theorem 34. *(Euler's Totient Theorem). Given a modulus n and a coprime to n, raising a to exponent $\varphi(n)$ gives 1 in $(\mathrm{mod}\ n)$.*

$$\vdash 1 < n \wedge \gcd(a,n) = 1 \Rightarrow a^{\varphi(n)} \equiv 1\ (\mathrm{mod}\ n)$$

and the Fermat's Little Theorem:

Corollary 35. *Given a prime p, raising any number a to exponent p gives a in $(\mathrm{mod}\ p)$.*

$$\vdash \mathsf{prime}\ p \Rightarrow a^p \equiv a\ (\mathrm{mod}\ p)$$

Proof. For a prime p, $\varphi(p) = p - 1$. We have $a^{p-1} \equiv 1\ (\mathrm{mod}\ p)$ by Theorem 34. Multiply both side by a gives the desired result. □

For another derivation based on Pascal's Triangle, see Theorem 38.

The modulo arithmetic of polynomials is patterned after the modulo arithmetic of numbers. In Section 2.5, we have introduced the polynomial ring $\mathcal{R}[\mathsf{X}]$. Picking a monic polynomial h with $\deg\ \mathsf{h} > 1$ as modulus (see criteria for polynomial modulus in Section 2.5.1), the polynomial remainders after division by h will have a degree less than $\deg\ \mathsf{h}$. With polynomial addition and multiplication results always reduced by the modulus h, these polynomial remainders after division by h form a quotient ring $\mathcal{R}[\mathsf{X}]/(\mathsf{h})$.

We shall see that for a prime p, $\mathbb{Z}_p$ is a finite field (Theorem 80, page 61), and for an irreducible h, the quotient ring becomes a quotient field (Theorem 81, page 62).

2.7.2 Pascal's Triangle and Primes

In the Pascal's Triangle, if the n-th row is replaced by remainders after division by n, for $n > 1$, those rows with all zeroes, except the first and the last, are precisely those with prime n.
Since remainder zero in $(\mathrm{mod}\ n)$ is equivalent to divisible by n, this feature of the modular Pascal's Triangle can be stated as:

Theorem 36. *Characterisation of a prime by binomial divisibility.*

$$\vdash \mathsf{prime}\ n \iff 1 < n \wedge \forall k.\ 0 < k \wedge k < n \Rightarrow n \mid \binom{n}{k}$$

$n=0$	1									—	1								
$n=1$	1	1								—	1	1							
$n=2$	1	2	1							(mod 2)	1	0	1						
$n=3$	1	3	3	1						(mod 3)	1	0	0	1					
$n=4$	1	4	6	4	1					(mod 4)	1	0	2	0	1				
$n=5$	1	5	10	10	5	1				(mod 5)	1	0	0	0	0	1			
$n=6$	1	6	15	20	15	6	1			(mod 6)	1	0	3	2	3	0	1		
$n=7$	1	7	21	35	35	21	7	1		(mod 7)	1	0	0	0	0	0	0	1	
$n=8$	1	8	28	56	70	56	28	8	1	(mod 8)	1	0	4	0	6	0	4	0	1

Figure 2.1: Pascal's Triangles: the one on the left shows the binomial coefficients, the one on the right has each row depicting the remainders under division by the corresponding row index. The colored rows, with remainders in-between all equal to zero, have row indexes that are *prime*.

Proof. Recall that the binomial coefficient $\binom{n}{k}$ is the number of ways to choose k elements from a set of n elements. Thus for $0 < k < n$:

$$\binom{n}{k} = \frac{n(n-1)\dots(n-k+1)}{k!} \tag{2.3}$$

For the if part $(\Rightarrow)$, n is prime. Rearrange (2.3) as:

$$\binom{n}{k} k! = n(n-1)\dots(n-k+1)$$

Clearly, n divides the right-hand side, so n must also divide the left-hand side. But a prime n cannot divide into $k!$ with $k < n$, so $n \mid \binom{n}{k}$.

For the only-if part $(\Leftarrow)$, assume that $n \mid \binom{n}{k}$ for $0 < k < n$. By contradiction, suppose n is not a prime. Then n has a proper prime factor p. Rearrange this specific binomial coefficient:

$$\binom{n}{p} = n\left[\frac{(n-1)\ \dots\ (n-p+1)}{p(p-1)!}\right]$$

As p is a proper factor of n, $1 < p < n$, so $n \mid \binom{n}{p}$ by assumption. Thus the term inside the square brackets has to be an integer. That requires the prime p to divide one of the numbers in the numerator. This is impossible because when n is a multiple of p, the nearest multiple preceding n that p can divide is $(n-p)$, yet all numbers in the numerator are in between. □

Corollary 37. *If rows of the Pascal's Triangle are taken as remainders of row index, only those*

rows with prime index will have zeroes except first and last.

$$\vdash \mathsf{prime}\ n \iff 1 < n \wedge \forall k.\ 0 < k \wedge k < n \Rightarrow \binom{n}{k} \equiv 0 \pmod{n}$$

All these lead to this nice gem:

Theorem 38. *Fermat's Little Theorem.*

$$\vdash \mathsf{prime}\ p \Rightarrow c^p \equiv c \pmod{p}$$

Proof. By induction on c. The base case $c = 0$ is trivial. Assuming that $c^p \equiv c \pmod{p}$, then

$$(c+1)^p \equiv c^p + 1 \equiv c + 1 \pmod{p}$$

The first congruence comes from binomial expansion with exponent a prime p (Theorem 36). □

Theorem 39. *Fermat's Little Theorem for monomials* $\mathsf{X} + c$.

$$\vdash \mathsf{prime}\ p \Rightarrow (\mathsf{X} + c)^p \equiv \mathsf{X}^p + c \pmod{p}$$

Proof. For a prime p, the binomial expansion of the left-hand side in $\pmod{p}$ will leave only the first and last terms (Theorem 36):

$$(\mathsf{X} + c)^p \equiv \mathsf{X}^p + c^p \pmod{p}$$

Also for a prime p, $c^p = c$ by Fermat's Little Theorem (Corollary 35). The result follows. □

This result is used in showing that the introspective limit is irrelevant for a prime (Theorem 69, page 50), and the characteristic of a finite field is an introspective exponent for monomials $\mathsf{X} + c$ (Theorem 98, page 77).

Another nice result concerning binomial expansion is this:

Theorem 40. *In a ring with characteristic prime, the binomial expansion with characteristic exponent for polynomials has a special form.*

$$\vdash \mathsf{Ring}\ \mathcal{R} \wedge \mathsf{prime}\ \mathsf{char}(\mathcal{R}) \Rightarrow \forall \mathsf{p}\ \mathsf{q}.\ \mathsf{poly}\ \mathsf{p} \wedge \mathsf{poly}\ \mathsf{q} \Rightarrow (\mathsf{p} + \mathsf{q})^{\mathsf{char}(\mathcal{R})} = \mathsf{p}^{\mathsf{char}(\mathcal{R})} + \mathsf{q}^{\mathsf{char}(\mathcal{R})}$$
$$\vdash \mathsf{Ring}\ \mathcal{R} \wedge \mathsf{prime}\ \mathsf{char}(\mathcal{R}) \Rightarrow \forall \mathsf{p}\ \mathsf{q}.\ \mathsf{poly}\ \mathsf{p} \wedge \mathsf{poly}\ \mathsf{q} \Rightarrow (\mathsf{p} - \mathsf{q})^{\mathsf{char}(\mathcal{R})} = \mathsf{p}^{\mathsf{char}(\mathcal{R})} - \mathsf{q}^{\mathsf{char}(\mathcal{R})}$$

Proof. Let $p = \mathsf{char}(\mathcal{R})$, and $n.\mathbb{1}$ be the addition of n copies of $\mathbb{1}$, the multiplicative identity. Then $n.\mathbb{1} = \mathbb{0}$ whenever n is a multiple of p, by definition of characteristic (Definition 18). The first one is due to the vanishing of binomial coefficients in $\pmod{p}$ (Theorem 36) upon binomial expansion of $(\mathsf{p} + \mathsf{q})^p$. Using the first one,

$$(\mathsf{p} - \mathsf{q})^p + \mathsf{q}^p = (\mathsf{p} - \mathsf{q} + \mathsf{q})^p = \mathsf{p}^p$$

A simple rearrangement gives the second one. □

Since the characteristic of a finite field is prime (Theorem 26), the above results are applicable to finite fields. The second result is used in the proof of Theorem 102, where a cofactor is shown to be an exponent for the introspective relation.

2.7.3 Generalised Fermat's Little Theorem

Note that both forms of Fermat's Little Theorem (Theorems 38 and 39) are not primality tests. However, the following generalisation is:

Theorem 41. *Primality Test based on Fermat's Little Theorem for monomials* $\mathsf{X} + c$.

$$\vdash 1 < n \wedge \gcd(c, n) = 1 \Rightarrow (\mathsf{prime}\ n \iff (\mathsf{X} + c)^n \equiv \mathsf{X}^n + c \pmod{n})$$

Proof. Using binomial expansion, the left-hand side is:

$$(\mathsf{X} + c)^n = \sum_{k=0}^{n} \binom{n}{k} c^k \mathsf{X}^{n-k}$$

Equating coefficients on both sides of $(\mathsf{X} + c)^n \equiv \mathsf{X}^n + c \pmod{n}$, the first terms from $k = 0$ are equal, and the last terms from $k = n$ are equal by Fermat's Little Theorem (Theorem 38). For the terms in between, note that $\gcd(c, n) = 1$ implies $c^k \not\equiv 0 \pmod{n}$, and, with $1 < n$, $\binom{n}{k} \equiv 0 \pmod{n}$ is equivalent to $n \mid \binom{n}{k}$. Thus the proof reduces to showing:

$$\mathsf{prime}\ n \iff n \mid \binom{n}{k} \qquad \text{for } 0 < k < n.$$

This is true by Theorem 36. □

This result resembles the introspective checks in Section 1.3, Equation 1.1. The difference is that this one is just $\pmod{n}$, while the introspectie checks have two moduli for $\pmod{n, \mathsf{X}^k - 1}$. As a primality test, this result is deterministic but not in polynomial-time, as the number of terms to check is of order $\mathcal{O}(n)$.

2.7.4 Consecutive LCM

The concept of the least common multiple (lcm) for two numbers can be readily extended to a list ℓ of numbers, denoted by $\mathsf{LCM}\ \ell$:

Definition 42. *The least common multiple of a numeric list, defined recursively.*

$$\begin{aligned} \mathsf{LCM}\ [\,] &\stackrel{\text{def}}{=} 1 \\ \mathsf{LCM}\ (h::t) &\stackrel{\text{def}}{=} \mathsf{lcm}\ h\ (\mathsf{LCM}\ t) \end{aligned}$$

The list ℓ should be a list of positives, denoted by POSITIVE ℓ, since zero has no common multiple with any nonzero. We can derive:

Lemma 43. *Some properties of* LCM *for lists.*

$$\vdash \mathsf{LCM}\ (l_1 +\!\!+ l_2) = \mathsf{lcm}\ (\mathsf{LCM}\ l_1)\ (\mathsf{LCM}\ l_2)$$
$$\vdash \mathsf{LCM}\ (\mathsf{REVERSE}\ \ell) = \mathsf{LCM}\ \ell$$
$$\vdash (\forall x.\ \mathsf{MEM}\ x\ \ell \Rightarrow x \mid m) \Rightarrow \mathsf{LCM}\ \ell \mid m$$
$$\vdash \mathsf{POSITIVE}\ \ell \Rightarrow \mathsf{SUM}\ \ell \leq \mathsf{LENGTH}\ \ell \times \mathsf{LCM}\ \ell$$

The third one just asserts that LCM ℓ is indeed the least common multiple of all list members: any common multiple m of the members is divisible by LCM ℓ. We shall establish the last one slightly differently to show how to get a lower bound for LCM ℓ:

Theorem 44. *The least common multiple of a non-empty positive list cannot be less than the integer average of its elements.*

$$\vdash \ell \neq [\,] \wedge \mathsf{POSITIVE}\ \ell \Rightarrow \mathsf{SUM}\ \ell\ \mathsf{div}\ \mathsf{LENGTH}\ \ell \leq \mathsf{LCM}\ \ell$$

Proof. Let z = LENGTH ℓ, and m = LCM ℓ, the least common multiple of its members. Each nonzero member $x_i \mid m$, so $x_i \leq m$. Then:

$$\mathsf{SUM}\ \ell = \sum_{i=1}^{z} x_i \ \leq\ \sum_{i=1}^{z} m = zm$$

With $\ell \neq [\,]$, its length $z \neq 0$. A simple integer division by z gives the desired result. □

Let $\ell = [1\ ..\ n]$, then LCM ℓ is the least common multiple of consecutive numbers up to n. This is of interest in the study of the AKS algorithm because a lower bound on LCM $[1\ ..\ n]$ gives a bound on the AKS parameter. This is essentially the first lemma invoked in the paper [Agrawal et al., 2004, Lemma 3.1]:

$$\vdash 2^n \leq \mathsf{LCM}\ [1\ ..\ n+1]$$

We shall sketch a proof of this result (Theorem 52) using the Leibniz Harmonic Triangle in denominator form. For the details, refer to Chan and Norrish [2019b].

2.7.5 From Pascal to Leibniz

For each row of the Pascal's Triangle, if we multiply the entry values by the number of elements in that row, we can form another number triangle (Figure 2.2). The transformed Pascal's Triangle is called Leibniz's Denominator Triangle, because it consists of all denominators of the usual Leibniz's Harmonic Triangle, shown on the right of Figure 2.3.

Recall that Pascal's Triangle has a building rule: each entry not on the boundary is the sum of its immediate parents. A similar building rule exists for Leibniz's Triangles, involving an entry and its immediate children – a triplet, a typical one is highlighted in Figure 2.3. Such a

					1						×1 ⇒	1						
				1		1					×2 ⇒	2	2					
			1		2		1				×3 ⇒	3	6	3				
		1		3		3		1			×4 ⇒	4	12	12	4			
	1		4		6		4		1		×5 ⇒	5	20	30	20	5		
1		5		10		10		5		1	×6 ⇒	6	30	60	60	30	6	
1		6		15		20		15		6	1	×7 ⇒	7	42	105	140	105	42	7

Figure 2.2: Pascal's Triangle to Leibniz's Denominator Triangle.

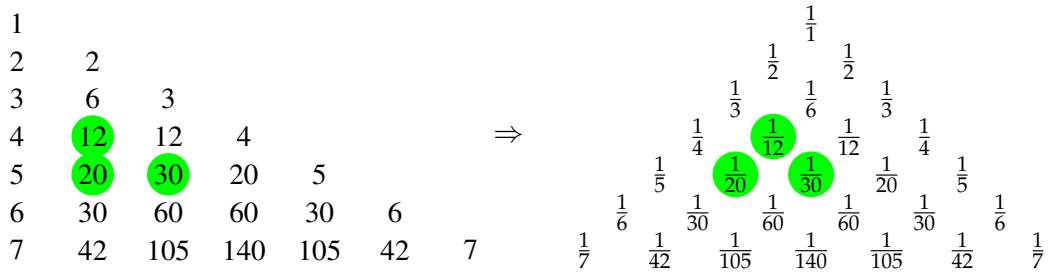


Figure 2.3: Leibniz's Denominator Triangle and Harmonic Triangle.

triplet, with denominators $a, b,$ and c, is called a Leibniz triplet, denoted by $\binom{a}{bc}$. Its basic property is:

$$\binom{a}{bc} : \quad \frac{1}{a} = \frac{1}{b} + \frac{1}{c} \tag{2.4}$$

which reflects that, in the Harmonic Triangle, each entry is the sum of its immediate children. To establish this, we need to set up a formal theory for the Denominator Triangle.

First, it is quite easy to formalise the Pascal's Triangle. Let $\mathcal{P}_{\text{row}}\ n$ denote the list of binomial coefficients along the n-th row. A basic property is this:

Theorem 45. *The sum across the n-th row of the Pascal's Triangle is* 2^n.

$$\vdash \mathsf{SUM}\ (\mathcal{P}_{\text{row}}\ n) = 2^n$$

Proof.

$$\mathsf{SUM}\ (\mathcal{P}_{\text{row}}\ n) = \sum_{k=0}^{n} \binom{n}{k} = (1+1)^n = 2^n \tag{2.5}$$

□

2.7.6 Leibniz Denominator Triangle

Let $\mathcal{L}$ denotes Leibniz's Denominator Triangle, as shown in Figure 2.4, with the entry $\mathcal{L}_{n,k}$ at n-th row, k-th column defined *via* the binomial coefficients (see Figure 2.2):

Definition 46. *Entries of the Leibniz Denominator Triangle by Figure 2.2.*

$$\mathcal{L}_{n,k} \stackrel{\text{def}}{=} (n+1)\binom{n}{k}$$

Note that $\mathcal{L}_{n,k} = 0$ when $k > n$, since in this case $\binom{n}{k} = 0$ by definition. Indeed, expressing the binomial coefficient in terms of factorials, we have:

$$\vdash k \leq n \Rightarrow \mathcal{L}_{n,k} = \frac{(n+1)n!}{k!(n-k)!} \tag{2.6}$$

$n = 0$	1					
$n = 1$	2	2				
$n = 2$	3	6	3			
$n = 3$	4	12	12	4		
$n = 4$	5	20	30	20	5	
$n = 5$	6	30	60	60	30	6
$k =$	0	1	2	3	4	5

Figure 2.4: Leibniz's Denominator Triangle

Evidently from Definition 46 and Figure 2.4, the left boundary of $\mathcal{L}$ (highlighted vertical) consists of consecutive numbers, from the denominators of the harmonic sequence:

$$\vdash \mathcal{L}_{n,0} = n + 1$$

Denoting the n-th row (highlighted horizontal) by $\mathcal{L}_{\text{row}}\ n$, we have:

Theorem 47. *In Leibniz's Denominator Triangle, the integer average of the n-th row is 2^n.*

$$\vdash \mathsf{SUM}\ (\mathcal{L}_{\text{row}}\ n)\ \mathsf{div}\ \mathsf{LENGTH}\ (\mathcal{L}_{\text{row}}\ n) = 2^n$$

Proof. From Definition 46, $(\mathcal{L}_{\text{row}}\ n)$ is just a multiple of $(\mathcal{P}_{\text{row}}\ n)$ by a factor of $(n+1)$, giving:

$$\vdash \mathsf{SUM}\ (\mathcal{L}_{\text{row}}\ n) = (n+1)(\mathsf{SUM}\ (\mathcal{P}_{\text{row}}\ n))$$
$$\vdash \mathsf{LENGTH}\ (\mathcal{L}_{\text{row}}\ n) = n + 1$$

The result follows by applying the binomial sum formula of Equation 2.5. □

This is a useful result because in Figure 2.4, as we shall demonstrate, the LCM of the vertical list is the same as the LCM of the horizontal list. By Theorem 44, the LCM of the horizontal list is bounded by its row average, and we have just proved that its row average is 2^n.

2.7.7 LCM Exchange

Within the vertical-horizontal format of the Denominator Triangle (Figure 2.5), we identify L-shaped "Leibniz triplets" (a typical one is marked) rooted at row n and column k, with 3 entries:

- the root of the triplet: $\alpha_{nk} = \mathcal{L}_{n,k}$ and,
- its two children on the next row: $\beta_{nk} = \mathcal{L}_{n+1,k}$, $\gamma_{nk} = \mathcal{L}_{n+1,k+1}$

Figure 2.5: The Leibniz triplet: in Denominator Triangle and in Harmonic Triangle.

Such a Leibniz triple is denoted by $\binom{\alpha_{nk}}{\beta_{nk}\gamma_{nk}}$. Our definition of $\mathcal{L}_{n,k}$ shows that a Leibniz triple satisfies Equation 2.4, or express linearly by clearing fractions:

Theorem 48. *An identity for a Leibniz triplet* $\binom{\alpha_{nk}}{\beta_{nk}\gamma_{nk}}$ *in the Denominator Triangle.*

$$\vdash \alpha_{nk}\beta_{nk} = \gamma_{nk}(\beta_{nk} - \alpha_{nk})$$

Proof. The vertical and horizontal pairs of the triplet have these relationships by Definition 46:

$$\begin{aligned} &\vdash (n+2)\alpha_{nk} = (n+1-k)\beta_{nk} && \text{vertical pair} \\ &\vdash (k+1)\gamma_{nk} = (n+1-k)\beta_{nk} && \text{horizontal pair} \end{aligned}$$

If $k > n$, these equations (with truncated natural number subtraction) show that both α_{nk} and γ_{nk}

are zero, and the identity is trivially true. Otherwise,

$$\begin{aligned}
& (k+1)\alpha_{nk}\beta_{nk} \\
= \; & (n+2-(n+1-k))\alpha_{nk}\beta_{nk} && \text{by } k \le n \\
= \; & (n+2)\alpha_{nk}\beta_{nk} - (n+1-k)\alpha_{nk}\beta_{nk} && \text{by distribution} \\
= \; & (n+1-k)\beta_{nk}\beta_{nk} - (n+1-k)\alpha_{nk}\beta_{nk} && \text{by vertical pair} \\
= \; & (n+1-k)\beta_{nk}\beta_{nk} - (n+1-k)\beta_{nk}\alpha_{nk} && \text{by commutativity} \\
= \; & (n+1-k)\beta_{nk}(\beta_{nk} - \alpha_{nk}) && \text{by distribution} \\
= \; & (k+1)\gamma_{nk}(\beta_{nk} - \alpha_{nk}) && \text{by horizontal pair}
\end{aligned}$$

Since $k+1 \neq 0$, the result follows by factor cancellation. □

There is a remarkable property for a Leibniz triplet:

Theorem 49. *In a Leibniz triplet* $\begin{pmatrix} \alpha_{nk} \\ \beta_{nk}\gamma_{nk} \end{pmatrix}$*, the vertical and horizontal pairs share the same least common multiple.*

$$\vdash \mathsf{lcm}\ \beta_{nk}\ \alpha_{nk} = \mathsf{lcm}\ \beta_{nk}\ \gamma_{nk}$$

Proof. Let $a = \alpha_{nk}$, $b = \beta_{nk}$, and $c = \gamma_{nk}$ form a Leibniz triplet, with b at the corner. If one (or more) of these entries is outside the usual range, the desired LCM exchange is trivially true:[9]

- if $b = 0$, then both $a = 0$ and $c = 0$.
- if $b \neq 0$, but $a = 0$ then $c = 0$, and *vice versa*.

Otherwise, all a, b, and c are nonzero. Note that $ab = c(b-a)$ by Theorem 48. Therefore,[10]

$$\begin{aligned}
& \mathsf{lcm}\ b\ c \\
= \; & bc \div \mathsf{gcd}(b,c) && \text{by property of LCM} \\
= \; & abc \div (a\ \mathsf{gcd}(b,c)) && \text{introduce factor } a \text{ above and below division} \\
= \; & bac \div \mathsf{gcd}(ab,ca) && \text{by common factor } a\text{, commutativity} \\
= \; & bac \div \mathsf{gcd}(c(b-a),ca) && \text{by Leibniz triplet identity (Theorem 48)} \\
= \; & bac \div (c\ \mathsf{gcd}(b-a,a)) && \text{extract common factor } c \\
= \; & ba \div \mathsf{gcd}(b,a) && \text{apply GCD subtraction and cancel factor } c \\
= \; & \mathsf{lcm}\ b\ a && \text{by property of LCM.}
\end{aligned}$$

□

Therefore we can exchange the LCM computation from vertical to horizontal.

[9] For any number n, $\mathsf{lcm}\ 0\ n = \mathsf{lcm}\ n\ 0 = 0$.

[10] Here, all divisions ($\div$) give integers, as the denominator divides the numerator.

2.7.8 LCM of Paths

Within the Denominator Triangle, a path is composed of entries. We treat paths as lists of numbers, without requiring the path to be connected. However, the paths we work with will be connected (refer to Figure 2.4) and include:

Definition 50. *Paths in Leibniz Denominator Triangle.*

- $(\mathcal{L}_{\text{down}}\ n)$*: the list* $[1\ ..\ n+1]$*, which happens to be the first* $(n+1)$ *elements of the leftmost column of the Denominator Triangle, reading downwards;*
- $(\mathcal{L}_{\text{up}}\ n)$*: the reverse of* $(\mathcal{L}_{\text{down}}\ n)$*, or the leftmost column of the triangle reading upwards; and*
- $(\mathcal{L}_{\text{row}}\ n)$*: the n-th row of the Denominator Triangle, reading from the left to right.*

Due to the LCM exchange within a Leibniz triplet (Theorem 49), we can prove the following:

Theorem 51. *In the Denominator Triangle, consider the first element (at the left boundary) of the n-th row. Then the least common multiple of the column of elements above it is equal to the least common multiple of elements along its row.*

$$\vdash \mathsf{LCM}\ (\mathcal{L}_{\text{down}}\ n) = \mathsf{LCM}\ (\mathcal{L}_{\text{row}}\ n)$$

The proof is done *via* a kind of zig-zag transformation, see Figure 2.6. In the Denominator Triangle, we represent the entries for LCM consideration as a path of black discs, and indicate the Leibniz triplets by discs marked with small gray dots. Recall that, by Theorem 49, the vertical pair of a Leibniz triplet can be swapped with its horizontal pair without affecting the least common multiple. This allows a path of black discs to zigzag to another, keeping the LCM of the whole path. By following the path transform at each step by the indicated Leibniz triplet, we can see that as the path formed by the black discs wriggles from being vertical in Step 1 to being horizontal at Step 7. Due to LCM exchange of a Leibniz triplet, the path LCM is kept unchanged, leading to an alternative way to estimate the consecutive LCM.

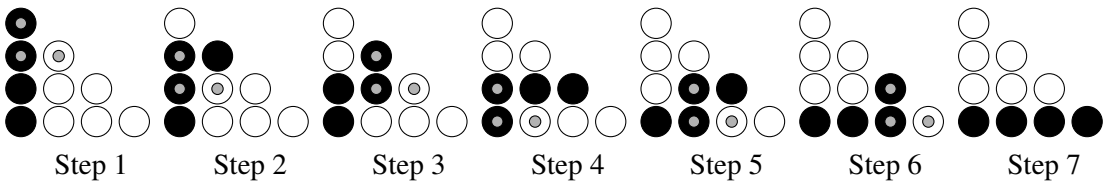


Figure 2.6: Transformation of a path from vertical to horizontal in the Denominator Triangle, stepping from left to right. The path is indicated by entries with black discs. The 3 gray-dotted discs in L-shape indicate the Leibniz triplet, which allows LCM exchange. Each step preserves the overall LCM of the path. Hence the black discs of Step 1 and of Step 7 have the same LCM.

After formalising zigzag paths and wriggle paths (see Chan and Norrish [2016]), we can prove:

Theorem 52. *A lower bound for the LCM of consecutive numbers from* 1 *to* $(n+1)$ *is* 2^n.

$$\vdash 2^n \leq \mathsf{LCM}\ [1\ ..\ n+1]$$

Proof.

$$\begin{array}{lll}
 & \mathsf{LCM}\ [1\ ..\ n+1] & \\
= & \mathsf{LCM}\ (\mathcal{L}_{\mathrm{down}}\ n) & \text{by path notation (Definition 50)} \\
= & \mathsf{LCM}\ (\mathcal{L}_{\mathrm{row}}\ n) & \text{by LCM transform (Theorem 51)} \\
\geq & \mathsf{SUM}\ (\mathcal{L}_{\mathrm{row}}\ n)\ \mathsf{div}\ \mathsf{LENGTH}\ (\mathcal{L}_{\mathrm{row}}\ n) & \text{by average lower bound (Theorem 44)} \\
= & 2^n & \text{by denominator row average (Theorem 47)}
\end{array}$$

□

This result is used to establish the bound on the AKS parameter (Theorem 65, page 47).

2.7.9 Powers and Coprimes

The following basic theorems are useful in providing the groundwork for the AKS proof.

The predicate n power_of b is defined as:

Definition 53. *A number* n *is a power of base* b *when* $n = b^e$ *for some exponent* e.

$$n\ \mathsf{power\_of}\ b \stackrel{\mathsf{def}}{=} \exists e.\ n = b^e$$

A number n is power-free if it can only be a trivial power of itself. Moreover,

Theorem 54. *Let* p *be a factor of a positive* n. *Then* n *is a power of* p *iff* n *is a power of its cofactor* n div p.

$$\vdash 0 < n \wedge p \mid n \Rightarrow (n\ \mathsf{power\_of}\ p \iff n\ \mathsf{div}\ p\ \mathsf{power\_of}\ p)$$

Proof. Let $q = n$ div p. Then $n = pq$. If n is a power of p, there is a k such that $n = p^k$ (Definition 53). If $k = 0$, then $n = 1$, so $p = q = 1$, and q is a power of p. Otherwise $k \neq 0$, then $k = h + 1$ for some h, so $q = p^h$, and q is a power of p. Conversely, if q is a power of p, then $n = pq$ must be a power of p. □

This is used in the proof of AKS Main Theorem, whereby instead of considering n is not a power of p, we show that the cofactor $q = n$ div p is not a power of p,

There is a simple condition that implies a number must be a power of a prime:

Theorem 55. *Given a prime* p *and a number* q, *if some nonzero power of* p *is a power of* q, *then* q *is a power of the prime* p.

$$\vdash \mathsf{prime}\ p \wedge (\exists x\ y.\ 0 < x \wedge p^x = q^y) \Rightarrow q\ \mathsf{power\_of}\ p$$

Proof. Note that $p^x = q^y$ means $p \mid q$, since p divides the left side, and a prime dividing a power must divide its base. Since $p \mid q$, divide q by p as many times as possible, and express $q = p^m u$ where m is the maximum possible exponent, and $p \nmid u$. The equation $p^x = q^y$ becomes

$$p^x = (p^m u)^y = p^{my} u^y.$$

By unique factorisation, with prime p and $p \nmid u$ and $x \neq 0$, it must be that $y \neq 0$, and $u^y = 1$, *i.e.*, $u = 1$. Thus $q = p^m$, and $m \neq 0$ shows that q is a power of p. □

This result in used in the proof of Theorem 114, when providing an expression for the cardinality of a certain introspective set.

Two numbers are coprime if their greatest common divisor is 1. This property obviously passes onto factors:

Theorem 56. *If m is a factor of n, then any number coprime to n is coprime to m.*

$$\vdash m \mid n \Rightarrow \forall k.\ \gcd(n,k) = 1 \Rightarrow \gcd(m,k) = 1$$

Proof. Let $d = \gcd(m,k)$. Then $d \mid m$ and $d \mid k$. Since $m \mid n$, so $d \mid n$. Hence d is a common factor of n and k, which makes $d = 1$. □

This is used in the proof of AKS Main Theorem, after introucing the cofactor $q = n \text{ div } p$ (page 89).

The following is used in the finding a special prime factor for n:

Theorem 57. *If a number n has a multiplicative order $1 < \text{order}_k(n)$ in modulo k, then among the prime factors of n, there is a prime factor p such that $1 < \text{order}_k(p)$.*

$$\vdash 0 < k \wedge 1 < \text{order}_k(n) \Rightarrow \exists p.\ \text{prime } p \wedge p \mid n \wedge 1 < \text{order}_k(p)$$

Proof. Let p be a prime factor of n, *i.e.*, $p \mid n$. Since $0 < \text{order}_k(n)$, we have $\gcd(k,n) = 1$. Then $\gcd(k,p) = 1$, as any common divisor of k, p is also a common divisor of k, n (Theorem 56). Thus $0 < \text{order}_k(p)$. If $\text{order}_k(p) = 1$, that implies $p \equiv 1 \pmod{k}$. Hence if every prime factor p of n has $\text{order}_k(p) = 1$, then n, as a product of its prime factors, will also have $\text{order}_k(n) = 1$. This contradicts $1 < \text{order}_k(n)$. Therefore some prime factor p of n must have $1 < \text{order}_k(p)$. □

This special prime factor is picked up in Section 3.6, about shifting of the introspective relation.

2.7.10 Phi Sum over Divisors

This is an identity for Euler's φ-function:

Theorem 58. *The sum of $\varphi(d)$ over the divisors d of n equals to n.*

$$\vdash n = \sum \varphi\ (\text{divisors } n) \quad \text{or} \quad n = \sum_{d \mid n} \varphi(d)$$

Proof. Let $S = \text{divisors } n$, and $T = \{1, \ldots, n\}$. Consider the function f for a divisor $d \in S$:

$$f(d) = \{j \mid j \in T \text{ and } \gcd(j, n) = d\}$$

We want to show that distinct divisors have distinct images of f. Let x, y be two divisors of n. Assume that $f(x) = f(y)$. Since $x \mid n$ gives $\gcd(x, n) = x$, $x \in f(x)$. That means $x \in f(y)$, with $\gcd(x, n) = y$, giving $x = y$.

Therefore, the images $f(d)$ of all divisors $d \in S$ form a partition on the set T. Moreover, if $\gcd(j, n) = d$, then $\gcd(j \text{ div } d, n \text{ div } d) = 1$. Define a permutation $\pi(d) = n \text{ div } d$ on the divisors S, then $|f(d)| = \varphi(\pi(d))$. Since π is a permutation, the sum over all $d \in S$ is identical to the sum over all $\pi(d) \in S$. Hence:

$$\sum_{d \mid n} \varphi(d) = \sum_{d \in S} \varphi(d) = \sum_{\pi(d) \in S} \varphi(\pi(d)) = \sum_{d \in S} |f(d)| = \left| \bigcup_{d \in S} f(d) \right| = |T| = n$$

□

This theorem is used in working out the order of elements in a finite field (Theorem 27, page 24), and the cyclotomic factors of $X^k - \mathbf{1}$ (Theorem 94, page 69).

Scripts Properties of monoids are proved in `algebra/monoid`, while properties of groups are proved in `algebra/group`. Similarly, properties of rings are proved in `algebra/ring`, while properties of fields and finite fields are located in `algebra/field`. For properties of the ring $\mathbb{Z}_k$, refer to `algebra/ring/ringInstances`.

Polynomial rings are formulated in `algebra/polynomial/polynomial`, and polynomial quotient rings are formulated in `algebra/polynomial/polyModuloRing`.

The remarkable results from number theory come from `algebra/lib`: `binomial` for binomials and prime, `triangle` for the consecutive LCM, `logPower` for power forms and cofactors, and `Gauss` for the Euler's φ-function.

2.8 Summary

We have covered the basic algebraic concepts required for the AKS algorithm. The hierarchy of algebraic structures has monoids, groups, rings, and fields. Each one is defined and the main properties elucidated. The concept of zero divisors leads to integeral domains, and we show that finite integral domains are finite fields. We use elements from rings or fields as coefficients for polynomials, and look at modular systems that give rise to quotient structures. We visit the Pascal's Triangle for Fermat's Little Theorem, look at the Leibniz Denominator Triangle for a consecutive LCM bound, check out some useful results about coprimes, and derive the sum of Euler's φ-function over the divisors of n. Equipped with this knowledge of basic algebra and number theory, we shall examine the AKS algorithm in detail, and see that it is indeed a primality test.

2.9 Remarks

Many theorem-proving systems have developed algebra libraries, including number theory and abstract algebra. These systems will be described at the end of Chapter 4, after we have covered our advanced algebra library.

Here, we just take an example to indicate the algebra libraries in various systems. These are the applications of finite fields to cryptography, include elliptic curve algorithms and polynomial factorisation algorithms.

In HOL, Joe Hurd et al. [2006] formalised elliptic curve cryptography based on a group structure. They defined algebraic structures for a generic type, but their work was based on concrete instances with numeric type.

In Mizar, Yuichi Futa et al. [2013] formalised some theorems on elliptic curve over a prime field by enriching the Mizar Mathematical Library.

In Coq, Evmorfia-Iro Bartzia and Strub [2014] had extended the Mathematical Components library for elliptic curves based on fields and projective geometry. Reynald Affeldt et al. [2016] reported on the formalisation of Reed-Solomon codes and their work on LDPC codes, using polynomials with coefficients from a field.

In Isabelle/HOL, Jose Divasón et al. [2017] had given a formalisation of the Berlekamp-Zassenhaus factorization algorithm. Their work involved the fields $\mathbb{Z}_p$ where p is a prime, and polynomials over such fields.

Formalisation of results in elementary number theory can be a substantial task, as evident in Asperti and Armentano [2008]. They discussed their work in Matita Interactive Theorem Prover to formalise just one page taken from a standard graduate textbook. Their work offered a proof of the Euler φ-function identity (Theorem 58).

Refer to Chan and Norrish [2012, 2013] for several approaches to formalise Fermat's Little Theorem (Theorem 38), including a combinatoric proof based on necklace counting and a group-theoretic proof based on Orbit-Stabilizer Theorem. In Chan and Norrish [2019b], the Leibniz Denominator Triangle was used to deduce, for the consecutive LCM, both a lower bound and an upper bound.

About the characterisation of a prime by binomial divisibility (Theorem 36), the only-if part is less well-known. In [Dietzfelbinger, 2004, Lemma 8.1.1], [Granville, 2004, Theorem 1] and [Shoup, 2008, Theorem 22.1], the only-if proof proceeds as follows. Let p be a prime factor of a composite n, so $p < n$. Put $n = p^k q$, where $k \geq 1$ is the greatest power of p that divides n. Then we can express:

$$\binom{n}{p} = p^k q \left[\frac{(n-1)\ \ldots\ (n-p+1)}{p(p-1)!}\right] = p^{k-1} q \left[\frac{(n-1)\ \ldots\ (n-p+1)}{(p-1)!}\right]$$

Since n is a multiple of p, the closest lesser number p can divide is $(n-p)$. Thus the numerator in the square brackets cannot have a factor p. This implies $\binom{n}{p}$ is only divisible by p^{k-1}, not

p^k. This guarantees that $n \not| \binom{n}{p}$. Our proof is a variation using the same idea.

Chapter 3

AKS Algorithm

This chapter begins with the pseudocode of the AKS algorithm, then relates the pseudocode to each phase of the algorithm. We shall derive the limit for power free check, prove the existence and bound of the AKS parameter, and investigate the role of introspective checks. It is relatively easy to show that a prime will pass the AKS tests, but the converse takes us onto a pathway meeting rings and fields. By getting hold of a special prime factor, we build up a finite field. We shift the introspective checks to polynomials with coefficients from this finite field. This pathway leads to a formulation of the AKS Main Theorem, the key to complete the proof of the correctness of the AKS algorithm.

The journey of a thousand miles
begins with a single step.
— Lao Tzu (老子) (*circa* 600 BC)[1]

3.1 AKS Pseudocode

Our journey to formalise the AKS algorithm begins with its pseudocode, see Algorithm 1. The pseudocode shows the details of each phase, discussed briefly in Section 1.3. We make use of several subroutines, *e.g.*, ROOT k n for the integer k-th root of n, $\mathsf{order}_k(n)$ for the multiplicative order of n in modulo k, and $\varphi(k)$ for the Euler-phi function of k, the number of coprimes to k not exceeding k. There are 3 FOR-loops, one for each phase. More explanation of these loops, and the justification of loop bounds, will be given in subsequent sections.

For now, note the upper bound for each loop with input n:

- for phase 1, the bound is $\lceil \log n \rceil$.
- for phase 2, the bound is roughly $\lceil \log n \rceil^5$, which gives the bound for parameter k.
- for phase 3, the bound is $s = (\mathsf{SQRT}\ \varphi(k))\lceil \log n \rceil$. As k is within $\lceil \log n \rceil^5$, s is bounded by some power in $\lceil \log n \rceil$.

[1] From chapter 64 of Tao Te Ching (道德經：「千里之行，始於足下。」).

Input: integer $n > 1$.
Output: whether n is PRIME or COMPOSITE.

1. **Power Free Test**
   For each k from 2 to $\lceil \log n \rceil$:
   - If $(\mathsf{ROOT}\ k\ n)^k = n$, return COMPOSITE.
2. **Parameter Search**
   For each k from 2 to $1 + \lceil \log n \rceil^5$ div 2:
   - If $k \mid n$, then if $k = n$, return PRIME, else return COMPOSITE.
   - If $k \geq \lceil \log n \rceil^2$ and $\mathsf{order}_k(n) \geq \lceil \log n \rceil^2$, go to Step 3 with this k.
3. **Introspective Checks**
   For each c from 1 to $(\mathsf{SQRT}\ \varphi(k))\lceil \log n \rceil$:
   - if $(\mathsf{X} + c)^n \not\equiv \mathsf{X}^n + c \pmod{n, \mathsf{X}^k - 1}$, return COMPOSITE.
4. return PRIME.

Algorithm 1: The AKS algorithm in pseudo-code

Input: integer $n > 1$.
Output: whether n is PRIME or COMPOSITE.

1. If ($n = b^m$ for $b \in \mathbb{N}$ and $m > 1$), return COMPOSITE.
2. Find the smallest k satisfying $\mathsf{order}_k(n) \geq \lceil \log n \rceil^2$.
3. For each $(1 < j \leq k)$, if $1 < \gcd(j, n) < n$, return COMPOSITE.
4. If $(k \geq n)$, return PRIME.
5. For each ($c = 1$ to $(\mathsf{SQRT}\varphi(k))\lceil \log n \rceil$)
   if $(\mathsf{X} + c)^n \not\equiv \mathsf{X}^n + c \pmod{n, \mathsf{X}^k - 1}$, return COMPOSITE.
6. return PRIME.

Algorithm 2: The algorithm in AKS revised paper, Agrawal et al. [2004].

Assuming that the subroutines have run-times bounded by polynomials in $\lceil \log n \rceil$, this suggests that the AKS algorithm runs in polynomial time of $\lceil \log n \rceil$. We shall formally prove this result as Theorem 4.

The pseudocode in Algorithm 1 is not optimized for efficiency. It is just simple enough, using only integer arithmetic, to carry out each phase. Our monadic style implementation in HOL4, aksM n closely matches this pseudocode (see Definition 3). This implementation is the basis of our computational complexity analysis of the AKS algorithm.

Our pseudocode differs slightly from the version given by the AKS team (see Algorithm 2), which in step 3 involves the computation of greatest common divisors. This effectively checks whether $d = \gcd(j, n)$ is 1 or n. If d is neither of these, then d is a proper factor of n, so n is composite. In Algorithm 1, we guarantee $\gcd(k, n) = 1$ upon entering step 3. This is because any common factor of k and n, which divides n, must have been encountered in step 2 and be dealt with in the $k \mid n$ check. This keeps our implementation GCD free, and spares us the need to analyse the computational complexity of GCD algorithms.[2]

For a better understanding of the AKS algorithm through the pseudocode, we shall study each phase in detail.

3.2 Power Free Test

Phase 1 of the AKS algorithm is to check whether the input n is power free. A number is power free if it is not a square, not a cube,*etc.*, *i.e.*, not a power, or formally:

Definition 59. *A power free number n has a trivial power form, with exponent* 1.

$$\text{power\_free } n \stackrel{\text{def}}{=} \forall b\ e.\ n = b^e \Rightarrow b = n \wedge e = 1$$

Since a power form b^e has its base b as factor, it is easy to see that:

Theorem 60. *A prime is power free.*

$$\vdash \text{prime } n \Rightarrow \text{power\_free } n$$

Here is one method to determine if a given number is power free:

Theorem 61. *Power free test by integer root and exponentiation.*

$$\vdash \text{power\_free } n \iff 1 < n \wedge \forall j.\ 1 < j \wedge j \leq \lceil \log n \rceil \Rightarrow (\text{ROOT } j\ n)^j \neq n$$

Proof. If $n = (\text{ROOT } j\ n)^j$ with $1 < j$, then n is not power free by Definition 59. We only need testing with j-th integer roots for j up to $\lceil \log n \rceil$, due to:

$$\vdash 0 < n \Rightarrow \forall b.\ n \text{ power\_of } b \iff \exists e.\ e \leq \lceil \log n \rceil \wedge n = b^e$$

[2] In principle, the binary GCD algorithm, also known as Stein's algorithm, can be analysed by our techniques, based on the complexity model described in Chapter 6. Refer to Knuth [1998] and Brent [2000] for the complexity analysis of GCD algorithms.

This is because for a fixed number $n = b^e$, the highest possible exponent e is attained by the smallest possible base $b = 2$. □

This result translates to the first FOR-loop in our AKS pseudocode (Algorithm 1).

3.3 AKS Parameter

Phase 2 of the AKS algorithm is to search for a parameter k, with three possible outcomes: nice k, good k, or bad (see Section 1.3). A good k is coprime to n with $a \leq \text{order}_k(n)$, where $a = \lceil \log n \rceil^2$, while a nice k is a factor of n.

3.3.1 Parameter Search

As depicted in the second FOR-loop in Algorithm 1, the search starts with $k = 2$. In successive iterations, each k is tested for its divisibility into n,

(i) if $k \mid n$, we have found the smallest non-trivial factor k of n. The search returns a nice k.

(ii) if $k \nmid n$, none of the smaller values of k is a factor of n. Thus $\gcd(k, n) = 1$, and $\text{order}_k(n)$ can be computed. Note that $\text{order}_k(n) \leq \varphi(k) \leq k$. If $k < a$, there is no hope for $a \leq \text{order}_k(n)$. Otherwise, if $a \leq \text{order}_k(n)$, the search returns a good k.

We define the parameter $k = \text{aks\_param}\ n$ as the result of a recursive search:

Definition 62. *Sequential search for the AKS parameter k of input n.*

$$
\begin{array}{l}
\text{aks\_param}\ n \stackrel{\text{def}}{=} \\
\quad (\textbf{let} \\
\quad\quad a\ =\ \lceil \log n \rceil\ ; \\
\quad\quad c\ =\ 1 + a^5\ \text{div}\ 2 \\
\quad\ \textbf{in} \\
\quad\quad \textbf{if}\ a \leq 1\ \textbf{then}\ \text{nice}\ n\ \textbf{else}\ \text{aks\_param\_search}\ n\ (a^2)\ 2\ c) \\
\text{where} \\
\quad \text{aks\_param\_search}\ n\ a\ k\ c \stackrel{\text{def}}{=} \\
\quad\quad \textbf{if}\ c < k\ \textbf{then}\ \text{bad} \\
\quad\quad \textbf{else if}\ k \mid n\ \textbf{then}\ \text{nice}\ k \\
\quad\quad \textbf{else if}\ a \leq k \wedge a \leq \text{order}_k(n)\ \textbf{then}\ \text{good}\ k \\
\quad\quad \textbf{else}\ \text{aks\_param\_search}\ n\ a\ (k+1)\ c
\end{array}
$$

The actual value of the cutoff c will be investigated in Section 3.3.2. For now, the use of a cutoff ensures that the search will always terminate. From the definition, we can show formally that:

Theorem 63. *If $n > 1$ has parameter* nice *k, then k is the smallest non-trivial factor of n.*

$$\vdash 1 < n \wedge \text{aks\_param}\ n = \text{nice}\ k \Rightarrow 1 < k \wedge k \mid n \wedge \forall j.\ 1 < j \wedge j < k \Rightarrow j \nmid n$$

Theorem 64. *If n has parameter* good *k, then k is coprime to n with a multiplicative order in* $(\mathrm{mod}\ k)$ *at least* $\lceil \log n \rceil^2$.

$$\vdash \mathsf{aks\_param}\ n = \mathsf{good}\ k \Rightarrow 1 < k \wedge k < n \wedge \gcd(k, n) = 1 \wedge \lceil \log n \rceil^2 \leq \mathsf{order}_k(n)$$

3.3.2 Parameter Exists

In the search for aks_param n by Definition 62, we use a particular cutoff c for termination, due to:

Theorem 65. *Let two numbers n, m be given, both at least 2. Up to a cutoff c expressed in n, m, there must be a value k in the range $1 < k \leq c$ such that, if k is not a factor of n, then k is a modulus with* $\mathsf{order}_k(n)$ *at least m.*

$$\begin{aligned} \vdash\ & 1 < n \wedge 1 < m \wedge c = 1 + (m^2\ \mathsf{div}\ 2)\lceil \log n \rceil \Rightarrow \\ & \exists k.\ 1 < k \wedge k \leq c \wedge (k \mid n \vee m \leq \mathsf{order}_k(n)) \end{aligned}$$

Proof. By contradiction, suppose for all k in the range $1 < k \leq c$, $k \nmid n$ and $\mathsf{order}_k(n) < m$. Then $\gcd(k, n) = 1$, since otherwise a common factor d will divide n, but $d \nmid n$ as $1 < d \leq k$. Let $j = \mathsf{order}_k(n)$, then $j < m$, and $0 < j$ because $\gcd(k, n) = 1$. Thus $0 < j < m$.

Consider the following set:

$$B_{nm} = \{n^j - 1 \quad | \quad 0 < j < m\}$$

Since $j = \mathsf{order}_k(n)$ implies $n^j \equiv 1\ (\mathrm{mod}\ k)$, or $k \mid n^j - 1$, k divides some element in B_{nm}. In other words, k divides the product of all the numbers in B_{nm}, which is:

$$\mathfrak{B}_{nm} = \prod_{0<j}^{j<m} (n^j - 1)$$

Now consider the consecutive numbers in $[1\ ..\ k]$. Not only does 1 divide $\mathfrak{B}_{nm}$ and k divide $\mathfrak{B}_{nm}$, but each intermediate value, say h, divides $\mathfrak{B}_{nm}$ too, as $\mathsf{order}_h(n) < m$ by assumption. Hence the consecutive least common multiple of $[1\ ..\ k]$, denoted by $\mathsf{LCM}\ [1\ ..\ k]$, divides $\mathfrak{B}_{nm}$, or:

$$\mathsf{LCM}\ [1\ ..\ k] \leq \mathfrak{B}_{nm} \tag{3.1}$$

Although enormous, $\mathfrak{B}_{nm}$ is a single number, depending only on n and m. Thus $\mathfrak{B}_{nm}$ is fixed, yet $\mathsf{LCM}\ [1\ ..\ k]$ grows with k. Therefore condition 3.1 cannot hold forever as k increases. To find the threshold, we need to obtains estimates for both sides of the condition.

For an estimate of $\mathfrak{B}_{nm}$, observe that $0 < j < m$ means j counts from 1 to $m - 1$. Hence,

$$\mathfrak{B}_{nm} = \prod_{j=1}^{m-1} (n^j - 1) < \prod_{j=1}^{m-1} n^j = n^{\sum_{j=1}^{m-1} j} = n^{m(m-1)\ \mathsf{div}\ 2} < n^{(m^2)\ \mathsf{div}\ 2}$$

A lower bound for $\mathsf{LCM}\ [1\ ..\ k]$ is given by Theorem 52, page 37:

$$\vdash 2^{k-1} \leq \mathsf{LCM}\ [1\ ..\ k]$$

Combining both estimates, to keep $\mathsf{LCM}\ [1\ ..\ k] \leq \mathfrak{B}_{nm}$ we must have:

$$\begin{array}{rcll} 2^{k-1} & < & n^{(m^2)\ \mathsf{div}\ 2} \leq 2^{\lceil \log n \rceil ((m^2)\ \mathsf{div}\ 2)} & \text{by } n \leq 2^{\lceil \log n \rceil}, \text{ or} \\ k & < & 1 + ((m^2)\ \mathsf{div}\ 2)\lceil \log n \rceil & \text{and cutoff } c = 1 + ((m^2)\ \mathsf{div}\ 2)\lceil \log n \rceil. \end{array}$$

If k advances to be equal to c, condition 3.1 will be violated. Thus within the range $1 < k \leq c$, we must have encountered either a factor k with $k \mid n$, or a modulus k with $\mathsf{order}_k(n) \geq m$. □

For the AKS parameter search, the cutoff is obtained by putting $m = \lceil \log n \rceil^2$ in Theorem 65. This gives the bound $1 + ((\lceil \log n \rceil^{2^2})\ \mathsf{div}\ 2)\lceil \log n \rceil$, which is simplified to $1 + \lceil \log n \rceil^5\ \mathsf{div}\ 2$. This bound is used in Definition 62. With the guarantee from Theorem 65, we have:

Corollary 66. *Our search for the AKS parameter is always successful, and the bound for* aks_param n *has order* $\lceil \log n \rceil^5$.

$$\begin{array}{l} \vdash \mathsf{aks\_param}\ n \neq \mathsf{bad} \\ \vdash 2 < n \wedge \mathsf{aks\_param}\ n = \mathsf{nice}\ k \Rightarrow k \leq 1 + \lceil \log n \rceil^5\ \mathsf{div}\ 2 \\ \vdash \mathsf{aks\_param}\ n = \mathsf{good}\ k \Rightarrow k \leq 1 + \lceil \log n \rceil^5\ \mathsf{div}\ 2 \end{array}$$

Some values of our AKS parameter are given in Table 3.1.

n	aks_param n	comment
1	nice 1	by definition
2	nice 2	prime
3	nice 3	prime
4	nice 2	factor
5	nice 5	prime
6	nice 2	factor
7	nice 7	prime
11	nice 11	prime
31	good 29	first prime with good
47	good 41	next prime with good
91	nice 7	factor
97	good 59	prime
95477	good 293	first composite with good[3]

Table 3.1: Selected values of the AKS parameter by aks_param n.

[3] Indeed, $95477 = 309^2 - 2^2 = 311 \times 307$.

Thus the only possible results of the parameter search are either a nice k or a good k. Note that our bound for k is $1 + \lceil \log n \rceil^5$ div 2. This is a minor improvement of the equivalent reuslt in the paper Agrawal et al. [2004], where Lemma 4.3 gives a bound of $\lceil \log n \rceil^5$.

The first exit in Phase 2 of Algorithm 1 is the result of:

Theorem 67. *When* aks_param n *returns* nice k*, then* n *is prime iff* $k = n$.

$$\vdash 1 < n \wedge \text{aks\_param } n = \text{nice } k \Rightarrow (\text{prime } n \iff k = n)$$

Proof. Since k is the least factor of n greater than 1 by Theorem 63, the result follows. □

This proves Theorem 2 when parameter is nice k. Hereafter, we shall take a parameter good k, with $1 < k < n$, $\gcd(k, n) = 1$ and $\lceil \log n \rceil^2 \leq \text{order}_k(n)$ by Theorem 64.

3.4 Introspective Checks

Phase 3 of the AKS algorithm corresponds to the third `FOR`-loop in Algorithm 1. The loop is initiated by a parameter good k, and performs a range of introspective checks (Equation 1.1):

$$\forall c. \quad 0 < c \wedge c \leq s \Rightarrow (\mathsf{X} + \boldsymbol{c})^n \equiv \mathsf{X}^n + \boldsymbol{c} \pmod{n, \mathsf{X}^k - \mathbf{1}}$$

where $s = (\text{SQRT } \varphi(k))\lceil \log n \rceil$ is called the introspective limit. The introspective checks are polynomial identities in $(\text{mod } n, \mathsf{X}^k - \mathbf{1})$. These introspective checks shall be denoted by poly_intro_range $\mathbb{Z}_n$ k n s:

$$\begin{array}{l} 0 < k \wedge 0 < n \Rightarrow \\ \quad (\text{poly\_intro\_range } \mathbb{Z}_n\ k\ n\ s \iff \\ \qquad \forall c.\ 0 < c \wedge c \leq s \Rightarrow (\mathsf{X} + \boldsymbol{c})^n \equiv \mathsf{X}^n + \boldsymbol{c} \pmod{n, \mathsf{X}^k - \mathbf{1}}) \end{array} \tag{3.2}$$

Since Phase 2 passes a parameter good k to Phase 3, the introspective limit s has an upper bound:

Theorem 68. *If input* n *gives a parameter* good k*, the introspective limit* s *is bounded by* k.

$$\vdash 1 < n \wedge \text{aks\_param } n = \text{good } k \wedge s = (\text{SQRT } \varphi(k))\lceil \log n \rceil \Rightarrow s \leq k$$

Proof. In the ring $\mathbb{Z}_n$, since coprimes of n form a multiplicative group (Theorem 33). Thus,

	$\text{order}_k(n) \mid \varphi(k)$	by multiplicative order (Theorem 16)
or	$\text{order}_k(n) \leq \varphi(k)$	by being a divisor
Also	$\lceil \log n \rceil^2 \leq \text{order}_k(n)$	by good k (Theorem 64)
Combining,	$\lceil \log n \rceil^2 \leq \varphi(k)$	by above
giving	$\lceil \log n \rceil \leq \text{SQRT } \varphi(k)$	by taking integer square roots

Multiplying both sides by SQRT $\varphi(k)$,

$$s = (\text{SQRT } \varphi(k))\lceil \log n \rceil \leq (\text{SQRT } \varphi(k))^2 \leq \varphi(k) \leq k$$

□

However, this introspective limit s is irrelevant when n is prime:

Theorem 69. *A prime n is introspective to all monomial $\mathsf{X} + c$.*

$$\vdash \text{prime } n \wedge 0 < k \Rightarrow \text{poly\_intro\_range } \mathbb{Z}_n\ k\ n\ s$$

Proof. From Fermat's Little Theorem for monomials (Theorem 39, page 29):

$$\vdash \text{prime } n \Rightarrow (\mathsf{X} + c)^n \equiv \mathsf{X}^n + c \pmod{n}$$

The equivalence evidently holds when both sides are reduced to remainders in $\mathsf{X}^k - \mathbf{1}$:

$$\vdash \text{prime } n \wedge 0 < k \Rightarrow (\mathsf{X} + c)^n \equiv \mathsf{X}^n + c \pmod{n, \mathsf{X}^k - \mathbf{1}}$$

Thus we have poly_intro_range $\mathbb{Z}_n\ k\ n\ s$ by (3.2). □

Therefore, a parameter good k plays a vital role in Phase 3: it specifies $\mathsf{X}^k - \mathbf{1}$ as the polynomial modulus for introspective checks, and it defines the introspective limit s. Overall, this good k is passed onto Phase 3 for poly_intro_range $\mathbb{Z}_n\ k\ n\ s$, with $s \leq k$ by Theorem 68.

3.5 AKS Primality Test

With the notion of poly_intro_range $\mathbb{Z}_n\ k\ n\ s$, we can recast aks n from Definition 1 as:

$$\begin{array}{l}
\vdash \text{aks } n \iff \\
\quad \text{power_free } n \wedge \\
\quad \textbf{case } \text{aks_param } n \textbf{ of} \\
\quad | \ \text{nice } k \triangleright k = n \\
\quad | \ \text{good } k \triangleright \text{poly_intro_range } \mathbb{Z}_n\ k\ n\ (\text{SQRT } \varphi(k))\lceil \log n \rceil \\
\quad | \ \text{bad} \triangleright \mathsf{F}
\end{array} \tag{3.3}$$

This enables an uncluttered proof that the AKS algorithm is indeed a primality test (Theorem 2):

$$\vdash \text{prime } n \iff \text{aks } n$$

One direction is easy:

Theorem 70. *A prime n shall pass all phases of the AKS algorithm.*

$$\vdash \mathsf{prime}\ n \Rightarrow \mathsf{aks}\ n$$

Proof. A prime n is power free by Theorem 60. We know that aks_param n will not return bad by Corollary 66. If aks_param n returns nice k, then $k = n$ by Theorem 67, since a prime has no proper factor. If aks_param n returns good k, then $0 < k$ since the search starts with $k = 2$. By Theorem 69, the introspective limit $s = (\mathsf{SQRT}\ \varphi(k))\lceil \log\ n \rceil$ is irrelevant, hence poly_intro_range $\mathbb{Z}_n\ k\ n\ s$. □

As noted in Section 1.4, this easy direction had been formalised by Campos et al. [2004] in ACL2, and de Moura and Tadeu [2008] attempted in Coq.

The other direction requires more work, with a change in perspective:

1. Find a special prime factor p of n to build a finite field $\mathbb{Z}_p$.
2. Verify that poly_intro_range $\mathbb{Z}_n\ k\ n\ s$ implies poly_intro_range $\mathbb{Z}_p\ k\ n\ s$.
3. Formulate AKS Main Theorem (Theorem 115) to prove the converse of Theorem 2.

The main reason for this perspective change is that, as algebraic structures and in a broad sense, a field is better behaved compared to a ring. In particular,

- all field elements are invertible. Thus any nonzero polynomial with coefficients taken from a field can be a divisor for polynomial division.
- a field has no zero divisors. This enforces that any nonzero polynomial with coefficients taken from a field cannot have more roots than its degree (Theorem 25).
- the characteristic of a finite field is always a prime (Theorem 26). This implies that the prime characteristic will be introspective to any monomial $\mathsf{X} + c$ (Theorem 69).
- in a finite field, the multiplicative group is cyclic (Theorem 28). Thus the multiplicative order of field elements are confined to be divisors of the order of the multiplicative group. This dictates the cyclotomic factorisation of the introspective modulus $\mathsf{X}^k - \mathbf{1}$.

As we shall see, all these contribute to the beautiful proof of AKS Main Theorem (Theorem 115).

3.6 Introspective Shift

Given the input number n, our starting point is the ring $\mathbb{Z}_n$. To find a field $\mathcal{F}$ to work with, note that $n > 1$ since it is power free. Therefore, n has a prime factor p. There may be several primes to choose from, but we shall pick one with the help of parameter good k. Recall that aks_param n = good k means $\gcd(k, n) = 1$ with $1 < k$ and $k < n$ (Theorem 64). Thus $2 < n$ and $1 < \lceil \log\ n \rceil^2 < \mathsf{order}_k(n)$. We choose a prime factor p of n with $1 < \mathsf{order}_k(p)$, provided by Theorem 57.

We shall show that, for a number n with parameter good k and a factor $p \mid n$, the introspective checks in modulo n:

$$(\mathsf{X} + \boldsymbol{c})^n \equiv \mathsf{X}^n + \boldsymbol{c} \pmod{n,\ \mathsf{X}^k - \mathbf{1}} \tag{3.4}$$

can be regarded as introspective checks in modulo p:

$$(\mathsf{X} + \boldsymbol{c})^n \equiv \mathsf{X}^n + \boldsymbol{c} \pmod{p,\ \mathsf{X}^k - \mathbf{1}} \tag{3.5}$$

When the constants c are in the same range $0 < c \leq s$, with limit $s = (\mathsf{SQRT}\ \varphi(i))\lceil \log n \rceil$, this is shifting of all checks from poly_intro_range $\mathbb{Z}_n\ k\ n\ s$ to poly_intro_range $\mathbb{Z}_p\ k\ n\ s$.

This introspective shift is barely mentioned in the AKS paper.[4] We provide a formal proof by invoking ring homomorphism, denoted by $(\mapsto_r)$, because $\mathbb{Z}_n$ and $\mathbb{Z}_p$ are two distinct rings with their own operations.

In order to treat the monomials $\mathsf{X} + c$ with $0 < c \leq s$ as elements of either $\mathbb{Z}_n[\mathsf{X}]$ or $\mathbb{Z}_p[\mathsf{X}]$, the limit s needs to satisfy:

$$s < n \quad \text{and} \quad s < p. \tag{3.6}$$

Note that $p \mid n$ gives $p \leq n$, and $s \leq k$ from Theorem 68. Thus for (3.6) to hold, we just need $k < p$. This is true because in the search for AKS parameter, if aks_param n returns good k, we have not encountered any factor of n, yet $p \mid n$. Hence the following applies:

Theorem 71. *For a suitable factor p of n, introspective checks based on $\mathbb{Z}_n$ are preserved as introspective checks based on $\mathbb{Z}_p$.*

$$\vdash 0 < n \wedge 0 < k \wedge 1 < p \wedge p \mid n \wedge s < p \wedge \mathsf{poly\_intro\_range}\ \mathbb{Z}_n\ k\ h\ s \Rightarrow \mathsf{poly\_intro\_range}\ \mathbb{Z}_p\ k\ h\ s$$

Proof. Consider the two rings, $\mathbb{Z}_n$ and $\mathbb{Z}_p$, where p is a factor of n. If $x \equiv y \pmod{n}$, then $(x - y)$ is a multiple of n, or a multiple of its factor p, showing that $x \equiv y \pmod{p}$ holds. Hence there is a homomorphism between these two rings (Theorem 31):

$$\vdash 0 < n \wedge p \mid n \Rightarrow (\lambda x.\ x\ \mathsf{mod}\ p) : \mathbb{Z}_n \mapsto_r \mathbb{Z}_p$$

This ring homomorphism preserves monomials $\mathsf{X} + \boldsymbol{c}$:

$$\vdash 0 < n \wedge 1 < p \wedge s < p \Rightarrow \forall c.\ 0 < c \wedge c \leq s \Rightarrow \forall f.\ f : \mathbb{Z}_n \mapsto_r \mathbb{Z}_p \Rightarrow f(\mathsf{X} + \boldsymbol{c}) = \mathsf{X} + \boldsymbol{c}$$

Here $f(\mathsf{X} + \boldsymbol{c})$ denotes the result of applying the ring homomorphism f to each coefficient of $\mathsf{X} + \boldsymbol{c}$. That $f(\mathsf{X} + \boldsymbol{c}) = \mathsf{X} + \boldsymbol{c}$ shows that the monomials $\mathsf{X} + \boldsymbol{c}$ with $0 < c \leq s$ can be treated as elements of either $\mathbb{Z}_n[\mathsf{X}]$ or $\mathbb{Z}_p[\mathsf{X}]$.

A ring homomorphism f preserves algebraic operations. Thus polynomial additions in

[4] Just Equation 3.4 "implies" Equation 3.5 in [Agrawal et al., 2004, page 4].

$\mathbb{Z}_n[\mathsf{X}]$ are preserved as the corresponding polynomial additions in $\mathbb{Z}_p[\mathsf{X}]$. Similarly, multiplication of polynomials by a scalar is preserved by homomorphism f. Since multiplication of polynomials is defined in terms of scalar multiplication by successive coefficients, it is preserved by homomorphism f. Furthermore, as exponentiation of a polynomial to a power is defined in terms of polynomial multiplication, it is preserved by homomorphism f.

Finally, the introspective relation just checks the result of a polynomial exponentiation. It follows that every introspective check in the quotient ring $\mathbb{Z}_n[\mathsf{X}]/(\mathsf{X}^k - \mathbf{1})$ is reflected as an introspective check in the quotient ring $\mathbb{Z}_p[\mathsf{X}]/(\mathsf{X}^k - \mathbf{1})$. □

3.7 AKS in Finite Field

The special prime p factor of n from Section 3.6 gives a finite field $\mathbb{Z}_p$, with characteristic the same as prime p. This will be the setting for the AKS Main Theorem (Chapter 5, Theorem 115). We collect the useful properties of such a finite field $\mathcal{F}$ as a set of criteria:

Definition 72. *The criteria for a field $\mathcal{F}$ to be applied in AKS Main Theorem.*

$$\begin{aligned}&\mathsf{aks\_criteria}\ \mathcal{F}\ n\ k \stackrel{\mathsf{def}}{=}\\ &\quad 0 < n \wedge 0 < k \wedge 1 < \mathsf{order}_k(\mathsf{char}(\mathcal{F})) \wedge \mathsf{char}(\mathcal{F}) \mid n \wedge k < \mathsf{char}(\mathcal{F}) \wedge\\ &\quad \lceil \mathsf{log}\ n \rceil^2 \leq \mathsf{order}_k(n) \wedge \mathsf{poly\_intro\_range}\ \mathcal{F}\ k\ n\ (\mathsf{SQRT}\ \varphi(k))\lceil \mathsf{log}\ n \rceil\end{aligned}$$

The criteria for the field $\mathbb{Z}_p$ becomes:

$$\begin{aligned}\vdash\ & 0 < p \wedge \mathsf{aks\_criteria}\ \mathbb{Z}_p\ n\ k \Rightarrow\\ & 0 < n \wedge 0 < k \wedge 1 < \mathsf{order}_k(p) \wedge p \mid n \wedge k < p \wedge \lceil \mathsf{log}\ n \rceil^2 \leq \mathsf{order}_k(n) \wedge\\ & \mathsf{poly\_intro\_range}\ \mathbb{Z}_p\ k\ n\ (\mathsf{SQRT}\ \varphi(k))\lceil \mathsf{log}\ n \rceil\end{aligned}$$

We can now provide an outline to establish the hard part of Theorem 2:

Theorem 73. *If a number n passes all phases of the AKS algorithm, then n is prime.*

$$\vdash \mathsf{aks}\ n \Rightarrow \mathsf{prime}\ n$$

Proof. Repeating the definition of $\mathsf{aks}\ n$ in (3.3):

$$\begin{aligned}\vdash\ & \mathsf{aks}\ n \iff\\ & \quad \mathsf{power\_free}\ n \wedge\\ & \quad \textbf{case}\ \mathsf{aks\_param}\ n\ \textbf{of}\\ & \quad \mid\ \mathsf{nice}\ k \triangleright k = n\\ & \quad \mid\ \mathsf{good}\ k \triangleright \mathsf{poly\_intro\_range}\ \mathbb{Z}_n\ k\ n\ (\mathsf{SQRT}\ \varphi(k))\lceil \mathsf{log}\ n \rceil\\ & \quad \mid\ \mathsf{bad} \triangleright \mathsf{F}\end{aligned}$$

If n is not power free, n cannot be a prime (Theorem 60). The search for $\mathsf{aks\_param}\ n$ will not return bad by Corollary 66. If $\mathsf{aks\_param}\ n$ returns $\mathsf{nice}\ k$, then n is prime only when $k = n$ by

Theorem 67. If aks_param n returns good k, we have shown in Section 3.6 that there is a special prime p factor of n that fits into the criteria of Definition 72:

$$\vdash \text{aks\_param } n = \text{good } k \wedge \text{poly\_intro\_range } \mathbb{Z}_n\ k\ n\ (\text{SQRT } \varphi(k))\lceil \log n \rceil \Rightarrow \\ \exists p.\ \text{prime } p \wedge \text{aks\_criteria } \mathbb{Z}_p\ n\ k$$

We shall prove this AKS Main Theorem for a finite field $\mathcal{F}$ (Theorem 115):

$$\vdash \text{FiniteField } \mathcal{F} \wedge |\mathsf{F}| = \text{char}(\mathcal{F}) \Rightarrow \forall n\ k.\ \text{aks\_criteria } \mathcal{F}\ n\ k \Rightarrow n \text{ power\_of char}(\mathcal{F})$$

Identifying the finite field $\mathcal{F}$ with $\mathbb{Z}_p$, a finite field with cardinality p and characteristic p, we conclude that n is a power of p. But n is power free, therefore n must be equal to p. In other words, n is prime. □

Scripts Scripts for the study of the computational aspect of the AKS algorithm are located in `aks/compute`. Power free test theorems are proved in `computeBasic`, Parameter search theorems are given in `computeParam`. Scripts for the study of the theoretical aspect of the AKS algorithm are located in `aks/theories`. Introspective relation and introspective shifting are discussed in `AKSintro` and `AKSshift`, respectively.

3.8 Summary

Starting with the pseudocode of the AKS algorithm, we study each phase: what are the key ideas behind, and how they fit together. We give the bound for the power free test, assert the existence of the AKS parameter with a search bound. For the introspective checks, proving that a prime n will pass all of them is straightforward, because the characterisation of primes by binomial divisibility (Theorem 36) shows that limit for the introspective checks is irrelevant. In order to prove that a composite n will fail at least one introspective check, we go through the tortuous journey of picking up a special prime factor p of n, then shift the introspective checks from $\mathbb{Z}_n[\mathsf{X}]/(\mathsf{X}^k - \mathbf{1})$ to $\mathbb{Z}_p[\mathsf{X}]/(\mathsf{X}^k - \mathbf{1})$. With this feat we formulate the AKS Main Thoerem in a finite field, and outline the proof of AKS theorem (Theorem 2) based on the Main Theorem. This concludes the first part of the theory for the AKS algorithm. The focus of the next part will be the AKS Main Theorem.

3.9 Remarks

The proof of Theorem 71 was done in a general context: how to shift poly_intro_range $\mathcal{R}\ k\ n\ s$ from one ring to another, then adapt to instances of ring like $\mathbb{Z}_n$ and $\mathbb{Z}_p$. This illustrates a general approach taken by us. We could have proved the theorem by appealing directly to the properties of modular arithmetic between $\mathbb{Z}_n$ and $\mathbb{Z}_p$, but we choose a ring-theoretic path using ring homomorphism. This puts the investigation in a proper algebraic setting, and provides an example to apply our generic ring library.

We follow the revised AKS paper (Agrawal et al. [2004]) to bound the parameter k by a lower bound for the consecutive LCM. Most expositions regarding this parameter (Terence Tao [2009]; Daleson [2006]; Crandall and Pomerance [2005]), following the original AKS paper (Agrawal et al. [2002]), make use of a Chebyshev-type estimate for the product of consecutive primes. This has been discussed in Section 1.2 (page 2). These two approaches, based on different methods, are equivalent because both are effectively weak forms of the prime number theorem (see Chan and Norrish [2019b]).

Although not directly related to our approach, note that the Prime Number Theorem has been formalised by Avigad et al. [2007] in Isabelle proof assistant, and independently in HOL Light by John Harrison [2009]. The formalization of Chebyshev's approach was taken up in Asperti and Ricciotti [2008] using Matita Interactive Theorem Prover.

Part II

Correctness

Part 2 is about the correctness of the AKS algorithm. The key is formulated in Part 1 as the Main Theorem. We first develop the required background theory of finite fields, then investigate the introspective relation by separating the exponents and polynomials into different sets. The interplay between these sets, together with the properties of polynomials with coefficients from a finite field, and the Pigeonhole Principle for injective map between finite sets, eventually stipulate the conclusion of the Main Theorem.

Advanced Algebra

This chapter develops more topics in finite fields that are crucial for the theory behind the AKS Main Theorem. Taking a geometric viewpoint, a finite field is a vector space over its subfields, with polynomial coefficients restricted to those of the subfields. The use of subfield polynomials give rise to irreducibility and adjoining roots, with a minimal polynomials for each field element. This leads to the existence and uniqueness of finite fields, which are essential for the cyclotomic factorisation of $\mathsf{X}^k - \mathbf{1}$*, where* k *is the AKS parameter and* $\mathsf{X}^k - \mathbf{1}$ *is the introspective modulus. We shall establish the existence of a special irreducible factor from the cyclotomic factors of* $\mathsf{X}^k - \mathbf{1}$*, which plays a key role in the proof of the AKS Main Theorem.*

> Algebra is nothing more than geometry, in words;
> geometry is nothing more than algebra, in pictures.
> — Sophie Germain (1831)

4.1 Finite Field Classification

The AKS Main Theorem, as formulate in the proof of Theorem 73, is:

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \wedge |\mathsf{F}| = \mathsf{char}(\mathcal{F}) \Rightarrow \forall n\ k.\ \mathsf{aks\_criteria}\ \mathcal{F}\ n\ k \Rightarrow n\ \mathsf{power\_of}\ \mathsf{char}(\mathcal{F})$$

It is based on a finite field $\mathcal{F}$ satisfying certain criteria, the $\mathsf{aks\_criteria}\ \mathcal{F}\ n\ k$. Its proof depends on a special irreducible factor h of $\mathsf{X}^k - \mathbf{1}$, the modulus of introspective checks. This irreducible factor h is used to form a polynomial quotient field $\mathcal{F}[\mathsf{X}]/(\mathsf{h})$, with an element X. There are many irreducible factors of $\mathsf{X}^k - \mathbf{1}$, but the proof, in one of its counting arguments, requires one with the multiplicative order of X in this quotient field to be k, *i.e.*, $\mathsf{order}_{\mathsf{h}}(\mathsf{X}) = k$. The existence of such a special irreducible h requires the full theory of finite field existence and uniqueness up to isomorphism, which are the classification theorems for finite fields.

Toward this goal, we shall take a field/subfield pair, treating the field as an extension of the subfield. These ideas involve the vector spaces, a geometric picture for finite fields.

4.1.1 Vector Spaces

A vector space $\mathsf{VSpace}\ \mathcal{S}\ \mathcal{G}\ f$ is another algebraic structure, with three components:

- a field $\mathcal{S}$ of scalars,
- an additive group $\mathcal{G}$ of vectors, and
- a multiplication f taking a scalar and a vector, resulting in a vector.

Together they must satisfy the vector space axioms, which are expressed in HOL4 as:

Definition 74. *Axioms for a vector space.*

$$\begin{aligned}&\mathsf{VSpace}\ \mathcal{S}\ \mathcal{G}\ f \stackrel{\mathsf{def}}{=}\\ &\quad \mathsf{Field}\ \mathcal{S} \wedge \mathsf{AbelianGroup}\ \mathcal{G} \wedge (\forall a\ v.\ a \in \mathsf{S} \wedge v \in \mathsf{G} \Rightarrow f\ a\ v \in \mathsf{G}) \wedge\\ &\quad (\forall a\ b\ v.\ a \in \mathsf{S} \wedge b \in \mathsf{S} \wedge v \in \mathsf{G} \Rightarrow f\ a\ (f\ b\ v) = f\ (\mathcal{S}.\mathsf{prod.op}\ a\ b)\ v) \wedge\\ &\quad (\forall v.\ v \in \mathsf{G} \Rightarrow f\ \mathbb{1}\ v = v) \wedge\\ &\quad (\forall a\ u\ v.\ a \in \mathsf{S} \wedge u \in \mathsf{G} \wedge v \in \mathsf{G} \Rightarrow f\ a\ (\mathcal{G}.\mathsf{op}\ u\ v) = \mathcal{G}.\mathsf{op}\ (f\ a\ u)\ (f\ a\ v)) \wedge\\ &\quad \forall a\ b\ v.\ a \in \mathsf{S} \wedge b \in \mathsf{S} \wedge v \in \mathsf{G} \Rightarrow f\ (a + b)\ v = \mathcal{G}.\mathsf{op}\ (f\ a\ v)\ (f\ b\ v)\end{aligned}$$

Our formalisation of the theory of vector spaces follows the approach given in Axler [2015]. The library is a standalone development consisting of basis, spanning subspace, and linear independence. The dimension of a vector space over its subspace is the minimal number of vectors in a subspace basis to span the vector space. The minimal requirement ensures that such basis vectors are linear independent.

4.1.2 Subfields

A subfield $\mathcal{S}$ of a field $\mathcal{F}$ has its carrier $\mathsf{S} \subseteq \mathsf{F}$, and itself is a field by keeping the same field additions and multiplications. The fact that multiplication is distributive over addition gives another view of the field/subfield relationship:

Theorem 75. *A field is a vector space over its subfield.*

$$\vdash \mathcal{S} \preccurlyeq \mathcal{F} \Rightarrow \mathsf{VSpace}\ \mathcal{S}\ \mathcal{F}.\mathsf{sum}\ (\times)$$

Proof. Given $\mathcal{S} \preccurlyeq \mathcal{F}$, we can:

- identify the elements of the subfield $\mathcal{S}$ as scalars.
- identity the elements of the abelian group $\mathcal{F}.\mathsf{sum}$ of field $\mathcal{F}$ as vectors.
- identify the field multiplication $*$ as the multiplication op of scalar to vector giving a vector.

Then it is a routine exercise to verify that all vector space axioms are satisfied. □

4.1.3 Prime Subfields

For a finite field $\mathcal{F}$, the elements given by repeated additions of $\mathbb{1}$ form a subfield. This subfield, which must be embedded in any subfield of $\mathcal{F}$, is also a field with no proper subfield. It is called its *prime field*, denoted by $\mathbf{PF}_{\mathcal{F}}$:

Theorem 76. *The prime field of a finite field is its smallest subfield.*

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \wedge S \preccurlyeq \mathcal{F} \Rightarrow \mathbf{PF}_{\mathcal{F}} \preccurlyeq S$$

The cardinality of $\mathbf{PF}_{\mathcal{F}}$ is $\mathsf{char}(\mathcal{F})$, the additive order of $\mathbb{1}$. The dimension of a finite field $\mathcal{F}$ over its prime field $\mathbf{PF}_{\mathcal{F}}$ is called its *degree*, denoted by $\mathsf{deg}(\mathcal{F})$. Treating a finite field $\mathcal{F}$ as a vector space over its prime field $\mathbf{PF}_{\mathcal{F}}$, its dimension is the number of basis vectors. Since every field element is uniquely expressed by a linear combination of basis vectors, we have:

Theorem 77. *The cardinality of a finite field equals its characteristic raised to its degree.*

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \Rightarrow |\mathsf{F}| = \mathsf{char}(\mathcal{F})^{\mathsf{deg}(\mathcal{F})}$$

With every field having its prime field as a subfield, this is the first key result about finite field cardinality:

Theorem 78. Cardinality — *The cardinality of a finite field must be a nontrivial prime power.*

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \Rightarrow \exists p\ d.\ \mathsf{prime}\ p \wedge 0 < d \wedge |\mathsf{F}| = p^d$$

Proof. Take $p = \mathsf{char}(\mathcal{F})$ and $n = \mathsf{deg}(\mathcal{F})$. Since the dimension of $\mathcal{F}$ over its subfield $\mathbf{PF}_{\mathcal{F}}$ is at least 1, we have $0 < n$. Now p is prime by Theorem 26, and $|\mathsf{F}| = p^n$ by Theorem 77. □

This particular result, that all subfields of a field share its characteristic, is trivial:

Theorem 79. *All subfields have the same characteristic as the field.*

$$\vdash \mathcal{S} \preccurlyeq \mathcal{F} \Rightarrow \mathsf{char}(\mathcal{S}) = \mathsf{char}(\mathcal{F})$$

Proof. All subfields have the same multiplicative identity $\mathbb{1}$ of the field. □

4.2 Existence of Finite Fields

The integers $\mathbb{Z}$ form a ring, an infinite one. Dividing the integers by a modulus $n \neq 0$, the remainders are in the range from 0 to $(n - 1)$. These are the elements of $\mathbb{Z}_n$, a finite ring with arithmetic operations in $(\mathrm{mod}\ n)$. There may be zero divisors in $\mathbb{Z}_n$, *e.g.*. $2 \times 3 \equiv 0\ (\mathrm{mod}\ 6)$. Thus $\mathbb{Z}_n$ cannot be a field if n is composite. However:

Theorem 80. *The ring $\mathbb{Z}_p$ for a prime p modulus is a finite field.*

$$\vdash \mathsf{prime}\ p \Rightarrow \mathsf{FiniteField}\ \mathbb{Z}_p$$

Proof. In the finite ring $\mathbb{Z}_p$, if $ab \equiv 0 \pmod{p}$, then $p \mid ab$. Since p is prime, $p \mid a$ or $p \mid b$. This means that $a \equiv 0 \pmod{p}$ or $b \equiv 0 \pmod{p}$. Thus $\mathbb{Z}_p$ has no zero divisors for a prime p, making it a finite integral domain, and a finite field by Theorem 21. □

Polynomials with coefficients from integers or a field both form Euclidean rings, which have the property that every ideal is generated by a single element. The ideal generated by a ring element a is written as (a). Let h be a polynomial, with coefficients from a field $\mathcal{F}$. The quotient ring by the ideal (h), denoted by $\mathcal{F}[\mathsf{X}]/(\mathsf{h})$, consists of all polynomial remainders after division by h. Like $\mathbb{Z}_p$ for a prime p, we have:

Theorem 81. *For a finite field, the polynomial quotient ring generated by an irreducible is a finite field.*

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \Rightarrow \forall \mathsf{h}.\ \mathsf{ipoly}\ \mathsf{h} \Rightarrow \mathsf{FiniteField}\ \mathcal{F}[\mathsf{X}]/(\mathsf{h})$$

Proof. [1] Note that $\mathcal{F}[\mathsf{X}]/(\mathsf{h})$ is a finite ring, as it consists of polynomials p with $\mathsf{deg\ p} < \mathsf{deg\ h}$. Let f and g be two such polynomials. If $\mathsf{f} \times \mathsf{g} \equiv \mathsf{0} \pmod{\mathsf{h}}$, this means $\mathsf{h} \mid \mathsf{f} \times \mathsf{g}$. Since h is irreducible, $\mathsf{h} \mid \mathsf{f}$ or $\mathsf{h} \mid \mathsf{g}$. Thus $\mathsf{f} \equiv \mathsf{0} \pmod{\mathsf{h}}$ or $\mathsf{g} \equiv \mathsf{0} \pmod{\mathsf{h}}$. Therefore $\mathcal{F}[\mathsf{X}]/(\mathsf{h})$ has no zero divisors, making it a finite integral domain, or a finite field by Theorem 21. □

Note that the order of $\mathcal{F}[\mathsf{X}]/(\mathsf{h})$ is $|\mathsf{F}|^d$, where $d = \mathsf{deg\ h}$, by counting the number of polynomial remainders, all with degree less than d. This provides a recipe to construct other finite fields, based on those of prime order, *e.g.*, $\mathbb{Z}_p$ for prime p, if there are irreducibles of every nonzero degree d.

4.2.1 Counting Irreducibles

For a finite field $\mathcal{F}$, let $(\mathcal{I}_{\mathcal{F}}\ n)$ denote the set of monic irreducibles of degree n in $\mathcal{F}[\mathsf{X}]$. We show that:

Theorem 82. *There are monic irreducible polynomials of any positive degree.*

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \Rightarrow \forall n.\ 0 < n \Rightarrow \exists p.\ \mathsf{monic}\ p \wedge \mathsf{ipoly}\ p \wedge \mathsf{deg}\ p = n$$

Proof. [2] For each divisor d of $n \neq 0$, let $\mathbf{\Psi}_d = \prod (\mathcal{I}_{\mathcal{F}}\ d)$, the product of all monic irreducibles of degree d. Multiplying these products over all divisors of n gives a remarkable result:[3]

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \wedge 0 < n \Rightarrow \mathsf{X}^{|\mathsf{F}|^n} - \mathsf{X} = \prod \{\ \mathbf{\Psi}_d \mid d \in \mathsf{divisors}\ n\ \}$$

[1] This proof works because polynomial rings over a field is a unique factorisation domain, where irreducibles are primes.

[2] This proof, based on degree and divisibility of special polynomials, is adapted from [Belk, 2016, Theorem 9].

[3] This result is nontrivial: it depends on a detailed analysis of the distinct irreducible factors of $\mathsf{X}^{|\mathsf{F}|^n} - \mathsf{X}$. For a formal proof, refer to our script.

A more pleasant typographic rendering, as one might find in textbooks, is:

$$\mathsf{X}^{|\mathsf{F}|^n} - \mathsf{X} = \prod_{d \mid n} \mathbf{\Psi}_d \tag{4.1}$$

The left-side is a polynomial of degree $|\mathsf{F}|^n$. By including more monic irreducibles on the right-side, the left polynomial becomes a divisor:

$$\left(\mathsf{X}^{|\mathsf{F}|^n} - \mathsf{X}\right) \mid \prod_{d=1}^{n} \mathbf{\Psi}_d$$

Now, suppose there are no monic irreducibles of degree n. Then $\mathcal{I}_{\mathcal{F}}\ n = \emptyset$, so that $\mathbf{\Psi}_n = \mathbf{1}$. We shall see how this leads to a contradiction, thus proving the result.

With $\mathbf{\Psi}_n = \mathbf{1}$, the left polynomial shall divide the product of all $\mathbf{\Psi}_d$, with subscript d from 1 to $n-1$. We claim that:

$$\left(\mathsf{X}^{|\mathsf{F}|^n} - \mathsf{X}\right) \mid \prod_{d=1}^{n-1} \mathbf{\Psi}_d \mid \prod_{d=1}^{n-1} \left(\mathsf{X}^{|\mathsf{F}|^d} - \mathsf{X}\right)$$

This is because each $(\mathsf{X}^{|\mathsf{F}|^d} - \mathsf{X})$ has a factor $\mathbf{\Psi}_d$ by Equation 4.1, and the $\mathbf{\Psi}_d$'s, being the product of all monic irreducibles of degree d, are pairwise coprime. However, the degree of the polynomial product on the right-side is less than the degree of the left polynomial:

$$\sum_{d=1}^{n-1} |\mathsf{F}|^d = \frac{|\mathsf{F}|^n - 1}{|\mathsf{F}| - 1} < |\mathsf{F}|^n \qquad \text{by geometric series, and } 1 < |\mathsf{F}|.$$

This is impossible since the left-side divides the right-side, and the right-side is not the zero polynomial. Thus $\mathcal{I}_{\mathcal{F}}\ n \neq \emptyset$, *i.e.*, there exists a monic irreducible polynomial of degree n. □

With the existence of monic irreducible polynomials of any positive degree, we can assert the existence of finite fields with prime power cardinality:

Theorem 83. Existence — *For each nontrivial prime power, there exists a finite field of that cardinality.*

$$\vdash \mathsf{prime}\ p \wedge 0 < d \Rightarrow \exists \mathcal{F}.\ \mathsf{FiniteField}\ \mathcal{F} \wedge |\mathsf{F}| = p^d$$

Proof. Note that $\mathbb{Z}_p$ is a finite field for a prime p, by Theorem 80. Applying Theorem 82, there is a monic irreducible h in $\mathbb{Z}_p[\mathsf{X}]$ with $\deg\ \mathsf{h} = n$. Then $\mathbb{Z}_p[\mathsf{X}]/(\mathsf{h})$ is a finite field by Theorem 81, with p^n elements. □

4.3 Uniqueness of Finite Fields

Putting $n = 1$ in Equation 4.1 gives $\mathsf{X}^{|\mathsf{F}|} - \mathsf{X} = \mathbf{\Psi}_1$. As $\mathbf{\Psi}_1$ is the product of linear monic irreducibles, we have:

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \Rightarrow \mathsf{X}^{|\mathsf{F}|} - \mathsf{X} = \prod \{\ \mathsf{X} - a \times \mathbf{1}\ \mid\ a \in \mathsf{F}\ \}$$

We use $a \times \mathbf{1}$ to represent the constant polynomial given by an element $a \in \mathsf{F}$, in accordance with types. In textbooks, this is presented simply as:

$$\mathsf{X}^{|\mathsf{F}|} - \mathsf{X} = \prod_{a \in \mathsf{F}} (\mathsf{X} - a) \tag{4.2}$$

The polynomial on the left has coefficients $\mathbb{1}$, $\mathbb{0}$ and $-\mathbb{1}$, which are present in any subfield $\mathcal{S} \preccurlyeq \mathcal{F}$. Therefore such a polynomial is for any subfield a subfield polynomial. However, as a field polynomial, its roots are all the field elements, each contributing a single factor, *i.e.*, there are no repeated roots.

4.3.1 Minimal Polynomials

Given a subfield $\mathcal{S} \preccurlyeq \mathcal{F}$, consider all the subfield polynomials, *i.e.*, elements in $\mathcal{S}[\mathsf{X}]$. Any field element $a \in \mathsf{F}$ is then a root of some subfield polynomial, *e.g.*, the polynomial of Equation 4.2. We write $\mathsf{poly}_{\mathcal{S}}\ \mathsf{h}$ to indicate a subfield polynomial with coefficients in $\mathcal{S}$. If h has no proper subfield polynomial factors, it is irreducible in the subfield, indicated by $\mathsf{ipoly}_{\mathcal{S}}\ \mathsf{h}$.

A monic subfield polynomial of the smallest degree having a as a root is called a *minimal* polynomial of a, denoted by $\mathfrak{m}_a$, with these properties (see, *e.g.*, [Belk, 2016, page 47]):

(1) A minimal polynomial of a finite field element a is irreducible over any subfield.

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \wedge \mathcal{S} \preccurlyeq \mathcal{F} \Rightarrow \forall a.\ a \in \mathsf{F} \Rightarrow \mathsf{ipoly}_{\mathcal{S}}\ \mathfrak{m}_a$$

(2) A minimal polynomial of a finite field element a divides every subfield polynomial with root a.

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \wedge \mathcal{S} \preccurlyeq \mathcal{F} \Rightarrow \forall \mathsf{h}\ a.\ \mathsf{poly}_{\mathcal{S}}\ \mathsf{h} \wedge a \in \mathsf{roots}\ \mathsf{h} \Rightarrow \mathfrak{m}_a \mid \mathsf{h}$$

From property (2), if there are two minimal polynomials of $a \in \mathsf{F}$, they will divide each other, hence equal, *i.e.*. $\mathfrak{m}_a$ is unique. We shall refer to $\mathfrak{m}_a$ as "the" minimal polynomial of a.

Recall from Equation 4.2 that all $a \in \mathsf{F}$ are roots of $\mathsf{X}^{|\mathsf{F}|} - \mathsf{X}$. By property (2), each $\mathfrak{m}_a$ divides $\mathsf{X}^{|\mathsf{F}|} - \mathsf{X}$, the dividend. Since each $\mathfrak{m}_a$ is irreducible by property (1), and irreducibles are coprime, the product of all $\mathfrak{m}_a$ also divides the dividend. Both product and dividend have identical roots, *i.e.*, all elements in F. With all possible elements taken as roots, there can be no more roots. Thus the product and the dividend differ only by a constant, which must be $\mathbf{1}$ as both are monic polynomials. Therefore:

Theorem 84. *With respect to any subfield of a finite field $\mathcal{F}$, the polynomial $\mathsf{X}^{|\mathsf{F}|} - \mathsf{X}$ is the product of distinct minimal polynomials from all the field elements.*

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \wedge \mathcal{S} \preccurlyeq \mathcal{F} \Rightarrow \mathsf{X}^{|\mathsf{F}|} - \mathsf{X} = \prod \{\ \mathfrak{m}_a \mid a \in \mathsf{F}\ \}$$

Note that in our formalisation, this product is over a set, which eliminates duplicates. This result shows that if a monic irreducible divides $\mathsf{X}^{|\mathsf{F}|} - \mathsf{X}$, it is the minimal polynomial of some field element, by unique factorisation.

4.3.2 Isomorphic Fields

We have shown that every finite field has prime characteristic p (Theorem 26), which is also the cardinality of its prime field. Note that the prime field operations by repeated addition of multiplicative identity are identical to modulo arithmetic in $\mathbb{Z}_p$. Hence,

Theorem 85. *For a finite field $\mathcal{F}$, its prime field is isomorphic to $\mathbb{Z}_p$ where $p = \mathsf{char}(\mathcal{F})$.*

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \Rightarrow \mathbf{PF}_{\mathcal{F}} \cong \mathbb{Z}_{\mathsf{char}(\mathcal{F})}$$

Consider two finite fields of the same cardinality. By Theorem 78, this means equal prime powers, which happens only when both the primes and powers are equal. Theorem 77 implies that both finite fields must have the same characteristic. Therefore:

Theorem 86. *Two finite fields of the same cardinality have isomorphic prime fields.*

$$\vdash \mathsf{FiniteField}\ \mathcal{F}_1 \wedge \mathsf{FiniteField}\ \mathcal{F}_2 \wedge |\mathsf{F}_1| = |\mathsf{F}_2| \Rightarrow \mathbf{PF}_{\mathcal{F}_1} \cong \mathbf{PF}_{\mathcal{F}_2}$$

Since the minimal polynomials $\mathfrak{m}_a$ for $a \in \mathsf{F}$ are subfield irreducibles (see property (1) in Section 4.3.1), they can be used to construct quotient fields $\mathcal{S}[\mathsf{X}]/(\mathfrak{m}_a)$. If $a \in \pi_{\mathcal{F}}$, a primitive of the finite field (see Section 2.6.3), this can be shown (*e.g.*, see [Belk, 2016, page 50, Theorem 5]):

Theorem 87. *A finite field is isomorphic to the quotient field by the minimal polynomial of a primitive.*

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \wedge \mathcal{S} \preccurlyeq \mathcal{F} \wedge a \in \pi_{\mathcal{F}} \Rightarrow \mathcal{F} \cong \mathcal{S}[\mathsf{X}]/(\mathfrak{m}_a)$$

These ideas lead directly to the uniqueness of finite fields up to isomorphism:

Theorem 88. Uniqueness — *Finite fields of the same cardinality are isomorphic, i.e., they are structurally the same.*

$$\vdash \mathsf{FiniteField}\ \mathcal{F}_1 \wedge \mathsf{FiniteField}\ \mathcal{F}_2 \wedge |\mathsf{F}_1| = |\mathsf{F}_2| \Rightarrow \mathcal{F}_1 \cong \mathcal{F}_2$$

Proof. [4] Let $|\mathsf{F}_1| = |\mathsf{F}_2| = q$. Note that their primes fields are isomorphic by Theorem 86, *i.e.*, there is an isomorphism: $\vartheta : \mathbf{PF}_{\mathcal{F}_1} \to \mathbf{PF}_{\mathcal{F}_2}$. Let $a \in \pi_{\mathcal{F}}$ be a primitive of $\mathcal{F}_1$. Its minimal

[4] This proof, based on quotient fields by minimal polynomials of primitives, is adapted from [Belk, 2016, page 56, Theorem 3]. Similar ideas are given in [Herstein, 1996, Theorem 6.4.2].

polynomial $\mathfrak{m}_a$ in the prime field has coefficients in $\mathbf{PF}_{\mathcal{F}_1}$. Then $\vartheta(\mathfrak{m}_a)$, the polynomial whose coefficients are the ϑ-image of the corresponding coefficients of $\mathfrak{m}_a$, is a subfield polynomial of $\mathcal{F}_2[\mathsf{X}]$ with coefficients in $\mathbf{PF}_{\mathcal{F}_2}$. The isomorphism ϑ ensures that:

- Note that $\mathfrak{m}_a$ is irreducible (property (1) in Section 4.3.1). Thus $\vartheta(\mathfrak{m}_a)$ is also irreducible in $\mathcal{F}_2[\mathsf{X}]$, and it divides $\vartheta(\mathsf{X}^q - \mathsf{X}) = \vartheta(\mathsf{X})^q - \vartheta(\mathsf{X})$, a polynomial in $\mathcal{F}_2[\mathsf{X}]$ which is a product of the minimal polynomials of the elements of $\mathcal{F}_2$ (Theorem 84). Therefore $\vartheta(\mathfrak{m}_a)$ equals to some minimal polynomial $\mathfrak{m}_b$ in $\mathcal{F}_2[\mathsf{X}]$, with $b \in \mathsf{F}_2$.
- both a and b have the same order, so b is a primitive in $\mathcal{F}_2$.
- polynomial division is preserved: the quotient and remainder by a polynomial modulus will map to their respective images, giving in general, when f is the isomorphism between $\mathcal{F}_1$ and $\mathcal{F}_2$:

$$\vdash \mathcal{F}_1 \cong_{(f)} \mathcal{F}_2 \Rightarrow \forall \mathsf{h}.\ \mathsf{ipoly}\ \mathsf{h} \Rightarrow \mathcal{F}_1[\mathsf{X}]/(\mathsf{h}) \cong_{(\mathsf{MAP}\ f)} \mathcal{F}_2[\mathsf{X}]/(f(\mathsf{h}))$$

  where $(\mathsf{MAP}\ f)$ is the isomorphism between polynomials, using f for each coefficient:

$$(\mathsf{MAP}\ f)\left(\sum_{j=0}^{n} c_j \mathsf{X}^j\right) = \sum_{j=0}^{n} f(c_j)\mathsf{X}^j \qquad \text{where } c_j \in \mathsf{F}_1 \text{ gives } f(c_j) \in \mathsf{F}_2$$

Using Theorem 87 and applying the last result for the isomorphic prime fields, taking $\mathsf{h} = \mathfrak{m}_a$, we have:

$$\mathcal{F}_1 \cong \mathbf{PF}_{\mathcal{F}_1}[\mathsf{X}]/(\mathfrak{m}_a) \cong \mathbf{PF}_{\mathcal{F}_2}[\mathsf{X}]/(\vartheta(\mathfrak{m}_a)) = \mathbf{PF}_{\mathcal{F}_2}[\mathsf{X}]/(\mathfrak{m}_b) \cong \mathcal{F}_2 \tag{4.3}$$

□

4.3.3 Existence and Uniqueness: Maximizing Type Generality

Our Theorem 88, stating the uniqueness of fields of the same order is "doubly polymorphic" in both fields: as we can see when we redisplay the theorem with extra type annotations, the related fields may be over different underlying types α and β:

$$\vdash \mathsf{FiniteField}\ (\mathcal{F}_1 : \alpha\ \mathtt{ring}) \wedge \mathsf{FiniteField}\ (\mathcal{F}_2 : \beta\ \mathtt{ring}) \wedge |\mathsf{F}_1| = |\mathsf{F}_2| \Rightarrow \mathcal{F}_1 \cong \mathcal{F}_2$$

However, our "base" Theorem 83, stating the existence of finite fields of prime power order, is over fields of numeric polynomial type:

$$\begin{aligned}\vdash\ & \mathsf{prime}\ (p : \mathtt{num}) \wedge (0 : \mathtt{num}) < (d : \mathtt{num}) \Rightarrow \\ & \exists (\mathcal{F} : \mathtt{num\ poly\ ring}).\ \mathsf{FiniteField}\ \mathcal{F} \wedge |\mathsf{F}| = ((p^d) : \mathtt{num})\end{aligned}$$

This is because we started with the finite field $\mathbb{Z}_p$ for a prime p (Theorem 80), with elements of numeric type. The resulting quotient field $\mathbb{Z}_p[\mathsf{X}]/(\mathsf{h})$, formed by an irreducible h of degree d

(Theorem 81), will have elements of numeric polynomial type.

If desired, we can lift the existence result to an arbitrary type α, as long as that type has enough elements. Here we satisfy that cardinality constraint by requiring the universe of that type (written $\mathcal{U}(:\alpha)$) to be infinite.

The first step exploits the fact that an infinite set A has the same cardinality as the set of all possible (finite) lists of elements drawn from A:

Lemma 89. *There is a bijection between all finite lists with elements type α and the elements themselves.*

$$\vdash \mathsf{INFINITE}\ \mathcal{U}(:\alpha) \Rightarrow \exists f.\, f\ :\ \mathcal{U}(:\alpha\ \texttt{list})\ \leftrightarrow\ \mathcal{U}(:\alpha)$$

We instantiate this lemma with α set to `num` and thereby show the existence of finite fields of the desired order over the type of natural numbers. Finally, because we know that our desired destination type has an infinite cardinality, we can inject homomorphically from the natural numbers into α, giving us:

Theorem 90. *Given an infinite type and a nontrivial prime power, there exist a finite field of that type and order.*

$$\begin{aligned}\vdash \mathsf{prime}\ (p : \texttt{num}) \wedge (0 : \texttt{num}) < (n : \texttt{num}) \wedge \mathsf{INFINITE}\ \mathcal{U}(:\alpha) \Rightarrow \\ \exists (\mathcal{F} : \alpha\ \texttt{ring}).\ \mathsf{FiniteField}\ \mathcal{F} \wedge |\mathsf{F}| = ((p^n) : \texttt{num})\end{aligned}$$

This establishes the existence of finite fields with prime power cardinality, for a generic type, provided its universe is infinite. As a point of interest, we can obtain the same result *via* a perhaps more traditional route: using extension fields and splitting fields.

4.3.4 Extension and Splitting Fields

Given a finite field $(\mathcal{F} : \alpha\ \texttt{ring})$, consider a polynomial t in $\mathcal{F}[\mathsf{X}]$ with $\mathsf{deg\ t} \neq 0$, *i.e.*, not a constant polynomial. Then t has an irreducible polynomial h as its factor. We shall first concentrate on this irreducible h, then show its relationship with t.

Note that the quotient field $\mathcal{F}[\mathsf{X}]/(\mathsf{h})$ from Theorem 81 has type $(\alpha\ \texttt{poly ring})$. It contains X when $\mathsf{deg\ X} < \mathsf{deg\ h}$, *i.e.* $1 < \mathsf{deg\ h}$. Because h divides itself, $\mathsf{h} = \mathsf{h}[\![\mathsf{X}]\!] \equiv \mathsf{0}\ (\mathrm{mod}\ \mathsf{h})$, showing that [5]:

Theorem 91. *Let* h *be a non-linear polynomial with coefficients from a field* $\mathcal{F}$*. Then* X *is a root of* h *in the quotient ring* $\mathcal{F}[\mathsf{X}]/(\mathsf{h})$*.*

$$\vdash \mathsf{Field}\ \mathcal{F} \wedge \mathsf{poly\ h} \wedge 1 < \mathsf{deg\ h} \Rightarrow \mathsf{h} \mid \mathsf{h}[\![\mathsf{X}]\!]$$

Assume that α type is infinite, *i.e.*, $\mathsf{INFINITE}\ \mathcal{U}(:\alpha)$. The bijection between $\mathcal{U}(:\alpha\ \texttt{poly})$ and $\mathcal{U}(:\alpha)$ from Lemma 89 allows the conversion of the quotient field to an extension field $\mathcal{E}$ of type

[5] When h is not required to be irreducible, $\mathcal{F}[\mathsf{X}]/(\mathsf{h})$ is a quotient ring, which becomes a quotient field when h is irreducible.

$(\alpha\ \texttt{ring})$. The constant polynomials in the quotient field becomes a subfield $\mathcal{S}$ of $\mathcal{E}$, which is isomorphic to the coefficients field $\mathcal{F}$ through a bijective map f:

$$\mathcal{F} \cong_f \mathcal{S} \wedge \mathcal{S} \preccurlyeq \mathcal{E}$$

At a high level, one might hope that $\mathcal{F}$ is a subset of $\mathcal{E}$ is true if and only if $\mathcal{E}$ is an extension of $\mathcal{F}$. However, the previous discussion shows that for a typed system like HOL4 there is a subtle difference:

- A field/subfield pair $\mathcal{S} \preccurlyeq \mathcal{F}$ relates fields of the same type, with the same field operations.
- A field $\mathcal{F}$ and its extension field $\mathcal{E}$, though possibly of the same type, are related through an isomorphic subfield $\mathcal{S}$ of $\mathcal{E}$, with extended field operations in $\mathcal{E}$.

Recall that polynomial t has an irreducible factor h, *i.e.*, h has no roots in $\mathcal{F}$. Since the coefficients of t and h are in $\mathcal{F}$, they have images $f(\mathsf{t})$ and $f(\mathsf{h})$ in $\mathcal{S}$. The root X of h in the quotient field $\mathcal{F}[\mathsf{X}]/(\mathsf{h})$ from Theorem 91 corresponds to some element in the extension field $\mathcal{E}$.

Note that any root of $f(\mathsf{h})$ is also a root of $f(\mathsf{t})$. Therefore, the extension field $\mathcal{E}$ contains a root for $f(\mathsf{t})$. The set of roots of $f(\mathsf{t})$ in $\mathcal{E}$ is denoted by $\mathsf{roots}_{\mathcal{E}}\ f(\mathsf{t})$:

$$\begin{aligned}\vdash\ &\mathsf{FiniteField}\ \mathcal{F} \wedge \mathsf{INFINITE}\ \mathcal{U}(:\alpha) \wedge \mathsf{poly\ t} \wedge 0 < \mathsf{deg\ t} \Rightarrow \\ &\exists \mathcal{S}\ \mathcal{E}\ f\ b.\ \mathcal{F} \cong_f \mathcal{S} \wedge \mathcal{S} \preccurlyeq \mathcal{E} \wedge \mathsf{FINITE\ E} \wedge b \in \mathsf{roots}_{\mathcal{E}}\ f(\mathsf{t})\end{aligned}$$

Having field $\mathcal{F}$ and its extension $\mathcal{E}$ of the same type enables successive extensions until $\mathcal{E}$ reduces $f(\mathsf{t})$ into linear factors, *i.e.*, containing all its roots. This is called a splitting field for t, denoted by $\mathsf{splitting}_{\mathcal{E}}\ f(\mathsf{t})$:

$$\begin{aligned}\vdash\ &\mathsf{FiniteField}\ \mathcal{F} \wedge \mathsf{INFINITE}\ \mathcal{U}(:\alpha) \wedge \mathsf{poly\ t} \wedge 0 < \mathsf{deg\ t} \Rightarrow \\ &\exists \mathcal{S}\ \mathcal{E}\ f.\ \mathcal{F} \cong_f \mathcal{S} \wedge \mathcal{S} \preccurlyeq \mathcal{E} \wedge \mathsf{FINITE\ E} \wedge \mathsf{splitting}_{\mathcal{E}}\ f(\mathsf{t})\end{aligned}$$

The splitting field provides another proof for the existence of finite fields of an infinite type:

Proof of Theorem 90. [6]
Given a prime p, a positive n, an infinite type α, one can start with a field $(\mathcal{F} : \alpha\ \texttt{ring})$ isomorphic to $\mathbb{Z}_p$. In this field $\mathcal{F}$, with $|\mathsf{F}| = p$, consider the polynomial $\mathsf{t} = \mathsf{X}^{p^n} - \mathsf{X}$. The smallest splitting field $\mathcal{E}$ of t has all the roots of t. Note that $\mathsf{char}(\mathcal{E}) = p$ by Theorem 79. Thus the formal derivative of t is a constant, sharing no root with t. This shows that t has no multiple roots, *i.e.*, all the p^n roots of t are distinct. Hence the field $(\mathcal{E} : \alpha\ \texttt{ring})$ has precisely p^n elements. □

We have shown that it is possible to follow what one might argue is a standard approach: making use of splitting fields to establish the existence of finite fields of order p^n for all prime p and positive n. However, the pleasant generality of the splitting field theorems is somewhat

[6] Such proofs can be found in, *e.g.*, [Herstein, 1996, Theorem 6.3.3], [Judson, 1994, Theorem 20.5], or [Robinson, 2008, Theorem 10.3.1].

undone by the requirement to add the various INFINITE $\mathcal{U}(:\alpha)$ preconditions. It is therefore a matter of taste as to which one to prefer.

4.3.5 Finite Fields of Finite Type

The requirement that the universe of α be infinite guarantees that an arbitrary amount of splitting can be carried out. Of course, the final splitting field is still finite, and so one is always able to construct an isomorphic finite field over a sufficiently large finite type. For example, if one started with the two-element finite field over `bool`, isomorphic to $\mathbb{Z}_2$, one could eventually carry this procedure out to construct a splitting field over a finite type of cardinality 2^n for any value of positive n. Though it is clear that this procedure can always be carried out, in HOL4 we cannot express the construction because we would need to be able to make a statement asserting the existence of a type of a specific finite cardinality.

4.4 Cyclotomic Polynomials

In a finite field $\mathcal{F}$, each nonzero element in F^* has a nonzero order, say n. The cardinality of $(\mathsf{orders}\ \mathcal{F}^*\ n)$, the set of elements with order equal to n, is given by Theorem 27:

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \Rightarrow \forall n.\ |\mathsf{orders}\ \mathcal{F}^*\ n| = \textbf{if}\ n \mid |\mathsf{F}^*|\ \textbf{then}\ \varphi(n)\ \textbf{else}\ 0$$

The product formed by multiplying all factors from elements of order n is called the n-th cyclotomic polynomial, denoted by $\mathbf{\Phi}_n$:

Definition 92. *Cyclotomic polynomials in a field.*

$$\mathsf{Field}\ \mathcal{F} \Rightarrow \mathbf{\Phi}_n = \prod \{\ \mathsf{X} - a \times \mathbf{1}\ \mid\ a \in \mathsf{F}^* \wedge \mathsf{order}_{\mathcal{F}^*}(a) = n\ \}$$

Since $\mathbf{\Phi}_n$ is a product of monic monomials, its degree is given by the number of factors:

Theorem 93. *The degree of a cyclotomic polynomial in a finite field.*

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \Rightarrow \forall n.\ \deg\ \mathbf{\Phi}_n = \textbf{if}\ n \mid |\mathsf{F}^*|\ \textbf{then}\ \varphi(n)\ \textbf{else}\ 0$$

Cyclotomic polynomials are related to the factorisation of $\mathsf{X}^n - \mathbf{1}$:

Theorem 94. *Let n be a divisor of the number of nonzero elements in a finite field. In the ring of polynomials with coefficients from this finite field, the polynomial $\mathsf{X}^n - \mathbf{1}$ is a product of cyclotomic factors $\mathbf{\Phi}_m$ where m divides n.*

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \wedge n \mid |\mathsf{F}^*| \Rightarrow \mathsf{X}^n - \mathbf{1} = \prod \{\ \mathbf{\Phi}_m\ \mid\ m \in \mathsf{divisors}\ n\ \}$$

Proof. [7] Note that $\mathbf{\Phi}_m$ consists of factors of elements with order m, *i.e.*, $x \in \mathsf{F}^*$ with $x^m = \mathbb{1}$. If

[7] This proof, based on divisibility of polynomials and pairwise coprime factors, is adapted from [Ireland and Rosen, 1990, Proposition 13.3.2].

$m \mid n$, then $x^n = x^{m\left(\frac{n}{m}\right)} = \mathbb{1}$. Therefore x is a root of the polynomial $\mathsf{X}^n - \mathbf{1}$. The factors are pairwise coprime, and their product, $\mathbf{\Phi}_m$, divides $\mathsf{X}^n - \mathbf{1}$:

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \Rightarrow \forall m\ n.\ m \mid n \Rightarrow \mathbf{\Phi}_m \mid \mathsf{X}^n - \mathbf{1}$$

Since two different cyclotomic polynomials, say $\mathbf{\Phi}_m$ and $\mathbf{\Phi}_k$, involve factors associated with elements of different orders n and k, the cyclotomic polynomials are pairwise coprime. It follows that their product over the divisors of n will divide $\mathsf{X}^n - \mathbf{1}$, or

$$\mathsf{X}^n - \mathbf{1} = \mathsf{h} \prod_{m \mid n} \mathbf{\Phi}_m \qquad \text{for some polynomial } \mathsf{h} \tag{4.4}$$

Equating the polynomial degree of both sides using Theorem 93, and applying Euler's φ-function identity (Theorem 58, page 38), h must be a constant. By Equation 4.4, $\mathsf{h} = \mathbf{1}$ because a product of monic polynomials, with a multiplier h, gives a monic result. □

We prove that cyclotomic polynomials have coefficients in any subfield, *i.e.*, they are subfield polynomials:

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \wedge \mathcal{S} \preccurlyeq \mathcal{F} \Rightarrow \forall n.\ \mathsf{poly}_{\mathcal{S}}\ \mathbf{\Phi}_n$$

Since the prime field $\mathbf{PF}_{\mathcal{F}}$ is the smallest subfield of a finite field $\mathcal{F}$ (Theorem 76), which is isomorphic to the field $\mathbb{Z}_{\mathsf{char}(\mathcal{F})}$ (Theorem 85), the coefficients of $\mathbf{\Phi}_n$ are integers. Indeed, using Equation 4.4 with $\mathsf{h} = \mathbf{1}$ and polynomial division, we can deduce, successively, that $\mathsf{X} - \mathbf{1} = \mathbf{\Phi}_1$, then $\mathsf{X}^2 - \mathbf{1} = \mathbf{\Phi}_1 \times \mathbf{\Phi}_2$ giving $\mathbf{\Phi}_2 = \mathsf{X} + \mathbf{1}$, then $\mathsf{X}^3 - \mathbf{1} = \mathbf{\Phi}_1 \times \mathbf{\Phi}_3$ giving $\mathbf{\Phi}_3 = \mathsf{X}^2 + \mathsf{X} + \mathbf{1}$, *etc.*, by keeping track of the divisors of n.

4.4.1 Alternative Presentations of $\mathbf{\Phi}_n$

Our cyclotomic polynomials are defined in a finite field, so that there are elements of various orders to give the factors and form the products $\mathbf{\Phi}_n$. As the statement of Theorem 94 shows, Equation 4.4 holds in a finite field $\mathcal{F}$ with a condition: $n \mid |\mathsf{F}^*|$. This is different from usual treatments [8], where Equation 4.4 holds for all $0 < n$.

Some textbooks, *e.g.*, Rotman [2010] and Herstein [1975], treat the cyclotomic polynomials $\mathbf{\Phi}_n$ as polynomials with rational coefficients, *i.e.*, elements of $\mathbb{Q}[\mathsf{X}]$. They define $\mathbf{\Phi}_n$ as a product of factors from all complex n-th primitive roots of unity:

$$\mathbf{\Phi}_n = \prod_{k=1}^{n} \left(\mathsf{X} - e^{2\pi i \frac{k}{n}}\right) \qquad \text{with } \gcd(k, n) = 1 \text{ and } i^2 = -1$$

then proving that, due to pairs of complex conjugate factors and properties of complex n-th primitive roots of unity, the resulting $\mathbf{\Phi}_n$'s have integer coefficients, *i.e.*, they are indeed in $\mathbb{Z}[\mathsf{X}]$, a subring of $\mathbb{Q}[\mathsf{X}]$. Others, *e.g.*, Herstein [1996] and Garrett [2004], simply define $\mathbf{\Phi}_n$

[8] See, *e.g.*, [Bastida and Lyndon, 1984, Proposition 3.5.6], or [Newman, 2012, Theorem 5.3].

recursively by Equation 4.4 with $\mathsf{h} = \mathbf{1}$. Either approach involves the use of a field (rationals $\mathbb{Q}$ or complex $\mathbb{C}$) which is not finite.

Within finite fields, we can prove that there is always one in which Equation 4.4 holds for any positive n:

Theorem 95. *For $n \neq 0$, the polynomial $\mathsf{X}^n - \mathbf{1}$ is a product of cyclotomic factors of the divisors of n in some finite field.*

$$\vdash 0 < n \Rightarrow \exists \mathcal{F}.\ \mathsf{X}^n - \mathbf{1} = \prod \{ \mathbf{\Phi}_m \mid m \in \mathsf{divisors}\ n \}$$

Proof. Given n, choose a prime p not a factor of n. Then $\gcd(n, p) = 1$, and we can compute $d = \mathsf{order}_n(p)$. This is the smallest exponent $d > 0$ such that $p^d \equiv 1 \pmod{n}$, so $n \mid p^d - 1$. By Theorem 83, there exists a finite field $\mathcal{F}$ of order p^d, and $|\mathsf{F}^*| = p^d - 1$. Apply Theorem 94 with this finite field $\mathcal{F}$ to obtain the desired factorisation of $\mathsf{X}^n - \mathbf{1}$. □

4.4.2 Cyclotomic Factors of $\mathsf{X}^n - 1$

The proof of the AKS Main Theorem (Theorem 115) requires the following result, an application of finite fields:

Theorem 96. *Let $\mathcal{F}$ be a finite field, n be positive, and $d = \mathsf{order}_n(|\mathsf{F}|)$ with $d > 1$. Then $\mathsf{X}^n - \mathbf{1}$ has a monic irreducible factor h of degree d, with the order of X equals to n in the quotient field $\mathcal{F}[\mathsf{X}]/(\mathsf{h})$.*

$$\begin{aligned}\vdash\ & \mathsf{FiniteField}\ \mathcal{F} \wedge 0 < n \wedge 1 < \mathsf{order}_n(|\mathsf{F}|) \Rightarrow \\ & \exists \mathsf{h}.\ \mathsf{monic}\ \mathsf{h} \wedge \mathsf{ipoly}\ \mathsf{h} \wedge \mathsf{h} \mid \mathsf{X}^n - \mathbf{1} \wedge \deg \mathsf{h} = \mathsf{order}_n(|\mathsf{F}|) \wedge \mathsf{order}_{\mathsf{h}}(\mathsf{X}) = n\end{aligned}$$

Proof. Let $d = \mathsf{order}_n(|\mathsf{F}|)$. This means $|\mathsf{F}|^d \equiv 1 \pmod{n}$, or $n \mid |\mathsf{F}|^d - 1$. By Theorem 82, there exists a monic irreducible polynomial z with $\deg \mathsf{z} = d$, giving a quotient field $\mathcal{F}[\mathsf{X}]/(\mathsf{z})$ of order $|\mathsf{F}|^d$.

Let $\mathcal{E} = \mathcal{F}[\mathsf{X}]/(\mathsf{z})$, the polynomial quotient field. Note that $\mathcal{E}$ has a subfield, the constant polynomials, which is isomorphic to $\mathcal{F}$. Therefore the finite field $\mathcal{E}$ can be taken as an extension field of $\mathcal{F}$ (see Section 4.3.4). In the discussion that follows, we identify the subfield of constant polynomials with $\mathcal{F}$, *i.e.*, treating $\mathcal{F} \preccurlyeq \mathcal{E}$.

Note that $|\mathsf{E}^*| = |\mathsf{F}|^d - 1$, and $n \mid |\mathsf{E}^*|$. This fact is significant:

(a) Since Theorem 94 applies, the subfield polynomial $\mathsf{X}^n - \mathbf{1}$ is a product of cyclotomic factors. In particular, $\mathbf{\Phi}_n \mid (\mathsf{X}^n - \mathbf{1})$.

(b) Moreover, Theorem 27 applies, giving some nonzero element a with order equal to n, *i.e.*, $a \in \mathsf{E}^*$ with $\mathsf{order}_{\mathcal{E}^*}(a) = n$.

Take $\mathsf{h} = \mathfrak{m}_a$, the minimal polynomial of this element a with order n. Then h is monic and irreducible in $\mathcal{F}$. Its degree is given by (see, *e.g.*, [Belk, 2016, page 59, Corollary 7 and 8]):

$$\vdash \mathsf{FiniteField}\ \mathcal{E} \wedge \mathcal{F} \preccurlyeq \mathcal{E} \Rightarrow \forall a.\ a \in \mathsf{E}^* \Rightarrow \deg \mathfrak{m}_a = \mathsf{order}_{\mathsf{order}_{\mathcal{E}^*}(a)}(|\mathsf{F}|) \tag{4.5}$$

In other words, $\mathsf{deg}\ \mathsf{h} = d$. Note that $\mathbf{\Phi}_n$ collects all the factors from elements of order n (see Definition 92), so it is a product of all minimal polynomials of these elements:

$$\vdash \mathsf{FiniteField}\ \mathcal{E} \wedge \mathcal{F} \preccurlyeq \mathcal{E} \Rightarrow \forall n.\ \mathbf{\Phi}_n = \prod \{\ \mathfrak{m}_a \mid a \in \mathsf{orders}\ \mathcal{E}^*\ n\ \} \tag{4.6}$$

Thus $\mathsf{h} \mid \mathbf{\Phi}_n$. With $\mathbf{\Phi}_n \mid (\mathsf{X}^n - \mathbf{1})$, this implies $\mathsf{h} \mid (\mathsf{X}^n - \mathbf{1})$.

Now X is a root of h in the quotient field $\mathcal{F}[\mathsf{X}]/(\mathsf{h})$ (Theorem 91). Note that $\mathsf{deg}\ \mathsf{h} = \mathsf{deg}\ \mathsf{z}$, and both h and z are monic and irreducible. Their quotient fields are isomorphic by uniqueness of finite fields of the same order (Theorem 88). Let $\mathsf{Y} \in \mathsf{E}^*$ be the isomorphic element corresponding to X. Then Y is a root of h in $\mathcal{E}$, where $\mathsf{h} \mid \mathbf{\Phi}_n$. Therefore Y is also a root of $\mathbf{\Phi}_n$. Again, $\mathbf{\Phi}_n$ is a product of factors of elements with order equal to n (Definition 92), hence $\mathsf{order}_{\mathsf{h}}(\mathsf{Y}) = n$. Since X is the counterpart of Y, and isomorphism preserves element orders, we have $\mathsf{order}_{\mathsf{h}}(\mathsf{X}) = n$. □

Note how this proof involves all the key topics in this chapter: existence and uniqueness of finite fields, subfields, extension fields, minimal polynomials, and cyclotomic polynomials.

Scripts Scripts for the classification of finite fields are located in `algebra/finitefield`, with `ffBasic` for basic properties, `ffAdvanced` for advanced properties, `ffCyclo` for cyclotomic polynomials, and `ffExist` for existence and uniqueness.

4.5 Summary

We treat a finite field as an extension of its subfields, using the vector space idea. We define the degree of a finite field relative to its prime subfield (Section 4.1.3), and show that the cardinality of a finite field is its characteristic raised to its degree (Theorem 77), *i.e.*, alway a prime power. Every element in a finite field is a linear combination of the basis from a subfield, showing that every element is associated with a minimal polynomial (Section 4.3.1) which is monic and irreducible when considered as a subfield polynomial. Counting monic irreducibles leads to the existence of finite fields (Theorem 83) by quotient fields of irreducibles (Theorem 81). Mapping minimal polynomials relative to prime subfields establish an isomorphism between two finite fields of the same cardinality (Theorem 87, hence finite fields of the same cardinality are unique up to isomorphism (Theorem 88). We define the cyclotomic polynomials, and establish their product over field orders (Theorem 95). By working with this cyclotomic product, we find a special irreducible factor h of $\mathsf{X}^k - \mathbf{1}$ with $\mathsf{order}_{\mathsf{h}}(\mathsf{X}) = k$ (Theorem 96). With the existence of this special irreducible h, we are ready to work on the proof of the AKS Main Theorem.

4.6 Remarks

Our formal proof of the classification of finite field, as well as subfields of finite fields, was presented in Chan and Norrish [2019a]. We take this opportunity to review the abstract algebra

libraries in other theorem provers.

There is a long tradition in the formalisation of abstract algebra, starting with Peter Aczel's Galois project (Aczel [1995]). The LEGO system was developed, from which Barthe [1994] gave a formal proof of the unsolvability of the symmetric group S_n with $n \geq 5$ and Bailey [1998] formalised part of Galois theory. These were pioneering efforts, with the first building up from monoids to rings, and the second containing basic vector spaces.

There was another attempt to formalise the Fundamental Theorem of Galois theory by Curiel [2011] in Isabelle/HOL. The work involved rings and fields, up to field automorphisms. Further progress had difficulties with adapting a polynomial library for the more advanced work.

The formalisation of the Odd Order Theorem by Gonthier et al. [2013] in Coq was the cumulative effort by many group theory experts. In this work, a great deal of abstract algebra was developed, with Galois theory playing a vital role, and finite fields were essential ingredients. Subjects covered by this project include vector spaces, extension fields, splitting fields, minimal polynomials and cyclotomic polynomials. These became the Mathematical Components library (Mahboubi and Tassi [2018]) in Coq.

The Isabelle/HOL Algebra Library (Ballarin et al. [2016]) is another substantial library for abstract algebra work, including rings, ideals, quotient rings, fields and polynomials. Based on polynomials and subfields of complex numbers, Thiemann and Yamada [2016] formalised algebraic numbers in Isabelle/HOL. Their theorems were not restricted to finite fields. Ongoing work on algebraic numbers in Coq based on field theory can be found in work done by Cohen [2012].

The Mizar Mathematical Library (Bancerek et al. [2018]) has several results covered in this chapter, including work on abelian groups, fields and vector spaces by Kusak et al. [1989], and primitive roots of unity and cyclotomic polynomials by Arneson and Rudnicki [2003]. Note that Arneson et al. [2003] formalised Witt's proof of the Wedderburn Theorem[9] using skew-fields and vector spaces over skew-fields, which are not treated in our work.

[9] Skew fields are fields without the commutative requirement for multiplication, and Wedderburn Theorem asserts that every finite skew field must be commutative, *i.e.*, a field.

Chapter 5

AKS Main Theorem

This chapter is devoted to this main result: given a number, if there is a finite field in which the number satisfies certain criteria, then the number must be a power of the field's characteristic. It is the key to the correctness proof of the AKS algorithm. The proof involves an ingenious play with various sets implied by the introspective relationship. Of particular interest is a finite set related to the introspective polynomials, with polynomial coefficients taken as elements from the finite field. The introspective checks put a condition on its cardinality, and the AKS parameters turn this condition into an injective map between two finite sets. This guarantees the main result, for otherwise the injective map would violate the Pigeonhole principle.

> *Reductio ad absurdum*, which Euclid loved so much,
> is one of a mathematician's favourite weapons.
> It is a far finer gambit than any chess gambit:
> a chess player may offer the sacrifice of a pawn
> or even a piece, but a mathematician offers the game.
> — Godfrey Harold Hardy (1940)[1]

5.1 Main Theorem

We shall finish the proof that a number n passing the AKS tests must be prime (Theorem 73):

$$\vdash \mathsf{aks}\ n \Rightarrow \mathsf{prime}\ n$$

by establishing this AKS Main Theorem:

[Theorem 115] *If a number n satisfies the AKS criteria in a finite field $\mathcal{F}$ of prime cardinality, then n is a power of the characteristic of the finite field.*

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \wedge |\mathsf{F}| = \mathsf{char}(\mathcal{F}) \Rightarrow \forall n\ k.\ \mathsf{aks\_criteria}\ \mathcal{F}\ n\ k \Rightarrow n\ \mathsf{power\_of}\ \mathsf{char}(\mathcal{F})$$

[1] From his book *A Mathematician's Apology*.

Compared to a similar theorem we have proved before ([Chan and Norrish, 2015, Theorem 11]), there are two improvements. We proved the main theorem without developing advanced features of finite fields (Chapter 4). Now the improvements require these topics: existence and uniqueness of finite fields, and cyclotomic factors.

5.1.1 Improvement #1: nature of parameter k

As depicted in the dependency diagram (Figure 1.1, page 10), the whole purpose of Chapter 4 is to establish the existence of a special irreducible factor h of $\mathsf{X}^k - \mathbf{1}$ such that $\mathsf{order}_\mathsf{h}(\mathsf{X}) = k$ (Theorem 96). In Chan and Norrish [2015], we proved the main theorem by assuming the parameter k is prime. This is in accordance with the original AKS paper (Agrawal et al. [2002])[2] and makes the existence of such an irreducible h simple.

Pick any monic irreducible factor h of $\mathsf{X}^k - \mathbf{1}$ with $k > 1$ and $\mathsf{deg}\ \mathsf{h} > 1$. Then in the quotient field $\mathcal{F}[\mathsf{X}]/(\mathsf{h})$, because $\mathsf{h} \mid \mathsf{X}^k - \mathbf{1}$, $\mathsf{X}^k - \mathbf{1} \equiv \mathbf{0} \pmod{\mathsf{h}}$, so $\mathsf{X}^k \equiv \mathbf{1} \pmod{\mathsf{h}}$. This shows that X raised to exponent k is the multiplicative identity in the multiplicative group of the quotient field. The multiplicative order of X, which is $\mathsf{order}_\mathsf{h}(\mathsf{X})$, must divide the exponent k (Theorem 10). Since $\mathsf{X} \not\equiv \mathbf{1} \pmod{\mathsf{h}}$, $\mathsf{order}_\mathsf{h}(\mathsf{X}) \neq 1$.

Assuming k is prime, this gives immediately $\mathsf{order}_\mathsf{h}(\mathsf{X}) = k$ for *any* monic irreducible factor h with $\mathsf{deg}\ \mathsf{h} > 1$. This is [Chan and Norrish, 2015, Theorem 14]. Otherwise, we have to go through Theorem 96 to obtain the special monic irreducible factor h. As noted after the proof (page 71), that result depends on all the advanced topics in Chapter 4.

5.1.2 Improvement #2: use of cofactor $q = n \mathsf{\ div\ } p$

In Chan and Norrish [2015], the constants formulated in the main theorem are roughly those in the original AKS paper Agrawal et al. [2002], but the constants in the AKS Main Theorem here are those in the revised AKS paper Agrawal et al. [2004]. The difference is a factor of 2, with the new constants smaller. This reflects our effort to match the better constants in the AKS paper, which can be seen by comparing Algorithm 1 with Algorithm 2.

The constants do not affect the main conclusion, that the input number n must be a power of its special prime factor p. However, the constants do affect the paths taken to arrive at the conclusion.

In Section 5.2, we shall define the introspective exponents, The bigger constants suit a proof just using n and p as introspective exponents, while the smaller constants suit a proof using n, p and cofactor $q = n \mathsf{\ div\ } p$ as introspective exponents. This is because two introspective exponents x and y can generate the set $(\widehat{\mathcal{N}}\ x\ y\ m)$ in Section 5.6. The choice for the generators affects the constants in main theorem. The bigger constants correspond to taking $x = n$ and $y = p$, and the smaller constants take $x = p$ and $y = q$.

That the cofactor q is an introspective exponent is proved in Theorem 102. In that proof, we use the fact that $\mathsf{X}^k - \mathbf{1}$ is a product of monic irreducible factors. This comes from the fact that $\mathsf{X}^k - \mathbf{1}$ is a product of cyclotomic factors (Section 4.4, Equation 4.4), and each cyclotomic

[2] This has been discusse in Section 1.2.

factor is a product of minimal polynomials (Section 4.4.2, Equation 4.6), which are monic and irreducible (Section 4.3.1). Thus the validity of the result depends on the advanced topics in Chapter 4.

Before embarking on a proof of the AKS Main Theorem, we need a deeper understanding of the introspective relationship.

5.2 Introspective Relation

As shown in Algorithm 1, the introspective checks take the following form (Equation 1.1):

$$\forall c. \quad 0 < c \wedge c \leq s \Rightarrow (\mathsf{X} + c)^n \equiv \mathsf{X}^n + c \pmod{n,\ \mathsf{X}^k - 1}$$

where $s = (\mathsf{SQRT}\ \varphi(k))\lceil \log n \rceil$. In the context of the ring $\mathbb{Z}_n$, the first $(\mathrm{mod}\ n)$ equivalence becomes an equality, *e.g.*, $x \equiv y \pmod{n}$ is the same as $x = y$ in $\mathbb{Z}_n$. This leaves the symbol $(\equiv)$ to indicate the polynomial modulo equivalence in $\mathbb{Z}_n[\mathsf{X}]$, the ring of polynomials with coefficients from $\mathbb{Z}_n$:

$$\forall c.\ 0 < c \wedge c \leq s \Rightarrow (\mathsf{X} + c)^n \equiv \mathsf{X}^n + c \pmod{\mathsf{X}^k - 1} \tag{5.1}$$

Thus the introspective checks can be viewed as polynomial modulo equivalence in $\mathbb{Z}_n[\mathsf{X}]$. Rewriting using the notation of polynomial substitution, both sides are strikingly similar:

$$(\mathsf{X} + c)^n[\![\mathsf{X}]\!] \equiv (\mathsf{X} + c)[\![\mathsf{X}^n]\!] \pmod{\mathsf{X}^k - 1} \tag{5.2}$$

since for any polynomial p, substitution gives $\mathsf{p}[\![\mathsf{X}]\!] = \mathsf{p}$ and $(\mathsf{X} + c)[\![\mathsf{p}]\!] = \mathsf{p} + c$. Superficially, by simply sliding the exponent n across the substitution brackets, the left side is transformed into the right side. We use a special notation for the introspective relationship:

Definition 97. *An exponent n is* introspective *to a polynomial p under modulus $(\mathsf{X}^k - 1)$, denoted by $n \overset{k}{\bowtie} \mathsf{p}$, when:*

$$n \overset{k}{\bowtie} \mathsf{p} \overset{\mathsf{def}}{=} \mathsf{poly}\ \mathsf{p} \wedge \mathsf{p}^n \equiv \mathsf{p}[\![\mathsf{X}^n]\!] \pmod{\mathsf{X}^k - 1}$$

Note that the symbol for introspective relation $(\overset{k}{\bowtie})$ hides the underlying ring $\mathcal{R}$, because we would like to investigate this relationship in a general context, not just for $\mathbb{Z}_n$, The underlying ring will be included as a subscript when it is of significance, *e.g.*, $\overset{k}{\bowtie}_{\mathbb{Z}_n}$.

With this notation, the introspective checks verify, for the input number n, the identities $n \overset{k}{\bowtie} \mathsf{X} + c$ in $\mathbb{Z}_n[\mathsf{X}]/(\mathsf{X}^k - 1)$ for $0 < c \leq s$ up to the introspective limit s. Moreover, the fact that a prime is introspective to any monomial (Theorem 69) can be restated as:

Theorem 98. *For a finite field, its characteristic is introspective to any monomial $\mathsf{X} + c$.*

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \wedge 0 < k \Rightarrow \mathsf{char}(\mathcal{F}) \overset{k}{\bowtie} \mathsf{X} + c$$

Proof. Let $p = \mathsf{char}(\mathcal{F})$, then p is prime (Theorem 26). Using Fermat's Little Theorem for monomials (Theorem 39) and polynomial evaluation,

$$(\mathsf{X} + c)^p \equiv \mathsf{X}^p + c = (\mathsf{X} + c)[\![\mathsf{X}^p]\!] \pmod{p}$$

Their difference, $0 \pmod{p}$, is divisible by $\mathsf{X}^k - 1$. By Definition 97, $p \overset{k}{\bowtie} \mathsf{X} + c$. □

The introspective relation ($\overset{k}{\bowtie}$) has some fundamental properties:

Theorem 99. *Introspective relation is multiplicative for introspective exponents.*

$$\vdash \mathsf{Ring}\ \mathcal{R} \Rightarrow \forall k.\ 0 < k \Rightarrow \forall m\ n\ \mathsf{p}.\ n \overset{k}{\bowtie} \mathsf{p} \wedge m \overset{k}{\bowtie} \mathsf{p} \Rightarrow nm \overset{k}{\bowtie} \mathsf{p}$$

Proof. Given $n \overset{k}{\bowtie} \mathsf{p}$, this means $\mathsf{p}^n \equiv \mathsf{p}[\![\mathsf{X}^n]\!] \pmod{\mathsf{X}^k - 1}$.
Also $m \overset{k}{\bowtie} \mathsf{p}$ means $\mathsf{p}^m \equiv \mathsf{p}[\![\mathsf{X}^m]\!] \pmod{\mathsf{X}^k - 1}$. Therefore:

$$(\mathsf{p}[\![\mathsf{X}]\!]^m - \mathsf{p}[\![\mathsf{X}^m]\!]) \text{ is divisible by } (\mathsf{X}^k - 1)$$

By substituting every X of the above statement by X^n,

$$(\mathsf{p}[\![\mathsf{X}^n]\!]^m - \mathsf{p}[\![(\mathsf{X}^n)^m]\!]) \text{ is divisible by } ((\mathsf{X}^n)^k - 1)$$

Note that:

- First term $\mathsf{p}[\![\mathsf{X}^n]\!]^m \equiv (\mathsf{p}^n)^m \pmod{\mathsf{X}^k - 1}$ by $n \overset{k}{\bowtie} \mathsf{p}$ above,
- Second term $\mathsf{p}[\![(\mathsf{X}^n)^m]\!] = \mathsf{p}[\![\mathsf{X}^{nm}]\!]$ by double exponentiation,
- Divisor $(\mathsf{X}^n)^k - 1 = (\mathsf{X}^k)^n - 1$ is also divisible by $\mathsf{X}^k - 1$, by geometric series.[3]

Overall, $\mathsf{p}^{nm} \equiv \mathsf{p}[\![\mathsf{X}^{nm}]\!] \pmod{\mathsf{X}^k - 1}$. By Definition 97, $nm \overset{k}{\bowtie} \mathsf{p}$. □

Theorem 100. *Introspective relation is multiplicative for introspective polynomials.*

$$\vdash \mathsf{Ring}\ \mathcal{R} \Rightarrow \forall k.\ 0 < k \Rightarrow \forall \mathsf{p}\ \mathsf{q}\ n.\ n \overset{k}{\bowtie} \mathsf{p} \wedge n \overset{k}{\bowtie} \mathsf{q} \Rightarrow n \overset{k}{\bowtie} \mathsf{p} \times \mathsf{q}$$

Proof. Given $n \overset{k}{\bowtie} \mathsf{p}$, this means $\mathsf{p}^n \equiv \mathsf{p}[\![\mathsf{X}^n]\!] \pmod{\mathsf{X}^k - 1}$.
Also given $n \overset{k}{\bowtie} \mathsf{q}$, this means $\mathsf{q}^n \equiv \mathsf{q}[\![\mathsf{X}^n]\!] \pmod{\mathsf{X}^k - 1}$.
Multiplying both polynomial congruences, and noting:

[3] The sum of a geometric sequence:

$$1 + r + r^2 + \cdots + r^{n-1} = \frac{1 - r^n}{1 - r}$$

corresponds to the following polynomial identity by long division:

$$\mathsf{Y}^n - 1 = (\mathsf{Y} - 1)(\mathsf{Y}^{n-1} + \cdots + \mathsf{Y}^2 + \mathsf{Y} + 1)$$

Putting $\mathsf{Y} = \mathsf{X}^k$, this shows $(\mathsf{X}^k)^n - 1$ is divisible by $\mathsf{X}^k - 1$.

- $\mathsf{p}^n \times \mathsf{q}^n = (\mathsf{p} \times \mathsf{q})^n$, and
- $\mathsf{p}[\![\mathsf{X}^n]\!] \times \mathsf{q}[\![\mathsf{X}^n]\!] = (\mathsf{p} \times \mathsf{q})[\![\mathsf{X}^n]\!]$

we have $(\mathsf{p} \times \mathsf{q})^n \equiv (\mathsf{p} \times \mathsf{q})[\![\mathsf{X}^n]\!] \pmod{\mathsf{X}^k - 1}$. By Definition 97, $n \overset{k}{\bowtie} \mathsf{p} \times \mathsf{q}$. □

There are two advantages of working in a finite field $\mathcal{F}$ with characteristic $p = \mathsf{char}(\mathcal{F})$:

- We get, for free, the result: $p \overset{k}{\bowtie} \mathsf{X} + c$, by Theorem 98 since p is prime (Theorem 26).
- The modulus polynomial $\mathsf{X}^k - 1$ shall have a special irreducible factor h by Theorem 96:[4]

$$\begin{aligned}&\vdash \mathsf{FiniteField}\ \mathcal{F} \wedge 0 < k \wedge 1 < \mathsf{order}_k(|\mathsf{F}|) \Rightarrow \\ &\quad \exists \mathsf{h}.\ (\mathsf{monic}\ \mathsf{h} \wedge \mathsf{ipoly}\ \mathsf{h} \wedge \mathsf{h} \mid \mathsf{X}^k - 1) \wedge \deg \mathsf{h} = \mathsf{order}_k(|\mathsf{F}|) \wedge \mathsf{order}_{\mathsf{h}}(\mathsf{X}) = k\end{aligned}$$

Both will play significant roles in the following. Let us take a closer look.

5.3 Introspective Sets

Our first advantage means that we now have two members in the introspective relation $\overset{k}{\bowtie}$, with $\mathsf{X} + c$ for $0 < c \leq s$:

- $n \overset{k}{\bowtie} \mathsf{X} + c$, by introspective checks on n and shifting (Section 3.6).
- $p \overset{k}{\bowtie} \mathsf{X} + c$, by Theorem 98, where $p = \mathsf{char}(\mathcal{F})$.

In view of this, these two sets will be of interest.

Definition 101. *The sets of introspective exponents and introspective polynomials.*

$$\begin{aligned}\mathcal{N} &\overset{\text{def}}{=} \{\ m \mid \gcd(m,k) = 1 \wedge \forall c.\ 0 < c \wedge c \leq s \Rightarrow m \overset{k}{\bowtie} \mathsf{X} + c\ \} \\ \mathcal{P} &\overset{\text{def}}{=} \{\ \mathsf{p} \mid \mathsf{poly}\ \mathsf{p} \wedge \forall m.\ m \in \mathcal{N} \Rightarrow m \overset{k}{\bowtie} \mathsf{p}\ \}\end{aligned}$$

The introspective checks give the following introspective monomials:

$$\forall c.\ 0 < c \wedge c \leq s \Rightarrow \mathsf{X} + c \in \mathcal{P}$$

Trivially, $1 \in \mathcal{P}$ and $\mathsf{X} \in \mathcal{P}$. Note that $\gcd(n,k) = 1$ by parameter $\mathsf{good}\ k$, and $\gcd(p,k) = 1$ by prime p being a factor of n:

$$\vdash 1 < n \wedge \mathsf{prime}\ p \wedge p \mid n \Rightarrow \forall k.\ \gcd(n,k) = 1 \Rightarrow \gcd(p,k) = 1$$

Hence there are at least two introspective exponents:

$$n \in \mathcal{N},\ \text{and}\ p \in \mathcal{N}$$

[4] This special irreducible factor eliminates the need for the parameter k to be prime, see Section 5.1.1.

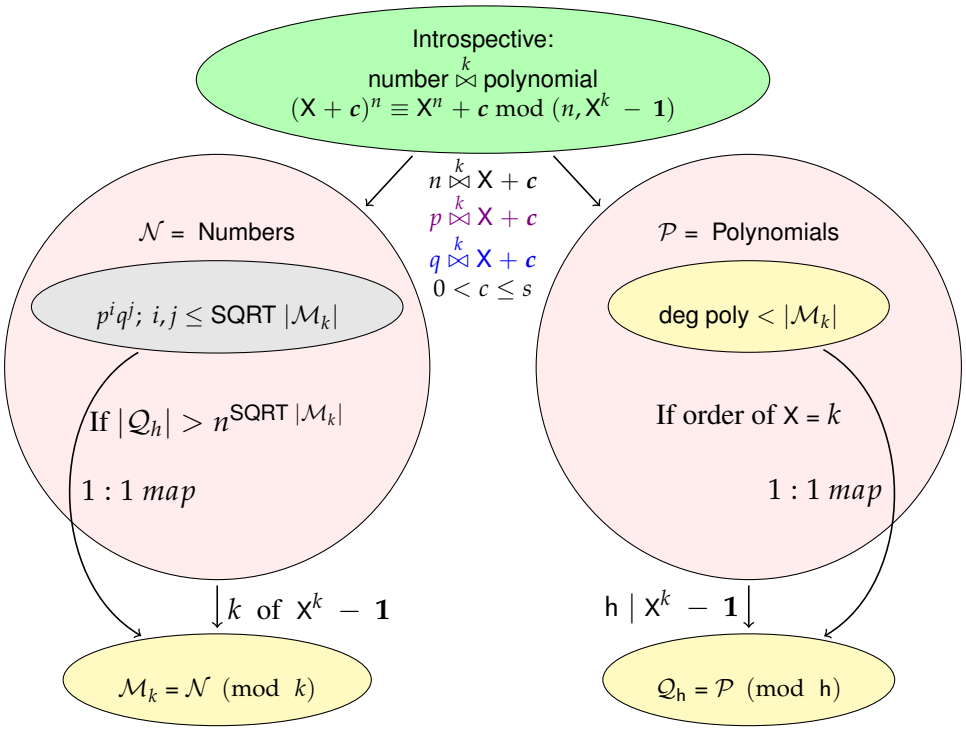


Figure 5.1: Sketch of the AKS proof. The introspective relations of n and p, a prime divisor of n, together with the cofactor $q = n \text{ div } p$, give rise to two sets $\mathcal{N}$ and $\mathcal{P}$ (Section 5.3). By taking modulo of k and h, an irreducible factor of $X^k - 1$, respectively, the sets $\mathcal{N}$ and $\mathcal{P}$ map, correspondingly, to two finite sets $\mathcal{M}_k$ and $\mathcal{Q}_h$ (Section 5.4). Two finite subsets of $\mathcal{N}$ and $\mathcal{P}$ can be crafted such that injective maps between finite sets can be constructed, as illustrated, if the parameters k and s are suitably chosen to satisfy the "if" conditions (Section 5.5 and Section 5.6). Once these "if" conditions are established, if n is not a perfect power of p, the grey set will have more than $|\mathcal{M}_k|$ elements. This is impossible as the injective map on the left will contradict the Pigeonhole Principle (Section 5.7). Therefore n must be a perfect power of its prime divisor p.

Since $p \mid n$, the quotient $q = n \text{ div } p$ is of special interest. We shall call q the *cofactor* of p with respective to n. Indeed, q is also an introspective exponent due to $\gcd(p, k) = 1$:[5]

Theorem 102. *Let a finite field have characteristic p. If n is an introspective exponent and p divides n, then under certain circumstances $q = n \text{ div } p$ is also an introspective exponent.*

$$\vdash \text{FiniteField } \mathcal{F} \wedge \gcd(k, |\mathsf{F}|) = 1 \wedge 1 < \text{order}_k(|\mathsf{F}|) \wedge p = \text{char}(\mathcal{F}) \wedge p \mid n \wedge n \overset{k}{\bowtie} \mathsf{X} + c \Rightarrow n \text{ div } p \overset{k}{\bowtie} \mathsf{X} + c$$

Proof. [6] Let $q = n \text{ div } p$. Working in $(\text{mod } \mathsf{X}^k - \mathbf{1})$, the steps are:

$$\begin{aligned} ((\mathsf{X} + c)^q)^p &\equiv (\mathsf{X} + c)^n && \text{by } qp = n \\ &\equiv \mathsf{X}^n + c && \text{by given } n \overset{k}{\bowtie} \mathsf{X} + c \\ &\equiv (\mathsf{X}^q)^p + c && \text{by } n = qp \\ &\equiv (\mathsf{X}^q + c)^p && \text{by Theorem 98 for } \mathsf{X}^q + c \end{aligned}$$

The last result shows that $((\mathsf{X} + c)^q)^p - (\mathsf{X}^q + c)^p \equiv \mathbf{0}$. Note that $p = \text{char}(\mathcal{F})$ is prime for a finite field (Theorem 26). Using the binomial expansion with exponent a prime characteristic (Theorem 40), we have $((\mathsf{X} + c)^q - (\mathsf{X}^q + c))^p \equiv \mathbf{0}$. Since $\mathsf{X}^k - \mathbf{1}$ has no repeated root, it is a product of a set of monic irreducibles. The exponent p can be dropped by applying Theorem 22:

$$\vdash \text{Field } \mathcal{F} \wedge \text{FINITE } S \wedge \text{miset } S \wedge \text{poly p} \wedge \mathsf{p}^n \equiv \mathbf{0} \ (\text{mod} \prod S) \Rightarrow \mathsf{p} \equiv \mathbf{0} \ (\text{mod} \prod S)$$

giving $((\mathsf{X} + c)^q - (\mathsf{X}^q + c)) \equiv \mathbf{0}$. Hence $(\mathsf{X} + c)^q \equiv \mathsf{X}^q + c$, and $q \overset{k}{\bowtie} \mathsf{X} + c$. □

Therefore, we have:

Corollary 103. *In a finite field $\mathcal{F}$ with $p = \text{char}(\mathcal{F})$, if $n \in \mathcal{N}$ and $p \mid n$ and other conditions, then $\mathcal{N}$ has both p and cofactor $q = n \text{ div } p$.*

$$\vdash \text{FiniteField } \mathcal{F} \wedge \gcd(k, |\mathsf{F}|) = 1 \wedge 1 < \text{order}_k(|\mathsf{F}|) \wedge \text{char}(\mathcal{F}) \mid n \wedge n \in \mathcal{N} \Rightarrow \text{char}(\mathcal{F}) \in \mathcal{N} \wedge n \text{ div } \text{char}(\mathcal{F}) \in \mathcal{N}$$

Recall the fundamental properties of the introspective relation, with multiplicative exponents (Theorem 99) and multiplicative polynomials (Theorem 100). This implies that the sets $\mathcal{N}$ and $\mathcal{P}$ are explosively large. Due to this, their finite counterparts are easier to investigate.

5.4 Modulo Sets

One way to get a finite counterpart from an infinite set is by looking at remainders after division, or image of the set under some modulus. For the exponents set $\mathcal{N}$, there is a natural modulus:

[5] This cofactor helps to match the constants in the AKS pseudocode, see Section 5.1.2.

[6] This proof is inspired *via* private communication with Terence Tao, who outlined an approach in Terence Tao [2009].

following Section 3.3.2, we use the parameter k that provides a big-enough $\mathsf{order}_k(n)$. We then define $\mathcal{M}_k$ to be the finite analogue of $\mathcal{N}$:

$$\mathcal{M}_k \stackrel{\text{def}}{=} \{\ m \bmod k \ \mid\ m \in \mathcal{N}\ \}$$

It is then easy to estimate the cardinality of $\mathcal{M}_k$:

Theorem 104. *The cardinality of set $\mathcal{M}_k$ is bounded.*

$$\vdash \mathsf{Ring}\ \mathcal{R} \wedge 1 < k \Rightarrow \forall n.\ n \in \mathcal{N} \Rightarrow \mathsf{order}_k(n) \leq |\mathcal{M}_k| \wedge |\mathcal{M}_k| \leq \varphi(k)$$

Proof. For the upper bound, by Definition 101, an introspective exponent $n \in \mathcal{N}$ is coprime to k. Its remainder under modulo k is less than k but still coprime to k:

$$\vdash \gcd(k, n) = \textbf{if}\ k = 0\ \textbf{then}\ n\ \textbf{else}\ \gcd(n \bmod k, k)$$

Since $\varphi(k)$ is the count of coprimes not exceeding k, it follows that $|\mathcal{M}_k| \leq \varphi(k)$.

For the lower bound, since $n \in \mathcal{N}$ implies $\forall j.\ n^j \in \mathcal{N}$ by Theorem 99, all their remainders under modulo k are in $\mathcal{M}_k$. The number of such distinct remainders is, by definition, $\mathsf{order}_k(n)$. Therefore $\mathsf{order}_k(n) \leq |\mathcal{M}_k|$. □

Similarly, our finite analogue of the polynomials set $\mathcal{P}$ is:

$$\mathcal{Q}_{\mathsf{h}} \stackrel{\text{def}}{=} \{\ \mathsf{p} \bmod \mathsf{h}\ \mid\ \mathsf{p} \in \mathcal{P}\ \}$$

where we make use of h, a special irreducible factor of $\mathsf{X}^k - \mathbf{1}$ from the second advantage (Section 5.2). Note that polynomial equivalences in modulus $\mathsf{z} = \mathsf{X}^k - \mathbf{1}$ are preserved when the modulus z is changed to its factor $\mathsf{h} \mid \mathsf{z}$:

Theorem 105. *Polynomial modulo equivalence holds for modulus factor.*

$$\begin{aligned}\vdash\ & \mathsf{Ring}\ \mathcal{R} \wedge \mathsf{Ulead}\ \mathcal{R}\ \mathsf{z} \wedge \mathsf{Ulead}\ \mathcal{R}\ \mathsf{h} \wedge \mathsf{h} \mid \mathsf{z} \Rightarrow \\ & \forall \mathsf{p}\ \mathsf{q}.\ \mathsf{poly}\ \mathsf{p} \wedge \mathsf{poly}\ \mathsf{q} \wedge \mathsf{p} \equiv \mathsf{q}\ (\bmod\ \mathsf{z}) \Rightarrow \mathsf{p} \equiv \mathsf{q}\ (\bmod\ \mathsf{h})\end{aligned}$$

Proof. When the difference $(\mathsf{p} - \mathsf{q})$ is divisible by z, and $\mathsf{h} \mid z$, the difference is also divisible by h by transitivity of divisibility. □

An irreducible polynomial h gives a polynomial modulo field $\mathcal{F}[\mathsf{X}]/(\mathsf{h})$, and its nonzero elements $\mathcal{F}_{\mathsf{h}}^*[\mathsf{X}]$ form a multiplicative group (Theorem 28):

$$\begin{aligned}&\vdash \mathsf{Field}\ \mathcal{F} \Rightarrow \forall \mathsf{h}.\ \mathsf{ipoly}\ \mathsf{h} \Rightarrow \mathsf{Field}\ \mathcal{F}[\mathsf{X}]/(\mathsf{h}) \\ &\vdash \mathsf{Field}\ \mathcal{F} \Rightarrow \forall \mathsf{h}.\ \mathsf{ipoly}\ \mathsf{h} \Rightarrow \mathsf{Group}\ \mathcal{F}_{\mathsf{h}}^*[\mathsf{X}]\end{aligned}$$

Recall that the special irreducible factor h of $\mathsf{X}^k - \mathbf{1}$ has the property $\mathsf{order}_{\mathsf{h}}(\mathsf{X}) = k$. With this condition we have:

Theorem 106. *In the polynomial field, powers of* X *are distinct.*

$$\vdash \mathsf{Field}\ \mathcal{F} \wedge \mathsf{ipoly}\ \mathsf{h} \wedge 1 < \deg \mathsf{h} \Rightarrow \\ \forall m\ n.\ m < \mathsf{order}_{\mathsf{h}}(\mathsf{X}) \wedge n < \mathsf{order}_{\mathsf{h}}(\mathsf{X}) \Rightarrow (\mathsf{X}^m \equiv \mathsf{X}^n \pmod{\mathsf{h}} \iff m = n)$$

Proof. Since $\mathbb{Z}_p$ is a finite field, the polynomial field by modulus h is also a finite field:

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \Rightarrow \forall \mathsf{p}.\ \mathsf{ipoly}\ \mathsf{p} \Rightarrow \mathsf{FiniteField}\ \mathcal{F}[\mathsf{X}]/(\mathsf{p})$$

Hence the multiplicative group of this finite field is a finite group. Thus we can apply:

$$\vdash \mathsf{Group}\ \mathcal{G} \Rightarrow \\ \forall x.\ x \in \mathsf{G} \Rightarrow \forall m\ n.\ m < \mathsf{order}_{\mathcal{G}}(x) \wedge n < \mathsf{order}_{\mathcal{G}}(x) \wedge x^m = x^n \Rightarrow m = n$$

since order is the minimal exponent to give

$$\vdash x^{\mathsf{order}_{\mathcal{G}}(x)} = \#e \quad \text{where } \#e \text{ is the group identity.}$$

Hence the powers of X are distinct. □

We shall see how the distinct powers of X help to establish a lower bound for $\mathcal{Q}_{\mathsf{h}}$.

5.5 Reduced Polynomials

With estimates of the cardinality of $\mathcal{M}_k$ given in Section 5.4, we turn to the cardinality of $\mathcal{Q}_{\mathsf{h}}$.

Its upper bound equals the size of the finite polynomial field, but this is irrelevant to the AKS proof. Only the lower bound is relevant, and for this we need another way to get something finite from an infinite set, a finite subset of $\mathcal{P}$:

$$\widehat{\mathcal{P}} \stackrel{\text{def}}{=} \{\ \mathsf{p}\ \mid\ \mathsf{p} \in \mathcal{P} \wedge \deg \mathsf{p} < |\mathcal{M}_k|\ \}$$

We shall prove that there is an injective map from $\widehat{\mathcal{P}}$ to $\mathcal{Q}_{\mathsf{h}}$, hence a lower bound on the cardinality of $\widehat{\mathcal{P}}$ will also be a lower bound for the cardinality of $\mathcal{Q}_{\mathsf{h}}$.

First, note an interesting interaction from $\mathcal{M}_k$ to $\mathcal{P}$, which is relevant to $\mathcal{Q}_{\mathsf{h}}$ since $\mathcal{Q}_{\mathsf{h}} \subseteq P$:

Theorem 107. *Each element in* $\mathcal{M}_k$ *gives a root for equivalent difference polynomials from* $\mathcal{P}$.

$$\vdash \mathsf{Ring}\ \mathcal{R} \wedge 0 < k \wedge \mathsf{Ulead}\ \mathcal{R}\ \mathsf{h} \wedge \mathsf{h} \mid \mathsf{X}^k - \mathbf{1} \Rightarrow \\ \forall \mathsf{p}\ \mathsf{q}.\ \\ \mathsf{p} \in \mathcal{P} \wedge \mathsf{q} \in \mathcal{P} \wedge \mathsf{p} \equiv \mathsf{q} \pmod{\mathsf{h}} \Rightarrow \\ \forall n.\ n \in \mathcal{M}_k \Rightarrow (\mathsf{p} - \mathsf{q})[\![\mathsf{X}^n]\!] \equiv \mathbf{0} \pmod{\mathsf{h}}$$

Proof. Given $n \in \mathcal{M}_k$, there is $m \in \mathcal{N}$ such that $n = m \bmod k$. For $\mathsf{p} \in \mathcal{P}$ and $\mathsf{q} \in \mathcal{P}$, $m \overset{k}{\bowtie} \mathsf{p}$ and

$m \overset{k}{\bowtie} \mathsf{q}$. Since $\mathsf{X}^k \equiv \mathbf{1} \pmod{\mathsf{X}^k - \mathbf{1}}$, it follows that:

$$\vdash \mathsf{Ring}\ \mathcal{R} \Rightarrow \forall k.\ 0 < k \Rightarrow \forall m.\ \mathsf{X}^m \equiv \mathsf{X}^{m \bmod k} \pmod{\mathsf{X}^k - \mathbf{1}}$$

Let $z = \mathsf{X}^k - \mathbf{1}$. This result shows $\mathsf{X}^m \equiv \mathsf{X}^n \pmod{z}$. We can proceed:

	$\mathsf{p}^m \equiv \mathsf{p}[\![\mathsf{X}^m]\!] \pmod{z}$	by $m \overset{k}{\bowtie} \mathsf{p}$
and	$\mathsf{p}[\![\mathsf{X}^m]\!] \equiv \mathsf{p}[\![\mathsf{X}^n]\!] \pmod{z}$	by $\mathsf{X}^m \equiv \mathsf{X}^n \pmod{z}$
so	$\mathsf{p}^m \equiv \mathsf{p}[\![\mathsf{X}^n]\!] \pmod{z}$	by transitivity
or	$\mathsf{p}^m \equiv \mathsf{p}[\![\mathsf{X}^n]\!] \pmod{\mathsf{h}}$	by Theorem 105 — [1]
Similarly,	$\mathsf{q}^m \equiv \mathsf{q}[\![\mathsf{X}^n]\!] \pmod{\mathsf{h}}$	from $m \overset{k}{\bowtie} \mathsf{q}$ — [2]
Since	$\mathsf{p}^m \equiv \mathsf{q}^m \pmod{\mathsf{h}}$	by given $\mathsf{p} \equiv \mathsf{q} \pmod{\mathsf{h}}$
so	$\mathsf{p}[\![\mathsf{X}^n]\!] \equiv \mathsf{q}[\![\mathsf{X}^n]\!] \pmod{\mathsf{h}}$	by [1] and [2] above
or	$(\mathsf{p} - \mathsf{q})[\![\mathsf{X}^n]\!] \equiv \mathbf{0} \pmod{\mathsf{h}}$	as claimed.

□

Due to this, an injective map between the two finite sets derived from $\mathcal{P}$ is possible:

Theorem 108. *There is an injective map from $\widehat{\mathcal{P}}$ to modulo set $\mathcal{Q}_\mathsf{h}$.*

$$\vdash \mathsf{Field}\ \mathcal{F} \wedge 0 < k \wedge (\mathsf{monic\ h} \wedge \mathsf{ipoly\ h} \wedge \mathsf{h} \mid \mathsf{X}^k - \mathbf{1}) \wedge \mathsf{order_h(X)} = k \Rightarrow (\lambda\, \mathsf{p}.\ \mathsf{p\ mod\ h}) : \widehat{\mathcal{P}} \hookrightarrow \mathcal{Q}_\mathsf{h}$$

Proof. For the special irreducible factor h, we have $\mathsf{order}_z(\mathsf{X}) = k$. Let $\mathsf{p} \in \widehat{\mathcal{P}}$ and $\mathsf{q} \in \widehat{\mathcal{P}}$, with $\mathsf{p} \equiv \mathsf{q} \pmod{h}$ in $\mathcal{Q}_\mathsf{h}$. To show that the map is injective is to show $\mathsf{p} = \mathsf{q}$.

Since $\widehat{\mathcal{P}} \subseteq \mathcal{P}$, $\mathsf{p} \in \mathcal{P}$ and $\mathsf{q} \in \mathcal{P}$. Then Theorem 107 applies: each $n \in \mathcal{M}_k$ gives a root X^n for polynomial $(\mathsf{p} - \mathsf{q})$. These roots are distinct when $\mathsf{order}_z(\mathsf{X}) = k$ by Theorem 106, *i.e.*, there are at least $|\mathcal{M}_k|$ distinct roots for $(\mathsf{p} - \mathsf{q})$. But $\mathsf{p} \in \widehat{\mathcal{P}}$ and $\mathsf{q} \in \widehat{\mathcal{P}}$, hence $\deg(\mathsf{p} - \mathsf{q}) < |\mathcal{M}_k|$ by subtraction of polynomials. By Theorem 25, this is possible only when the difference is $\mathbf{0}$, *i.e.*, $\mathsf{p} = \mathsf{q}$. □

With this injective map we can estimate a lower bound for the cardinality of $\mathcal{Q}_\mathsf{h}$.

Theorem 109. *The modulo set $\mathcal{Q}_\mathsf{h}$ has a nice lower bound.*

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \wedge (\mathsf{monic\ h} \wedge \mathsf{ipoly\ h} \wedge \mathsf{h} \mid \mathsf{X}^k - \mathbf{1}) \wedge \mathsf{order_h(X)} = k \wedge 1 < k \wedge 0 < s \wedge s < \mathsf{char}(\mathcal{F}) \Rightarrow 2^{\min s\, |\mathcal{M}_k|} \leq |\mathcal{Q}_\mathsf{h}|$$

Proof. The pre-conditions matches Theorem 108, hence there is an injective map from $\widehat{\mathcal{P}}$ to $\mathcal{Q}_\mathsf{h}$. This means $|\widehat{\mathcal{P}}| \leq |\mathcal{Q}_h|$.

By its definition, $|\widehat{\mathcal{P}}|$ is the count of polynomials $\mathsf{p} \in \mathcal{P}$ having $\deg\ \mathsf{p} < |\mathcal{M}_k|$. Since

$1 < |\mathcal{M}_k|$ by Theorem 104, $\widehat{\mathcal{P}}$ has at least these monomials from $\mathcal{P}$ (see Section 5.3):

$$\mathsf{X} + c \text{ with } 0 < c \leq s \tag{5.3}$$

Including $\mathsf{X} \in \widehat{\mathcal{P}}$, there are $s + 1$ monomials. By Theorem 100, any product of these $s + 1$ monomials is in $\widehat{\mathcal{P}}$, as long as the degree of the product is less than $|\mathcal{M}_k|$. Thus a lower bound for $|\widehat{\mathcal{P}}|$ is to count those monomial products with degree $d < |\mathcal{M}_k|$. Applying standard combinatorial techniques, the count is $\binom{|\mathcal{M}_k| + s}{|\mathcal{M}_k| - 1}$. However, to avoid handling and later approximating binomials, a crude expression for the lower bound suffices for our purpose.

Counting only products of monomials in 5.3 with the following form:

$$\mathsf{X}^{e_0}(\mathsf{X} + \mathbf{1})^{e_1} \ldots (\mathsf{X} + c)^{e_c} \ldots (\mathsf{X} + s)^{e_s} \tag{5.4}$$

where the exponents $e_0, e_1, \ldots e_c \ldots e_s$ are 0 or 1, we consider two cases:

- If $s < |\mathcal{M}_k|$, count all products in (5.4), since all constants have $c \leq s$. Each product has a degree less than $|\mathcal{M}_k|$. There are 2^s such products in this case.
- If $|\mathcal{M}_k| \leq s$, count only products in (5.4) with $0 < c \leq |\mathcal{M}_k|$. There is a product taking every monomial with constants in this range, which gives a degree equal to $|\mathcal{M}_k|$, so this must be excluded. However, $\mathbf{1} \in \widehat{\mathcal{P}}$ but $\mathbf{1}$ cannot be formed by multiplying monomials, so this can be included. There are $2^{|\mathcal{M}_k|}$ elements in this case.

Considering both cases, we conclude that $2^{\min s\ |\mathcal{M}_k|} \leq |\mathcal{Q}_h|$. □

5.5.1 Parameters Condition

The AKS parameters provide a crucial inequality involving $|\mathcal{M}_k|$:

Theorem 110. *The AKS parameters a, k and s give an upper bound for an introspective exponent n raised to* $(\mathsf{SQRT}\ |\mathcal{M}_k|)$ *power.*

$$\begin{aligned} \vdash\ & \mathsf{Ring}\ \mathcal{R} \wedge 1 < k \wedge 1 < n \wedge \neg(n\ \mathsf{power\_of}\ 2) \wedge a = \lceil \log n \rceil^2 \wedge \\ & s = (\mathsf{SQRT}\ \varphi(k))\lceil \log n \rceil \wedge a \leq \mathsf{order}_k(n) \wedge n \in \mathcal{N} \Rightarrow \\ & n^{\mathsf{SQRT}\ |\mathcal{M}_k|} < 2^{\min s\ |\mathcal{M}_k|} \end{aligned}$$

Proof. Let $j = \mathsf{order}_k(n)$, and $m = \lceil \log n \rceil$. Note that

$$j \leq |\mathcal{M}_k|,\ |\mathcal{M}_k| < \varphi(k) \text{ and } m^2 \leq j.$$

the first two come from the bounds on cardinality of $\mathcal{M}_k$ (Theorem 104), and the third is given. Taking integer square roots, and reversing inequality directions,

$$\mathsf{SQRT}\ |\mathcal{M}_k| \geq \mathsf{SQRT}\ j,\ \mathsf{SQRT}\ \varphi(k) \geq \mathsf{SQRT}\ |\mathcal{M}_k| \text{ and } \mathsf{SQRT}\ j \geq m.$$

Therefore:

- $s = (\mathsf{SQRT}\ \varphi(k))m \ \geq\ m(\mathsf{SQRT}\ |\mathcal{M}_k|)$
- $|\mathcal{M}_k| \geq (\mathsf{SQRT}\ |\mathcal{M}_k|^2) \ \geq\ (\mathsf{SQRT}\ j)(\mathsf{SQRT}\ |\mathcal{M}_k|) \geq m(\mathsf{SQRT}\ |\mathcal{M}_k|)$

Thus $\min s\ |\mathcal{M}_k| \geq m(\mathsf{SQRT}\ |\mathcal{M}_k|)$. With $2^m > n$ for integer logarithm, we have:

$$2^{\min s\ |\mathcal{M}_k|} \geq 2^{m(\mathsf{SQRT}\ |\mathcal{M}_k|)} = (2^m)^{\mathsf{SQRT}\ |\mathcal{M}_k|} > n^{\mathsf{SQRT}\ |\mathcal{M}_k|}.$$

□

5.6 Reduced Exponents

Having success in finding an injective map from $\widehat{\mathcal{P}}$ to $\mathcal{Q}_\mathsf{h}$, we would like to find an injective map to $\mathcal{M}_k$ from a similar reduced subset of introspective exponents.

It turns out there is such a situation, with the following set of reduced exponents:

$$\widehat{\mathcal{N}}\ p\ q\ m \stackrel{\text{def}}{=} \{\ p^i q^j\ \mid\ i \leq m \wedge j \leq m\ \} \tag{5.5}$$

This reduced subset is generated by the two known elements $p \in \mathcal{N}$ and $q \in \mathcal{N}$ (Section 5.3), with some cut-off m in their exponents. By multiplicative closure of introspective exponents (Theorem 99), $\widehat{\mathcal{N}}\ p\ q\ m \subseteq \mathcal{N}$. The following property for elements is obvious by monotonicity of exponential function:

Theorem 111. *An upper bound for an element in $\widehat{\mathcal{N}}$.*

$$\vdash 1 < p \wedge 1 < q \Rightarrow \forall n\ m.\ n \in \widehat{\mathcal{N}}\ p\ q\ m \Rightarrow n \leq (pq)^m$$

Proof. Each element $n \in \widehat{\mathcal{N}}\ p\ q\ m$ can be expressed in the form $n = p^i q^j$, where $i, j \leq m$. Thus $n \leq p^m q^m \leq (pq)^m$. □

Note another interesting interaction from $\mathcal{Q}_\mathsf{h}$ to $\mathcal{N}$, which is relevant to $\widehat{\mathcal{N}}\ p\ q\ m \subseteq \mathcal{N}$:

Theorem 112. *Each element in $\mathcal{Q}_\mathsf{h}$ gives a root for a special polynomial from $\mathcal{N}$.*

$$\begin{array}{l}\vdash \mathsf{Field}\ \mathcal{F} \wedge 0 < k \wedge \mathsf{monic}\ \mathsf{h} \wedge \mathsf{ipoly}\ \mathsf{h} \wedge \mathsf{h} \mid \mathsf{X}^k - \mathbf{1} \Rightarrow \\ \quad \forall n\ m. \\ \qquad n \in \mathcal{N} \wedge m \in \mathcal{N} \wedge n\ \equiv\ m\ (\mathrm{mod}\ k) \Rightarrow \\ \qquad\quad \forall \mathsf{p}.\ \mathsf{p} \in \mathcal{Q}_\mathsf{h} \Rightarrow (\mathsf{X}^n - \mathsf{X}^m)[\![\mathsf{p}]\!]\ \equiv\ \mathbf{0}\ (\mathrm{mod}\ \mathsf{h})\end{array}$$

Proof. Given $\mathsf{p} \in \mathcal{Q}_\mathsf{h}$, there is $\mathsf{q} \in \mathcal{P}$ such that $\mathsf{p} = \mathsf{q} \bmod \mathsf{h}$. For $n \in \mathcal{N}$ and $m \in \mathcal{N}$, $n \overset{k}{\bowtie} \mathsf{q}$ and $m \overset{k}{\bowtie} \mathsf{q}$. Since $\mathsf{X}^k\ \equiv\ \mathbf{1}\ (\mathrm{mod}\ \mathsf{X}^k - \mathbf{1})$, it follows that:

$$\vdash \mathsf{Ring}\ \mathcal{R} \Rightarrow \forall k.\ 0 < k \Rightarrow \forall n\ m.\ n\ \equiv\ m\ (\mathrm{mod}\ k) \Rightarrow \mathsf{X}^n\ \equiv\ \mathsf{X}^m\ (\mathrm{mod}\ \mathsf{X}^k - \mathbf{1})$$

Let $z = \mathsf{X}^k - 1$. This result shows $\mathsf{X}^n \equiv \mathsf{X}^m \pmod z$. We can proceed:

	$\mathsf{q}^n \equiv \mathsf{q}[\![\mathsf{X}^n]\!] \pmod z$	by $n \overset{k}{\bowtie} \mathsf{q}$
	$\mathsf{q}^m \equiv \mathsf{q}[\![\mathsf{X}^m]\!] \pmod z$	by $m \overset{k}{\bowtie} \mathsf{q}$
and	$\mathsf{q}[\![\mathsf{X}^m]\!] \equiv \mathsf{q}[\![\mathsf{X}^n]\!] \pmod z$	by $\mathsf{X}^n \equiv \mathsf{X}^m \pmod z$
so	$\mathsf{q}^n \equiv \mathsf{q}^m \pmod z$	by equivalence
or	$\mathsf{q}^n - \mathsf{q}^m \equiv 0 \pmod z$	by subtraction
Since	$(\mathsf{X}^n - \mathsf{X}^m)[\![\mathsf{p}]\!] \equiv (\mathsf{X}^n - \mathsf{X}^m)[\![\mathsf{q}]\!] \pmod h$	by $\mathsf{p} = \mathsf{q} \bmod \mathsf{h}$
and	$(\mathsf{X}^n - \mathsf{X}^m)[\![\mathsf{q}]\!] = \mathsf{q}^n - \mathsf{q}^m$	by substitution
so	$(\mathsf{X}^n - \mathsf{X}^m)[\![p]\!] \equiv 0 \pmod z$	by combining above
*Modu*or	$(\mathsf{X}^n - \mathsf{X}^m)[\![p]\!] \equiv 0 \pmod h$	by Theorem 105

□

Due to this, an injective map between the two finite sets derived from $\mathcal{N}$ is possible:

Theorem 113. *There is an injective map from $\widehat{\mathcal{N}}\ p\ q\ (\mathsf{SQRT}\ |\mathcal{M}_k|)$ to modulo set $\mathcal{M}_k$, where p, q are two introspective exponents.*

$$\begin{array}{l} \vdash \mathsf{Field}\ \mathcal{F} \wedge \mathsf{monic\ h} \wedge \mathsf{ipoly\ h} \wedge \mathsf{h} \mid \mathsf{X}^k - 1 \Rightarrow \\ \quad \forall p\ q. \\ \quad\quad 1 < p \wedge 1 < q \wedge p \in \mathcal{N} \wedge q \in \mathcal{N} \wedge (pq)^{\mathsf{SQRT}\ |\mathcal{M}_k|} < |\mathcal{Q}_{\mathsf{h}}| \Rightarrow \\ \quad\quad (\lambda m.\ m \bmod k)\ :\ \widehat{\mathcal{N}}\ p\ q\ (\mathsf{SQRT}\ |\mathcal{M}_k|) \hookrightarrow \mathcal{M}_k \end{array}$$

Proof. Let $i \in \widehat{\mathcal{N}}\ p\ q\ (\mathsf{SQRT}\ |\mathcal{M}_k|)$, and $j \in \widehat{\mathcal{N}}\ p\ q\ (\mathsf{SQRT}\ |\mathcal{M}_k|)$ with $i \equiv j \pmod k$. To show the map is injective is to show $i = j$.

Since $\widehat{\mathcal{N}}\ p\ q\ (\mathsf{SQRT}\ |\mathcal{M}_k|) \subseteq N$, both $i \in \mathcal{N}$ and $j \in \mathcal{N}$. Theorem 112 applies: every $\mathsf{p} \in \mathcal{Q}_h$ is a root of $\mathsf{X}^i - \mathsf{X}^j$. Hence there are at least $|\mathcal{Q}_h|$ roots. But i and j in $\widehat{\mathcal{N}}\ n\ p\ (\mathsf{SQRT}\ |\mathcal{M}_k|)$ are bounded by $n^{\mathsf{SQRT}\ |\mathcal{M}_k|}$ by Theorem 111. Therefore, with the pre-condition inequality, there are more roots than degree for the polynomial $\mathsf{X}^i - \mathsf{X}^j$ with coefficient from finite field $\mathcal{F}$. This is not possible by Theorem 25, unless it is 0, which means $i = j$. □

5.7 Punch Line

The cardinality of $\widehat{\mathcal{N}}\ p\ q\ m$ is simple to express when its generators have a special property:

Theorem 114. *Cardinality of $\widehat{\mathcal{N}}\ p\ q\ m$ when generators p, q are not related by perfect power.*

$$\begin{array}{l} \vdash \mathsf{Field}\ \mathcal{F} \wedge 0 < k \wedge p \in \mathcal{N} \wedge q \in \mathcal{N} \wedge \mathsf{prime}\ p \wedge 1 < q \wedge \neg(q\ \mathsf{power\_of}\ p) \Rightarrow \\ \quad \forall m.\ |\widehat{\mathcal{N}}\ p\ q\ m| = (1+m)^2 \end{array}$$

Proof. Let $f(i,j) = p^i \times q^j$, and $t = \{0, 1, \ldots, m\}$. By definition (5.5), it is easy to see that

$\widehat{\mathcal{N}}\ p\ q\ m$ is the image of $t \times t$ under f:

$$f(\!|t \times t|\!) = \{\ p^i \times q^j\ \mid\ i \le m \wedge j \le m\ \} = \widehat{\mathcal{N}}\ p\ q\ m$$

More interesting is that the given conditions imply an injective map:

$$f\ :\ t \times t\ \hookrightarrow\ \widehat{\mathcal{N}}\ p\ q\ n$$

To show this map is injective, pick two ordered pairs y and z in $t \times t$. Let $y = (i,j)$, and $z = (u,v)$. Assume that $p^i \times q^j = p^u \times q^v$, we need to show $y = z$, *i.e.* $i = u$ and $j = v$. This comes down to case analysis:

- If $i = u$, the equality reduces to $q^j = q^v$, hence $j = v$.
- If $i < u$, the equality reduces to $q^j = p^d \times q^v$, where $d = u - i$. Now $j \neq v$, for otherwise this leaves with $1 = p^d$, which is impossible for prime p. Thus there are two sub-cases:
  - If $j < v$, this becomes $1 = p^d \times q^e$, where $e = v - j$, which is clearly impossible.
  - If $j > v$, this will give $q^e = p^d$, where $e = j - v$. Since $e \neq 0$, Theorem 55 (page 37) applies:
    $$\vdash \mathsf{prime}\ p \wedge (\exists x\ y.\ 0 < x \wedge p^x = q^y) \Rightarrow q\ \mathsf{power\_of}\ p$$
    Hence q is a power of p, contradicting the given $\neg(q\ \mathsf{power\_of}\ p)$. This case is also impossible.
- If $i > u$, this is similar to the case $i < u$ by reversing i and u.

The only possible case makes this map injective. □

This property is crucial to establish this key result:

Theorem 115. ***(AKS Main Theorem).*** *If a number n satisfies the AKS criteria in a finite field $\mathcal{F}$ of prime cardinality, then n is a power of the characteristic of the finite field.*

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \wedge |\mathsf{F}| = \mathsf{char}(\mathcal{F}) \Rightarrow \forall n\ k.\ \mathsf{aks\_criteria}\ \mathcal{F}\ n\ k \Rightarrow n\ \mathsf{power\_of}\ \mathsf{char}(\mathcal{F})$$

Proof. Let $p = \mathsf{char}(\mathcal{F})$, $a = \lceil \log\ n \rceil^2$, and $s = (\mathsf{SQRT}\ \varphi(k))\lceil \log\ n \rceil$. By the AKS criteria in a finite field $\mathcal{F}$ (Definition 72), we have:

$$\begin{array}{l} 0 < n \wedge 0 < k \wedge 1 < \mathsf{order}_k(p) \wedge p \mid n \wedge k < p \wedge \\ a \le \mathsf{order}_k(n) \wedge \mathsf{poly\_intro\_range}\ \mathcal{F}\ k\ n\ s \end{array} \tag{5.6}$$

The goal is to show that n is a power of p.

A finite field $\mathcal{F}$ has its characteristic p prime. Note $k \neq 1$ since $1 < \mathsf{order}_k(p)$, and $n \neq 1$ due to $p \mid n$. Therefore $1 < n$ and $1 < k$. Since $p \mid n$, $p \le n$. If $p = n$, then certainly n is a

power of p. Otherwise, $p < n$. If n is a power 2, its only prime factor is 2, so $p = 2$ and n is a power of p trivially.

Now that we have $p < n$ and n is not a power of 2, it is time to introduce the sets $\mathcal{N}$ and $\mathcal{P}$, for introspective exponents and polynomials, respectively (Definition 101). The conditions in 5.6 give poly_intro_range $\mathcal{F}\ k\ n\ s$, and $\gcd(n,k) = 1$ by $1 < \mathsf{order}_k(n)$. Thus $n \in \mathcal{N}$ by Definition 101. Using $k > 1$ and a special irreducible factor h of $\mathsf{X}^k - \mathbf{1}$ such that $\mathsf{order}_{\mathsf{h}}(\mathsf{X}) = k$ (Theorem 96), we have the sets $\mathcal{M}_k$ and $\mathcal{Q}_{\mathsf{h}}$ (Section 5.4) with k and h as modulus. Let $t = |\mathcal{M}_k|$. Since $n \in \mathcal{N}$, $a \leq \mathsf{order}_k(n)$, and n is not a power of 2, we have the following (Theorem 110):

$$n^{\mathsf{SQRT}\ t} < 2^{\mathsf{min}\ s\ t}$$

Note $s \leq k$ by $\gcd(k,n) = 1$ and $a \leq \mathsf{order}_k(n)$ (Theorem 68), and $k < p$ from 5.6. Moreover, $0 < s$ since $1 < n$, thus $0 < s < p$. This provides a lower bound for $|\mathcal{Q}_{\mathsf{h}}|$ from properties of the special irreducible factor h of $\mathsf{X}^k - \mathbf{1}$ with $\mathsf{order}_{\mathsf{h}}(\mathsf{X}) = k$ (Theorem 109):

$$2^{\mathsf{min}\ s\ t} \leq |\mathcal{Q}_{\mathsf{h}}|$$

From these two inequalities,

$$n^{\mathsf{SQRT}\ t} < |\mathcal{Q}_{\mathsf{h}}| \tag{5.7}$$

Let $q = n\ \mathsf{div}\ p$, then $n = pq$, and $1 < q$ since $n \neq p$. Since $\gcd(n,k) = 1$ and $p \mid n$, we have $\gcd(p,k) = 1$ (Theorem 56, page 38), so $1 < \mathsf{order}_k(p)$. Applying Corollary 103, we have $p \in \mathcal{N}$ and $q \in \mathcal{N}$. Use p and q to form the set $\widehat{\mathcal{N}}\ p\ q\ (\mathsf{SQRT}\ t)$. Note that $pq = n$, so $(pq)^{\mathsf{SQRT}\ t} < |\mathcal{Q}_{\mathsf{h}}|$ by 5.7. This ensures an injective map to $\mathcal{M}_k$ by Theorem 113:

$$(\lambda\, m.\ m\ \mathsf{mod}\ k)\ :\ \widehat{\mathcal{N}}\ p\ q\ (\mathsf{SQRT}\ t)\ \hookrightarrow\ \mathcal{M}_k \tag{5.8}$$

With this injective map in place, we can show that n must be a power of p.

The proof proceeds by contradiction. Suppose n is not a power of p, then q is not a power of p (Theorem 54, page 37). With prime p and $1 < q$, the cardinality of $\widehat{\mathcal{N}}\ p\ q\ (\mathsf{SQRT}\ t)$ is given by (Theorem 114):

$$|\widehat{\mathcal{N}}\ p\ q\ (\mathsf{SQRT}\ t)| = (1 + \mathsf{SQRT}\ t)^2$$

Note that $(1 + \mathsf{SQRT}\ t)^2 > t$ for integer square-roots, and $t = |\mathcal{M}_k|$. Therefore:

$$|\widehat{\mathcal{N}}\ p\ q\ (\mathsf{SQRT}\ t)| > |\mathcal{M}_k|$$

revealing that the injective map (5.8) would violate the Pigeonhole Principle. The only way to resolve this is that n is a power of its prime factor p. □

With AKS Main Theorem (Theorem 115) in place, the proof of Theorem 73 is complete:

$$\vdash \mathsf{aks}\ n \Rightarrow \mathsf{prime}\ n$$

Together with Theorem 70:

$$\vdash \mathsf{prime}\ n \Rightarrow \mathsf{aks}\ n$$

we conclude that the AKS algorithm is correct (Theorem 2):

$$\vdash \mathsf{prime}\ n \iff \mathsf{aks}\ n$$

Scripts Scripts for the correctness proof of the AKS Main Theorem have many versions. They are located in `aks/theories`, with `AKStheorem` for the original Chan and Norrish [2015], `AKSrevised` incorporting the first improvement (see Section 5.1), `AKSimproved` having both improvements, and `AKSclean` a cleanup version for presentation.

5.8 Summary

We have shown that the three phases of the AKS algorithm correctly give a primality test for any input number n. This is true especially for those n that enter into the third phase of introspective checks, whereby the theory shifts its focus to a finite field $\mathcal{F}$, whose characteristic p is a special prime that divides n. We study the form of the introspective relation, then define the introspective sets, with exponents on one hand, and polynomials on the other (Section 5.3). With the AKS parameter k, and the special irreducible factor h of $\mathsf{X}^k - \mathbf{1}$ having $\mathsf{order}_{\mathsf{h}}(\mathsf{X}) = k$, we can define their modular equivalent (Section 5.4). As a result of the interplay between these sets, injective maps between their finite counterparts can be constructed through the conditions imposed by the AKS algorithm (Theorems 108 and 113). In the end, if n can pass a sufficient number of introspective checks, the Pigeonhole principle for injective map between finite sets enforces that n must be a power of p (AKS Main Theorem 115). The power must be trivial because n is power free after the check in phase 1, thus $n = p$, and n is prime. This concludes the second part of the theory for the AKS algorithm. We venture to the next part, the computational complexity of the AKS algorithm.

5.9 Remarks

All correctness proofs of the AKS algorithm, including ours, are elaborations of the proof given in the AKS papers Agrawal et al. [2002, 2004]. The two improvements of Section 5.1 in the revised paper are absent in the original paper, but they do not affect the main conclusion: that the AKS algorithm is a deterministic polynomial-time primality test. Hence, depending on their intended audience, not all of expositions of the AKS algorithm incorporate the improvements. For example, both are absent in Dietzfelbinger [2004]; Rempe-Gillen and Waldecker [2014], although the later had a section on cyclotomic polynomials and indicated in an exercise that the parameter k need not be prime. Those including both improvements are Crandall and Pomerance [2005]; Daleson [2006]. In Linowitz [2006] the fact that the cofactor $q = n \ \mathsf{div}\ p$ is introspective was shown by applying the Chinese Remainder Theorem.

Following the proof in the AKS paper, these expositions first derive a lower bound for $\mathcal{Q}_h$ (Theorem 109), then derive an upper bound for $\mathcal{Q}_h$ by assuming n is not a power of its prime factor p, using the Pigeonhole principle. The incompatibility of the two bounds on $\mathcal{Q}_h$ shows that n must be a power of p.

In this thesis we take a slightly different approach, as described in Figure 5.1 and given in Section 5.7. From the lower lower bound for $\mathcal{Q}_h$, we derive a further inequality from the conditions on AKS parameters (Section 5.5.1, Theorem 110). The two inequalities combine to give (5.7). When n is not a power of its prime factor p, we construct an injective map between finite sets (Theorem 114), and show that (5.7) implies a violation of the Pigeonhole principle.

Our proof scripts include both approaches to prove the AKS Main Theorem (Theorem 115). There are versions with and without one or two of the improvements.

When estimating a lower bound for $\mathcal{Q}_h$, the AKS paper used standard combinatorics to obtain the bound as a single binomial coefficient, which was later simplified to (5.7). We show directly (5.7) for simplicity.

Our formulation of the AKS Main Theorem in Finite Field is distilled from the AKS paper. In the AKS paper, although the proof is based on rings and finite fields, the statement of the main theorem is in number-theoretic terms. Some expositions (Dietzfelbinger [2004]; Crandall and Pomerance [2005]; Daleson [2006]; Shoup [2008]) state the main theorem in either $\mathbb{Z}_n[X]$ or $\mathbb{Z}_p[X]$, while Bernstein [2002] uses the quotient ring $\mathbb{Z}_n[X]/(X^k - \mathbf{1})$. We seem to be the first to state the main theorem (Theorem 115) in terms the properties of an abstract finite field.

Part III

Complexity

Part 3 is about the computational complexity of the AKS algorithm. We devise a model of computation in monadic style, develop a general recurrence theory of loop count, and apply the theory to analyse the AKS algorithm. The key idea is to break down into subroutines, and examine each subroutine to formally conclude that the AKS algorithm belongs to the polynomial-time class.

Chapter 6

Complexity Models

This chapter develops the models for the study of computational complexity. We use a logging monad to keep track on computations built up from elementary operations, including a running count of the number of steps. We define our complexity measure, introduce the $\mathcal{O}$-notation, and describe the elementary operations in our machine model, and how they are utilised to build common subroutines. We illustrate our technique of complexity analysis by an example, from which we can see the necessity of a general theory of recurrence in order to provide estimates to bound the number of steps for various patterns of program loops. These will be our tools to analyse the computational complexity of the AKS algorithm.

> A Writer monad value is a pair: (computation, log).
> Binding replaces the computation value
> with the result of applying the bound function
> to the previous value and appends any log data
> from the computation to the existing log data.
> — All About Monads, HaskellWiki[1]

6.1 Monadic Computation

Monads are generally captured by two functions:

$$\begin{array}{l}(\mathsf{unit} : \alpha \ \rightarrow \ \alpha\ \mathtt{M}\ ) \\ (\mathsf{bind} : \alpha\ \mathtt{M} \ \rightarrow \ (\alpha \ \rightarrow \ \beta\ \mathtt{M}) \ \rightarrow \ \beta\ \mathtt{M}\ )\end{array}$$

where $(\alpha\ \mathtt{M})$ is the type of a monadic computation yielding a value of type α. The pretty **do** ... **od** notation is a syntactic sugar for combinations of bind calls. In particular,

$$\textbf{do}\ v\ \leftarrow\ m_1;\ m_2\ \textbf{od} \quad \text{means} \quad \mathsf{bind}\ m_1(\lambda\, v.\ m_2)$$

where v is presumably a free variable, present in m_2. When this is written as **do** $m_1;\ m_2$ **od** without v, it indicates that v is not present in m_2 and the result of m_1 is being ignored. This

[1] From https://wiki.haskell.org/All_About_Monads

syntax can be extended to

$$\textbf{do } v_1 \leftarrow m_1;\ v_2 \leftarrow m_2;\ m_3 \textbf{ od} \quad \text{means} \quad \mathsf{bind}\ m_1 (\lambda\, v_1.\ \mathsf{bind}\ m_2\ (\lambda\, v_2.\ m_3))$$

The monad we used is based on the "writer" model, similar to the Writer monad in Haskell. The logging is used to keep track of the number of steps, called *ticks*, as a computation progresses. The ticks are given a type `counter` = Count num, which is identical to the natural numbers. We are cautious to separate ticks from any numerical computations.

With these ideas and notations, we introduce our basic ingredients of complexity analysis:

Definition 116. *Components for monadic computation.*

$$\begin{aligned}
&\mathsf{unit}\ x \stackrel{\text{def}}{=} (x,\mathsf{Count}\ 0)\\
&\mathsf{tick}\ c \stackrel{\text{def}}{=} ((),\mathsf{Count}\ c)\\
&\mathsf{valueOf}\ (v,\mathsf{Count}\ c) \stackrel{\text{def}}{=} v\\
&\mathsf{stepsOf}\ (v,\mathsf{Count}\ c) \stackrel{\text{def}}{=} c\\
&\mathsf{bind}\ (v_1,\mathsf{Count}\ c_1)\ f \stackrel{\text{def}}{=}\\
&\quad \textbf{case } f\ v_1 \textbf{ of } (v_2,\mathsf{Count}\ c_2) \triangleright (v_2,\mathsf{Count}\ (c_1 + c_2))
\end{aligned}$$

As can be seen from this definition, tick is a primitive to advance the clock without returning anything. Writing unit x as **return** x when x is the final result of a computation, we have, *e.g.*, **do** tick 10; **return** 3 **od** giving this pair $(3,\mathsf{Count}\ 10)$. The components of the pair can be extracted by the functions valueOf and stepsOf.

6.2 Complexity Analysis

To express the AKS computations with input n in monadic style, we shall design aksM n (see Section 7.1). The result, $\mathsf{valueOf}\ (\mathsf{aksM}\ n)$ is a boolean asserting whether n is prime. The number of steps of computation, $\mathsf{stepsOf}\ (\mathsf{aksM}\ n)$, is a function of the input n. Our aim is to measure the growth of $\mathsf{stepsOf}\ (\mathsf{aksM}\ n)$ in terms of the input size.

6.2.1 Complexity Measure

A convenient measure of the size of a number n is the length of its binary representation:

Definition 117. *Except for* $n = 0$ *or* 1*, halving a number* n *reduces its binary length by* 1.

$$\mathsf{size}\ n \stackrel{\text{def}}{=} \textbf{if } n \leq 1 \textbf{ then } 1 \textbf{ else } 1 + \mathsf{size}\ (n\ \mathsf{div}\ 2)$$

While size n is precise, it is not a familiar size measure. It is customary to use instead log n, the base 2 logarithm of n. We shall use its integer round-up version $\lceil \log n \rceil$:

Definition 118. *Of all exponentials of* 2, $2^{\lceil \log n \rceil}$ *is unique between* n *and* $2n$ *for* $n > 0$.

$$\lceil \log 0 \rceil = 0$$
$$0 < n \Rightarrow \forall m.\ \lceil \log n \rceil = m \iff n \leq 2^m \wedge 2^m < 2n$$

We define $\lceil \log 0 \rceil = 0$, so that $\lceil \log n \rceil$ is a total function. These two measures are almost the same, as shown in Table 6.1.

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
size n	1	1	2	2	3	3	3	3	4	4	4	4	4	4	4	4	5	5	5
$\lceil \log n \rceil$	0	0	1	2	2	3	3	3	3	4	4	4	4	4	4	4	4	5	5

Table 6.1: Comparison of measures size n and $\lceil \log n \rceil$.

When n is a power of 2, $n = 2^e$ for some exponent e. In this case, $\lceil \log n \rceil = e$, but the binary representation of n has e zeroes with a leading 1, *i.e.*, size $n = e + 1$. This explains the adjustment of size n to $\lceil \log n \rceil$, and shows that they differ by at most 1:

$$\vdash \lceil \log n \rceil \leq \text{size } n \wedge \text{size } n \leq 1 + \lceil \log n \rceil$$

We can pin this down precisely:

Theorem 119. *The measure* size n *differs from* $\lceil \log n \rceil$ *by* 1 *at* $n = 0$ *and* n *a power of* 2.

$$\vdash \text{size } n = \lceil \log n \rceil + \textbf{if } n = 0 \vee n \text{ power\_of } 2 \textbf{ then } 1 \textbf{ else } 0$$

We prefer the precise complexity measure by size n, and we shall derive the complexity classes of algorithms based on this measure. The conversion to the customary complexity classes based on $\lceil \log n \rceil$ is trivial in view of the above result.

6.2.2 Complexity Notation

The complexity classes of an algorithm is expressed using the big-$\mathcal{O}$ notation:

Definition 120. *The class* $\mathcal{O}(f)$ *is the set of functions asymptotically bounded by* f*, up to a multiplicative constant.*

$$\mathcal{O}(f) \stackrel{\text{def}}{=} \{\ g \mid \exists k\, c.\ \forall n.\ k < n \Rightarrow g\ n \leq c(f\ n)\ \}$$

Thus if we have proved that $\forall n.\ g\ n \leq 3\lceil \log n \rceil^2$, we should conclude $g \in \mathcal{O}(\lambda n.\ \lceil \log n \rceil^2)$. This is conventionally written as $g \in \mathcal{O}(\lceil \log n \rceil^2)$, with the λ-quantification being understood.

In the AKS paper some complexity results are expressed as, *e.g.*, $\tilde{\mathcal{O}}(\lceil \log n \rceil^2)$. This notation ignores logarithmic powers of the variable, *e.g.*$\mathcal{O}(f(n)\lceil \log n \rceil^3)$ is shortened to $\tilde{\mathcal{O}}(f(n))$. We do not formalise this notation, and use it only to quote results from literature.

With size n and $\lceil \log n \rceil$ off by 1 at most (Theorem 119), we have this corollary:

Corollary 121. *The class of polynomial* size n *is equal to the class of polynomial* $\lceil \log n \rceil$.

$$\vdash \mathcal{O}((\text{size } n)^k) = \mathcal{O}(\lceil \log n \rceil^k)$$

For faithfulness in size measure, we shall first derive the bounds on computation steps in terms of size n. We use the above corollary to convert the complexity class to be based on $\lceil \log n \rceil$, for ease of comparison with results in literature.

6.3 Machine Model

To analyse the number of steps in a computation in a formal setting, we need to define precisely what constitutes a step. The ultimate model of computation is the Turing machine, with each step corresponding to a basic action of the machine. However, using Turing machines for the AKS algorithm is too tedious. We therefore turn to a more convenient, but still low-level model of computation.

In our model, we have a clock ticking off the progress of the computation. The number of steps is expressed in ticks. We treat the basic arithmetical operations as elementary, and just define the steps based on the bit-size of the numeric operands. Thus the steps are expressed in terms of the size function.

These elementary operations are expressed in monadic style. The monad keeps track of the ticks required for the computation, and returns the result of the computation as value.

Definition 122. *Ticks and values of elementary arithmetic operations.*

$$\begin{array}{l}
\text{addM } x\ y \stackrel{\text{def}}{=} \textbf{do}\ \text{tick}\ (\max\ (\text{size } x)\ (\text{size } y));\ \textbf{return}\ (x + y)\ \textbf{od} \\
\text{subM } x\ y \stackrel{\text{def}}{=} \textbf{do}\ \text{tick}\ (\max\ (\text{size } x)\ (\text{size } y));\ \textbf{return}\ (x - y)\ \textbf{od} \\
\text{mulM } x\ y \stackrel{\text{def}}{=} \textbf{do}\ \text{tick}\ ((\text{size } x)(\text{size } y));\ \textbf{return}\ xy\ \textbf{od} \\
\text{divM } x\ y \stackrel{\text{def}}{=} \\
\quad \textbf{if}\ y = 0\ \textbf{then return}\ (x\ \text{div}\ 0) \\
\quad \textbf{else do}\ \text{tick}\ ((\text{size } x)(\text{size } y));\ \textbf{return}\ (x\ \text{div}\ y)\ \textbf{od} \\
\text{modM } x\ y \stackrel{\text{def}}{=} \\
\quad \textbf{if}\ y = 0\ \textbf{then return}\ (x\ \text{mod}\ 0) \\
\quad \textbf{else do}\ \text{tick}\ ((\text{size } x)(\text{size } y));\ \textbf{return}\ (x\ \text{mod}\ y)\ \textbf{od}
\end{array}$$

For division by 0, the monad just returns expressions like x div 0 and x mod 0, which are valid expressions of type `num` in HOL4.[2] Note that the ticks for these elementary arithmetic operations are quite conservative. For example, long multiplication and division are used for the estimate of their ticks. In addition or subtractiion, we take care to use the maximum of the two input sizes.

We also define basic list operations, and basic boolean operations:

[2] For divM $x\ y$ and modM $x\ y$, the test $y = 0$ is needed for a total function in HOL4.

Definition 123. *Ticks and values of elementary list operations.*

$$\begin{aligned}
&\mathsf{nullM}\ ls \stackrel{\text{def}}{=} \textbf{do}\ \mathsf{tick}\ 1;\ \textbf{return}\ (ls = [\,])\ \textbf{od}\\
&\mathsf{headM}\ ls \stackrel{\text{def}}{=} \textbf{do}\ \mathsf{tick}\ 1;\ \textbf{return}\ (\mathsf{HD}\ ls)\ \textbf{od}\\
&\mathsf{tailM}\ ls \stackrel{\text{def}}{=} \textbf{do}\ \mathsf{tick}\ 1;\ \textbf{return}\ (\mathsf{TL}\ ls)\ \textbf{od}\\
&\mathsf{consM}\ x\ ls \stackrel{\text{def}}{=} \textbf{do}\ \mathsf{tick}\ 1;\ \textbf{return}\ (x{::}ls)\ \textbf{od}
\end{aligned}$$

Definition 124. *Ticks and values of elementary boolean operations.*

$$\begin{aligned}
&\mathsf{eqM}\ x\ y \stackrel{\text{def}}{=} \textbf{do}\ \mathsf{tick}\ (\max\ (\mathsf{size}\ x)\ (\mathsf{size}\ y));\ \textbf{return}\ (x = y)\ \textbf{od}\\
&\mathsf{notM}\ b \stackrel{\text{def}}{=} \textbf{do}\ \mathsf{tick}\ 1;\ \textbf{return}\ (\neg b)\ \textbf{od}\\
&\mathsf{boolM}\ b \stackrel{\text{def}}{=} \textbf{do}\ \mathsf{tick}\ 1;\ \textbf{return}\ (\textbf{if}\ b\ \textbf{then}\ 1\ \textbf{else}\ 0)\ \textbf{od}
\end{aligned}$$

Except for the equality check, these simple operations are 1 tick operations.

6.4 Subroutines

We develop several useful subroutines, in order to express algorithms better:

Definition 125. *Subroutines in monadic style.*

$$\begin{aligned}
&\mathsf{zeroM}\ n \stackrel{\text{def}}{=} \mathsf{eqM}\ n\ 0\\
&\mathsf{oneM}\ n \stackrel{\text{def}}{=} \mathsf{eqM}\ 1\ n\\
&\mathsf{twiceM}\ n \stackrel{\text{def}}{=} \mathsf{mulM}\ n\ 2\\
&\mathsf{halfM}\ n \stackrel{\text{def}}{=} \mathsf{divM}\ n\ 2\\
&\mathsf{parityM}\ n \stackrel{\text{def}}{=} \mathsf{modM}\ n\ 2\\
&\mathsf{evenM}\ n \stackrel{\text{def}}{=} \textbf{do}\ z \leftarrow \mathsf{parityM}\ n;\ \mathsf{zeroM}\ z\ \textbf{od}\\
&\mathsf{sqM}\ n \stackrel{\text{def}}{=} \mathsf{mulM}\ n\ n\\
&\mathsf{incM}\ n \stackrel{\text{def}}{=} \mathsf{addM}\ n\ 1\\
&\mathsf{decM}\ n \stackrel{\text{def}}{=} \mathsf{subM}\ n\ 1\\
&\mathsf{leqM}\ n\ m \stackrel{\text{def}}{=} \textbf{do}\ z \leftarrow \mathsf{subM}\ n\ m;\ \mathsf{zeroM}\ z\ \textbf{od}
\end{aligned}$$

Their values and number of steps are listed in Table 6.2. Other algorithms are built upon these subroutines for formal correct proof and complexity analysis.

6.5 Integer Logarithm

We shall illustrate our approach to complexity analysis and design of an algorithm by working through an example: the computation of $\lceil \log n \rceil$. Given an input n, the value of $\lceil \log n \rceil$ is used in all the phases of the AKS algorithm.

Value	Number of Steps
$\vdash$ valueOf (zeroM n) $\iff$ n = 0	$\vdash$ stepsOf (zeroM n) = size n
$\vdash$ valueOf (oneM n) $\iff$ n = 1	$\vdash$ stepsOf (oneM n) = size n
$\vdash$ valueOf (twiceM n) = $2n$	$\vdash$ stepsOf (twiceM n) = 2(size n)
$\vdash$ valueOf (halfM n) = n div 2	$\vdash$ stepsOf (halfM n) = 2(size n)
$\vdash$ valueOf (parityM n) = n mod 2	$\vdash$ stepsOf (parityM n) = 2(size n)
$\vdash$ valueOf (evenM n) $\iff$ EVEN n	$\vdash$ stepsOf (evenM n) = 2(size n) + 1
$\vdash$ valueOf (sqM n) = (n^2)	$\vdash$ stepsOf (sqM n) = (size n)^2
$\vdash$ valueOf (incM n) = $n + 1$	$\vdash$ stepsOf (incM n) = size n
$\vdash$ valueOf (decM n) = $n - 1$	$\vdash$ stepsOf (decM n) = size n
$\vdash$ valueOf (leqM n m) $\iff$ $n \leq m$	$\vdash$ stepsOf (leqM n m) = size (max n m) + size ($n - m$)

Table 6.2: Subroutines values and number of steps.

6.5.1 Logarithm Computation

By reformulating Theorem 119, we can compute $\lceil \log n \rceil$:

Theorem 126. *Computing* $\lceil \log n \rceil$ *from* size n *with adjustment.*

$$\vdash \lceil \log n \rceil = \textbf{if } n = 0 \textbf{ then } 0 \textbf{ else } \text{size } n - \textbf{if } n \text{ power\_of } 2 \textbf{ then } 1 \textbf{ else } 0$$

This requires two subroutines: one to compute size n, and one to check if n is a power of 2. We shall first consider the power check computation.

6.5.2 Power Check

Consider the definition of n power_of b (Definition 53, page 37):

$$n \text{ power\_of } b \stackrel{\text{def}}{=} \exists e.\ n = b^e$$

This definition is not computationally effective, but we can check if n is a power of b using repeated division by base b. This can be expressed in pseudo-code, which is recursive:

$\vdash$ n power_of b $\iff$
  if $n = 0$ **then** $b = 0$
  else if $n = 1$ **then** T
  else if $b = 0$ **then** $n \leq 1$
  else if $b = 1$ **then** $n = 1$
  else if $n \equiv 0 \pmod{b}$ **then** n div b power_of b
  else F

(6.1)

Implementing this in monadic style is straight-forward, using primitives and subroutines:

```
power_ofM b n ≝
 do
  n0 ← zeroM n;
  n1 ← oneM n;
  b0 ← zeroM b;
  b1 ← oneM b;
  if n0 then return b0
  else if n1 then return T
  else if b0 then return (n0 ∨ n1)
  else if b1 then return n1
  else
   do
    m ← modM n b;
    gd ← zeroM m;
    if gd then do q ← divM n b; power_ofM b q od
    else return F
   od
 od
```

Given the resemblance of the implementation and the pseudo-code, we can establish:

$$\vdash \mathsf{valueOf}\ (\mathsf{power\_ofM}\ b\ n) \iff n\ \mathsf{power\_of}\ b$$

$$\begin{aligned}
\vdash\ & \mathsf{stepsOf}\ (\mathsf{power_ofM}\ b\ n) = \\
& 2(\mathsf{size}\ n) + 2(\mathsf{size}\ b) + \\
& \textbf{if}\ n = 0 \vee n = 1 \vee b = 0 \vee b = 1\ \textbf{then}\ 0 \\
& \textbf{else} \\
& \quad (\mathsf{size}\ n)(\mathsf{size}\ b) + \mathsf{size}\ (n\ \mathsf{mod}\ b) + \\
& \quad \textbf{if}\ n \not\equiv 0\ (\mathrm{mod}\ b)\ \textbf{then}\ 0 \\
& \quad \textbf{else} \\
& \quad\quad (\mathsf{size}\ n)(\mathsf{size}\ b) + \mathsf{stepsOf}\ (\mathsf{power_ofM}\ b\ (n\ \mathsf{div}\ b))
\end{aligned}$$

We would like to obtain an upper bound for $\mathsf{stepsOf}\ (\mathsf{power\_ofM}\ b\ n)$, which at present is only expressed recursively.

6.5.3 Size Computation

We have seen that the number of steps of the primitives, as well as subroutines in Table 6.2, are expressed in terms of $\mathsf{size}\ n$, with respect to an input n. It is ironic that, although $\mathsf{size}\ n$ is used

to express the complexity results, we do need to analyse its own computational complexity.

As a function, size n can be computed recursively (Definition 117):

$$\text{size } n \stackrel{\text{def}}{=} \textbf{if } n \leq 1 \textbf{ then } 1 \textbf{ else } 1 + \text{size } (n \text{ div } 2)$$

By implementing this recursive procedure in monadic style as (sizeM n):

```
sizeM n ≝
 do
  b0 ← zeroM n;
  b1 ← oneM n;
  if b0 ∨ b1 then return 1
  else do x ← halfM n; y ← sizeM x; incM y od
 od
```

Again, given the resemblance of the implementation and the pseudo-code, we can prove:

$$\begin{aligned}
&\vdash \text{valueOf } (\text{sizeM } n) = \text{size } n \\
&\vdash \text{stepsOf } (\text{sizeM } n) = \\
&\quad 2(\text{size } n) + \\
&\quad \textbf{if } n \leq 1 \textbf{ then } 0 \\
&\quad \textbf{else } 2(\text{size } n) + \text{size } (\text{size } n - 1) + \text{stepsOf } (\text{sizeM } (n \text{ div } 2))
\end{aligned}$$

We would like to extract an upper bound for stepsOf (sizeM n) from this recursive expression.

6.5.4 Recurrence Pattern

The expression for stepsOf (sizeM n) can be recasted in the following form:

$$\begin{aligned}
&\vdash \textbf{let} \\
&\quad \text{body } n \;=\; \textbf{if } n \leq 1 \textbf{ then } 2 \textbf{ else } 4(\text{size } n) + \text{size } (\text{size } n - 1)\ ; \\
&\quad \text{exit } n \;=\; (n = 1) \\
&\textbf{in} \\
&\quad \forall n. \\
&\qquad \text{stepsOf } (\text{sizeM } n) = \\
&\qquad\quad \textbf{if } n = 0 \textbf{ then } 2 \\
&\qquad\quad \textbf{else } \text{body } n + \textbf{if } \text{exit } n \textbf{ then } 0 \textbf{ else } \text{stepsOf } (\text{sizeM } (n \text{ div } 2))
\end{aligned} \tag{6.2}$$

This shows better the conditions under which stepsOf (sizeM n) recurs with a smaller argument stepsOf (sizeM (n div 2)). Basically, we identify the recursive argument n, and use a simple condition ($n = 0$) on the recursive argument to differentiate between whether recursion will occur or stop. We would like a simple condition on the recursive argument at the start in order

to apply the same pattern for all similar patterns of recursion.

If the stopping condition is not being satisfied for the argument n, we treat the subsequent proceeding in two parts: a *body* and an *exit* check before actual recursion. In this case, the price for a simple stopping condition $n = 0$ is a slightly complicated expression for *body*, and the need for *exit*, but we have a recurrence pattern that fits well into others. For example, the same pattern fits into a recasting of stepsOf (power_ofM b n):

$$
\begin{array}{l}
\vdash \textbf{let} \\
\quad \text{body } n \;= \\
\quad\; 2(\text{size } n) + 2(\text{size } b) + \\
\quad\; \textbf{if } n \le 1 \vee b \le 1 \textbf{ then } 0 \\
\quad\; \textbf{else} \\
\quad\;\; (\text{size } n)(\text{size } b) + \text{size } (n \bmod b) + \\
\quad\;\; \textbf{if } n \not\equiv 0 \pmod{b} \textbf{ then } 0 \textbf{ else } (\text{size } n)(\text{size } b)\ ; \\
\quad \text{exit } n \;=\; (n = 1 \vee b = 0 \vee b = 1 \vee n \not\equiv 0 \pmod{b}) \\
\textbf{in} \\
\quad \forall n. \\
\quad\;\; 1 < b \Rightarrow \\
\quad\;\; \text{stepsOf } (\text{power\_ofM } b\ n) = \\
\quad\;\; \textbf{if } n = 0 \textbf{ then } 2 + 2(\text{size } b) \\
\quad\;\; \textbf{else} \\
\quad\;\;\; \text{body } n + \textbf{if } \text{exit } n \textbf{ then } 0 \textbf{ else } \text{stepsOf } (\text{power\_ofM } b\ (n \text{ div } b))
\end{array}
\tag{6.3}
$$

We call these patterns of recurrence of stepsOf *recurrence loops*. With these examples in mind, we shall develop a general theory for recurrence loops. Our aim is to solve the recurrence for an explicit result, or, more commonly, to extract an upper bound for the result.

6.6 Recurrence Loops

We shall use loop x to denote a generic step-counting recurrence in argument x. The predicate guard x is the stopping condition for recurrence: if guard x is true, the step-count is given by quit x. If guard x is false, the step-count for each iteration is given by body x. In the simplest form, we let the step-count to recur immediately with loop (modify x), where modify x changes the argument x to bring it "closer" to satisfy guard x. Due to the cooperation between guard x and modify x, the recurrence of loop x will terminate eventually. This is the general form of the basic loop x recurrecnce:

$$\forall x.\ \text{loop } x = \textbf{if } \text{guard } x \textbf{ then } \text{quit } x \textbf{ else } \text{body } x + \text{loop } (\text{modify } x) \tag{6.4}$$

To solve this recurrence, we first need to define the number of iterations through the loop. This is readily provided by loop_count, parameterised by guard, modify, and x, which is modelled after the behaviour of loop x:

Definition 127. *The iteration count of* loop *for a well-defined relation* R *between* guard *and* modify.

$$\begin{array}{l}\mathsf{WF}\ R \wedge (\forall x.\ \neg\mathsf{guard}\ x \Rightarrow R\ (\mathsf{modify}\ x)\ x) \Rightarrow \\ \quad \forall x. \\ \qquad \mathsf{loop\_count\ guard\ modify}\ x = \\ \qquad\quad \textbf{if}\ \mathsf{guard}\ x\ \textbf{then}\ 0\ \textbf{else}\ (1 + \mathsf{loop\_count\ guard\ modify}\ (\mathsf{modify}\ x))\end{array}$$

To ensure termination, we need a well-founded relation R, *i.e.*, WF R with R (modify x) x behaves like the "less-than" relation. Moreover, as long as guard x is false, the relation R shall persist this "less-than" behaviour through successive iterations, so that eventually guard x is true, and the recursion stops.

Bearing these prerequisites in mind, the counting process is straight-forward. This is simply storing the recursion on the stack, then upon quit when the condition of guard is met, start the count from 0, and as the stack unwinds, increment the count.

6.6.1 Basic Loop

The loop_count is useful for solving the recurrence, because we can unfold the recurrence until guard z is true, where z = modify$^n(x)$ and n = loop_count guard modify x:

$$\begin{array}{ll} & \mathsf{loop}\ x \\ = & \mathsf{body}\ x + \mathsf{loop}\ (\mathsf{modify}\ x) \\ = & \mathsf{body}\ x + \mathsf{body}\ (\mathsf{modify}\ x) + \mathsf{loop}\ (\mathsf{modify}\ (\mathsf{modify}\ x)) \\ = & \mathsf{body}\ x + \mathsf{body}\ (\mathsf{modify}\ x) + \mathsf{body}\ (\mathsf{modify}\ (\mathsf{modify}\ x)) + \mathsf{loop}\ (\mathsf{modify}^3(x)) \\ = & \ldots \\ = & \mathsf{body}\ x + \mathsf{body}\ (\mathsf{modify}\ x) + \mathsf{body}\ (\mathsf{modify}\ (\mathsf{modify}\ x)) + \ldots + \mathsf{body}\ z + \mathsf{quit}\ z \end{array}$$

This gives the basic theorem for a simple recurrence loop:

Theorem 128. *Explicit solution of basic recurrence* loop x *of Equation 6.4.*

$$\begin{array}{l}\vdash \mathsf{WF}\ R \wedge (\forall x.\ \neg\mathsf{guard}\ x \Rightarrow R\ (\mathsf{modify}\ x)\ x) \wedge \\ \quad (\forall x.\ \mathsf{loop}\ x = \textbf{if}\ \mathsf{guard}\ x\ \textbf{then}\ \mathsf{quit}\ x\ \textbf{else}\ \mathsf{body}\ x + \mathsf{loop}\ (\mathsf{modify}\ x)) \Rightarrow \\ \qquad \forall x. \\ \qquad\quad (\textbf{let} \\ \qquad\qquad n\ =\ \mathsf{loop\_count\ guard\ modify}\ x \\ \qquad\quad \textbf{in} \\ \qquad\qquad \mathsf{loop}\ x = \mathsf{quit}\ (\mathsf{modify}^n(x)) + \sum_{j=0}^{j<n} (\lambda j.\ \mathsf{body}\ (\mathsf{modify}^j(x))))\end{array}$$

For application to complexity analysis of algorithms, we identify several types of recurrence patterns for the step-counting loop x, see Table 6.3. The recurrence loop for list x makes use of HD x, the head of x, and TL x, the tail of x.

Name	Type of argument x	guard x	modify x	condition for WF R
Decreaing Loop	$(x : \mathtt{num})$	$x = 0$	$x - b$	$0 < b$ for termination
Increasing Loop	$(x : \mathtt{num})$	$m \leq x$	$x + b$	$0 < b$ to exceed max m
Dividing Loop	$(x : \mathtt{num})$	$x = 0$	x div b	$1 < b$ for termination
Multiplying Loop	$(x : \mathtt{num})$	$m \leq x$	bx	$1 < b$ to exceed max m
List Reduction Loop	$(x : \alpha\ \mathtt{poly})$	$x = []$	TL x	TL x has shorter LENGTH x

Table 6.3: Types of Recurrence Loop

.

6.6.2 General Loop

The basic recurrence loop of Equation 6.4 can be extended in at least two ways. Sometimes inside an iteration, after completing body x steps, the computation can abort when a condition exit x is met (see the examples of stepsOf (power_ofM b n) and stepsOf (sizeM n)):

$$\forall x.\ \text{loop}\ x = \textbf{if}\ \text{guard}\ x\ \textbf{then}\ \text{quit}\ x\ \textbf{else}\ \text{body}\ x + \textbf{if}\ \text{exit}\ x\ \textbf{then}\ 0\ \textbf{else}\ \text{loop}\ (\text{modify}\ x) \qquad (6.5)$$

If the function body x is complicated, its value may be dominated by another function cover x:

$$\forall x.\ \text{body}\ x \leq \text{cover}\ x$$

Thus the general recurrence results for step-counting loop x are:

Theorem 129. *A bound for the step-counting* loop x *with* $\forall x.\ x \leq \text{modify}\ x$.

$$\begin{array}{l}
\vdash \text{WF}\ R \wedge (\forall x.\ \neg\text{guard}\ x \Rightarrow R\ (\text{modify}\ x)\ x) \wedge (\forall x.\ x \leq \text{modify}\ x) \wedge \\
\quad (\forall x. \\
\qquad \text{loop}\ x = \\
\qquad\quad \textbf{if}\ \text{guard}\ x\ \textbf{then}\ \text{quit}\ x \\
\qquad\quad \textbf{else}\ \text{body}\ x + \textbf{if}\ \text{exit}\ x\ \textbf{then}\ 0\ \textbf{else}\ \text{loop}\ (\text{modify}\ x)) \Rightarrow \\
\qquad \forall x\ \text{cover}. \\
\qquad\quad (\textbf{let} \\
\qquad\qquad n\ =\ \text{loop_count guard modify}\ x \\
\qquad\quad \textbf{in} \\
\qquad\qquad (\forall x.\ \text{body}\ x \leq \text{cover}\ x) \wedge (\forall x\ y.\ x \leq y \Rightarrow \text{cover}\ x \leq \text{cover}\ y) \Rightarrow \\
\qquad\qquad\ \text{loop}\ x \leq \text{quit}\ (\text{modify}^n(x)) + n(\text{cover}\ (\text{modify}^n(x))))
\end{array}$$

Theorem 130. *A bound for the step-counting* loop x *with* $\forall x.$ modify $x \leq x$.

$$\begin{array}{l}
\vdash \mathsf{WF}\ R \wedge (\forall x.\ \neg\mathsf{guard}\ x \Rightarrow R\ (\mathsf{modify}\ x)\ x) \wedge (\forall x.\ \mathsf{modify}\ x \leq x) \wedge \\
\quad (\forall x. \\
\qquad \mathsf{loop}\ x = \\
\qquad\quad \textbf{if}\ \mathsf{guard}\ x\ \textbf{then}\ \mathsf{quit}\ x \\
\qquad\quad \textbf{else}\ \mathsf{body}\ x + \textbf{if}\ \mathsf{exit}\ x\ \textbf{then}\ 0\ \textbf{else}\ \mathsf{loop}\ (\mathsf{modify}\ x)) \Rightarrow \\
\quad \forall x\ \mathsf{cover}. \\
\qquad (\textbf{let} \\
\qquad\quad n\ =\ \mathsf{loop_count}\ \mathsf{guard}\ \mathsf{modify}\ x \\
\qquad \textbf{in} \\
\qquad\quad (\forall x.\ \mathsf{body}\ x \leq \mathsf{cover}\ x) \wedge (\forall x\ y.\ x \leq y \Rightarrow \mathsf{cover}\ x \leq \mathsf{cover}\ y) \Rightarrow \\
\qquad\quad \mathsf{loop}\ x \leq \mathsf{quit}\ (\mathsf{modify}^n(x)) + n(\mathsf{cover}\ x))
\end{array}$$

Theorem 129 is suitable for increasing and multiplying loops. For decreasing and dividing loops, Theorem 130 is more appropriate. For list reduction loop, the corresponding result is:

Theorem 131. *A bound for the step-counting* loop x *with* $(x : \alpha\ \mathtt{poly})$.

$$\begin{array}{l}
\vdash (\forall x.\ \mathsf{body}\ x \leq \mathsf{cover}\ x) \wedge (\forall x\ y.\ x \leq y \Rightarrow \mathsf{cover}\ x \leq \mathsf{cover}\ y) \wedge \\
\quad (\forall x. \\
\qquad \mathsf{loop}\ x = \\
\qquad\quad \textbf{if}\ x = [\,]\ \textbf{then}\ c\ \textbf{else}\ \mathsf{body}\ x + \textbf{if}\ \mathsf{exit}\ x\ \textbf{then}\ 0\ \textbf{else}\ \mathsf{loop}\ (\mathsf{TL}\ x)) \Rightarrow \\
\quad \forall x.\ \mathsf{loop}\ x \leq c + (\mathsf{cover}\ x)(\mathsf{LENGTH}\ x)
\end{array}$$

Here $(x : \alpha\ \mathtt{poly}) \leq (y : \alpha\ \mathtt{poly})$ denotes that x being a sublist of y.

6.6.3 Extended Loop

The AKS algorithm involves some sophisticated computations with stepsOf having more than one varying argument. To handle this situation, we extend the basic recurrence loop x to take two arguments x and y: loop $x\ y$.

We use transform x to modify the argument x, keeping modify y for argument y. The simple loop iteration count loop_count x is extended in HOL4 as:

Definition 132. *The iteration count of loop for two arguments x and y.*

$$\begin{array}{l}
\mathsf{loop2_count}\ \mathsf{guard}\ \mathsf{modify}\ \mathsf{transform}\ x\ y \stackrel{\mathsf{def}}{=} \\
\quad \mathsf{loop_count}\ (\lambda\,(x,y).\ \mathsf{guard}\ x\ y) \\
\qquad (\lambda\,(x,y).\ (\mathsf{transform}\ x, \mathsf{modify}\ y))\ (x,y)
\end{array}$$

from which we can show:

$$\begin{array}{l}\mathsf{WF}\ R \wedge (\forall x\ y.\ \neg\mathsf{guard}\ x\ y \Rightarrow R\ (\mathsf{transform}\ x,\mathsf{modify}\ y)\ (x,y)) \Rightarrow \\ \ \forall x\ y. \\ \quad \mathsf{loop2\_count\ guard\ modify\ transform}\ x\ y = \\ \quad\ \textbf{if}\ \mathsf{guard}\ x\ y\ \textbf{then}\ 0 \\ \quad\ \textbf{else}\ (1 + \mathsf{loop2\_count\ guard\ modify\ transform}\ (\mathsf{transform}\ x)\ (\mathsf{modify}\ y))\end{array}$$

Each recurrence type in Table 6.3 can be paired with

$$\text{either } \forall x.\ x \leq \mathsf{transform}\ x \quad \text{or } \forall x.\ \mathsf{transform}\ x \leq x.$$

Typical generalisations of the upper bound for loop x (Theorems 129 and 130) become:

Theorem 133. *A bound for the step-counting* loop $x\ y$ *with* $\forall x.\ x \leq \mathsf{transform}\ x$.

$$\begin{array}{l}\vdash \mathsf{WF}\ R \wedge (\forall x\ y.\ \neg\mathsf{guard}\ x\ y \Rightarrow R\ (\mathsf{transform}\ x,\mathsf{modify}\ y)\ (x,y)) \wedge \\ \quad (\forall x.\ x \leq \mathsf{transform}\ x) \wedge (\forall x.\ \mathsf{modify}\ x \leq x) \wedge \\ \quad (\forall x\ y. \\ \qquad \mathsf{loop}\ x\ y = \\ \qquad\ \textbf{if}\ \mathsf{guard}\ x\ y\ \textbf{then}\ \mathsf{quit}\ x\ y \\ \qquad\ \textbf{else} \\ \qquad\quad \mathsf{body}\ x\ y + \textbf{if}\ \mathsf{exit}\ x\ y\ \textbf{then}\ 0\ \textbf{else}\ \mathsf{loop}\ (\mathsf{transform}\ x)\ (\mathsf{modify}\ y)) \Rightarrow \\ \ \forall x\ y\ \mathsf{cover}. \\ \qquad (\textbf{let} \\ \qquad\quad n\ =\ \mathsf{loop2\_count\ guard\ modify\ transform}\ x\ y \\ \qquad\ \textbf{in} \\ \qquad\quad (\forall x\ y.\ \mathsf{body}\ x\ y \leq \mathsf{cover}\ x\ y) \wedge \\ \qquad\quad (\forall x_1\ y_1\ x_2\ y_2.\ x_1 \leq x_2 \wedge y_1 \leq y_2 \Rightarrow \mathsf{cover}\ x_1\ y_1 \leq \mathsf{cover}\ x_2\ y_2) \Rightarrow \\ \qquad\quad \mathsf{loop}\ x\ y \leq \\ \qquad\quad \mathsf{quit}\ (\mathsf{transform}^n(x))\ (\mathsf{modify}^n(y)) + n(\mathsf{cover}\ (\mathsf{transform}^n(x))\ y))\end{array}$$

Theorem 134. *A bound for the step-counting* loop $x\ y$ *with* $\forall x.$ transform $x \leq x$.

$$
\begin{array}{l}
\vdash \mathsf{WF}\ R \wedge (\forall x\ y.\ \neg\mathsf{guard}\ x\ y \Rightarrow R\ (\mathsf{transform}\ x, \mathsf{modify}\ y)\ (x,y)) \wedge \\
\quad (\forall x.\ \mathsf{transform}\ x \leq x) \wedge (\forall x.\ x \leq \mathsf{modify}\ x) \wedge \\
\quad (\forall x\ y. \\
\qquad \mathsf{loop}\ x\ y = \\
\qquad\quad \textbf{if}\ \mathsf{guard}\ x\ y\ \textbf{then}\ \mathsf{quit}\ x\ y \\
\qquad\quad \textbf{else} \\
\qquad\quad\quad \mathsf{body}\ x\ y + \textbf{if}\ \mathsf{exit}\ x\ y\ \textbf{then}\ 0\ \textbf{else}\ \mathsf{loop}\ (\mathsf{transform}\ x)\ (\mathsf{modify}\ y)) \Rightarrow \\
\quad \forall x\ y\ \mathsf{cover}. \\
\qquad (\textbf{let} \\
\qquad\quad n\ =\ \mathsf{loop2\_count}\ \mathsf{guard}\ \mathsf{modify}\ \mathsf{transform}\ x\ y \\
\qquad\quad \textbf{in} \\
\qquad\quad (\forall x\ y.\ \mathsf{body}\ x\ y \leq \mathsf{cover}\ x\ y) \wedge \\
\qquad\quad (\forall x_1\ y_1\ x_2\ y_2.\ x_1 \leq x_2 \wedge y_1 \leq y_2 \Rightarrow \mathsf{cover}\ x_1\ y_1 \leq \mathsf{cover}\ x_2\ y_2) \Rightarrow \\
\qquad\quad\ \mathsf{loop}\ x\ y \leq \\
\qquad\quad\ \mathsf{quit}\ (\mathsf{transform}^n(x))\ (\mathsf{modify}^n(y)) + n(\mathsf{cover}\ x\ (\mathsf{modify}^n(y))))
\end{array}
$$

There are similar results for list recurrences with $(x : \beta\ \mathtt{poly})$ and $(y : \alpha\ \mathtt{poly})$. These are the main tools to extract an upper bound for the number of steps in our algorithm analysis.

In a similar fashion, we can generalise the theory of recurrence loops further to handle the situation of a step-counting *loop* with 3 varying arguments: loop $x\ y\ z$. Luckily, for the complexity analysis of the AKS algorithm, just 2 varying arguments suffice.

6.7 Complexity Results

Continuing with the $\lceil \log n \rceil$ computation example in Section 6.5, we shall apply the recurrence theory to obtain bounds on the number of steps of the algorithms involved in the example.

6.7.1 Size Complexity

For the computation of size n using sizeM n, the recurrence of stepsOf $(\mathsf{sizeM}\ n)$, as shown in Equation 6.2, fits into the pattern of recurrence by dividing loop (see Table 6.3), with the divisor $b = 2$. Applying the recurrence loop theory of dividing loop, we find:

$$\vdash \mathsf{stepsOf}\ (\mathsf{sizeM}\ n) \leq 7(\mathsf{size}\ n)^2$$

This gives the complexity class:

Theorem 135. *Our algorithm (see Section 6.5.3) to compute* $(\mathsf{size}\ n)$ *belongs to the polynomial-time class.*

$$\vdash \mathsf{stepsOf} \circ \mathsf{sizeM} \in \mathcal{O}(\lceil \log n \rceil^2)$$

Therefore the pseudocode of size n (see Definition 117) results in a polynomial-time algorithm.

6.7.2 Power Check Complexity

As for the computation of n power_of b using power_ofM b n, Equation 6.3 shows that the recurrence of stepsOf (power_ofM b n) is a dividing loop (see Table 6.3) with b as the divisor. Applying the recurrence theory for dividing loops, we show that:

$$\vdash \mathsf{stepsOf}\ (\mathsf{power\_ofM}\ b\ n) \leq 11(\mathsf{size}\ b)(\mathsf{size}\ n)^2$$

Thus obtaining its complexity class:

Theorem 136. *Our algorithm (see Section 6.5.2) to compute* (n power_of b) *belongs to the polynomial-time class.*

$$\vdash \mathsf{stepsOf} \circ \mathsf{power\_ofM}\ b \in \mathcal{O}(\lceil \log n \rceil^2)$$

Hence the pseudocode of Equation 6.1 for testing whether n is the power of a base b leads to a polynomial-time algorithm.

6.7.3 Logarithm Complexity

Recall the computation of $\lceil \log n \rceil$ by size n (Theorem 126):

$$\lceil \log n \rceil = \textbf{if}\ n = 0\ \textbf{then}\ 0\ \textbf{else}\ \mathsf{size}\ n - \textbf{if}\ n\ \mathsf{power\_of}\ 2\ \textbf{then}\ 1\ \textbf{else}\ 0 \tag{6.6}$$

We have the sizeM n subroutine (Section 6.5.3). We can specialise the power_ofM n b subroutine (Section 6.5.2) to base $b = 2$, getting:

$$\mathsf{power\_twoM} \stackrel{\text{def}}{=} \mathsf{power\_ofM}\ 2$$
$$\vdash \mathsf{stepsOf} \circ \mathsf{power\_twoM} \in \mathcal{O}(\lceil \log n \rceil^2)$$

Using these subroutines, $\lceil \log n \rceil$ can be implemented in monadic style as logM n:

```
logM n ≝
 do
  gd ← zeroM n;
  if gd then return 0
  else
   do
    x ← sizeM n;
    b ← power_twoM n;
    y ← boolM b;
    subM x y
   od
 od
```

With the previous results of sizeM n and power_twoM n', we have:

$$\vdash \mathsf{valueOf}\ (\mathsf{logM}\ n) = \lceil \log\ n \rceil$$
$$\vdash \mathsf{stepsOf}\ (\mathsf{logM}\ n) \leq 28(\mathsf{size}\ n)^2$$

This leads directly to its complexity class:

Theorem 137. *Our computation of* $\lceil \log\ n \rceil$ *by Equation 6.6 is polynomial-time.*

$$\vdash \mathsf{stepsOf} \circ \mathsf{logM} \in \mathcal{O}(\lceil \log\ n \rceil^2)$$

Scripts For the formalisation of recurrence theory, refer to `algorithm/loop`. It depends on library scripts in `algorithm/lib`, treating bit-size and the big-$\mathcal{O}$ notation. The complexity analysis of $\lceil \log\ n \rceil$ is given in `aks/machine/countBasic`.

6.8 Summary

For a formal study of computational complexity of algorithms, our starting point is a machine with only elementary operations. Each computation is expressed in monadic style, relying on the binding of Writer monads to count the number of steps. Our complexity measure is based on the length of the binary representation of a number. We build subroutines, and implement recursive algorithms using subroutines. In order to count the number of steps in recursive algorithms modelled upon recursive definitions, we develop a theory of recurrence loop counts for some typical loop patterns. The theory deals with the recurrence equation for the number of steps with one or more varying parameters, and extracts an upper bound. With all necessary tools at hand, we are ready to investigate the computational complexity of the AKS algorithms.

6.9 Remarks

Note that our complexity analysis is based on bounds on the number of steps, rather than relying on composition of complexity classes of subroutines. This has certain advantages. For example, consider the deterministic primality test by trial division looking for a factor:

$$\vdash \mathsf{prime}\ n \iff 1 < n \wedge \forall q.\ 1 < q \wedge q \leq \mathsf{SQRT}\ n \Rightarrow q \not| n \tag{6.7}$$

This test for prime n can be implemented in monadic style as primeM n. Since our loop recurrence can provide estimates for either upper bound or lower bound, we can prove formally that:

$$\vdash \mathsf{valueOf}\ (\mathsf{primeM}\ n) \iff \mathsf{prime}\ n$$
$$\vdash \mathsf{prime}\ n \Rightarrow \mathsf{SQRT}\ n \leq \mathsf{stepsOf}\ (\mathsf{primeM}\ n)$$

This confirms the primality test by Equation 6.7 is not a polynomial-time algorithm.

There are other developments in formalisation related to computational complexity. For example, in a series of papers (Charguéraud and Pottier [2015, 2017]; Guéneau et al. [2018]) the authors described their formal proofs of amortized complexity and asymptotic complexity in Coq. An informative summary of their work was presented by Armaël Guéneau [2017].

Manuel Eberl [2017] had formalised the Akra-Bazzi theorem in Isabelle/HOL. This is the master theorem for the complexity analysis of divde-and-conquer algorithms. Applying this work for imperative programs in Isabelle/HOL was done by Haslbeck and Zhan and Haslbeck [2018]. Our work does not require such an advanced tool. The AKS algorithm can be implemented with subroutines consisting of while-loops with up to two varying parameters (see Chapter 7). These can be handled by the simple recurrence theory outlined in this chapter.

The asympotic big-$\mathcal{O}$ notation has been formalised in Avigad and Donnelly [2004] using the Isabelle/HOL proof assistant. The work is base on ordered rings, and is applied to simplification of expressions, *e.g.*, obtaining the leading term of a serie expansion.

Maximilian Haslbeck and Nipkow [2018] formalised a meta theory in Isabelle/HOL to study different Hoare logics for reasoning about time bound in the analysis of programs. The idea is to extend the Hoare triple for program correctness with time credits to reason about the temporal behaviour of the program. There are merits in this approach, especially the use of separation logic, which we may pursue in future work.

Chapter 7

AKS Complexity

This chapter delivers the proof about the computational complexity of the AKS algorithm. We shall implement the AKS algorithm, with all necessary subroutines for each phase. We ensure that each subroutine is correctly implemented, and apply the recurrence theory to provide an upper bound for its number of steps. Based on these results, we prove the correctness of our implementation, and show that it belongs to the class of polynomial-time algorithms.

We may say most aptly that
the Analytical Engine weaves algebraic patterns,
just as the Jacquard loom weaves flowers and leaves.
— Ada Lovelace (1843)[1]

7.1 AKS Implementation

We shall weave the 3 phases of the AKS algorithm in monadic style (see Definition 3) for an implementation (compare with the pseudocode in Algorithm 1):

```
aksM n ≝
 do
  b ← power_freeM n;
  if b then
   do
    c ← paramM n;
    case c of
    | nice k ▷ eqM k n
    | good k ▷ poly_intro_rangeM n k k
    | bad ▷ return F
   od
  else return F
 od
```

[1] In *Notes on the Analytical Engine*, her appendices to an acount of Charles Babbage's 1842 lectures in Turin.

(7.1)

Each phase is implemented as a subroutine, and further subroutines may arise through their implementations. By analysing the computational complexity of each subroutine, we shall prove that this AKS implementation is a polynomial-time algorithm.

7.2 Power Free Check

The power free test constitutes Phase 1 of the AKS algorithm. Recall the definition of power_free n (Definition 59, page 45):

$$\text{power\_free } n \stackrel{\text{def}}{=} \forall m\ e.\ n = m^e \Rightarrow m = n \wedge e = 1$$

This is not computationally effective, but the power free test (Theorem 61) is:

$$\vdash \text{power\_free } n \iff 1 < n \wedge \forall j.\ 1 < j \wedge j \leq \lceil \log n \rceil \Rightarrow (\text{ROOT } j\ n)^j \neq n \tag{7.2}$$

To implement this, we need to develop a few subroutines:

(1) a computation of the limit $\lceil \log n \rceil$

(2) a computation of ROOT j n, the integer j-th root of n.

(3) a computation of n^j, the j-th power of n.

We have developed the logM n subroutine (Section 6.5.1) to compute $\lceil \log n \rceil$. We shall first consider the subroutine for exponentiation, *i.e.*, n^j, as this will be useful for the later ROOT j n.

7.2.1 Integer Exponentiation

The exponential form can be computed by "repeat squaring":

$$\vdash b^n = \textbf{if } n = 0 \textbf{ then } 1 \textbf{ else } (\textbf{if } \text{EVEN } n \textbf{ then } 1 \textbf{ else } b)(b^2)^{n \text{ div } 2}$$

We could also check $n = 1$ and return the base b. This is optimisation, which can complicate the complexity analysis. Our aim is just to establish the complexity class of an algorithm. We are not interested to improve its performance, unless the improvement affects the complexity class.[2]

[2]We use repeat squaring for integer power precisely for this purpose.

Implementing this simple recursive form directly as $(\mathsf{expM}\ b\ n)$, we have:

$$
\begin{array}{l}
\vdash \mathsf{valueOf}\ (\mathsf{expM}\ b\ n) = b^n \\
\vdash \textbf{let} \\
\quad \mathsf{body}\ b\ n\ = \\
\quad\quad 1 + 5(\mathsf{size}\ n) + (\mathsf{size}\ b)^2 + \textbf{if}\ \mathsf{EVEN}\ n\ \textbf{then}\ 0\ \textbf{else}\ (\mathsf{size}\ b)(\mathsf{size}\ (b^2)^{n\ \mathsf{div}\ 2}) \\
\quad \textbf{in} \\
\quad\quad \forall b\ n. \\
\quad\quad\quad \mathsf{stepsOf}\ (\mathsf{expM}\ b\ n) = \\
\quad\quad\quad\quad \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ \mathsf{body}\ b\ n + \mathsf{stepsOf}\ (\mathsf{expM}\ (b^2)\ (n\ \mathsf{div}\ 2)) \\
\vdash \mathsf{stepsOf}\ (\mathsf{expM}\ b\ n) \leq 15(\mathsf{max}\ 1\ n)^3(\mathsf{size}\ b)^2(\mathsf{size}\ n)^2
\end{array}
$$

The bound for $\mathsf{stepsOf}\ (\mathsf{expM}\ b\ n)$ is a result of our theory of recurrence loops with 2 arguments. Note the factor $(\mathsf{max}\ 1\ n)$ in the bound. This is because the recurrence theory initially gives:

$$\vdash \mathsf{stepsOf}\ (\mathsf{expM}\ b\ n) \leq 1 + \mathsf{size}\ n + 5(\mathsf{size}\ n)^2 + 8(\mathsf{size}\ n)n^3(\mathsf{size}\ b)^2$$

In order to simplify this for later use, either we have to make $0 < n$ a pre-condition, or simply replace n by $(\mathsf{max}\ 1\ n)$ to cater for the case $n = 0$. This upper bound shows:

Theorem 138. *Computation of an integer raised to exponent n is polynomial-time in* $n^{\frac{3}{2}\lceil \mathsf{log}\ n \rceil}$.

$$\vdash \mathsf{stepsOf} \circ \mathsf{expM}\ b \in \mathcal{O}((\lambda n.\ n^{3\lceil \mathsf{log}\ n \rceil^2}))$$

This result is for a fixed base b, and hides the constant $(\mathsf{size}\ b)^2$. When the exponent n is bounded, but the base b varies, the complexity class becomes $\mathcal{O}(\lceil \mathsf{log}\ b \rceil^2)$.

7.2.2 Integer Root

Working out the k-th integer root of a number n by hand is well-known before the availability of logarithmic tables or electronic calculators:

Definition 139. *Extraction of the binary k-th root of n, bit by bit.*

$$
\begin{array}{l}
\mathsf{ROOT}\ k\ n \stackrel{\mathsf{def}}{=} \\
\quad \textbf{if}\ k = 0\ \textbf{then}\ \mathsf{ROOT}\ 0\ n \\
\quad \textbf{else if}\ n = 0\ \textbf{then}\ 0 \\
\quad \textbf{else}\ (\textbf{let}\ m\ =\ 2(\mathsf{ROOT}\ k\ (n\ \mathsf{div}\ 2^k))\ \textbf{in}\ m + \textbf{if}\ (1+m)^k \leq n\ \textbf{then}\ 1\ \textbf{else}\ 0)
\end{array}
\tag{7.3}
$$

This means $m = 2(\mathsf{ROOT}\ k\ (n\ \mathsf{div}\ 2^k))$, which can be computed recursively, is a very good approximation to $\mathsf{ROOT}\ k\ n$. It differs from the true integer root only by 1, thus a simple check can adjust the value to make it correct.

To implement this, we need two subroutines: one to compute n div 2^k and another to compute n^k. The subroutine power_ofM b n in Section 6.5.2 can be adapted to compute n div b^k by counting the number of divisions by base b. The subroutine expM n k in Section 7.2.1 computes n^k. With these subroutines, the computation of ROOT k n can be implemented in monadic style as rootM k n:

$$\vdash \mathsf{valueOf}\ (\mathsf{rootM}\ k\ n) = \mathsf{ROOT}\ k\ n$$
$$\vdash \mathsf{stepsOf}\ (\mathsf{rootM}\ k\ n) \leq 157(\max\ 1\ k)^3(\mathsf{size}\ k)^2(\mathsf{size}\ n)^3$$

The bound is obtained by recurrence theory of dividing loops with cover for body, leading to:

Theorem 140. *Integer root extraction by Equation 7.3 is a polynomial-time algorithm.*

$$\vdash \mathsf{stepsOf} \circ \mathsf{rootM}\ k \in \mathcal{O}(\lceil \log\ n \rceil^3)$$

Other algorithms to compute the integer k-th root of a number n are known (for example, see [von zur Gathen and Gerhard, 2013, Section 9.5]). Newton's method based on successive approximation can be adapted to compute ROOT k n.[3] Our integer root algorithm is well-suited for our simple loop-counting method of complexity analysis. Note that an implementation of integer square root n by bisection has been analysed by Haslbeck and Nipkow [2018] in an example. This bisection algorithm was formally shown to be of the order $\mathcal{O}(\lceil \log\ n \rceil)$.

7.2.3 Power Free Complexity

We have all the subroutines logM n, rootM k n, and expM n k available. The power free test of Equation 7.2 can be implemented in monadic style as power_freeM n, with:

$$\vdash \mathsf{valueOf}\ (\mathsf{power\_freeM}\ n) \iff \mathsf{power\_free}\ n$$
$$\vdash \mathsf{stepsOf}\ (\mathsf{power\_freeM}\ n) \leq 207(\mathsf{size}\ n)^9$$

The bound is obtained by recurrence theory of decreasing loop with cover for body, leading to:

Theorem 141. *The Power Free Test of Equation 7.2 is a polynomial-time algorithm.*

$$\vdash \mathsf{stepsOf} \circ \mathsf{power\_freeM} \in \mathcal{O}(\lceil \log\ n \rceil^9)$$

The best known power-free test has run-time of $\widetilde{\mathcal{O}}(\lceil \log\ n \rceil^3)$, as given in *Modern Computer Algebra* by [von zur Gathen and Gerhard, 2013, Theorem 9.28].

[3] Newton's iteration to compute ROOT k n makes use of a sequence m_j, where

$$m_0 = n, \quad \text{and} \quad m_{j+1} = \left((k-1)m_j + n \ \mathsf{div}\ (m_j^{(k-1)})\right)\ \mathsf{div}\ k.$$

This sequence converges to the integer k-th root of n. When $k = 2$, this corresponds to the square-root iteration $m_{j+1} = (m_j + n \ \mathsf{div}\ m_j)\ \mathsf{div}\ 2$, also known as integer square-root by bisection.

7.3 AKS Parameter

Phase 2 of the AKS algorithm is about searching a parameter k for the input number n. A nice k is a divisor of n. A good k, useful for Phase 3, is a modulus. The condition is that the multiplicative order of n in modulus k, $\mathsf{order}_k(n)$, is at least $\lceil \log n \rceil^2$.

Before we can implement this parameter search, we need to build subroutines for modular arithmetic from the primitives of machine model (Section 6.3).

7.3.1 Modular Arithmetic

We develop the library for modular arithmetic for a modulus m:

- Modular addition:

$$\begin{array}{l} \mathsf{maddM}\ m\ x\ y \stackrel{\mathsf{def}}{=} \textbf{do}\ z\ \leftarrow\ \mathsf{addM}\ x\ y;\ \mathsf{modM}\ z\ m\ \textbf{od} \\ \vdash \mathsf{valueOf}\ (\mathsf{maddM}\ m\ x\ y) = (x + y)\ \mathsf{mod}\ m \\ \vdash \mathsf{stepsOf}\ (\mathsf{maddM}\ m\ x\ y) \leq 3(\mathsf{size}\ m)(\mathsf{size}\ (\mathsf{max}\ x\ y)) \end{array}$$

- Modular subtraction:

$$\begin{array}{l} \mathsf{msubM}\ m\ x\ y \stackrel{\mathsf{def}}{=} \textbf{do}\ z\ \leftarrow\ \mathsf{subM}\ x\ y;\ \mathsf{modM}\ z\ m\ \textbf{od} \\ \vdash \mathsf{valueOf}\ (\mathsf{msubM}\ m\ x\ y) = (x - y)\ \mathsf{mod}\ m \\ \vdash \mathsf{stepsOf}\ (\mathsf{msubM}\ m\ x\ y) \leq 2(\mathsf{size}\ m)(\mathsf{size}\ (\mathsf{max}\ x\ y)) \end{array}$$

- Modular multiplication:

$$\begin{array}{l} \mathsf{mmulM}\ m\ x\ y \stackrel{\mathsf{def}}{=} \textbf{do}\ z\ \leftarrow\ \mathsf{mulM}\ x\ y;\ \mathsf{modM}\ z\ m\ \textbf{od} \\ \vdash \mathsf{valueOf}\ (\mathsf{mmulM}\ m\ x\ y) = xy\ \mathsf{mod}\ m \\ \vdash \mathsf{stepsOf}\ (\mathsf{mmulM}\ m\ x\ y) \leq 3(\mathsf{size}\ m)(\mathsf{size}\ x)(\mathsf{size}\ y) \end{array}$$

For modular arithmetic, the number of steps is dominated by the final division to find the remainder. In $\mathsf{maddM}\ m\ x\ y$, it is to find the remainder of $x + y$, which is estimated to be less than $2(\mathsf{max}\ x\ y)$. In $\mathsf{msubM}\ m\ x\ y$, it is to find the remainder of $x - y$, which is known to be less than x. This results in a slight difference in $\mathsf{stepsOf}\ (\mathsf{maddM}\ m\ x\ y)$ and $\mathsf{stepsOf}\ (\mathsf{msubM}\ m\ x\ y)$.

7.3.2 Modular Exponentiation

This is computed using basic modular arithmetic with repeating squaring:

$$\begin{array}{l} \vdash 1 < m \Rightarrow \\ \quad b^n\ \mathsf{mod}\ m = \\ \quad\ \textbf{if}\ n = 0\ \textbf{then}\ 1 \\ \quad\ \textbf{else}\ (\textbf{let}\ c\ =\ (b^2)^{n\ \mathsf{div}\ 2}\ \mathsf{mod}\ m\ \textbf{in if}\ \mathsf{EVEN}\ n\ \textbf{then}\ c\ \textbf{else}\ bc\ \mathsf{mod}\ m) \end{array}$$

Implementing this in monadic style as mexpM $m\ b\ n$, we formally proving that:

$$\vdash \mathsf{valueOf}\ (\mathsf{mexpM}\ m\ b\ n) = b^n \bmod m$$
$$\vdash \mathsf{stepsOf}\ (\mathsf{mexpM}\ m\ b\ n) \leq 17(\mathsf{size}\ n)^2(\mathsf{size}\ m)^2(\mathsf{size}\ (\max\ b\ m))^2$$

The bound on steps is achieved by using a cover for the body. From this we obtain:

Theorem 142. *Modular exponentiation can be implemented as a polynomial-time algorithm.*

$$\vdash \mathsf{stepsOf} \circ \mathsf{mexpM}\ m\ b \in \mathcal{O}(\lceil \log n \rceil^2)$$

7.3.3 Multiplicative Order

Given a number n, and a modulus m, the multiplicative order of n in the ring $\mathbb{Z}_m$, denoted by $\mathsf{order}_m(n)$, is given by:

Definition 143. *The multiplicative* $\mathsf{order}_m(n)$ *is the least index* k *such that* $n^k \equiv 1 \pmod m$.

$$1 < m \Rightarrow$$
$$(\mathsf{order}_m(n) = k \iff$$
$$n^k \equiv 1 \pmod m \wedge \forall j.\ 0 < j \wedge j < (\textbf{if}\ k = 0\ \textbf{then}\ m\ \textbf{else}\ k) \Rightarrow n^j \bmod m \neq 1)$$

The least requirement suggests the following basic search, with cutoff c and starting value j, and compute $\mathsf{order}_m(n)$ by searching from 1 onwards to modulus m:

$$\begin{array}{l}
\mathsf{ordz_compute}\ m\ n \stackrel{\mathsf{def}}{=} \\
\quad \textbf{if}\ m = 0\ \textbf{then}\ \mathsf{order}_0(n) \\
\quad \textbf{else if}\ m = 1\ \textbf{then}\ 1 \\
\quad \textbf{else}\ \mathsf{ordz_seek}\ m\ n\ m\ 1 \\
\textbf{where} \\
\quad \mathsf{ordz_seek}\ m\ n\ c\ j \stackrel{\mathsf{def}}{=} \\
\quad \textbf{if}\ m = 0 \vee c \leq j\ \textbf{then}\ 0 \\
\quad \textbf{else if}\ n^j \equiv 1 \pmod m\ \textbf{then}\ j \\
\quad \textbf{else}\ \mathsf{ordz_seek}\ m\ n\ c\ (1+j)
\end{array} \tag{7.4}$$

Note that for a total function, we let $m = 0$ to give an undefined value $\mathsf{order}_0(n)$.

Theorem 144. *The order search by Equation 7.4 always yields the correct result.*

$$\vdash \mathsf{ordz\_compute}\ m\ n = \mathsf{order}_m(n)$$

Proof. The search range from 1 to modulus m is sufficient because $\mathsf{order}_m(n) \leq m$:

$$\vdash 0 < m \Rightarrow (\mathsf{order}_m(n) = 0 \iff \mathsf{gcd}(m,n) \neq 1)$$
$$\vdash 0 < m \wedge 0 < \mathsf{order}_m(n) \Rightarrow \mathsf{order}_m(n) \mid \varphi(m)$$
$$\vdash \varphi(m) \leq m$$

□

The first two of the listed results show that there is room for optimisation: *(i)* we could first check if $\mathsf{gcd}(m,n) = 1$, otherwise $\mathsf{order}_m(n) = 0$; *(ii)* we could have computed $\varphi(m)$, then only consider the divisors of $\varphi(m)$. These optimisations, while improving performance, will introduce more subroutines, making the computational analysis more complicated. The sequential search to compute $\mathsf{order}_m(n)$ can be implemented in monadic style, giving:

$$\vdash \mathsf{valueOf}\ (\mathsf{ordz\_seekM}\ m\ n\ c\ j) = \mathsf{ordz\_seek}\ m\ n\ c\ j$$
$$\vdash \mathsf{valueOf}\ (\mathsf{ordzM}\ m\ n) = \mathsf{order}_m(n)$$
$$\vdash \mathsf{stepsOf}\ (\mathsf{ordz\_seekM}\ m\ n\ c\ j) \leq 26(\mathsf{max}\ 1\ c)(\mathsf{size}\ (\mathsf{max}\ j\ c))(\mathsf{size}\ m)^2(\mathsf{size}\ (\mathsf{max}\ n\ m))^2(\mathsf{size}\ c)^3$$
$$\vdash \mathsf{stepsOf}\ (\mathsf{ordzM}\ m\ n) \leq 29(\mathsf{max}\ 1\ m)(\mathsf{size}\ n)(\mathsf{size}\ m)^7$$

The last one shows that:

Theorem 145. *Our computation of* $\mathsf{order}_m(n)$ *is a polynomial-time algorithm.*

$$\vdash \mathsf{stepsOf} \circ \mathsf{ordzM}\ m \in \mathcal{O}(\lceil \mathsf{log}\ n \rceil)$$

Note that this is the complexity to compute $\mathsf{order}_m(n)$ for a fixed modulus m. When the modulus m varies, as this will happen in the AKS parameter search, we have to use the bound to see the full picture, which is $\mathcal{O}((m\lceil \mathsf{log}\ m \rceil)^7)$.

7.3.4 Search for Parameter

To compute the parameter $\mathsf{aks\_param}\ n$, we use a straight-forward search from $k = 2$ onwards, up to some cutoff c, and look for modulus k such that $m \leq \mathsf{order}_k(n)$ for some maximum m.

We first check if $k \mid n$, or equivalently, if $n \equiv 0 \pmod{k}$. If $k \mid n$, then n has a factor k, which is a proper factor if $k < n$. This factor k certainly tells a lot about the primality of n, and we label it as a nice k. Otherwise, $k \nmid n$, and because we are searching by k sequentially, $\mathsf{gcd}(k,n) = 1$, as any common divisor of k and n, if present, will have already been revealed. Thus $\mathsf{order}_k(n) \neq 0$, and its computation has been analysed in Section 7.3.3. The only question is whether $m \leq \mathsf{order}_k(n)$. If it does, we label it as a good k, and make use of this parameter for Phase 3 of the AKS algorithm. Otherwise, we increment k and repeat this search. If the cutoff c is reached without having any nice k or good k, we have the situation bad.

These 3 possible situations are wrapped up as:

$$
\begin{array}{l}
\mathsf{param\_seek}\ m\ c\ n\ k \stackrel{\mathsf{def}}{=} \\
\quad \textbf{if}\ k = 0\ \textbf{then}\ \mathsf{bad} \\
\quad \textbf{else if}\ c \leq k\ \textbf{then}\ \mathsf{bad} \\
\quad \textbf{else if}\ n \equiv 0 \pmod{k}\ \textbf{then}\ \mathsf{nice}\ k \\
\quad \textbf{else if}\ m \leq \mathsf{order}_k(n)\ \textbf{then}\ \mathsf{good}\ k \\
\quad \textbf{else}\ \mathsf{param\_seek}\ m\ c\ n\ (k+1) \\
\textbf{where} \\
\quad \mathsf{param}\ n \stackrel{\mathsf{def}}{=} \\
\quad\ (\textbf{let} \\
\qquad m = \lceil \log n \rceil\ ; \\
\qquad c = 2 + m^5\ \mathsf{div}\ 2 \\
\quad\ \textbf{in} \\
\qquad \textbf{if}\ m \leq 1\ \textbf{then}\ \mathsf{nice}\ n\ \textbf{else}\ \mathsf{param\_seek}\ (m^2)\ c\ n\ 2)
\end{array}
\tag{7.5}
$$

Note that, although we do not expect the caller of $\mathsf{param\_seek}\ m\ c\ n\ k$ to give $k = 0$, for a total function we have to deal with the situation when $k = 0$, so we just set that to bad. Similarly, when m is too small the search is trivial, but then the result will just be $\mathsf{nice}\ k$, not $\mathsf{good}\ k$.

When $\mathsf{param}\ n$ puts the cutoff $c = 2 + \lceil \log n \rceil^5\ \mathsf{div}\ 2$, only $\mathsf{nice}\ k$ or $\mathsf{good}\ k$ can result, due to:

$$
\begin{array}{l}
\vdash \mathsf{param}\ n = \mathsf{aks\_param}\ n \\
\vdash \mathsf{param}\ n \neq \mathsf{bad}
\end{array}
$$

For details about the existence of parameter, see Section 3.3.2, especially Corollary 66.

The search for the AKS parameter is implemented in monadic style with $\mathsf{paramM}\ n$ calling $\mathsf{param\_seekM}\ m\ c\ n\ k$ with initial values. Our complexity analysis shows:

$$
\begin{array}{l}
\vdash \mathsf{valueOf}\ (\mathsf{param\_seekM}\ m\ c\ n\ k) = \mathsf{param\_seek}\ m\ c\ n\ k \\
\vdash \mathsf{valueOf}\ (\mathsf{paramM}\ n) = \mathsf{param}\ n \\
\vdash \mathsf{stepsOf}\ (\mathsf{param\_seekM}\ m\ c\ n\ k) \leq \\
\quad 41(\mathsf{max}\ 1\ (c-k))(\mathsf{max}\ 1\ c)(\mathsf{size}\ (\mathsf{max}\ c\ k))(\mathsf{size}\ m)(\mathsf{size}\ n)(\mathsf{size}\ c)^7 \\
\vdash \mathsf{stepsOf}\ (\mathsf{paramM}\ n) \leq 1418157969(\mathsf{size}\ n)^{20}
\end{array}
$$

the last one shows this computational complexity result:

Theorem 146. *Our AKS parameter search by (7.5) is a polynomial-time algorithm.*

$$\vdash \mathsf{stepsOf} \circ \mathsf{paramM} \in \mathcal{O}(\lceil \log n \rceil^{20})$$

The AKS paper analysed the parameter search, giving a run-time estimate of $\widetilde{\mathcal{O}}(\lceil \log n \rceil^7)$. The discrepancy is due to our use of long multiplication in subroutines, see Section 7.5.1.

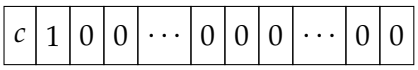
7.4 Introspective Checks

In Phase 3 of the AKS algorithm, Equation 1.1 shows the introspective checks:

$$(\mathsf{X} + c)^n \equiv \mathsf{X}^n + c \pmod{n,\ \mathsf{X}^k - \mathbf{1}}$$

Since only remainders under modulus $\mathsf{X}^k - \mathbf{1}$ are considered, the polynomial computations are performed in the quotient ring $\mathfrak{R}_{n,k} = \mathbb{Z}_n[\mathsf{X}]/(\mathsf{X}^k - \mathbf{1})$, where polynomial equivalence $(\equiv)$ becomes equality $(=)$ between remainders (see Section 5.2). In this section, we shall take this view, and hide away the double moduli $(\bmod n, \mathsf{X}^k - \mathbf{1})$ notation for brevity.

For the sake of simplifying introspective computations in $\mathfrak{R}_{n,k}$, we shall treat polynomials as unnormalised remainders with respect to the modulus $\mathsf{X}^k - \mathbf{1}$. We write `weak p` to represent an unnormalised polynomial `p`, in contrast to its normalised counterpart `poly p`. Each unnormalised polynomial in $\mathfrak{R}_{n,k}$ is represented by a list of coefficients of length k, with the j-th entry being the j-th coefficient (see Figure 7.1).



unnormalised $\mathsf{X} + c$ in $\mathfrak{R}_{n,k}$

Figure 7.1: Polynomial as a list of cofficients, with the least significant coefficient on the left.

A polynomial computation in this quotient ring $\mathfrak{R}_{n,k}$ will give a result as another unnormalised polynomial, *i.e.*, another list of coefficients with the same length k. The computation of an introspective check for a constant c proceed as (see Figure 7.2):

1. Start with a list representing the monomial $\mathsf{z} = \mathsf{X} + c$.
2. Prepare the final result with a list representing $\mathsf{q} = \mathsf{X}^n + c$ in $\mathfrak{R}_{n,k}$.
3. Manipulate the list `z` to mimic the computation of $\mathsf{p} = (\mathsf{X} + c)^n$ in $\mathfrak{R}_{n,k}$.
4. If the two lists are equal, *i.e.*, $\mathsf{p} = \mathsf{q}$, then this constant c passes the introsepctive check.

Table 7.1: Steps to perform introspective computations for unnormalised $\mathsf{X} + c$.

Before implementing the algorithm for introspective checks, we shall develop subroutines for unnormalised polynomials, tailored to computations in the quotient ring $\mathfrak{R}_{n,k}$.

7.4.1 Polynomial Equality

Two polynomials are equal if their coefficients are the same. Thus the equality check of two polynomials `p` and `q` can be implemented recursively in monadic style as `poly_eqM p q`, based on

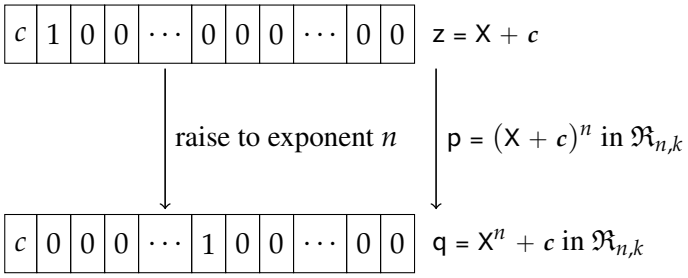


Figure 7.2: Polynomial introspective check: p = q in $\mathfrak{R}_{n,k} = \mathbb{Z}_n[X]/(X^k - 1)$.

primitives for lists (see Definition 123):

```
poly_eqM p q ≝
 do
  p0 ← nullM p;
  q0 ← nullM q;
  if p0 then return q0
  else if q0 then return p0
  else
   do
    h1 ← headM p;
    t1 ← tailM p;
    h2 ← headM q;
    t2 ← tailM q;
    e0 ← eqM h1 h2;
    if e0 then poly_eqM t1 t2 else return F
   od
 od
```

Our complexity analysis of this implementation gives these results:

$$\vdash \text{valueOf}\ (\text{poly\_eqM p q}) \iff \text{p} = \text{q}$$
$$\vdash \text{LENGTH p} = k \wedge \text{LENGTH q} = k \wedge \text{weak p} \wedge \text{weak q} \Rightarrow \text{stepsOf}\ (\text{poly\_eqM p q}) \leq 9(\text{max}\ 1\ k)(\text{size}\ n)$$

7.4.2 Polynomial Modular Addition

The unnormalised addition $\bigoplus_n$ of two polynomials in the quotient ring $\mathfrak{R}_{n,k}$ can be achieved by pairwise addition of coefficients in $(\text{mod}\ n)$:

$$p \oplus_n q \stackrel{\text{def}}{=} \text{MAP2}\ (\lambda\, x\, y.\ (x + y)\ \text{mod}\ n)\ p\ q$$

Implementing this in monadic style as poly_addM p q, the results are:

$$\vdash \mathsf{weak\ p} \wedge \mathsf{weak\ q} \wedge \mathsf{LENGTH\ p} = \mathsf{LENGTH\ q} \Rightarrow \mathsf{valueOf}\ (\mathsf{poly\_addM}\ n\ \mathsf{p\ q}) = \mathsf{p} \oplus_n \mathsf{q}$$
$$\vdash \mathsf{LENGTH\ p} = k \wedge \mathsf{LENGTH\ q} = k \wedge \mathsf{weak\ p} \wedge \mathsf{weak\ q} \Rightarrow$$
$$\mathsf{stepsOf}\ (\mathsf{poly\_addM}\ n\ \mathsf{p\ q}) \leq 12(\mathsf{max}\ 1\ k)(\mathsf{size}\ n)^2$$

7.4.3 Polynomial Modular Scalar Multiplication

In the quotient ring $\mathfrak{R}_{n,k}$, to multiply a polynomial by a scalar c, an operation denoted by $\odot_n$, corresponds to multiplying of each coefficient by c in $(\mathrm{mod}\ n)$:

$$c \odot_n p \stackrel{\mathsf{def}}{=} \mathsf{MAP}\ (\lambda x.\ cx\ \mathsf{mod}\ n)\ p$$

Implementing this in monadic style as poly_cmultM c p, the results are:

$$\vdash \mathsf{weak\ p} \wedge c < n \Rightarrow \mathsf{valueOf}\ (\mathsf{poly\_cmultM}\ n\ c\ \mathsf{p}) = c \odot_n \mathsf{p}$$
$$\vdash \mathsf{LENGTH\ p} = k \wedge \mathsf{weak\ p} \Rightarrow$$
$$\mathsf{stepsOf}\ (\mathsf{poly\_cmultM}\ n\ c\ \mathsf{p}) \leq 8(\mathsf{max}\ 1\ k)(\mathsf{size}\ (\mathsf{max}\ c\ n))(\mathsf{size}\ n)$$

7.4.4 Polynomial Modular Multiplication by X

The multiplication of an unnormalised polynomial of length k by X in $\mathfrak{R}_{n,k}$ is equivalent to shifting the coefficient of X^j to be the coefficient of X^{j+1}, for $0 \leq j < k$, with the coefficient of X^k to become the constant. We call this a *turn* for the polynomial:

$$\mathsf{turn\ p} \stackrel{\mathsf{def}}{=} \textbf{if}\ \mathsf{p} = [\,]\ \textbf{then}\ [\,]\ \textbf{else}\ \mathsf{LAST\ p::FRONT\ p}$$

Implementing this in monadic style as poly_turnM p, the results are:

$$\vdash \mathsf{valueOf}\ (\mathsf{poly\_turnM\ p}) = \mathsf{turn\ p}$$
$$\vdash \mathsf{LENGTH\ p} = k \Rightarrow \mathsf{stepsOf}\ (\mathsf{poly\_turnM\ p}) \leq 13(\mathsf{max}\ 1\ k)$$

7.4.5 Polynomial Modular Multiplication

Given an unnormalised polynomial q = h::t, the head h corresponds to the constant term, and the tail t the quotient of q divided by X (see Figure 7.1). Hence $\mathsf{q} = \mathsf{t} \times \mathsf{X} + h$, and the long multiplication of two polynomials p and q is based on this identity, with a partial result r:

$$\mathsf{p} \times \mathsf{q} + \mathsf{r} = \mathsf{p} \times (\mathsf{t} \times \mathsf{X} + h) + \mathsf{r} = \mathsf{p} \times \mathsf{X} \times t + (h \times \mathsf{p} + \mathsf{r})$$

Define a function $f(\mathsf{r}, \mathsf{p}, \mathsf{q}) = \mathsf{p} \times \mathsf{q} + \mathsf{r}$. The above identity can be expressed recursively:

$$f(\mathsf{r}, \mathsf{p}, \mathsf{q}) = f(h \times \mathsf{p} + \mathsf{r}, \mathsf{turn\ p}, \mathsf{t})$$

where turn p is the multiplication of p by X (see Section 7.4.4). Initialising the partial result r = zero_k, a list of zeros with length k, the unnormalised multiplication $\bigotimes_{n,k}$ of two polynomials p and q in $\mathfrak{R}_{n,k}$ is defined as:

$$\begin{array}{l}\mathsf{p} \bigotimes_{n,k} \mathsf{q} \stackrel{\mathsf{def}}{=} \mathsf{iterate}\ n\ \mathsf{zero}_k\ \mathsf{p}\ \mathsf{q}\\ \mathsf{iterate}\ n\ p_1\ p_2\ [\,] \stackrel{\mathsf{def}}{=} p_1\\ \mathsf{iterate}\ n\ p_1\ p_2\ (h{::}t) \stackrel{\mathsf{def}}{=} \mathsf{iterate}\ n\ (h \odot_n p_2 \oplus_n p_1)\ (\mathsf{turn}\ p_2)\ t\end{array}$$

Implementing this in monadic style as poly_multM p q, the results are:

$$\begin{array}{l}\vdash 0 < n \wedge \mathsf{weak\ p} \wedge \mathsf{weak\ q} \wedge \mathsf{q} \neq [\,] \wedge \mathsf{LENGTH\ p} = k \Rightarrow \mathsf{valueOf}\ (\mathsf{poly\_multM}\ n\ \mathsf{p}\ \mathsf{q}) = \mathsf{p} \bigotimes_{n,k} \mathsf{q}\\ \vdash 0 < n \wedge \mathsf{LENGTH\ p} = k \wedge \mathsf{LENGTH\ q} = k \wedge \mathsf{weak\ p} \wedge \mathsf{weak\ q} \Rightarrow\\ \quad \mathsf{stepsOf}\ (\mathsf{poly\_multM}\ n\ \mathsf{p}\ \mathsf{q}) \leq 37(\mathsf{max}\ 1\ k)^2(\mathsf{size}\ n)^2\end{array}$$

Squaring a polynomial is just a special case of multiplication:

$$\bigotimes^2_{n,k} \mathsf{p} \stackrel{\mathsf{def}}{=} \mathsf{p} \bigotimes_{n,k} \mathsf{p}$$

Implementing this in monadic style as poly_sqM p, the results are:

$$\begin{array}{l}\vdash 0 < n \wedge \mathsf{weak\ p} \wedge \mathsf{p} \neq [\,] \wedge \mathsf{LENGTH\ p} = k \Rightarrow \mathsf{valueOf}\ (\mathsf{poly\_sqM}\ n\ \mathsf{p}) = \bigotimes^2_{n,k} \mathsf{p}\\ \vdash 0 < n \wedge \mathsf{LENGTH\ p} = k \wedge \mathsf{weak\ p} \Rightarrow \mathsf{stepsOf}\ (\mathsf{poly\_sqM}\ n\ \mathsf{p}) \leq 37(\mathsf{max}\ 1\ k)^2(\mathsf{size}\ n)^2\end{array}$$

7.4.6 Polynomial Modular Exponentiation

Raising an unnormalised polynomial p to exponent j in $\mathfrak{R}_{n,k}$, denoted by $\mathsf{p}\ \Delta_{n,k}\ j$, is achieved by repeated use of modular squaring, $\bigotimes^2_{n,k}$:

$$\begin{array}{l}\mathsf{p}\ \Delta_{n,k}\ j \stackrel{\mathsf{def}}{=}\\ \quad \textbf{if}\ j = 0\ \textbf{then}\ [1]\\ \quad \textbf{else}\ (\textbf{let}\ \mathsf{q}\ =\ \bigotimes^2_{n,k}\ \mathsf{p}\ \Delta_{n,k}\ (j\ \mathsf{div}\ 2)\ \textbf{in if}\ \mathsf{EVEN}\ j\ \textbf{then}\ \mathsf{q}\ \textbf{else}\ \mathsf{p} \bigotimes_{n,k} \mathsf{q})\end{array}$$

Implementing this in monadic style as $\mathsf{poly\_expM}\ n\ \mathsf{p}\ j$, the results are:

$$\begin{array}{l}\vdash 1 < n \wedge \mathsf{weak\ p} \wedge \mathsf{p} \neq [\,] \wedge \mathsf{LENGTH\ p} = k \Rightarrow \mathsf{valueOf}\ (\mathsf{poly\_expM}\ n\ \mathsf{p}\ j) = \mathsf{p}\ \Delta_{n,k}\ j\\ \vdash 0 < n \wedge \mathsf{weak\ p} \wedge \mathsf{LENGTH\ p} = k \Rightarrow\\ \quad \mathsf{stepsOf}\ (\mathsf{poly\_expM}\ n\ \mathsf{p}\ j) \leq 83(\mathsf{max}\ 1\ k)^2(\mathsf{size}\ j)^2(\mathsf{size}\ n)^2\end{array}$$

7.4.7 Modular Introspective Checks

In Figure 7.2 we describe the polynomial computation of introspective checks. It is obvious that the initial $\mathsf{z} = \mathsf{X} + c$ has the form $\mathsf{X}^m + c$ with exponent $m = 1$. The final $\mathsf{q} = \mathsf{X}^n + c$ is also of the form $\mathsf{X}^m + c$, but with exponent $m = n \bmod k$, due to:

$$\vdash 1 < n \wedge 0 < k \Rightarrow \mathsf{X}^n\ +\ c\ \mathsf{mod}\ (\mathsf{X}^k\ -\ \mathbf{1}) = \mathsf{X}^{n\ \mathsf{mod}\ k}\ +\ c$$

Instead of building the first z and final q individually, we construct the general $\mathsf{X}^m + c$, which is an unnormalised polynomial in the quotient ring $\mathfrak{R}_{n,k}$, in monadic style by the subroutine poly_X_exp_addM $n\ k\ m\ c$. This subroutine takes the input exponent m and computes $j = m \bmod k$ internally to put 1 at the j-th coefficient, ensuring the result is in the quotient ring $\mathfrak{R}_{n,k}$. Using our analysis, the number of steps for the subroutine is given by:

$$\vdash 0 < n \Rightarrow \\ \quad \mathsf{stepsOf}\ (\mathsf{poly\_X\_exp\_addM}\ n\ k\ m\ c) \leq 34(\mathsf{max}\ 1\ k)(\mathsf{size}\ k)(\mathsf{size}\ m)(\mathsf{size}\ c)(\mathsf{size}\ n)^2$$

The initial z can be built as poly_X_exp_addM $n\ k\ 1\ c$, and the final q can be built as poly_X_exp_addM $n\ k\ n\ c$. With poly_expM $n\ z\ n$ to compute z^n in quotient ring $\mathfrak{R}_{n,k}$, the introspective check of $\mathsf{X} + c$ can be put into monadic style as poly_introM $n\ k\ c$, in accordance with the steps in Table 7.1:

```
poly_introM n k c ≝
 do
  z ← poly_X_exp_addM n k 1 c;
  p ← poly_expM n z n;
  q ← poly_X_exp_addM n k n c;
  poly_eqM p q
 od
```

we have these results by recurrence analysis:

$$\vdash 1 < n \wedge 1 < k \Rightarrow (\mathsf{valueOf}\ (\mathsf{poly\_introM}\ n\ k\ c) \iff n \overset{k}{\bowtie}_{\mathbb{Z}_n} (\mathsf{X} + \boldsymbol{c}))$$
$$\vdash 0 < n \Rightarrow \mathsf{stepsOf}\ (\mathsf{poly\_introM}\ n\ k\ c) \leq 160(\mathsf{max}\ 1\ k)^2(\mathsf{size}\ k)(\mathsf{size}\ c)(\mathsf{size}\ n)^4$$

In order to collect all introspective checks over a range of constants c from 1 to a limit s, *i.e.*, $1 \leq c \leq s$, we run poly_intro_rangeM $n\ k\ s$ recursively with a decreasing s down to 1, and calling the subroutine poly_introM $n\ k\ s$:

```
poly_intro_rangeM n k s ≝
 do
  c0 ← zeroM s;
  if c0 then return T
  else
   do
    b1 ← poly_introM n k s;
    if b1 then do j ← decM s; poly_intro_rangeM n k j od
    else return F
   od
 od
```

with these results:

$$\vdash 1 < n \wedge 1 < k \Rightarrow (\mathsf{valueOf}\ (\mathsf{poly\_intro\_rangeM}\ n\ k\ s) \iff \mathsf{poly\_intro\_range}\ \mathbb{Z}_n\ k\ n\ s)$$
$$\vdash 0 < n \Rightarrow$$
$$\mathsf{stepsOf}\ (\mathsf{poly\_intro\_rangeM}\ n\ k\ s) \leq 163(\max\ 1\ s)(\max\ 1\ k)^2(\mathsf{size}\ k)(\mathsf{size}\ s)(\mathsf{size}\ n)^4$$

It is remarkable that all the polynomial subroutines can be implemented in monadic style, and be simple enough for analysis by our elementary recurrence loop theory. There is no need to implemenat a general polynomial division algorithm to find the remainder, due to the special form of the modulus $\mathsf{X}^k - \mathbf{1}$ (see Section 7.4.4). This feature of the AKS algorithm is very convenient, and is the key to our simplistic implementation.

7.5 Complexity Analysis

Recall that the AKS algorithm from Definition 3 can be expressed in monadic style:

```
aksM n ≝
 do
  b ← power_freeM n;
  if b then
   do
    c ← paramM n;
    case c of
    | nice k ▷ eqM k n
    | good k ▷ poly_intro_rangeM n k k
    | bad ▷ return F
    od
  else return F
 od
```

(7.6)

Careful comparision this implementaton with the definition of aks n in Section 3.5 (page 50) shows a change in the introspective limit:

```
⊢ aks n ⟺
    power_free n ∧
    case aks_param n of
    | nice k ▷ k = n
    | good k ▷ poly_intro_range ℤ_n k n (SQRT φ(k)) ⌈log n⌉
    | bad ▷ F
```

(7.7)

In the definition of aks n, the introspective limit is $s = (\mathsf{SQRT}\ \varphi(k))\lceil \log n \rceil$. In our aksM n implementation (7.6), this limit is replaced by k. This is because $s \leq k$ (Theorem 68), and for the introspective checks of $\mathsf{X} + c$, a composite n will fail at some $c \leq s$ (Theorem 73), while a prime n will pass regardless of the value of c (Theorem 69). Indeed, we prove that this is valid:

Theorem 147. *The introspective limit of the AKS algorithm, treat as a function of its parameter k and input n, can be replaced by any value exceeding $s = (\mathsf{SQRT}\ \varphi(k))\lceil \log n \rceil$.*

$$\begin{array}{l}
\vdash (\forall k.\ 0 < k \wedge \lceil \log n \rceil^2 \leq \mathsf{order}_k(n) \Rightarrow (\mathsf{SQRT}\ \varphi(k))\lceil \log n \rceil \leq \mathsf{limit}\ k\ n) \Rightarrow \\
\quad (\mathsf{prime}\ n \iff \\
\quad\ \mathsf{power_free}\ n \wedge \\
\quad\ \textbf{case}\ \mathsf{aks_param}\ n\ \textbf{of} \\
\quad\ |\ \mathsf{nice}\ k \rhd k = n \\
\quad\ |\ \mathsf{good}\ k \rhd \forall c.\ 0 < c \wedge c \leq \mathsf{limit}\ k\ n \Rightarrow (\mathsf{X} + \boldsymbol{c})^n \equiv \mathsf{X}^n + \boldsymbol{c}\ (\mathrm{mod}\ n,\ \mathsf{X}^k - \mathbf{1}) \\
\quad\ |\ \mathsf{bad} \rhd \mathsf{F})
\end{array}$$

Therefore, this modification of the introspective limit keeps the algorithm as a primality test. It has an impact on the performance of the algorithm, but it simplifies the subsequent complexity analysis.

With all the required subroutines for each phase ready, we can establish our goal:

Theorem 148. *Our AKS implementation is correct, and shows the algorithm is polynomial-time.*

$$\begin{array}{c}
\vdash \mathsf{valueOf}\ (\mathsf{aksM}\ n) \iff \mathsf{aks}\ n \\
\vdash \mathsf{stepsOf} \circ \mathsf{aksM} \in \mathcal{O}(\lceil \log n \rceil^{21})
\end{array}$$

Proof. The correctness of the implementation aksM n in (7.6) follows from its resemblence to the aks n definition in (7.7), and the change of the introspective limit to k is justified by Theorem 147 due to $s \leq k$ (Theorem 68). The total number of steps for the AKS algorithm can be deduced from these results, established previously:

- Phase 1 Power Free Test, from Theorem 141:

$$\vdash \mathsf{stepsOf} \circ \mathsf{power\_freeM} \in \mathcal{O}(\lceil \log n \rceil^9)$$

- Phase 2 Parameter Search, from Theorem 146:

$$\vdash \mathsf{stepsOf} \circ \mathsf{paramM} \in \mathcal{O}(\lceil \log n \rceil^{20})$$

- Phase 3 Introspective Checks, we have:

$$\begin{array}{l}
\vdash 0 < n \Rightarrow \\
\quad \mathsf{stepsOf}\ (\mathsf{poly_intro_rangeM}\ n\ k\ s) \leq 163(\max\ 1\ s)(\max\ 1\ k)^2(\mathsf{size}\ k)(\mathsf{size}\ s)(\mathsf{size}\ n)^4
\end{array}$$

As explained before , our implementation takes $s = k$ to have poly_intro_rangeM $n\ k\ k$. By Corollary 66, the AKS parameter search never returns bad in Phase 2. A good k enters Phase 3, with $0 < k < n$. With these conditions on s and k, the bound on the number of steps of poly_intro_rangeM $n\ k\ s$ simplifies to:

$$\vdash s \leq k \wedge k < n \Rightarrow \mathsf{stepsOf}\ (\mathsf{poly\_intro\_rangeM}\ n\ k\ s) \leq 163(\max\ 1\ k)^3(\mathsf{size}\ n)^6$$

Thus the bound on the number of steps of poly_intro_rangeM $n\ k\ k$ becomes:

$$\mathsf{stepsOf}\ (\mathsf{poly\_intro\_rangeM}\ n\ k\ k) \leq 163k^3(\mathsf{size}\ n)^6 \tag{7.8}$$

The bounds on the parameter k are given by (from Corollary 66):

$$\vdash 2 < n \wedge \mathsf{param}\ n = \mathsf{nice}\ k \Rightarrow k \leq 1 + \lceil \log\ n \rceil^5\ \mathsf{div}\ 2$$
$$\vdash \mathsf{param}\ n = \mathsf{good}\ k \Rightarrow k \leq 1 + \lceil \log\ n \rceil^5\ \mathsf{div}\ 2$$

Substituting this bound for k, we have:

$$\mathsf{stepsOf}\ (\mathsf{poly\_intro\_rangeM}\ n\ k\ k) \leq 4075(\mathsf{size}\ n)^{21}. \tag{7.9}$$

Of the three phases in the algorithm, the number of steps is dominated by the introspective checks with $\mathcal{O}((\mathsf{size}\ n)^{21})$ in (7.9). Since $\mathcal{O}(\mathsf{size}\ n)$ equals $\mathcal{O}(\lceil \log\ n \rceil)$ by Corollary 121, the result follows. □

We have achieved the formalisation of the AKS algorithm. The resulting order $\mathcal{O}(\lceil \log\ n \rceil^{21})$ is rather high, when compared to the revised AKS paper showing an order of $\widetilde{\mathcal{O}}(\lceil \log\ n \rceil^{\frac{21}{2}})$ by similar analysis of each phase. We outline the reason for this discrepancy.

7.5.1 Complexity Review

In the revised AKS paper, the computational complexities for the introspective checks in Phase 3, which dominates all other phases, is $\widetilde{\mathcal{O}}(k^{\frac{3}{2}} \lceil \log\ n \rceil^3)$, where the parameter k is estimated to be $\mathcal{O}(\lceil \log\ n \rceil^5)$, giving the final result.

We deduced similarly that Phase 3 has a dominant order $\mathcal{O}(k^3 \lceil \log\ n \rceil^6)$ (refer to 7.8). Our exponents are twice of those in the AKS results. This is due to our multiplication subroutines, both for numbers and polynomials. The AKS results assume fast numerical multiplication of $\widetilde{\mathcal{O}}(\mathsf{size}\ n)$ for two n-bit numbers, and fast polynomial multiplication of $\widetilde{\mathcal{O}}(k(\mathsf{size}\ n))$ for two degree k polynomials with n-bit coefficients. We use long multiplication in both cases, and our results are $\mathcal{O}((\mathsf{size}\ n)^2)$ (Definition 122) and $\mathcal{O}((k(\mathsf{size}\ n))^2)$ (Section 7.4.5), respectively. Note that both exponents are twice of those derived from fast multiplications. We also have the parameter k estimated to be $\mathcal{O}(\lceil \log\ n \rceil^5)$ (Theorem 65), but the previous results ultimately give an overall complexity of order $\mathcal{O}(\lceil \log\ n \rceil^{21})$.

Improvements of the AKS algorithm to order $\mathcal{O}(\lceil \log\ n \rceil^6)$, but using Gaussian periods, is described in [Crandall and Pomerance, 2005, Section 4.5.3]. Based on the AKS idea, but using an introspective modulus of $X^k - \boldsymbol{b}$, leads to a randomized primality test essentially of order $\mathcal{O}(\lceil \log\ n \rceil^4)$, as discussed in [Crandall and Pomerance, 2005, Section 4.5.4].

Scripts The upper bounds for the steps of various subroutines are formulated in scripts located in the folder `aks/machine`. Each script analyses the number of steps for a certain class of subroutines, *e.g.*, parameter search in `countParam`, polynomials and introspective checks in `countPoly`, and AKS computations in `countAKS`.

7.6 Summary

We formalise "PRIMES is in P", by proving the correctness of an implemenation of the AKS algorithm, and establishing its computational complexity in the order of polynomial time. All the work has been done in the theorem-prover HOL4, backed up by various libraries we developed for general use in algebraic topics, and simple recurrence loop analysis. We are proud of our achievement, having formally verified a milestone in primality testing.

7.7 Remarks

The discussion of complexity analysis in the AKS paper occupies just half a page. A careful analysis of the complexity of the AKS algorithm, including various versions and improvements, is reported in Brent [2010].

Improvements of the AKS algorithm by Hendrik W. Lenstra, Jr. and Carl Pomerance, first published in Lenstra [2002], but updated as late as 2016, lower the effective time bound to $\mathcal{O}(\lceil \log\ n \rceil^6)$. Their method is based on Gaussian periods, which is more advanced than our theory of finite fields.

We present only a formal implementation of the AKS algorithm for the purpose of analysing its computational complexity. The implementation can run in HOL4, but only for very small numbers. Because the introspective checks are independent, they can be carried out in parallel. An investigation of a parallel implementation of the AKS algorithm is given in Bronder [2006].

Due to the high order $\widetilde{\mathcal{O}}(\lceil \log\ n \rceil^{\frac{21}{2}})$, the AKS algorithm is not a practical primality test. Many practical primality tests have been formalised, some even using theories of rings and finite fields, overlapping our work.

The formalisation of Lehmer's primality criterion by Simon Wimmer [2013] in Isabelle/HOL was based on ideals, ring homomorphisms, module and polynomials with coefficients from a ring. They included a proof that the multiplicative group of a finite field is cyclic (Theorem 28) from Euler's φ-function identity (Theorem 58).

A formal verification of the Miller-Rabin probabilistic primality test was carried out by Hurd [2003] in HOL. The work relied on the group structure of modulo arithmetic, both addition and multiplication, in a prime modulus. Although there was no explicit mention of fields, the two

groups are the main components of the finite field $\mathbb{Z}_p$ for prime p. A proof of the cyclic nature of the multiplicative group of $\mathbb{Z}_p$ (Theorem 28) was given.

In Coq, Théry and Hanrot delivered *Primality Proving with Elliptic Curves* at TPHOL 2007 (Théry and Hanrot [2007]). They formalised elliptic curves defined over a field, and Théry [2007] proved the group structure for "addition" of rational points on elliptic curves. The proof made use of basic properties polynomials with coefficients from a ring. Their subsequent work was based on the group law, and using prime certificates to derive a checker for elliptic curve certificates.

It is worth mentioning that Pepin's test for the primality of Fermat numbers was formalized by Fujisawa et al. [1998] in Mizar. Their work was purely number-theoretic, but did involve the role of order (see Section 2.2) for the invertibles in $\mathbb{Z}_n$.

Chapter 8

Conclusion

This conclusion examines the development of our work in formalising the AKS algorithm. We step back and take a look at what paths have been taken, what alternate paths are possible, and what lies ahead in our direction.

> We shall not cease from exploration,
> and the end of all our exploring
> will be to arrive where we started
> and know the place for the first time.
> — Thomas Stearns Eliot (1942)[1]

8.1 Overall Summary

We shall return to the starting dependency diagram (reproduced in Figure 8.1) to see how far we have explored the wonders of the AKS algorithm.

This thesis begins with the Introduction (Chapter 1), giving an overview. Starting with the 3-phase description of the AKS algorithm, we discuss the breakthrough by the AKS team, review the formalisation attempts by the theorem-proving community, formulate our main goals, and indicate a path for our formalisation. The path consists of 3 parts.

Part 1 is about laying the foundation for the AKS algorithm, with two chapters.

Chapter 2 covers the basic abstract algebraic structures essential to develop the concepts behind the operation of the AKS algorithm. The algebraic structures include monoids, groups, rings, fields, and integral domains. We go further to discuss finite fields, showing that it has a prime characteristic, and a cyclic multiplicative group, giving primitives for a finite field. We than apply the algebraic results to the ring $\mathbb{Z}_n$ of remainders modulus n. Useful properties of this ring are derived, including the Fermat's Little Theorem. We also prove several results from Number Theory, including a bound for the consecutive LCM which is useful for the size estimation of AKS parameter k.

Chapter 3 shows the pseudocode of the AKS algorithm, gives details of the power free test, and establishes the existence of the AKS parameter k. Then we show that the AKS algorithm is

[1] From "Little Gidding V", the fourth and final poem of his *Four Quartets*.

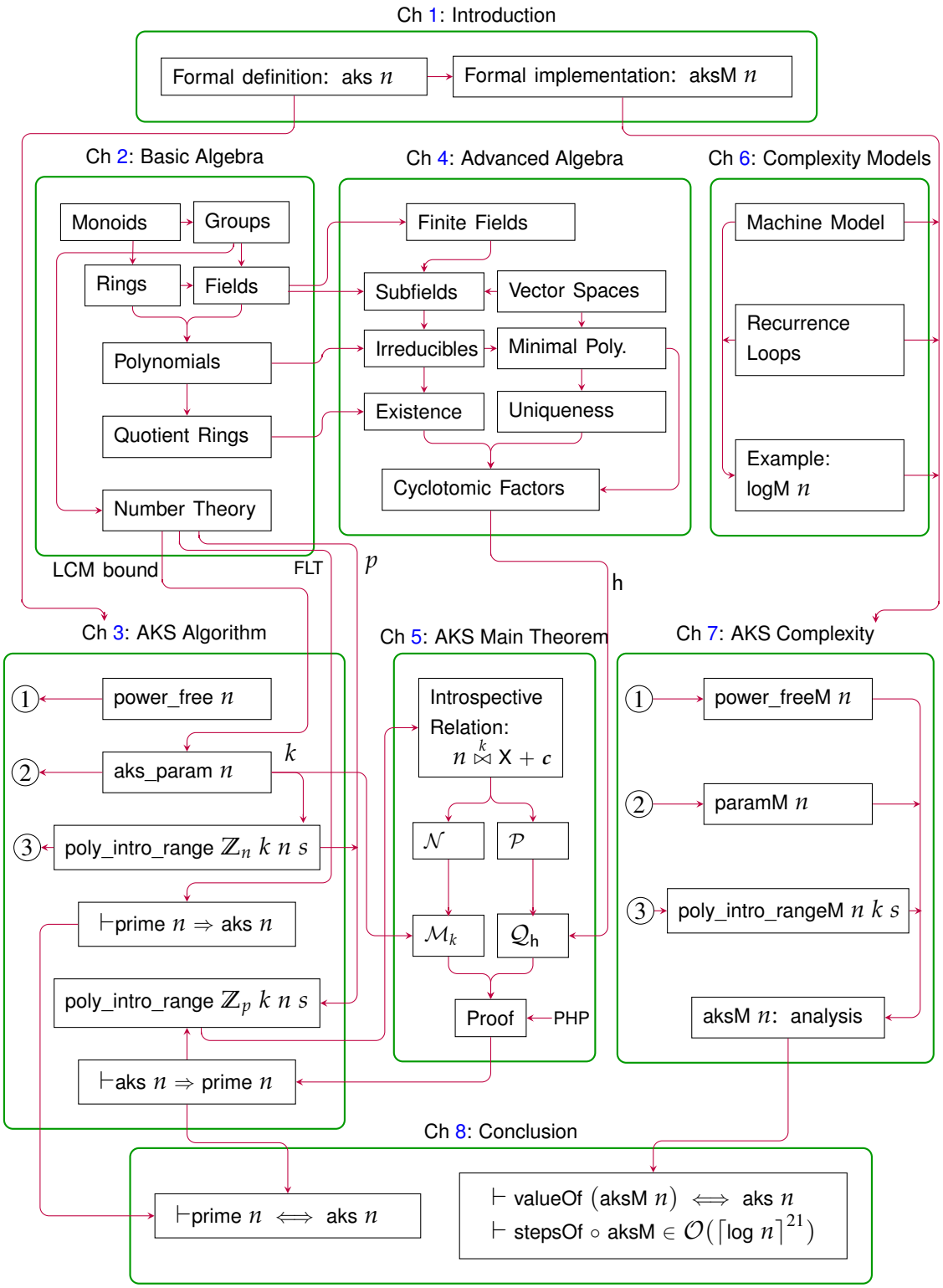


Figure 8.1: Dependency Diagram of Section 1.6, page 10.

indeed a primality test: it is easy to show that a prime must pass the tests of AKS, but to show the other direction requires shifting the introspective checks from the ring $\mathbb{Z}_n$ to a field $\mathbb{Z}_p$, where p is a special prime factor of n. This prepares for the AKS Main Theorem, which is formulated but not proved.

Part 2 is about the correctness proof of the AKS Main Theorem, with two chapters.

Chapter 4 gives the background necessary for the study of the AKS Main Theorem. These are the advanced properties of finite fields, including subfields, prime fields, irreducibles, and minimal polynomials. This leads to the existence of finite fields and uniqueness by cardinality up to isomorphism. This enables an investigation of the cyclotomic factors of $\mathsf{X}^k - \mathbf{1}$, the polynomial modulus of the AKS algorithm, ultimately providing an irreducible factor h giving a polynomial quotient field with the property that $\mathsf{order}_{\mathsf{h}}(\mathsf{X}) = k$.

Chapter 5 delivers the proof of the AKS Main Theorem, by first identifying sets involving introspective exponents and polynomials. The finite counterparts of these sets, under modulo k for exponents and modulo h for polynomials, respectively, lead to a study of their cardinalities. This in turn leads to a study of the injective maps between these finite sets, showing that if n can pass all the introspective checks then n must be a power of its prime factor p: otherwise one of the injective maps would violate the Pigeonhole principle. This conclusion by the AKS Main Theorem, together with the power free test in phase 1 and the existence of parameter from phase 2, implies that $n = p$.

Part 3 is about the computational complexity of the AKS algorithm, with two chapters.

Chapter 6 develops our tools: a machine model and a theory of recurrence. The machine model is based on a writer monad, to keep track of computations and accumulating step counts. We state the properties of the primitives for our machine: elementary arithmetic, simple boolean tests, and basic list operators. We build subroutines, and begin counting the number of steps for each. We note the patterns of recurrence in step counting of loops, and devise a simple recurrence theory to give upper bounds for the step counts.

Chpater 7 applies the techniques developed in Chapter 6 to analyse the AKS algorithm. We code each phase in monadic style as subroutines, and weave the subroutines together as an implementation of the AKS algorithm. We prove that the implementation accurately executes the AKS algorithm, and for an input n, the number of steps is bounded as $\mathcal{O}(\lceil \log n \rceil^{21})$.

This Conclusion (Chapter 8) conveys the final verdict: through our mechanisation of the AKS algorithm, "PRIMES is in P" is formally verified.

8.2 Formalisation Issues

Along the paths we have taken, we strive for elementary approaches and simple methods. We make choices, and encounter various issues about formalisation.

8.2.1 Simple Numbers

In this formalisation work, we use only natural numbers, avoiding any use of integers, rationals, real numbers and complex numbers. We make this choice despite the fact that HOL4 has well-established libraries beyond the basic natural numbers. The initial reason is to keep our work simple, but in the end it keeps our work elementary. There are two examples:

- *No Prime Number Theorem.*

  We establish the consecutive LCM bound in the existence proof of parameter k by the Leibniz Denominator Triangle (Section 2.7.4, Theorem 52). As noted in the remarks in Section 3.9, this result is equivalent to a weak form of the Prime Number Theorem (PNT). At the beginning of this work, we were not aware of this alternative. We had plans to either rework the formalisation of PNT [2] in HOL4, or just incorporate the consecutive LCM bound as a lemma, as was done in the AKS paper. The second choice will be unsatisfactory from a completeness point of view, but the first choice will suggest using the real libraries of HOL4. Our desire to use only natural numbers prompted us to look for an elementary approach to prove the consecutive LCM bound. That we eventually found the proof *via* the Leibniz Denominator Triangle is a big relief.

- *No use of calculus.*

  Besides the episode of the consecutive LCM bound just mentioned, there is one occasion where we almost have to use calculus: to show that $\mathsf{X}^k - \mathbf{1}$ has no repeated roots. This involves the idea of a derivative, treating $\mathsf{X}^k - \mathbf{1}$ as a function in X. After consultating the literature (*e.g.*, Lidl and Niederreiter [1986]), we found the use of formal deriviative of polynomials to solve this problem.

However, there are consequences for our insistence on using only natural numbers. Some of our theorems cannot be formulated in full generality:

- Although our theories of algebraic structures are formulated with a generic type α, either finite or infinite, we only work with finite instances of these structures. For example, we only proved that $\mathbb{Z}_n$ is a ring for $n > 0$ (Theorem 30). Since we do not use integers, rationals, real or complex numbers, we lack the ring $\mathbb{Z}$, and the fields $\mathbb{Q}, \mathbb{R}$, and $\mathbb{C}$. The multiplicative group of a field is cyclic, but we can prove only the version for finite fields (Theorem 28). Without rationals, this is beyond our reach: the prime subfield of a field with characteristic 0 is isomorphic to $\mathbb{Q}$. The finite field version is proved (Theorem 85).

- The complex field $\mathbb{C}$ has elements of every order, by the primitive complex roots of the cyclotomic equation. In a finite field, the order of elements are restricted (Theorem 27). This restriction is reflected in our formalisation of the cyclotomic polynomials, which is different from the textbook approach. See Section 4.4.1 for a thorough discussion.

[2] The Prime Number Theorem has been formalised by Avigad *et al.* in Isabelle Avigad et al. [2007] and independently by John Harrison in HOL Light Harrison [2009].

8.2.2 Simple Methods

Generally, if there are several approaches to formalise a topic, we choose the simplest one. Once an approach is committed but an obstacle appears, there is a dilemma: either abandon the approach and start over, or endure and find a way out to the very end. Yet the simple methods often keep our work elementary, as these examples show:

- *Algebraic structures as records*.

  In Section 2.1, we have described our formulation of the hierarchy of algebraic structures in HOL4. Our approach is based on records, *e.g.*, $\mathcal{G}$.carrier denotes the set of elements in a group $\mathcal{G}$. This is different from other group definitions already in HOL4 examples[3] using an ordered pair of a set G and a binary operator $(\times)$.

  Using records is convenient because HOL4 has a parsing/printing mechanism that allows additional functions to appear as if they are additional record fields. Indeed, we use this mechanism to give the group record a .inv field for the inverse of an element.

  So we cannot use the existing HOL4 libraries. We certainly cannot use algebra libraries from another theorem prover. Our work is to be done with minimal support, which means building a lot of libraries, especially algebraic ones, for this project from ground up. This approach has an advantage: our self-developed libraries are well-tuned for this project. We can keep reusing our own theorems, or polish our theorems to maximize reusability.

- *Polynomial as lists*.

  Polynomials consist of coefficients (Section 2.5, (2.1)), but theorems about polynomials are independent of the underlying representation. For simplicity, we just use a list of coefficients to represent a polynomial, since there is a standard list library in HOL4. This works fine until we come to define the degree of a polynomial. The constants, which are singletons, have degree 0. The zero polynomial $\mathbf{0}$, represented by the empty list $[]$, shall have degree -1. Since we are not using integers (see Section 8.2.1), we define the degree of $\mathbf{0}$ to be 0. The upshot is that for polynomial division in our library, a constant cannot be a divisor, as the remainder $\mathbf{0}$ does not have a degree strictly less than that of the divisor.

  Taking polynomials as lists means polynomial operations are defined recursively on lists. These definitions look at the head and tail of a list, which is a very local view, as oppose to the global view of a polynomial as a map of index to coefficients. As a result, it is a chore to prove polynomial addition and multiplication are commutative and associative. Moreover, keeping the intermediate polynomials always normalised is such a headache that we end up with first defining these operations over unnormalised polynomials, then normalise the final result.

  We thought the library of unnormalised polynomials should be hidden once the library of normalised polynomials is in full use, building theories of polynomial division, factors

[3] Joe Hurd et al. [2006] formalised elliptic curve cryptography based on a group structure in HOL.

and roots. It was a real surprise to find that the unnormalised polynomials are useful when implementing the introspective checks for the AKS algorithm (see Section 7.4). The list representation actually simplifies the implementation and analysis of polynomial algorithms, since the recursive nature of lists fits well with our recurrence loop pattern (see Table 6.3).

- *Complexity analysis by loops.*

  The analysis of the complexity of algorithms often lead to recurrence equations. There are many techniques to solve or extract bounds from such recurrences (see the remarks in Section 6.9), *e.g.*, the Akra-Bazzi theorem formalised by Manuel Eberl [2017]. We use none of them in Section 6.6. To estimate an upper bound on the number of steps, we just count the number of iterations in the recurrence loop, and multiply by the number of steps of the most time-consuming iteration (Theorems 129 and 129). As a result, our complexity results for various algorithms are far from the best.

  Our simple recurrence theory of loops means we are really restrictive in the choice of subroutines to use in a formal implementation of the AKS algorithm for analysis. There is no Euclidean algorithm to compute the greatest commond divisor. There is also no fast multiplication, and no Newton's method for root extraction.

  We emphasize that optimisation for performance is never our goal. Our aim is just to establish "PRIMES is in P", not the best algorithm for "PRIMES is in P". In fact, most of our algorithm implementations are plainly unoptimised. For example, in Section 7.2.1 our computation of b^n only checks $b^0 = 1$ for exit, not bothering also checking $b^1 = b$. This is not because we are lazy, but because a single equality-exit check fits well with our simple theory of recurrence loop (refer to guard x in Table 6.3).

8.2.3 Simple Types

We work within the simple types of the theorem prover HOL4. Every construct in a typed system has a type. This works fine to keep statements about constructs to make sense, but it causes difficulties during our formalisation of finite fields (Chapter 4). We find it hard to use extension field as freely as we would like. This point has been fully discussed in Section 4.3.3. This is not a weakness of HOL4, as this will happen for any typed system.

In traditional mathematics, the idea of the type of an extension field is very fluid. Take an example from the "Bible"[4]: *Finite Fields* by Lidl and Niederreiter [1997]. In section 2.2, the heading says, "Roots of irreducible polynomials", but a polynomial with a root is reducible. Of course, the Bible is serious. The irreducible ipoly h with coefficients from a field $\mathcal{F}$ has no root, but it forms a quotient field $\mathcal{E} = \mathcal{F}[X]/(h)$, and it is in this extension field $\mathcal{E}$ that the irreducible has a root.

However, for a typed system, the quotient field $\mathcal{E}$ and the original field $\mathcal{F}$ have different types. The irreducible h has only one type, so it cannot be a polynomial over both systems. The

[4] As referred to by an MAA reviewer, see https://www.maa.org/press/maa-reviews/lectures-on-finite-fields.

coefficients of h has to be lifted from $\mathcal{F}$ to $\mathcal{E}$ (each element in $\mathcal{F}$ becomes a constant in $\mathcal{E}$), and it is this lifted version of h that has a root in $\mathcal{E}$. One can thus appreciate the trouble in a typed system to straighten everything out.

We avoid these issues by sticking to a field/subfield pair, because both field and subfield are of the same type, and share the same operations: $(+)$ and $(\times)$. In the Bible, section 1.4 effectively says "$\mathcal{F}$ is called a subfield of $\mathcal{E}$, and $\mathcal{E}$ is called an extension of $\mathcal{F}$". There is a subtle difference for a typed system, which is carefully delineated in Section 4.3.4. Our resolution is awkward, but accurate, as shown in the statement of Theorem 90 about the existence of finite fields of an infinite type.

There is a flip side to this. Every finite field textbook reads like the Bible.[5] Because we are seeking an alternate route to formalise the classification theorems of finite fields, it is hard to find resources not following the Bible. In the end, we manage to complete our work on Chapter 4 by understanding the ideas in the lecture notes of Belk [2016].

8.3 Alternative Tactics

During our formalisation work, there are good times and bad times. We can forge ahead when the supporting libraries are in good order. When we are blocked, we try alternate paths. Some paths we have taken, some we trace but return empty-handed, and some we could have taken if we have time. In hindsight, even the failed attempts suggest the correct direction.

8.3.1 On Finite Fields

During the formalisation of finite fields leading to their classification (Chapter 4), we have pointed out (Section 8.2.3) that, due to type issues, we use a field/subfield pair instead of field extensions. The following is a list of topics along this less familiar path:

- *Subfield elements and polynomials.*

  Given a field/subfield pair $\mathcal{S} \preccurlyeq \mathcal{F}$, a subfield polynomial poly_S h is a polynomial h with all its coefficients in the subfield $\mathcal{S}$. For example, the minimal polynomial $\mathfrak{m}_a$ of an element $a \in$ F is a subfield polynomial (Section 4.3.1). The Frobenius map takes a field element x to $x^{|\mathsf{S}|}$. A field element is in the subfield if and only if it is fixed by the Frobenius map:

  $$\vdash \mathsf{FiniteField}\ \mathcal{F} \Rightarrow \forall \mathcal{S}.\ \mathcal{S} \preccurlyeq \mathcal{F} \Rightarrow \forall x.\ x \in \mathsf{F} \Rightarrow (x \in \mathsf{S} \iff x^{|\mathsf{S}|} = x)$$

  This is the basis of the following, to check if a field polynomial is a subfield polynomial:

  $$\vdash \mathsf{FiniteField}\ \mathcal{F} \Rightarrow \forall \mathcal{S}.\ \mathcal{S} \preccurlyeq \mathcal{F} \Rightarrow \forall \mathsf{h}.\ \mathsf{poly}\ \mathsf{h} \Rightarrow (\mathsf{poly}_S\ \mathsf{h} \iff \mathsf{h}^{|\mathsf{S}|} = \mathsf{h}[\![\mathsf{X}^{|\mathsf{S}|}]\!])$$

  We use the last criterion to establish the fact that the cyclotomic polynomials $\mathbf{\Phi}_n$ have integer coefficients (Section 4.4).

[5] Moreover, raising questions on finite fields on a math forum usually gets the answer, "refer to the Bible".

- *Conjugates and minimal polynomials.*

  With respect to a field/subfield pair $\mathcal{S} \preccurlyeq \mathcal{F}$, the successive images of a field element under the Frobenius map are its conjugates. The set of all conjugates of a field element a is denoted by $\mathsf{Conj}_{\mathcal{S}}\ a$:

$$\mathsf{Conj}_{\mathcal{S}}\ a \stackrel{\text{def}}{=} \{\ a^{|\mathsf{S}|^n}\ \mid\ n \in \mathcal{U}(:\mathtt{num})\ \}$$

  The minimal polynomial $\mathfrak{m}_a$ of an element a is a product of its conjugate factors:

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \wedge \mathcal{S} \preccurlyeq \mathcal{F} \Rightarrow \forall a.\ a \in \mathsf{F} \Rightarrow \mathfrak{m}_a = \prod \{\ \mathsf{X} - c \times \mathbf{1}\ \mid\ c \in \mathsf{Conj}_{\mathcal{S}}\ a\ \}$$

  Many properties of the minimal polynomial of a finite field element, *e.g.*, those given in Section 4.3.1, are established through this expression for the minimal polynomial.

- *Polynomials of special forms.*

  In Chapter 4, the equations 4.1, 4.2 and 4.4 are not isolated results. They are deductions from a study of polynomials of special forms: $\mathsf{X}^n - \mathsf{X}$ and $\mathsf{X}^n - \mathbf{1}$. For example, a polynomial of the first form with exponent $|\mathsf{F}|^d$ has all monic irreducibles of degree d as factors:

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \Rightarrow \forall \mathsf{h}.\ \mathsf{monic\ h} \wedge \mathsf{ipoly\ h} \Rightarrow \mathsf{h} \mid \mathsf{X}^{|\mathsf{F}|^{\deg \mathsf{h}}} - \mathsf{X}$$

  and polynomials of the second form have an interesting divisibility condition depending only on their exponents when the coefficients come from a non-trivial ring:

$$\vdash \mathsf{Ring}\ \mathcal{R} \wedge \mathbb{1} \neq \mathbb{0} \Rightarrow \forall n\ m.\ \mathsf{X}^n - \mathbf{1} \mid \mathsf{X}^m - \mathbf{1} \iff n \mid m$$

  This result provide a proof [6] to establish the classification of all subfields of a finite field:

$$\vdash \mathsf{FiniteField}\ \mathcal{F} \Rightarrow \forall n.\ (\exists \mathcal{S}.\ \mathcal{S} \preccurlyeq \mathcal{F} \wedge \deg(\mathcal{S}) = n) \iff n \mid \deg(\mathcal{F})$$

8.3.2 On Algebra

Finite fields are at the top of the hierarchy of algebraic structures (Section 2.1) in our library. Besides the change to field/subfield perspective in Section 8.3.1, there are other meanders that we go through to work out the whole hierarchy. We shall give a few examples.

- *Cyclic Multiplicative Group.*

  Based on the properties of orders of an element in a field (Theorem 27), and an identity of the Euler φ-function (Theorem 58), we have proved that the multiplicative group of finite field is cyclic (Theorem 28). At first we cannot show the Euler φ-function identity, so we get this result by another route. The proof [7] is based on the properties of a finite abelian group, which is the case for $\mathcal{F}^*$ of a finite field.

[6] Such a proof is given in [McEliece, 1987, Theorem 6.6], or [Ireland and Rosen, 1990, Proposition 7.1.5].
[7] Such a proof is given in [Justesen and Høholdt, 2004, Theorem 2.1.2].

Let $\mathcal{G}$ be a finite abelian group. An element $a \in \mathsf{G}$ has an order, since the sequence $a, a^2, a^3, \ldots$ must repeat. Of all the order of elements in $\mathcal{G}$, the maximum is called the maximal order of the finite group, denoted by $\mathsf{maximal\_order}\ \mathcal{G}$. We can show that the order of any group element must divide the maximal order:

$$\vdash \mathsf{FiniteAbelianGroup}\ \mathcal{G} \Rightarrow \forall x.\ x \in \mathsf{G} \Rightarrow \mathsf{order}_{\mathcal{G}}(x) \mid \mathsf{maximal\_order}\ \mathcal{G}$$

Combining this result with the fact that a nonzero polynomial with coefficients from a field has the number of its roots bounded by its degree (Theorem 25), the maximal order must be equal to the cardinality of the group carrier G. Therefore, in a finite field $\mathcal{F}$, the element with the maximal order in its multiplicative group $\mathcal{F}^*$ is a generator, giving $\mathsf{cyclic}\ \mathcal{F}^*$.

- *Ideals, Quotient Rings and Fields.*

  We have briefly mentioned these entities in Section 4.2. We develope a whole theory around these concepts, and establish some useful results. For example, denoting a ring and an ideal pair by $\mathcal{I} \prec \mathcal{R}$, we prove that the quotient ring $\mathcal{R}\ \mathsf{div}\ \mathcal{I}$ is indeed a ring:

  $$\vdash \mathsf{Ring}\ \mathcal{R} \wedge \mathcal{I} \prec \mathcal{R} \Rightarrow \mathsf{Ring}\ (\mathcal{R}\ \mathsf{div}\ \mathcal{I})$$

  We define $\mathsf{EuclideanRing}\ \mathcal{R}\ f$ for a ring $\mathcal{R}$ equipped with a norm function f, so that both the quotient and remainder of division can be determined through the norm. We also define $\mathsf{PrincipalIdealRing}\ \mathcal{R}$ for a ring $\mathcal{R}$ where every ideal is generated by a ring element. We prove that:

  $$\vdash \mathsf{EuclideanRing}\ \mathcal{R}\ f \Rightarrow \mathsf{PrincipalIdealRing}\ \mathcal{R}$$

  Let $\mathcal{R} \simeq_f \mathcal{S}$ denote that rings $\mathcal{R}$ and $\mathcal{S}$ have a homomorphism f. Then the kernel of f is an ideal of $\mathcal{R}$:

  $$\vdash \mathcal{R} \simeq_f \mathcal{S} \Rightarrow \mathsf{kernel}\ f\ \mathcal{R}\ \mathcal{S} \prec \mathcal{R}$$

  We also define the notion of a maximal ideal:

  $$\mathsf{maximal}\ \mathcal{R}\ \mathcal{I} \stackrel{\mathsf{def}}{=} \mathcal{I} \prec \mathcal{R} \wedge \forall \mathcal{J}.\ \mathcal{I} \prec \mathcal{J} \wedge \mathcal{J} \prec \mathcal{R} \Rightarrow \mathcal{I} = \mathcal{J} \vee \mathcal{J} = \mathcal{R}$$

  and prove that a quotient ring is a field if and only if the ideal is maximal:

  $$\vdash \mathsf{Ring}\ \mathcal{R} \Rightarrow \forall \mathcal{I}.\ \mathcal{I} \prec \mathcal{R} \wedge \mathsf{Field}\ (\mathcal{R}\ \mathsf{div}\ \mathcal{I}) \iff \mathcal{I} \neq \mathcal{R} \wedge \mathsf{maximal}\ \mathcal{R}\ \mathcal{I}$$

  This result is the basis of our original proof [8] of Theorem 81: the quotient ring by an irreducible is a quotient field because the ideal by an irreducible is maximal.

- *Finite Field Isomorphism.*

  Our first formalisation of the uniqueness of finite field (Theorem 88) is to make an effort to

[8] Our proof follows the approach given in [Herstein, 1996, Theorem 4.5.11].

construct an explicit isomorphism between two finite fields of equal cardinality. Although the explicit isomorphic map is clumsy to express, it is essentially the chain of isomorphic maps (4.3) given before the end of the proof given on page 65.

8.3.3 On Combinatorics

There are several occasions where we need some counting results for our theorems. We only use bijection between finite sets and the rudimentary product rule. We could have develop a library for combinatorics, but we are able to work around in each case.

- *Möbius Inversion*

  In Section 4.2.1, we prove the existence of monic irreducible polynomials of any positive degree Theorem 82 by a counting argument. Most textbooks[9] arrive at the same result by another route. After deducing Equation 4.1, the usual method is to equate the polynomial degree on the left with the sum of counts of monic irreducibles on the right. By applying Möbius inversion, one can extract the individual monic irreducibles counts, and show that these counts are nonzero. Our proof avoids the use of Möbius inversion formula.

  Note that Möbius inversion had been formalised, *e.g.*, by Asperti and Armentano [2008]. In that paper, the authors offered a proof of an identity for Euler's φ-function

  $$\sum_{d \mid n} \varphi(d) = n$$

  based on Möbius inversion. We use instead a counting argument (Theorem 58, page 38).

- *Counting Monomial Products*

  In Section 5.5, when estimating a lower bound for the cardinality of $\mathcal{Q}_{\mathsf{h}}$ (Theorem 109), we would like to count the number of monomial products with the following form:

  $$\mathsf{X}^{e_0}(\mathsf{X}+1)^{e_1}\ldots(\mathsf{X}+c)^{e_c}\ldots(\mathsf{X}+s)^{e_s}$$

  where the exponents are nonnegative, satisfying:

  $$e_0 + e_1 + \cdots + e_c \cdots + e_s < |\mathcal{M}_k|$$

  Using the technique "stars and bars"[10], the exact count is: $\binom{|\mathcal{M}_k|+s}{|\mathcal{M}_k|-1}$. In this case, we did not attempt to obtain this exact count, as the advantage is minimal (see the remarks in Section 5.9).

The theory of combinatorics is an active area of research, with many interesting results and applications. Some results depend on the theory of finite partially ordered sets (posets), an

[9] See, *e.g.*, Lidl and Niederreiter [1986], Ireland and Rosen [1990], and McEliece [1987].
[10] Popularized by William Feller in his classic book on probability (Feller [1968]).

algebraic structure that could be developed from our abstract algebra library. Refer to Singh [2017] for recent works on formalisation of combinatorics in Coq.

8.4 Future Work

As one session draws to a close, another begins. This thesis presents a lot of ideas and viewpoints, and covers a lot of ground. There are more topics for exploration. We list 3 possible directions.

8.4.1 AKS Variations

There is a good discussion of AKS improvements in Bedodi [2010], including Berrizbeitia's Algorithms. The idea is simple, and interesting.

It is clear there is only one even prime, and Euclid[11] showed that there are infinitely many odd primes. The AKS algorithm is effectively testing if an odd n is prime, since any even n will return a factor 2 at Phase 2 (see pseduocode in Algorithm 1).

Odd numbers have only two forms: either $4n + 1$ or $4n + 3$, for $n = 0, 1, 2, 3, \ldots$. Since there are an infinite number of primes in an arithmetic progression by Dirichlet's theorem,[12] the primes for each form is infinite. Only a prime of the first form can be the hypotenuse of an integer right triangle, by Fermat's Two Square Theorem.[13] Thus primes of the first form are called Pythagorean primes, and primes of the second form are non-Pythagorean primes.

Due to additional properties for Pythagorean and non-Pythagorean primes, adapting the AKS algorithm to these number families results in some novel modifications. These have investigated by Pedro Berrizbeitia [2005], producing two versions of AKS-like primality tests: one for each form of odd numbers.

A formalisation of Berrizbeitia's Algorithms is possible by extending our existing library. There are other ideas from Berrizbeitia et al. [2004] concerning primality tests, so there should be plenty of venues to explore in this area.

8.4.2 Complexity Improvements

In Section 7.5.1, we review the discrepancy between our AKS complexity order $\mathcal{O}(\lceil \log n \rceil^{21})$ with $\widetilde{\mathcal{O}}(\lceil \log n \rceil^{\frac{21}{2}})$ in the AKS paper. To improve our complexity bounds, we might work on the formalisation fast multiplication. For example, Karatsuba's polynomial multiplication has been implemented in Coq by Dénès et al. [2012].

It would be nice if the elementary steps in our machine model (see Section 6.3) could be implemented in terms of some low-level operations. When this is done, the number of steps in Definitions 122, 124 and 123 can be derived rather than defined. There are two ways to proceed:

[11] The product of all prime up to some maximum is even, adding 1 becomes odd.

[12] Dirichlet's theorem has been formalised by John Harrison [2010] in HOL Light, and Mario Carneiro [2016] in Metamath.

[13] This is #20 in the list of Formalizing 100 Theorems, at http://www.cs.ru.nl/~freek/100/.

- *Using Machine Codes*

  It is possible to define instruction sets for execution, memory cells for storage, and use separation logic to formally verify machine-code programs. Research work in this area has been carried out by Myreen [2008], with a published book Myreen [2012].

- *Using a Turing Machine*

  Turing Machines are the definitive models of computation, but it is a challenge to code simple arithmetic algorithms for a Turing Machine. Nevertheless, formalisation work on Turing Machines being done; one by Asperti and Ricciotti [2012] in Matita Theorem Prover, and another using separation logic by Xu et al. [2013] in Isabelle/HOL.

We might also borrow some ideas from Asperti [2013], using a reverse methodological approach to obtain a formal treatment of Complexity Theory at a comfortable level of abstraction and logical rigor.

8.4.3 Library Applications

Every formalisation project takes time to accomplish, and the time and effort being spent is crystallised as the development of an extensive collection of supporting libraries. Besides being utilised for the original project, these libraries often find application in other projects.

Our library structure is documented in Section A.2:

- the aks folder (Section A.2.1) for the AKS development,
- the algebra folder (Section A.2.2) for our algebra library, and
- the algorithm folder (Section A.2.3) for complexity analysis.

The AKS development have been thoroughly discussed in Chapters 3 and 5. The rudimentary algorithm library can be enhanced as suggested in Section 8.4.2.

Our algebra libraries contain enhancement to existing HOL libraries for numbers, sets, and lists, as well as the hierarchy of algebraic structures introduced in Chapter 2. Part of the HOL improvements have been incorported to current HOL releases, and the algebraic structures have been extended to include finite fields, as described in Chapter 4, to support the improvements in our AKS work (see Section 5.1). Our group library include subgroups, normal subgroups, group homomorphisms, and group actions. We have proved the Orbit-Stabilizer Theorem, which has been used to formalise a necklace proof of Fermat's Little Theorem (see Chan and Norrish [2013]).

Algebra libraries in other theorem provers (see Section 4.6) have been the foundation of further formalisation of algebraic topics. For example, the formal construction of real algebraic numbers in Coq by Cyril Cohen [2012], and the formalisation in Isabelle/HOL of Berlekamp-Zassenhaus factorization by Jose Divasón et al. [2017]. In Mizar, there is work on commutative algebra by Rudnicki et al. [2001], and lattice theory by Järvinen [2007].

We might make use of our abstract algebra libraries and do further researches on finite fields. Already in an undergraduate research project, a student has made use of our group library to prove the Correspondence Theorem.[14] We could use our algebra library to establish the theory of Galois Correspondence, thus completing the formalisation of the Fundamental Theorem of Galois Theory, the current progress of which has been mentioned in Section 4.6.

8.5 Afterword

To formalise is to understand, in detail. De Bruijn challenged us to explain the mathematics to a machine. Galois reminded us to study in order to seek the truth. We have been pursuing an understanding of the AKS algorithm from theory to implementation, finding the beauty at heart.

A Milestone

Beauty is truth,
truth beauty.
— John Keats (1819)[15]

Beauty is in the eye of the beholder, but the following are definitely true:

$$\begin{aligned}
&\vdash \mathsf{prime}\ n \iff \mathsf{aks}\ n \\
&\vdash \mathsf{valueOf}\ (\mathsf{aksM}\ n) \iff \mathsf{aks}\ n \\
&\vdash \mathsf{stepsOf} \circ \mathsf{aksM} \in \mathcal{O}(\lceil \log n \rceil^{21})
\end{aligned}$$

They have been formalised, with a verification by the theorem-prover HOL4. It is a joy to witness these results coming out from the theorem-prover with the verdict "goal proved." The beauty of formalisation is this: you can always rerun the scripts through the theorem-prover, and relive the joy and excitement of the first time through.

These results mark a milestone in mechanisation: a formal proof of "PRIMES in P". In 2006, the AKS team was awarded the Gödel Prize for their "simple and elementary" 2004 paper of the AKS algorithm. Now this thesis adds a finishing touch with our elementary formalisation of the AKS algorithm.

[14] This is the bijection between the set of all subgroups of group $\mathcal{G}$ containing a normal subgroup $\mathcal{N}$, and the set of all subgroups of the quotient group $\mathcal{G}$ div $\mathcal{N}$. The proof is contributed by Yiming Xu.

[15] From the ending lines of his poem, *Ode on a Grecian Urn*.

Appendix

A.1 Script References

This is a table listing the actual names of theorems in the repository scripts.[16] The interactive scripts have suffix `.hol`, and the HOL4 scripts have suffix `.sml`. The location of a script is described by a folder path. Refer to Section A.2 for the organisation of the folders.

This thesis is prepared under the repository tag `phd-thesis-02`, and is compiled using `Holmakefile` with HOL4 version tag `6711e409`.

Location of ...	Script	Name
Definition 1	`aks/theories/AKSclean`	`aks_def`
Theorem 2	`aks/theories/AKSclean`	`aks_thm`
Definition 3	`aks/machine/countAKS`	`aksM_def`
Theorem 4	`aks/machine/countAKS`	`aksM_steps_big_O`
Definition 5	`algebra/monoid/monoid`	`Monoid_def`
Definition 6	`algebra/group/group`	`group_def_by_inverse`
Definition 7	`algebra/ring/ring`	`Ring_def`
Definition 8	`algebra/field/field`	`Field_def`
Definition 9	`algebra/monoid/monoidOrder`	`order_alt`
Theorem 10	`algebra/monoid/monoidOrder`	`monoid_order_divides_exp`
Theorem 11	`algebra/monoid/monoidOrder`	`monoid_order_power_eqn`
Definition 12	`algebra/group/subgroup`	`Subgroup_def`
Theorem 13	`algebra/group/subgroup`	`subgroup_thm`
Theorem 14	`algebra/group/subgroup`	`Lagrange_thm`
Theorem 15	`algebra/group/groupOrder`	`generated_group_card`
Theorem 16	`algebra/group/groupOrder`	`group_order_divides`
Theorem 17	`algebra/group/groupOrder`	`finite_group_Fermat`

[16] Repository location: http://bitbucket.org/jhlchan/hol/src/. Names are hyperlinked to the actual script location.

Definition 18	algebra/ring/ring	char_def
Theorem 19	algebra/ring/ringUnit	ring_units_group
Definition 20	algebra/ring/integralDomain	IntegralDomain_def
Theorem 21	algebra/field/field	finite_integral_domain_is_field
Theorem 22	algebra/polynomial/polyFieldModulo	poly_distinct_irreducibles_mod_exp_eq_zero
Theorem 23	algebra/polynomial/polyDivides	poly_root_factor_thm
Lemma 24	algebra/polynomial/polyRoot	poly_roots_count_id
Theorem 25	algebra/polynomial/polyRoot	poly_roots_count
Theorem 26	algebra/field/field	finite_field_char
Theorem 27	algebra/finitefield/ffUnity	field_orders_card
Theorem 28	algebra/finitefield/ffAdvanced	finite_field_mult_group_cyclic
Definition 29	algebra/field/fieldOrder	field_orders_primitive
Theorem 30	algebra/ring/ringInstances	ZN_char
Theorem 31	algebra/ring/ringInstances	ZN_to_ZN_ring_homo
Definition 32	algebra/group/groupInstances	Estar_alt
Theorem 33	algebra/ring/ringInstances	ZN_coprime_order_divides_phi
Theorem 34	algebra/group/groupInstances	Euler_Fermat_alt
Corollary 35	algebra/group/groupInstances	Fermat_little_eqn
Theorem 36	algebra/lib/binomial	prime_iff_divides_binomials
Corollary 37	algebra/lib/binomial	prime_iff_divides_binomials_alt
Theorem 38	algebra/group/groupInstances	Fermat_little_eqn
Theorem 39	algebra/polynomial/polyBinomial	poly_ZN_freshman_fermat

Theorem 40	algebra/polynomial/polyBinomial	poly_freshman_thm, poly_freshman_thm_sub
Theorem 41	algebra/polynomial/polyBinomial	poly_ZN_prime_identity
Definition 42	algebra/lib/triangle	list_lcm_def
Lemma 43	algebra/lib/triangle	list_lcm_lower_bound_alt
Theorem 44	algebra/lib/triangle	list_lcm_nonempty_lower_alt
Theorem 45	algebra/lib/binomial	binomial_horizontal_sum
Definition 46	algebra/lib/triangle	leibniz_def
Theorem 47	algebra/lib/triangle	leibniz_horizontal_average_eqn
Theorem 48	algebra/lib/triangle	leibniz_triplet_property
Theorem 49	algebra/lib/triangle	leibniz_triplet_lcm
Definition 50	algebra/lib/triangle	leibniz_vertical, leibniz_up, leibniz_horizontal
Theorem 51	algebra/lib/triangle	leibniz_lcm_property
Theorem 52	algebra/lib/triangle	leibniz_vertical_lcm_lower
Definition 53	algebra/lib/logPower	perfect_power_def
Theorem 54	algebra/lib/logPower	perfect_power_cofactor_alt
Theorem 55	algebra/lib/logPower	perfect_power_condition
Theorem 56	algebra/lib/helperFunction	coprime_factor_coprime
Theorem 57	algebra/ring/ringInstances	ZN_order_gt_1_property
Theorem 58	algebra/lib/Gauss	Gauss_little_thm
Definition 59	algebra/lib/logPower	power_free_def
Theorem 60	algebra/lib/logPower	prime_is_power_free
Theorem 61	algebra/lib/logPower	power_free_test_upto_ulog

Definition 62	aks/compute/computeParam	aks_param_alt
Theorem 63	aks/compute/computeParam	aks_param_nice
Theorem 64	aks/compute/computeParam	aks_param_good_coprime_order
Theorem 65	aks/compute/computeParam	ZN_order_modulus_exists_4
Corollary 66	aks/compute/computeParam	aks_param_exists
Theorem 67	aks/compute/computeParam	aks_param_nice_for_prime
Theorem 68	aks/theories/AKSclean	aks_param_good_intro_limit_alt
Theorem 69	aks/theories/AKSintro	ZN_intro_range_by_prime
Theorem 70	algebra/theories/AKSclean	aks_thm_1
Theorem 71	aks/theories/AKSshift	ring_homo_intro_range_ZN_to_ZN_X_add_c_alt
Definition 72	aks/theories/AKSclean	aks_criteria_def
Theorem 73	algebra/theories/AKSclean	aks_thm_2
Definition 74	algebra/linear/VectorSpace	VSpace_def
Theorem 75	algebra/finitefield/ffAdvanced	field_is_vspace_over_subfield
Theorem 76	algebra/finitefield/ffBasic	prime_field_minimal_subfield
Theorem 77	algebra/finitefield/ffAdvanced	finite_field_card_alt
Theorem 78	algebra/finitefield/ffExist	finite_field_card_eq_prime_power
Theorem 79	algebra/field/fieldMap	subfield_char
Theorem 80	algebra/field/fieldInstances	ZN_finite_field
Theorem 81	algebra/polynomial/polyFieldModulo	poly_mod_irreducible_finite_field
Theorem 82	algebra/finitefield/ffMaster	poly_monic_irreducible_exists_alt
Theorem 83	algebra/finitefield/ffExist	finite_field_card_prime_power_exists
Theorem 84	algebra/finitefield/ffConjugate	poly_master_eq_poly_minimal_product_alt

Theorem 85	algebra/finitefield/ffExist	finite_field_isomorphic_ZN_PF_alt
Theorem 86	algebra/finitefield/ffExist	finite_field_prime_field_isomorphic
Theorem 87	algebra/finitefield/ffExtend	poly_mod_ring_by_primitives_element_field_isomorphism
Theorem 88	algebra/finitefield/ffExist	finite_field_eq_card_isomorphic
Lemma 89	algebra/finitefield/ffExist	A_LIST_BIJ_A
Theorem 90	algebra/finitefield/ffExist	finite_field_existence
Theorem 91	algebra/polynomial/polyFieldModulo	poly_field_mod_lift_has_root_X_alt
Definition 92	algebra/finitefield/ffCyclo	poly_cyclo_alt
Theorem 93	algebra/finitefield/ffUnity	poly_cyclo_deg_eqn
Theorem 94	algebra/finitefield/ffUnity	poly_unity_eq_poly_cyclo_product_alt
Theorem 95	algebra/finitefield/ffExist	poly_unity_eq_poly_prod_image_cyclo_num
Theorem 96	algebra/finitefield/ffExist	poly_unity_special_factor_exists_alt
Definition 97	aks/theories/AKSintro	poly_intro_def
Theorem 98	aks/theories/AKSintro	poly_intro_X_add_c_finite_field
Theorem 99	aks/theories/AKSintro	poly_intro_mult
Theorem 100	aks/theories/AKSintro	poly_intro_compose
Definition 101	aks/theories/AKSsets	setN_def, setP_def
Theorem 102	aks/theories/AKSintro	poly_intro_X_add_c_prime_char_3
Corollary 103	aks/theories/AKSsets	setN_has_char_and_cofactor
Theorem 104	aks/theories/AKSsets	modN_card_bounds_better
Theorem 105	algebra/polynomial/polyDivision	poly_mod_eq_divisor
Theorem 106	algebra/polynomial/polyFieldModulo	poly_mod_field_exp_eq_0

Theorem 107	aks/theories/AKSmaps	setP_poly_modN_divisor_eq
Theorem 108	aks/theories/AKSmaps	reduceP_mod_modP_inj_1
Theorem 109	aks/theories/AKSmaps	modP_card_lower_1
Theorem 110	aks/theories/AKSmaps	modN_card_in_exp_lt_bound_3_alt
Theorem 111	aks/theories/AKSsets	reduceN_element_upper_better
Theorem 112	aks/theories/AKSmaps	setN_mod_eq_gives_modP_roots
Theorem 113	aks/theories/AKSmaps	reduceN_mod_modN_inj_2
Theorem 114	aks/theories/AKSsets	reduceN_card_field
Theorem 115	algebra/theories/AKSclean	aks_main_punch
Definition 116	aks/machine/countMonad	bind_alt
Definition 117	algorithm/lib/bitsize	size_alt
Definition 118	algebra/lib/logPower	ulog_def_alt
Theorem 119	algorithm/lib/bitsize	size_by_ulog_alt
Definition 120	algorithm/lib/complexity	big_O_def
Corollary 121	algorithm/lib/complexity	big_O_size_eq_big_O_ulog
Definition 122	aks/machine/countMacro	addM_def, subM_def, mulM_def
Definition 123	aks/machine/countMacro	nullM_def, headM_def, tailM_def
Definition 124	aks/machine/countMacro	eqM_def, notM_def, boolM_def
Definition 125	aks/machine/countMacro	zeroM_def, oneM_def
Theorem 126	algorithm/lib/bitsize	ulog_by_size_eqn
Definition 127	algorithm/loop/loop	loop_count_thm
Theorem 128	algorithm/loop/loop	loop_modify_count_eqn_1
Theorem 129	algorithm/loop/loop	loop_rise_count_cover_exit_le

Theorem 130	algorithm/loop/loop	loop_fall_count_cover_exit_le
Theorem 131	algorithm/loop/loopList	loop_list_count_cover_exit_le
Definition 132	algorithm/loop/loop	loop2_count_def
Theorem 133	algorithm/loop/loop	loop2_rise_fall_count_cover_exit_le
Theorem 134	algorithm/loop/loop	loop2_fall_rise_count_cover_exit_le
Theorem 135	aks/machine/countBasic	sizeM_steps_big_O
Theorem 136	aks/machine/countBasic	power_ofM_steps_big_O
Theorem 137	aks/machine/countBasic	ulogM_steps_big_O
Theorem 138	aks/machine/countPower	expM_steps_big_O
Definition 139	algebra/lib/logPower	ROOT_EVAL
Theorem 140	aks/machine/countPower	rootM_steps_big_O
Theorem 141	aks/machine/countPower	power_freeM_steps_big_O
Theorem 142	aks/machine/countModulo	mexpM_steps_big_O
Definition 143	algebra/ring/ringInstances	ZN_order_test_4
Theorem 144	aks/compute/computeOrder	ordz_compute_eqn
Theorem 145	aks/machine/countOrder	ordzM_steps_big_O
Theorem 146	aks/machine/countParam	paramM_steps_big_O
Theorem 147	aks/theories/AKSclean	aks_variation_thm
Theorem 148	aks/machine/countAKS	aksM_steps_big_O

Locations of Definition, Lemma, Theorem, and Corollary.

A.2 Script Libraries

The repository library is well-organised and well-documented. The following is assembled from various `README` files. The proof scripts make up a complete HOL4[17] development for the thesis: mechanisation of the AKS Algorithm.

A.2.1 The *aks* folder

`/aks` — the theoretical basis and computational aspects of the AKS algorithm.

- `/theory` — for the theoretical basis of the AKS Main Theorem.
  - `/AKSintro`, introspective relation essential to the AKS proof.
  - `/AKSshift`, shifting introspective relation between rings.
  - `/AKSsets`, sets involved in he AKS proof.
  - `/AKSmaps`, mappings involved in the AKS proof.
  - `/AKStheorem`, the main theorem in the AKS proof.
  - `/AKSrevised`, revise the AKS proof for a general AKS parameter.
  - `/AKSimproved`, improve the bound constants in the AKS proof.
  - `/AKSclean`, a clean rewrite of the AKS proof.
- `/compute` — for a computational study of the AKS algorithm.
  - `/computeInteger`, computations of $\lceil \log n \rceil$, the round-up value of $\log_2 n$.
  - `/computeBasic`, computations of exponentials, root, powers and integer roots.
  - `/computeOrder`, computation of the modular multiplicative order.
  - `/computeParam`, computation of an AKS parameter.
  - `/computePoly`, polynomial computations with modulus $X^k - 1$.
  - `/computeRing`, modulo polynomial computations in ring $\mathbb{Z}_n$.
  - `/computeAKS`, polynomial checks and all part of the AKS algorithm.
- `/machine` — for a computational complexity analysis of AKS algorithm.
  - `/countMonad`, a monad to track computational values and counts.
  - `/countMacro`, macro operations as subroutines in monadic style.
  - `/countBasic`, basic arithmetic functions in monadic style.
  - `/countModulo`, modulo arithmetic in monadic style.

[17] Avaliable at http://bitbucket.org/jhlchan/hol/src/, repository tag `phd-thesis-02`. This thesis is compiled using `Holmakefile` with HOL4 version tag `6711e409`. Folders are hyperlinked to the actual repository location.

- `/countPower`, exponentiation, root extraction and power free test.
- `/countPrime`, traditional primality testing algorithm.
- `/countOrder`, modular exponentiation and multiplicative order.
- `/countParam`, parameter search for AKS algorithm.
- `/countPoly`, polynomial algorithmes for AKS algorithm.
- `/countAKS`, combine with polynomial checks for AKS algorithm.

A.2.2 The *algebra* folder

`/algebra` — various algebra topics to support the theory behind the AKS algorithm.

- `/lib` — the extension of existing HOL libraries.
  - `/helperNum`, more theorems on arithmetic, divisibility, and `GCD` and `LCM`.
  - `/helperSet`, more theorems about sets: mappings, sums and products.
  - `/helperList`, more theorems about list manipulations, *e.g.*, `turn`.
  - `/helperFunction`, theorems on function equivalence.
  - `/binomial`, properties of Pascal's triangle, for binomial coefficients.
  - `/triangle`, properties of Leibniz's Denominator triangle, for consecutive `LCM`.
  - `/Euler`, properties of number-theoretic sets, and Euler's φ-function.
  - `/Gauss`, more theorems about coprimes, φ-function, and an identity.
  - `/primes`, properties of two-factors, and a primality test.
  - `/primePower`, properties of prime powers and divisors, for consecutive `LCM`.
  - `/logPower`, properties of perfect power, power free, and upper logarithm.
  - `/sublist`, order-preserving sublist and properties.
- `/monoid` — an algebraic structure with a useful binary operation.
  - `/monoid`, properties of a monoid with identity.
  - `/monoidInstances`, examples of monoids, in particular modulo numbers.
  - `/monoidOrder`, order of element in a monoid and its properties.
  - `/monoidMap`, homomorphism and isomorphism between monoids.
  - `/submonoid`, properties of submonoids of a monoid.
- `/group` — a monoid with all its elements invertible.
  - `/group`, properties of group, as an invertible monoid.
  - `/groupInstances`, examples of groups, in particular prime modulo systems.

  - `/groupOrder`, extends monoidOrder to groupOrder.
  - `/groupCyclic`, theorems on cyclic group and generators.
  - `/subgroup`, theorems on subgroups, including cosets.
  - `/quotientGroup`, theorems of a group partitioned by a normal subgroup.
  - `/groupMap`, homomorphism and isomorphism between groups.
  - `/congruences`, application to number theory, mainly Fermat's Little Theorem.
  - `/symmetryGroup`, the group of all permutations of a set of symbols.
  - `/finiteGroup`, properties of a finite group.
  - `/groupAction`, action of a group on a target.

- `/ring` — an algebraic structure with two related binary operations.
  - `/ring`, properties of a ring, made up of a group and a monoid.
  - `/ringInstances`, examples of rings, in particular modulo arithmetic systems.
  - `/ringUnit`, properties of invertible elements in a ring (they form a group).
  - `/ringDivides`, theorems of division for invertible elements in a ring.
  - `/ringIdeal`, the concept of a well-behaved sub-structure of a ring.
  - `/quotientRing`, theorems of a ring partitioned by an ideal.
  - `/ringMap`, homomorphism and isomorphism between rings.
  - `/ringInteger`, the similarity of the multiples of ring unity and the integers.
  - `/ringBinomial`, the binomial theorem in terms of ring integers.
  - `/integralDomain`, properties of integral domain, a non-trivial ring without zero divisors.
  - `/integralDomainInstances`, examples of integral domains, *e.g.*, arithmetic in modulo prime.

- `/field` — a nontrivial ring with all nonzero elements invertible.
  - `/field`, properties of a field, as an invertible (zero excluded) ring.
  - `/fieldInstances`, examples of fields, mainly arithmetic in prime modulus.
  - `/fieldIdeal`, extends the ringIdeal and quotientRing to fields.
  - `/fieldBinomial`, extends the ring binomial theorem to fields.
  - `/fieldOrder`, multiplicative order of elements in a field.
  - `/fieldMap`, homomorphism and isomorphism between fields.
  - `/fieldProduct`, product of a set of field elements.
  - `/symmetryField`, the group of field automorphisms fixing a subfield.

- `/polynomial` — made up of coefficients taken from a ring or a field.
  - `/polyWeak`, properties of un-normalized polynomials.
  - `/polynomial`, properties of normalized polynomials.
  - `/polyRing`, theorems for polynomials with coefficients from a ring.
  - `/polyField`, theorems for polynomials with coefficients from a field.
  - `/polyDivision`, theorems for polynomial division in general.
  - `/polyFieldDivision`, theorems for field polynomial division.
  - `/polyRingModulo`, theorems for ring polynomial division remainders.
  - `/polyFieldModulo`, theorems for field polynomial division remainders.
  - `/polyRoot`, properties of polynomial roots and factors.
  - `/polyMonic`, properties of monic polynomials.
  - `/polyEval`, polynomial evaluation, acting as a function.
  - `/polyBinomial`, binomial theorem for polynomials.
  - `/polyDivides`, divisibility of polynomials.
  - `/polyIrreducible`, properties of irreducible polynomials.
  - `/polyDerivative`,formal derivative for polynomials, symbolic term by term.
  - `/polyProduct`, product of polynomials, properties, evaluation, and divisibility.
  - `/polyCyclic`, properties of the quotient of $\mathsf{X}^n - \mathbf{1}$ by $\mathsf{X} - \mathbf{1}$.
  - `/polyGCD`, greatest common divisor of polynomials.
  - `/polyMultiplicity`, multiplicity of polynomial factors and roots.
  - `/polyMap`, maps between polynomials, under homomorphism or isomorphism of their coefficient rings or fields.
- `/linear` — the study of linear spaces.
  - `/VectorSpace`, properties of vector space from its scalars and vectors.
  - `/SpanSpace`, represents elements by linear combination of basis vectors.
  - `/LinearIndep`, theorems for change of basis and linear independence.
  - `/FiniteVSpace`, dimension over subspace and linear independent basis.
- `/finitefield` — an intricate structural theory due to its finite nature.
  - `/ffBasic`, properties of the prime subfield of a finite field.
  - `/ffAdvanced`, extend a finite field by an irreducible polynomial.
  - `/ffInstances`, examples of finite field construction, *e.g.*, $\mathbb{GF}_4$.
  - `/ffPoly`, subring polynomials and roots of subfield polynomials.

- `/ffMaster`, master polynomials, relationship with irreducible polynomials of a subfield.
- `/ffCyclo`, cyclotomic polynomials, the order of its roots.
- `/ffUnity`, roots of unity and the number of elements of each field order.
- `/ffMinimal`, minimal polynomials, its existence by linear independence, and its properties.
- `/ffConjugate`, conjugates of field elements, their orders; also the product of conjugate factors.
- `/ffExist`, the classification of finite fields: existence and uniqueness.
- `/ffExtend`, field extension by isomorphic subfield.
- `/ffSplit`, splitting field of a field polynomial.

A.2.3 The *algorithm* folder

`/algorithm` — computational complexity of algorithms to analyse the AKS algorithm.

- `/lib` — common library for basic notions of complexity.
  - `/bitsize`, bit size of number representations.
  - `/complexity`, big-$\mathcal{O}$ notation for complexity class.
  - `/recurrence`, recurrence theory for step counting loops.
- `/loop` — library for patterns of loop recurrence.
  - `/loop`, general theory of loop recurrence, with body and exit.
  - `/loopIncrease`, recurrence theory of increasing loops.
  - `/loopDecrease`, recurrence theory of decreasing loops.
  - `/loopDivide`, recurrence theory of dividing loops.
  - `/loopMultiply`, recurrence theory of multiplying loops.
  - `/loopList`, recurrence theory of list reduction loops.

Bibliography

AARONSON, S., 2003. The Prime Facts: From Euclid to AKS. (cited on page 1)

ACZEL, P., 1995. Galois: A Theory Development Project. Technical report, Manchester University, U.K. Available from http://www.cs.man.ac.uk/~petera/galois.ps.gz. (cited on page 73)

AFFELDT, R.; GARRIGUE, J.; AND SAIKAWA, T., 2016. Formalization of Reed-Solomon codes and progress report on formalization of LDPC codes. In *The 2016 International Symposium on Information Theory and Its Applications (ISITA 2016)*, 532–536. (cited on page 40)

AGRAWAL, M.; KAYAL, N.; AND SAXENA, N., 2002. PRIMES is in P. Original paper. (cited on pages 1, 2, 55, 76, and 90)

AGRAWAL, M.; KAYAL, N.; AND SAXENA, N., 2004. PRIMES is in P. *Annals of Mathematics*, 160, 2 (September 2004), 781–793. (cited on pages xix, 1, 2, 31, 44, 49, 52, 55, 76, and 90)

ARNESON, B.; BAAZ, M.; AND RUDNICKI, P., 2003. Witt's Proof of the Wedderburn Theorem. *Journal of Formalized Mathematics*, 15 (December 2003). (cited on page 73)

ARNESON, B. AND RUDNICKI, P., 2003. Primitive Roots of Unity and Cyclotomic Polynomials. *Journal of Formalized Mathematics*, 15 (December 2003). (cited on page 73)

ASPERTI, A., 2009. A survey on Iteractive Theorem Proving. http://cs.unibo.it/~asperti/SLIDES/itp.pdf. Talk given at the Tata Institute of Technology, Mumbai, India. (cited on page 9)

ASPERTI, A., 2013. A Formal Proof of Borodin-Trakhtenbrot's Gap Theorem. In *Certified Programs and Proofs*, vol. 8307 of *Lecture Notes in Computer Science*, 163–177. Springer International Publishing, Cham. doi:10.1007/978-3-319-03545-1_11. https://link.springer.com/chapter/10.1007/978-3-319-03545-1_11. (cited on page 142)

ASPERTI, A. AND ARMENTANO, C., 2008. A Page in Number Theory. *Journal of Formalized Reasoning*, 1, 1 (2008), 1–23. doi:10.6092/issn.1972-5787/385. https://jfr.unibo.it/article/view/385. (cited on pages 40 and 140)

ASPERTI, A. AND AVIGAD, J., 2011. Zen and the art of formalisation. *Mathematical Structures in Computer Science*, 21, 4 (August 2011), 679–682. doi:10.1017/S0960129511000065. (cited on page 1)

ASPERTI, A. AND RICCIOTTI, W., 2008. About the Formalization of Some Results by Chebyshev in Number Theory. In *Types for Proofs and Programs, International Conference, TYPES 2008, Torino, Italy, March 26-29, 2008, Revised Selected Papers*, 19–31. doi:10.1007/978-3-642-02444-3_2. https://doi.org/10.1007/978-3-642-02444-3_2. (cited on page 55)

ASPERTI, A. AND RICCIOTTI, W., 2012. Formalizing Turing Machines. In *Logic, Language, Information and Computation*, 1–25. Springer Berlin Heidelberg, Berlin, Heidelberg. doi:10.1007/978-3-642-32621-9_1. (cited on page 142)

AVIGAD, J. AND DONNELLY, K., 2004. Formalizing O Notation in Isabelle/HOL. In *Automated Reasoning*, vol. 3097 of *Lecture Notes in Computer Science*, 357–371. Springer Berlin Heidelberg, Berlin, Heidelberg. doi:10.1007/978-3-540-25984-8_27. https://link.springer.com/chapter/10.1007/978-3-540-25984-8_27. IJCAR 2004. (cited on page 111)

AVIGAD, J.; DONNELLY, K.; GRAY, D.; AND RAFF, P., 2007. A Formally Verified Proof of the Prime Number Theorem. *ACM Transactions on Computational Logic*, 9, 1 (December 2007). doi:10.1145/1297658.1297660. (cited on pages 55 and 134)

AVIGAD, J. AND HARRISON, J., 2014. Formally Verified Mathematics. *Communications of the ACM*, 57, 4 (April 2014), 66–75. doi:10.1145/2591012. (cited on page 9)

AXLER, S., 2015. *Linear Algebra Done Right*. Undergraduate Texts in Mathematics. Springer International Publishing. ISBN 9783319307657. doi:10.1007/978-3-319-11080-6. ISBN: 9783319307657. (cited on page 60)

BAILEY, A., 1998. *The Machine-Checked Literate Formalisation of Algebra in Type Theory*. Ph.D. thesis, Department of Computer Science, University of Manchester. (cited on page 73)

BALLARIN, C.; KAMMÜLLER, F.; AND PAULSON, L. C., 2016. The Isabelle/HOL Algebra Library. Available from http://isabelle.in.tum.de/library/HOL/HOL-Algebra/index.html. (cited on page 73)

BANCEREK, G.; BYLIŃSKI, C.; GRABOWSKI, A.; KORNIŁOWICZ, A.; MATUSZEWSKI, R.; NAUMOWICZ, A.; AND PĄK, K., 2018. The Role of the Mizar Mathematical Library for Interactive Proof Development in Mizar. *Journal of Automated Reasoning*, 61, 1 (June 2018), 9–32. doi:10.1007/s10817-017-9440-6. https://doi.org/10.1007/s10817-017-9440-6. (cited on page 73)

BARTHE, G., 1994. A formal proof of the unsolvability of the symmetric group over a set with five or more elements. Technical report, University of Nijmegen, the Netherlands. Available from ftp://ftp.cs.ru.nl/pub/CompMath.Found/sn.ps.Z. (cited on page 73)

BARTZIA, E.-I. AND STRUB, P.-Y., 2014. A Formal Library for Elliptic Curves in the Coq Proof Assistant. In *Interactive Theorem Proving: 5th International Conference, ITP*

2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings, ITP 2014 (Vienna, Austria, 2014), 77–92. Springer International Publishing, Cham. doi:10.1007/978-3-319-08970-6_6. https://doi.org/10.1007/978-3-319-08970-6_6. (cited on page 40)

BASTIDA, J. R. AND LYNDON, R., 1984. *Field Extensions and Galois Theory*. Encyclopedia of Mathematics and its Applications. Cambridge University Press. ISBN 9781107340749. doi:10.1017/CBO9781107340749. ISBN: 9781107340749. (cited on page 70)

BEDODI, A., 2010. *Primality Tests in Polynomial Time*. Master's thesis, Universitá Degli Studi Roma TRE. (cited on page 141)

BELK, J., 2016. Classification of Finite Fields. *Number Theory Course: Math 318, Bard College*, (Spring 2016). Available from http://faculty.bard.edu/belk/math318/ClassificationFiniteFieldsRevised.pdf. (cited on pages 62, 64, 65, 71, and 137)

BERNSTEIN, D. J., 2002. An Exposition of the Agrawal-Kayal-Saxena Primality Proving Theorem. (cited on pages 1 and 91)

BERRIZBEITIA, P., 2005. Sharpening "Primes Is in P" for a Large Family of Numbers. *Mathematics of Computation*, 7, 252 (October 2005), 2043–2059. https://www.jstor.org/stable/4100226. (cited on page 141)

BERRIZBEITIA, P.; MÜLLER, S.; AND WILLIAMS, H. C., 2004. Pseudocubes and Primality Testing. In *Algorithmic Number Theory*, vol. 3076 of *Lecture Notes in Computer Science*, 102–116. Springer Berlin Heidelberg, Berlin, Heidelberg. doi:10.1007/978-3-540-24847-7_7. (cited on page 141)

BORNEMANN, F., 2003. PRIMES Is in P: A Breakthrough for Everyman. *Notices of the AMS*, 50, 5 (May 2003), 545–552. (cited on page 2)

BRENT, R. P., 2000. Twenty years' analysis of the Binary Euclidean Algorithm. In *Millenial Perspectives in Computer Science*, Proceedings of the 1999 Oxford-Microsoft Symposium in Honour of Professor Sir Antony Hoare, 41–53. (cited on page 45)

BRENT, R. P., 2010. Primality Testing. Technical report, Mathematical Sciences Institute, ANU. Available from https://maths-people.anu.edu.au/~brent/pd/comp4600_primality.pdf. (cited on page 129)

BRONDER, J. S., 2006. *The AKS Class of Primality Tests: A Proof of Correctness and Parallel Implemenation*. Master's thesis, The University of Maine. (cited on page 129)

CAMPOS, C.; MODAVE, F.; AND ROACH, S., 2004. Towards the Verification of the AKS Primality Test in ACL2. Fifth International Conference on Intelligent Technologies. (cited on pages 7 and 51)

CARNEIRO, M., 2016. Formalization of the prime number theorem and Dirichlet's theorem. *CICM 2016, FMM track*, (August 2016). https://arxiv.org/abs/1608.02029. (cited on page 141)

CHAN, H. L. AND NORRISH, M., 2012. A String of Pearls: Proofs of Fermat's Little Theorem. In *Proceedings of Certified Programs and Proofs*, no. 7679 in LNCS, 188–207. Springer. doi:10.1007/978-3-642-35308-6_16. (cited on page 40)

CHAN, H. L. AND NORRISH, M., 2013. A String of Pearls: Proofs of Fermat's Little Theorem. *Journal of Formalized Reasoning*, 6, 1 (December 2013), 63–87. doi:10.6092/issn.1972-5787/3728. (cited on pages 40 and 142)

CHAN, H. L. AND NORRISH, M., 2015. Mechanisation of AKS Algorithm: Part 1 — the Main Theorem. In *Interactive Theorem Proving, ITP 2015*, no. 9236 in LNCS, 117–136. Springer. doi:10.1007/978-3-319-22102-1_8. (cited on pages 7, 76, and 90)

CHAN, H. L. AND NORRISH, M., 2016. Proof Pearl: Bounding Least Common Multiples with Triangles. In *Interactive Theorem Proving, ITP 2016*, no. 9807 in LNCS, 140–150. Springer. doi:10.1007/978-3-319-43144-4_9. (cited on page 36)

CHAN, H. L. AND NORRISH, M., 2019a. Classification of Finite Fields with Applications. *Journal of Automated Reasoning*, 63, 3 (October 2019), 667–693. doi:10.1007/s10817-018-9485-1. (cited on page 72)

CHAN, H. L. AND NORRISH, M., 2019b. Proof Pearl: Bounding Least Common Multiples with Triangles. *Journal of Automated Reasoning*, 62, 2 (February 2019), 171–192. doi:10.1007/s10817-017-9438-0. (cited on pages 31, 40, and 55)

CHARGUÉRAUD, A. AND POTTIER, F., 2015. Machine-Checked Verification of the Correctness and Amortized Complexity of an Efficient Union-Find Implementation. In *Conference on Interactive Theorem Proving (ITP)*, vol. 9236 of *Lecture Notes in Computer Science*, 137–153. Springer. http://gallium.inria.fr/~fpottier/publis/charguer aud-pottier-uf.pdf. (cited on page 111)

CHARGUÉRAUD, A. AND POTTIER, F., 2017. Verifying the Correctness and Amortized Complexity of a Union-Find Implementation in Separation Logic with Time Credits. *Journal of Automated Reasoning*, (September 2017). doi:10.1007/s10817-017-9431-7. https://link.springer.com/article/10.1007/s10817-017-9431-7. (cited on page 111)

COHEN, C., 2012. Construction of Real Algebraic Numbers in Coq. In *Interactive Theorem Proving, ITP 2012*, no. 7406 in LNCS, 67–82. Springer. (cited on pages 73 and 142)

CRANDALL, R. AND POMERANCE, C., 2005. *Prime Numbers: A Computational Perspective*. Springer. ISBN 9780387252827. (cited on pages 1, 55, 90, 91, and 129)

CURIEL, N., 2011. *Formalizing Galois Theory: I automorphism groups of fields*. Master's thesis, California State University San Marcos. Available from http://csusm-dspace.calstate.edu/handle/10211.8/107. (cited on page 73)

CURIEN, P.-L. (Ed.), 2011. *Interactive Theorem Proving and the Formalisation of Mathematics*. Mathematical Structures in Computer Science. Cambridge University Press. (cited on page 9)

DALESON, G., 2006. *Deterministic Primality Testing in Polynomial Time*. Master's thesis, Portland State University. (cited on pages 55, 90, and 91)

DE MOURA, F. L. C. AND TADEU, R., 2008. The Correctness of the AKS Primality Test in Coq. Available from http://www.cic.unb.br/~flavio/AKS.pdf. (cited on pages 7 and 51)

DÉNÈS, M.; MÖRTBERG, A.; AND SILES, V., 2012. A Refinement-Based Approach to Computational Algebra in Coq. In *Interactive Theorem Proving*, 83–98. Springer Berlin Heidelberg, Berlin, Heidelberg. doi:10.1007/978-3-642-32347-8_7. (cited on page 141)

DIETZFELBINGER, M., 2004. *Primality Testing in Polynomial Time: From Randomized Algorithms to 'PRIMES is in P'*. Lecture Notes in Computer Science. Springer. ISBN 9783540403449. (cited on pages 1, 40, 90, and 91)

DIVASÓN, J.; JOOSTEN, S.; THIEMANN, R.; AND YAMADA, A., 2017. A Formalization of the Berlekamp-Zassenhaus Factorization Algorithm. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, CPP 2017 (Paris, France, 2017), 17–29. ACM, New York, NY, USA. doi:10.1145/3018610.3018617. http://doi.acm.org/10.1145/3018610.3018617. (cited on pages 40 and 142)

EATCS, 2006. Gödel Prize, by European Association for Theoretical Computer Science. http://eatcs.org/index.php/component/content/article/502. 33rd International Colloquium on Automata, Languages and Programming, ICALP 2006. (cited on page 3)

EBERL, M., 2017. Proving Divide and Conquer Complexities in Isabelle/HOL. *Journal of Automated Reasoning*, 58, 4 (April 2017), 483–508. doi:10.1007/s10817-016-9378-0. https://doi.org/10.1007/s10817-016-9378-0. (cited on pages 111 and 136)

FELLER, W., 1968. *An Introduction to Probability Theory and Its Applications*, vol. 1 of *Wiley Series in Probability and Statistics*. Wiley, 3 edn. ISBN 978-0-471-25708-0. (cited on page 140)

FUJISAWA, Y.; FUWA, Y.; AND SHIMIZU, H., 1998. Public-Key Cryptography and Pepin's Test for the Primality of Fermat Numbers. *Journal of Formalized Mathematics*, 10 (December 1998). (cited on page 130)

FUTA, Y.; OKAZAKI, H.; AND SHIDAMA, Y., 2013. Formalization of Definitions and Theorems Related to an Elliptic Curve Over a Finite Prime Field by Using Mizar. *Journal of*

Automated Reasoning, 50, 2 (Febuary 2013), 161–172. doi:10.1007/s10817-012-9265-2. https://doi.org/10.1007/s10817-012-9265-2. (cited on page 40)

GALOIS, É., 1830. On the theory of numbers. *Bulletin des Sciences Mathématiques, Physiques et Chimiques*, 13 (June 1830), 428–435. Reprinted by Liouville in 1846, English translation in Neumann [2013], pages 61–75. (cited on page 15)

GARRETT, P. B., 2004. *The Mathematics of Coding Theory: Information, Compression, Error Correction, and Finite Fields*. Pearson Prentice Hall, Upper Saddle River, NJ. ISBN 9780131019676. ISBN: 9780131019676. (cited on page 70)

GONTHIER, G.; ASPERTI, A.; AVIGAD, J.; BERTOT, Y.; COHEN, C.; GARILLOT, F.; LE ROUX, S.; MAHBOUBI, A.; O'CONNOR, R.; OULD BIHA, S.; PASCA, I.; RIDEAU, L.; SOLOVYEV, A.; TASSI, E.; AND THÉRY, L., 2013. A Machine-Checked Proof of the Odd Order Theorem. In *Interactive Theorem Proving: 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings*, 163–179. Springer Berlin Heidelberg, Berlin, Heidelberg. doi:10.1007/978-3-642-39634-2_14. https://doi.org/10.1007/978-3-642-39634-2_14. (cited on page 73)

GRANVILLE, A., 2004. It is easy to determine whether a given integer is prime. *Bulletin of the American Mathematical Society*, 42, 1 (September 2004), 3–38. (cited on page 40)

GUÉNEAU, A., 2017. Formal proofs of asymptotic complexity. http://gallium.inria.fr/~agueneau/talks/masterclass_philippa_gardner.pdf. (cited on page 111)

GUÉNEAU, A.; CHARGUÉRAUD, A.; AND POTTIER, F., 2018. A Fistful of Dollars: Formalizing Asymptotic Complexity Claims via Deductive Program Verification. In *European Symposium on Programming (ESOP)*, vol. 10801 of *Lecture Notes in Computer Science*, 533–560. Springer International Publishing, Cham. doi:10.1007/978-3-319-89884-1_19. https://link.springer.com/chapter/10.1007/978-3-319-89884-1_19. (cited on page 111)

HARRISON, J., 1996. Formalized Mathematics. http://www.cl.cam.ac.uk/~jrh13/papers/form-math3-times.ps.gz. Technical Report 36, Turku Centre for Computer Science (TUCS). (cited on page 9)

HARRISON, J., 2002. Real Numbers in Real Applications. https://www.cl.cam.ac.uk/~jrh13/slides/tpholsr-19aug02/slides.pdf. Theorem Proving in Higher Order Logics, 15th International Conference. (cited on page 6)

HARRISON, J., 2009. Formalizing an analytic proof of the Prime Number Theorem. *Journal of Automated Reasoning*, 43 (2009), 243–261. (cited on pages 55 and 134)

HARRISON, J., 2010. A formalized proof of Dirichlet's theorem on primes in arithmetic progression. *Journal of Formalized Reasoning*, 2, 1 (2010), 63–83. doi:10.6092/issn.1972-5787/1558. https://jfr.unibo.it/article/view/1558. (cited on page 141)

HARRISON, J., 2013. A Survey of Automated Theorem Proving. https://www.lektorium.tv/course/23011. PDMI Computer Science Club mini-course, St. Petersburg, Russia. (cited on page 9)

HARRISON, J., 2015. Formalization of Mathematics for Fun and Profit. https://www.cl.cam.ac.uk/~jrh13/slides/wollic-23jul15/slides.pdf. Invited talk at WoLLIC 2015, Indiana University, Bloomington. (cited on page 9)

HASLBECK, M. P. L. AND NIPKOW, T., 2018. Hoare Logics for Time Bounds: A Study in Meta Theory. In *Tools and Algorithms for the Construction and Analysis of Systems*, 155–171. Springer International Publishing, Cham. doi:10.1007/978-3-319-89960-2_9. (cited on pages 111 and 116)

HERSTEIN, I. N., 1975. *Topics In Algebra*. John Wiley & Sons. ISBN 9780471010906. ISBN: 9780471010906. (cited on page 70)

HERSTEIN, I. N., 1996. *Abstract Algebra*. John Wiley & Sons. ISBN 9780471368793. ISBN: 9780471368793. (cited on pages 65, 68, 70, and 139)

HURD, J., 2003. Verification of the Miller-Rabin Probabilistic Primality Test. *Elsevier Science Inc.*, (2003). doi:10.1016/S1567-8326(02)00065-6. DOI: 10.1016/S1567-8326(02)00065-6. (cited on pages 6 and 129)

HURD, J.; GORDON, M.; AND FOX, A., 2006. Formalized Elliptic Curve Cryptography. *High Confidence Software and Systems*, (April 2006). (cited on pages 40 and 135)

IRELAND, K. AND ROSEN, M., 1990. *A Classical Introduction to Modern Number Theory*, vol. 84 of *Graduate Texts in Mathematics*. Springer-Verlag New York. ISBN 9781441930941. doi:10.1007/978-1-4757-2103-4. ISBN: 9781441930941. (cited on pages 69, 138, and 140)

JÄRVINEN, J., 2007. *Lattice Theory for Rough Sets*, 400–498. Springer-Verlag, Berlin, Heidelberg. ISBN 978-3-540-71200-8. doi:10.1007/978-3-540-71200-8_22. https://doi.org/10.1007/978-3-540-71200-8_22. (cited on page 142)

JUDSON, T. W., 1994. *Abstract Algebra: Theory and Applications*. The Prindle, Weber & Schmidt Series in Advanced Mathematics. PWS Publishing Company. ISBN 9780534936846. ISBN: 9780534936846. (cited on page 68)

JUSTESEN, J. AND HØHOLDT, T., 2004. *A Course in Error-Correcting Codes*. EMS Textbooks in Mathematics. European Mathematical Society, New York, Berlin, Heidelberg, 2nd edn. ISBN 3037190019. ISBN: 9783037190012. (cited on page 138)

KNUTH, D. E., 1998. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. ISBN 9780201896848. (cited on page 45)

KUSAK, E.; LEONCZUK, W.; AND MUZALEWSKI, M., 1989. Abelian Groups, Fields and Vector Spaces. *Journal of Formalized Mathematics*, 1 (November 1989). (cited on page 73)

LENSTRA, H. W., 2002. Primality Testing with Gaussian Periods. In *FST TCS 2002: Foundations of Software Technology and Theoretical Computer Science*, vol. 2556 of *Lecture Notes in Computer Science*, 1–1. Springer Berlin Heidelberg, Berlin, Heidelberg. doi:10.1007/3-540-36206-1_1. https://math.dartmouth.edu/~carlp/aks111216.pdf. (cited on page 129)

LIDL, R. AND NIEDERREITER, H., 1986. *Introduction to Finite Fields and Their Applications (2nd Edition)*. Cambridge University Press, New York, NY, USA. ISBN 9780521460941. ISBN: 9780521460941. (cited on pages 134 and 140)

LIDL, R. AND NIEDERREITER, H., 1997. *Finite Fields (2nd Edition)*. No. 20 in Encyclopedia of Mathematics and its Applications. Cambridge University Press, New York, NY, USA. ISBN 9780521392310. ISBN: 9780521392310. (cited on page 136)

LINOWITZ, B., 2006. *An Exposition of the AKS Polynomial Time Primality Testing*. Master's thesis, University of Pennsylvania. (cited on page 90)

MAGID, A. (Ed.), 2008. *A Special Issue on Formal Proof*, vol. 55, Issue 11 of *Notices of the AMS*. American Mathematical Society. (cited on page 9)

MAHBOUBI, A. AND TASSI, E., 2018. *Mathematical Components*. https://math-comp.github.io/mcb/. Available from https://math-comp.github.io/mcb/book.pdf. (cited on page 73)

MCELIECE, R. J., 1987. *Finite Fields for Computer Scientists and Engineers*. The Kluwer International Series in Engineering and Computer Science. Springer, New York, Berlin, Heidelberg. ISBN 9781461291855. ISBN: 9781461291855. (cited on pages 24, 25, 138, and 140)

MYREEN, M. O., 2008. *Formal Verification of Machine-Code Programs*. Ph.D. thesis, University of Cambridge, Computer Laboratory, Trinity College. Supervised by Professor Mike Gordon. (cited on page 142)

MYREEN, M. O., 2012. *Formal Verification of Machine-Code Programs*. Distinguished Dissertation. BCS (February 14, 2011), 3 edn. ISBN 978-1906124816. (cited on page 142)

NEUMANN, P. M., 2013. *The Mathematical Writings of Évariste Galois*. Heritage of European Mathematics. European Mathematical Society. ISBN 9783037191040. ISBN: 9783037191040. (cited on pages 15 and 162)

NEWMAN, S. C., 2012. *A Classical Introduction to Galois Theory*. John Wiley & Sons. ISBN 9781118091395. ISBN: 9781118091395. (cited on page 70)

REMPE-GILLEN, L. AND WALDECKER, R., 2014. *Primality Testing for Beginners*, vol. 70 of *Student Mathematical Library*. American Mathematical Society. ISBN 9780821898833. (cited on pages 1 and 90)

ROBINSON, D. J. S., 2008. *An Introduction to Abstract Algebra*. De Gruyter Textbook. De Gruyter. ISBN 9783110198164. ISBN: 9783110198164. (cited on page 68)

ROBINSON, S., 2002. New Method Said to Solve Key Problem In Math. Available from https://www.nytimes.com/2002/08/08/us/new-method-said-to-solve-key-problem-in-math.html. (cited on page 2)

ROTMAN, J. J., 2010. *Advanced Modern Algebra: Second Edition*, vol. 114 of *Graduate Studies in Mathematics*. American Mathematical Society. ISBN 9781470411763. ISBN: 9781470411763. (cited on page 70)

RUDNICKI, P.; SCHWARZWELLER, C.; AND TRYBULEC, A., 2001. Commutative Algebra in the Mizar System. *Journal of Symbolic Computation*, 32, 1 (Jul. 2001), 143–169. doi:10.1006/jsco.2001.0456. http://dx.doi.org/10.1006/jsco.2001.0456. Special issue on computer algebra and mechanized reasoning: selected St. Andrews' ISSAC/Calculemus 2000 contributions. (cited on page 142)

SAPTHARISHI, R., 2007. Primality Testing: the AKS Algorithm. Available from http://www.cmi.ac.in/~ramprasad/studenttalks/primality/primality.pdf. (cited on page 1)

SHOUP, V., 2008. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, 2nd edn. ISBN 9780511814549. doi:10.1017/CBO9780511814549. (cited on pages 1, 40, and 91)

SIMON WIMMER, L. N., 2013. A Formalisation of Lehmer's Primality Criterion. *Archive of Formal Proofs, Isabelle*, (July 2013). (cited on page 129)

SINGH, A. K., 2017. Formalization of some central theorems in combinatorics of finite sets. *Kalpa Publications in Computing*, 1 (2017), 43–57. LPAR-21S: IWIL Workshop and LPAR Short Presentations. (cited on page 141)

TERENCE TAO, 2009. The AKS primality test. http://terrytao.wordpress.com/2009/08/11/the-aks-primality-test/. (cited on pages 1, 55, and 81)

THÉRY, L., 2003. Proving Pearl: Knuth's Algorithm for Prime Numbers. In *Theorem Proving in Higher Order Logics*, 304–318. Springer Berlin Heidelberg, Berlin, Heidelberg. doi:https://doi.org/10.1007/10930755_20. https://link.springer.com/chapter/10.1007/10930755_20. (cited on page 6)

THÉRY, L., 2007. Proving the group law for elliptic curves formally. Technical Report RT-0330, INRIA. https://hal.inria.fr/inria-00129237. Available from hhttps://hal.inria.fr/inria-00129237/en/. (cited on page 130)

THÉRY, L. AND HANROT, G., 2007. Primality Proving with Elliptic Curves. In *TPHOL 2007*, vol. 4732 of *LNCS*, 319–333. Springer-Verlag, Kaiserslautern, Germany. doi:10.1007/978-3-540-74591-4. https://hal.inria.fr/inria-00138382. (cited on page 130)

THIEMANN, R. AND YAMADA, A., 2016. Algebraic Numbers in Isabelle/HOL. In *Interactive Theorem Proving: 7th International Conference, ITP 2016, Nancy, France, August 22-25, 2016, Proceedings*, 391–408. Springer International Publishing, Cham. doi:10.1007/978-3-319-43144-4_24. https://doi.org/10.1007/978-3-319-43144-4_24. (cited on page 73)

TRYBULEC, A.; KORNILOWICZ, A.; NAUMOWICZ, A.; AND KUPERBERG, K. (Eds.), 2013. *Special Issue: Formal Mathematics for Mathematicians*, vol. 50, Issue 2 of *Journal of Automated Reasoning*. Springer Netherlands. (cited on page 9)

URBAN, J., 2016. Advances in Formal Mathematics. http://prague_logic_2016.math.cas.cz/slides/urban.pdf. (cited on page 9)

VON ZUR GATHEN, J. AND GERHARD, J., 2013. *Modern Computer Algebra*. Cambridge University Press, 3 edn. ISBN 9781139856065. doi:10.1017/CBO9781139856065. (cited on page 116)

WIEDIJK, F., 2003. John Harrison's Formalization of the Agrawal-Kayal-Saxena Primality Test. http://www.cs.ru.nl/~freek/talks/intercity.dvi. Intercity Number Theory Seminar, Nijmegen. (cited on page 6)

XU, J.; ZHANG, X.; AND URBAN, C., 2013. Mechanising Turing Machines and Computability Theory in Isabelle/HOL. In *Interactive Theorem Proving*, 147–162. Springer Berlin Heidelberg, Berlin, Heidelberg. doi:10.1007/978-3-642-32621-9_1. (cited on page 142)

ZHAN, B. AND HASLBECK, M. P. L., 2018. Verifying Asymptotic Time Complexity of Imperative Programs in Isabelle. In *Automated Reasoning*, vol. 10900 of *Lecture Notes in Computer Science*, 532–548. Springer International Publishing, Cham. doi:10.1007/978-3-319-94205-6_35. https://link.springer.com/chapter/10.1007/978-3-319-94205-6_35. (cited on page 111)

Index