Resolving Zero-Divisors of Radical Triangular Sets using Hensel Lifting and Applications

by

John Kluesner

M.Sc. in Scientific Computing, Richard Stockton University, 2014
B.Sc. in Scientific Computing, Richard Stockton University, 2013

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
Department of Mathematics
Faculty of Science

SIMON FRASER UNIVERSITY
Summer 2017

Approval

Name:	**John Kluesner**
Degree:	**Master of Science (Mathematics)**
Title:	**Resolving Zero-Divisors of Radical Triangular Sets using Hensel Lifting and Applications**
Examining Committee:	**Chair:** Jonathan Jedwab Professor
	Michael Monagan Senior Supervisor Professor _____
	Nathan Ilten Supervisor Professor _____
	Imin Chen Examiner Professor _____
Date Defended:	**August 16 2017**

Abstract

This thesis aims to create efficient algorithms for computing in the ring $R = \mathbb{Q}[z_1, \ldots, z_n]/T$ where T is a zero-dimensional triangular set. The presence of zero-divisors in R makes it a computational challenge to use modular algorithms. In particular, there has never been a proper modular algorithm for computing greatest common divisors of polynomials in $R[x]$. We present two new ways of resolving zero-divisors: Hensel lifting and fault tolerant rational reconstruction, which allows us to create a new modular gcd algorithm for $R[x]$ as well as a new inversion algorithm for R. We have implemented our algorithms in Maple using the RECDEN library, and we show that they outperform the methods currently implemented in Maple's `RegularChains` package. The method of Hensel lifting for resolving zero-divisors should give rise to other new modular algorithms for computing modulo triangular sets and our applications show that this approach is fruitful.

Keywords: Computer Algebra, Modular Algorithms, Triangular Sets, Radical Ideals, Inversion, Greatest Common Divisors, Maple

Dedication

I dedicate this thesis to my wife Lauren who I am looking forward to spending my post-graduate life with.

Acknowledgements

I would like to acknowledge all my friends, family, and teachers who have supported me in my journey to complete this and my past degrees. In particular, I would like to thank

- My parents for giving me the freedom to pursue my dreams.
- My supervisor Dr. Michael Monagan for being patient, kind, and understanding, inside and outside of work.
- Dr. Nathan Ilten for being a great professor in two of my favorite courses I have ever enrolled.
- My wife Lauren for being supportive and patient with me while I finished this thesis.
- My Aunt Jeanne and Uncle Mike for always checking up on me and sending me maple syrup while I have been away from home.
- Drs. Judith Vogel, Bradley Forrest, and Robert Olsen from my undergraduate institution for helping me get into graduate school.
- My good friends Jonathan Toscano, Kelly O'Neill, and Daniel King.
- My peers Marshall Law, Brett Nasserden, and Matthew Lynn for working with me through difficult coursework.

Table of Contents

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

Suppose that we seek to find the greatest common divisor of two polynomials $a, b \in \mathbb{Q}(\alpha_1, \ldots, \alpha_n)[x]$ where α_i are algebraic numbers. Since $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ is a field, this can be done using the Euclidean algorithm, but since the coefficients in the polynomial remainder sequence grow quickly, as noted in [5], this is very inefficient. A better approach is to use a modular algorithm. That is, reduce a and b modulo multiple prime numbers p_i and compute $g_i \equiv \gcd(a, b) \pmod{p}$. Then use Chinese remaindering to combine all g_i of lowest degree. This is the approach of Langemyr and McCallum [18] and later improved by Encarnacion [12] by introducing rational reconstruction. A problem with their algorithms is their solution to the multiple extension case is to find a primitive element and then apply an algorithm for one extension. Monagan and van Hoeij [15] improved the multiple extension case by circumventing the primitive element.

The computational model used for an algebraic number field is $\mathbb{Q}[z_1, \ldots, z_n]/T$ where $T = \langle t_1(z_1), t_2(z_1, z_2), \ldots, t_n(z_1, \ldots, z_n) \rangle$ and each t_i is the minimal polynomial of α_i, hence irreducible, over $\mathbb{Q}(\alpha_1, \ldots, \alpha_{i-1})$. A natural generalization is to consider the same problem when each t_i is possibly reducible and hence zero-divisors may be encountered while computing in the ring $\mathbb{Q}[z_1, \ldots, z_n]/T$.

The generators of T form what is known as a triangular set. Let $R = \mathbb{Q}[z_1, \ldots, z_n]/T$. This thesis proposes a new algorithm for computing $\gcd(a, b)$ with $a, b \in R[x]$. The backbone of it is the Euclidean algorithm. However, the EA can not always be used in this ring. For example, suppose $T = \langle z_1^2 + 1, z_2^2 + 1 \rangle$ and $R = \mathbb{Q}[z_1, z_2]/T$. Notice that $z_1^2 - z_2^2 = 0$ in R hence $z_1 - z_2$ and $z_1 + z_2$ are zero-divisors in R. Consider computing the gcd of

$$a = x^4 + (z_1 + 18\,z_2)\,x^3 + (-z_2 + 3\,z_1)\,x^2 + 324\,x + 323$$
$$b = x^3 + (z_1 + 18\,z_2)\,x^2 + (-19\,z_2 + 2\,z_1)\,x + 324$$

using the Euclidean algorithm. The remainder of $a \div b$ is

$$r_1 = (z_1 + 18z_2)x^2 + 323.$$

Since $z_1 + 18z_2$ is a unit, a division can be performed; dividing b by r_1 gives

$$r_2 = (z_1 - z_2)x + 1.$$

The next step in the Euclidean algorithm would be to invert $z_1 - z_2$, but it is a zero-divisor, so it cannot continue. A correct approach would be to factor $z_2^2 + 1 = (z_2 - z_1)(z_2 + z_1) \pmod{z_1^2 + 1}$ to split the triangular set T into $\{z_1^2 + 1, z_2 - z_1\}$ and $\{z_1^2 + 1, z_2 + z_1\}$. After that, finish the EA modulo each of these new triangular sets. It is possible to combine the results using the Chinese remainder theorem, but that is costly so it is common practice to instead return the output of the EA along with the associated triangular set. For example, see the definition of pseudo-gcd in [17] and regular-gcd in [19]. We follow this trend with our definition componentwise-gcd in section 5.1.

Now, consider trying to compute the $\gcd(a, b)$ above using a modular algorithm. One would expect to hit the modular image of the same zero-divisor at each prime and hence one could combine them using Chinese remaindering and rational reconstruction. For instance, the EA modulo 13 will terminate with the zero-divisor $z_1 + 12z_2 \pmod{13}$ as expected. However, running the EA modulo 17 terminates earlier because $\mathrm{lc}(r_1) = z_1 + 18z_2 \equiv z_1 + z_2 \pmod{17}$ is a zero-divisor. This presents a problem: $z_1 + z_2 \pmod{17}$ and $z_1 + 12z_2 \pmod{13}$ will never combine into a zero-divisor no matter how many more primes are chosen.

To circumvent, our algorithm finds a monic zero-divisor and lifts it using Hensel lifting to a zero-divisor over $\mathbb{Q}$. Our technique handles both the expected zero-divisors (such as $z_1 + 12z_2 \pmod{13}$ in the above example) and the unexpected zero-divisors (such as $z_1 + z_2 \pmod{17}$). A different approach that we tried is Abbott's fault tolerant rational reconstruction as described in [1]; although this is effective, we prefer Hensel lifting as it enables us to split the triangular set immediately thus saving work.

We also consider the inversion problem for triangular sets: Given a triangular set $T \subset \mathbb{Q}[z_1, \ldots, z_n]$ and a polynomial $a \in \mathbb{Q}[z_1, \ldots, z_n]/T$, compute a^{-1} or determine it does not exist. This has been solved by Maza, Schost, and Vrbik in [21]. We propose a new algorithm that uses Hensel lifting to resolve zero-divisors, a modular-gcd algorithm to determine invertibility, and Newton iteration to compute the inverse.

We would like to motivate triangular sets from an algebraic geometry perspective. Note that we will be omitting details which the reader may refer to hapter 2 of [8] for. Consider a set of polynomials $f_1, \ldots, f_s \in \mathbb{Q}[z_1, \ldots, z_n]$. Further, assume the variety

$$V = \mathbf{V}(f_1, \ldots, f_s) = \{\alpha \in \mathbb{C}^n : f_i(\alpha) = 0 \text{ for all } i\}$$

is finite. We can create an ideal $\mathbf{I}(V) = \{f \in \mathbb{Q}[z_1, \ldots, z_n] : f(\alpha) = 0 \text{ for all } \alpha \in V\}$. In general, $\mathbf{I}(V) \supset \langle f_1, \ldots, f_s \rangle$. Using resultants or Grobner bases, one can show that $\mathbf{I}(V) = \langle t_1, t_2, \ldots, t_n \rangle$ for polynomials $t_1 \in \mathbb{Q}[z_1]$, $t_2 \in \mathbb{Q}[z_1, z_2]$, etc. where $\deg_{z_i}(t_i) > 0$. This is a natural setting where triangular sets occur.

Continuing, we will show how working modulo $\mathbf{I}(V)$ relates to the roots of $f_1, \ldots, f_s$. Suppose $f \in \mathbb{Q}[z_1, \ldots, z_n]$ satisfied $\emptyset \subsetneq V \cap \mathbf{V}(f) \subsetneq V$. We can find $g \in \mathbb{Q}[z_1, \ldots, z_n]$ where $g(\beta) = 0$ for all $\beta \in V - \mathbf{V}(f)$ but $g \notin \mathbf{I}(V)$. Then $fg \in \mathbf{I}(V)$ and so f would be a zero-divisor modulo $\mathbf{I}(V)$. Conversely, if f is a zero-divisor modulo $\mathbf{I}(V)$, it must contain a root in V for similar reasoning. This establishes that $f \notin \mathbf{I}(V)$ is a zero-divisor modulo $\mathbf{I}(V)$ if and only if $\emptyset \subsetneq V \cap \mathbf{V}(f) \subsetneq V$. We can similarly prove $f \notin \mathbf{I}(V)$ is a unit modulo $\mathbf{I}(V)$ if and only if $V \cap \mathbf{V}(f) = \emptyset$.

Thus, computation modulo $\mathbf{I}(V)$ is intrinsically related to the roots of system of polynomials; in particular, the equivalences in the last paragraph show that efficient computation modulo triangular sets is fundamental to studying these roots. A large amount of research in algebraic geometry concerns roots of systems of polynomials, see chapters 2,3,7 in Using Algebraic Geometry [8] by Cox, Little, and O'Shea for instance.

1.2 New Results

With the motivation in mind, we would like to share what new results we have contributed. Our main contribution comes in the way of using Hensel lifting to resolve zero-divisors encountered modulo a prime p. We give two new applications: a modular gcd algorithm and an inversion algorithm. Using timing tests, we show that our approach is a significant improvement over the algorithms used in the `RegularChains` package in Maple. We also reprove some known results about zero-dimensional radical triangular sets using an approach motivated by algebraic number theory while the norm has been techniques from algebraic geometry in the past. We give detailed descriptions of the algorithms created as well as proofs of correctness. In particular,

- Chapter 4 gives two new ways of resolving zero-divisors modulo triangular sets: one based on Abbott's fault tolerant rational reconstruction (see [1]), and the other based on Hensel lifting.

- Proposition 14 shows a new algorithm for multiplying polynomials modulo a triangular set. While this is not better than other methods asymptotically, we show it is a practical improvement.

- Lemma 32 is a new result about the prime numbers that appear in fractions of monic factors of monic polynomials modulo a triangular set. This generalizes known results from algebraic number theory.

- ModularC-GCD is a new algorithm for computing greatest common divisors of polynomials with coefficients modulo a radical triangular set. We have time tests that show it is faster than the method implemented in Maple.

- Inversion is a new algorithm for computing the inverse of a polynomial modulo a radical triangular set. We have time tests that show it is faster than the method implemented in Maple when working over multiple extensions, which is of major interest.

Chapter 2

Abstract Algebra

The goal of this thesis is to create new algorithms for efficient computation for polynomials over $\mathbb{Q}$. For this purpose, we will make use of more general algebraic structures. The first of these will be commutative rings with unity. We will continue with the monic Euclidean algorithm over a commutative ring with unity. Next, we review rational reconstruction and the p-adic representation of integers because of their importance. We conclude with some field theory to motivate triangular sets, the key object of study in this thesis. The material in section 2.1 can be found in chapter 7 of Dummit and Foote's Abstract Algebra textbook [11]. Also see the first two sections of chapter 13 in [11] for the material in section 2.5.

2.1 Rings

Definition 1. A commutative ring R with unity is a nonempty set with two binary operations, addition (denoted by $a + b$) and multiplication (denoted by ab), that satisfy the following axioms:

i)	$a + b = b + a$ for all $a, b \in R$;	(commutativity of $+$)
ii)	$(a + b) + c = a + (b + c)$ for all $a, b, c \in R$;	(associativity of $+$)
iii)	there exists $0 \in R$ where $a + 0 = a$ for all $a \in R$;	(additive identity)
iv)	for any $a \in R$, there exists $-a \in R$ where $a + (-a) = 0$;	(additive invertibility)
v)	$ab = ba$ for all $a, b \in R$;	(commutativity of multiplication)
vi)	$(ab)c = a(bc)$ for all $a, b, c \in R$;	(associativity of multiplication)
vii)	$a(b + c) = ab + ac$ for all $a, b, c \in R$;	(distributive property)
viii)	there exists $1 \in R$ where $1a = a$ for all $a \in R$;	(multiplicative identity)
ix)	$1 \neq 0$.	(identity distinction)

We will never consider noncommutative rings with or without unity; so when we refer to a ring, we always mean commutative with unity. Here are some examples:

- The rational numbers $\mathbb{Q}$ form a ring.

- The integers $\mathbb{Z}$ form a ring.
- Polynomials with coefficients from a ring R and indeterminate x form a ring; this will be denoted as $R[x]$.
- The set $\mathbb{Q}[z]/\langle z^2-2\rangle$ forms what is known as a quotient ring. It is in fact isomorphic to the algebraic number field $\mathbb{Q}(\sqrt{2}) = \{a+b\sqrt{2} : a, b \in \mathbb{Q}\}$. These concepts are reviewed in later sections.
- Given two rings R_1 and R_2, one can form a new ring $R_1 \times R_2$ under component-wise addition and multiplication.

Many natural properties come easily from the axioms, such as uniqueness of identities and inverses and that $a0 = 0$ for all $a \in R$. The proofs are straightforward. It should be noted that rings do *not* enjoy the existence of multiplicative inverses. The set of elements of R with multiplicative inverses is called the units of R and is denoted as R^*.

We now turn our attention to important subsets of a ring: ideals. These will be used when we define the coefficients of polynomials to which this thesis pertains.

Definition 2. Consider a ring R. An ideal $I \subset R$ is a nonempty set where (i) $a - b \in I$ for all $a, b \in I$ and (ii) $ra \in I$ for all $r \in R$ and $a \in I$.

Ideals occur very naturally in any ring R. For example, the even integers form an ideal of $\mathbb{Z}$. The set $\{0\}$ forms an ideal of any ring R and is sometimes called the trivial ideal. Fix an element $a \in R$ and the set $\langle a\rangle = \{ar : r \in R\}$ forms the principal ideal generated by a. Fix elements $a_1, \ldots, a_n \in R$ and the set $\langle a_1, \ldots, a_n\rangle = \{r_1a_1 + \cdots + r_na_n : r_i \in R,\ i = 1..n\}$ forms the ideal generated by $a_1, \ldots, a_n$. More concretely, if we let $R = \mathbb{Z}$, then $\langle 23993\rangle$ is all integer multiples of 23993.

Given ideals $I, J \subset R$, new ideals can formed as

- the sum of the ideals $I + J = \{a + b : a \in I,\ b \in J\}$;
- the intersection of the ideals $I \cap J$; and
- the product of the ideals $IJ = \{\sum_{i=1}^{n} a_ib_i : a_i \in I,\ b_j \in J,\ n \in \mathbb{Z}, n \geq 1\}$, which is the set of all finite sums of two-products of elements of I and J.

Proving each is an ideal is straightforward. In general, the constructions above form the chain $IJ \subset I \cap J \subset I \subset I + J$. One can also characterize $I + J$ as the smallest ideal containing both I and J, while IJ is the smallest ideal containing all products of elements in $I \cup J$. An interesting question is when the inclusion $IJ \subset I \cap J$ is actually an equality. The next definition gives the answer.

Definition 3. Let I and J be ideals of a ring R. Then I and J are called *comaximal* if $I + J = R$. Equivalently, I and J are comaximal if $1 \in I + J$.

Lemma 1. Let $I_1, \ldots, I_n \subset R$ be pairwise comaximal ideals of R. Then $\bigcap_{i=1}^{n} I_i = \prod_{i=1}^{n} I_i$.

Proof. Work by induction on n. Consider the base case $n = 2$. Take an arbitrary $r \in I_1 \cap I_2$. Since I_1 and I_2 are comaximal, there exist $a \in I_1$ and $b \in I_2$ where $a + b = 1$. Then, $r = ar + br \in I_1 I_2$ by definition. Continue with the inductive step. The inductive hypothesis states $\bigcap_{i=1}^{n-1} I_i = \prod_{i=1}^{n-1} I_i$. I claim I_n and $\prod_{i=1}^{n-1} I_i$ are comaximal. To see this, note that there are $a_i \in I_i$ and $b_i \in I_n$ where $a_i + b_i = 1$ for $i < n$. Then, $1 = \prod_{i-1}^{n-1}(a_i + b_i)$. If one expands this product, one term is $a_1 a_2 \cdots a_{n-1}$ while all others contain some b_i and so must be contained in I_n. This shows $1 \in I_n + \prod_{i=1}^{n-1} I_i$ and so indeed I_n and $\prod_{i=1}^{n-1} I_i$ are comaximal. Thus, by the base case using $\prod_{i=1}^{n-1} I_i$ and I_n,

$$\prod_{i=1}^{n} I_i = I_n \prod_{i=1}^{n-1} I_i = I_n \bigcap_{i=1}^{n-1} I_i = I_n \cap \bigcap_{i=1}^{n-1} I_i = \bigcap_{i=1}^{n} I_i$$

□

We now show how to create quotient rings. Consider a ring R with a nonempty subset I. The relation $\equiv$ on R with respect to I is defined by $a \equiv b \pmod{I}$ if $a - b \in I$.

Proposition 2. Let R be a ring with a nonempty subset I. The relation $\equiv$ is an equivalence relation if and only if I is closed under subtraction.

Proof. ($\Longrightarrow$) Take $a, b \in I$. Note that $a \equiv a \pmod{I}$ by reflexivity. This gives $0 = a - a \in I$. Also, $a - 0 \in I$ and $b - 0 \in I$. Therefore, $a \equiv 0 \pmod{I}$ and $b \equiv 0 \pmod{I}$. By symmetry, $0 \equiv b \pmod{I}$. Finally, using transitivity, $a \equiv 0 \equiv b \pmod{I}$. Thus, $a - b \in I$ and so I is indeed closed under subtraction.

($\Longleftarrow$) First, consider $a \in I$. Since I is closed under subtraction, $0 = a - a \in I$. Now, for reflexive, take $r \in R$ and note that $r - r = 0 \in I$ gives $r \equiv r \pmod{I}$. For symmetric, take $r, s \in R$ where $r \equiv s \pmod{I}$ and observe that $s - r = 0 - (r - s) \in I$ since $0 \in I$ and $r - s \in I$. This implies $s \equiv r \pmod{I}$. For transitive, suppose $r \equiv s \pmod{I}$ and $s \equiv t \pmod{I}$, so $r - s \in I$ and $t - s \in I$ using symmetry. Then, $r - t = (r - s) - (t - s) \in I$. Thus, $\equiv$ is an equivalence relation. □

If I is a set closed under subtraction we can thus form equivalence classes

$$[a] := \{b \in I : b \equiv a \pmod{I}\}.$$

We can multiply and add equivalence classes by multiplying their representatives. For this to be useful, we need $[a][b] = [ab]$ and $[a] + [b] = [a + b]$, but this is only true if I is an ideal.

Proposition 3. Consider a ring R with a nonempty subset I that is closed under subtraction. Then addition and multiplication of the equivalence classes of $\equiv$ are well-defined if and only if I is an ideal.

Proof. ($\Longrightarrow$) First, $0 \in I$ since I is closed under subtraction. With that in mind, take any $r \in R$ and $a \in I$. Note that $a \equiv 0 \pmod{I}$. Then, using that multiplication is well-defined, $ra \equiv r0 \equiv 0 \pmod{I}$ and so $ra = ra - 0 \in I$, as desired.

($\Longleftarrow$) Suppose $a, b_1, b_2 \in R$ with $b_1 \equiv b_2 \pmod{I}$. To show addition is well-defined, note that $b_1 - b_2 \in I$ and so $a + b_1 - (a + b_2) = b_1 - b_2 \in I$ shows $a + b_1 \equiv a + b_2 \pmod{I}$. For multiplication, note that $b_1 - b_2 \in I$ and so $ab_1 - ab_2 = a(b_1 - b_2) \in I$ by the second defining property of an ideal. Thus, $ab_1 \equiv ab_2 \pmod{I}$. □

The equivalence classes of $\equiv$ form the quotient ring of R modulo I. This is often denoted as R/I. All the ring axioms of R/I are inherited from the ring axioms of R because multiplication and addition of the equivalence classes are well-defined.

Note that if $a \in I$, then $a \equiv 0 \pmod{I}$. This reveals the intuition behind quotient rings: all the elements in the ideal act as the additive identity in R/I. For example, consider the ring $\mathbb{Q}[x]$ with ideal $\langle x \rangle$. Then a polynomial $a_0 + a_1 x + \cdots + a_n x^n$ can be identified with its constant coefficient a_0 when working in $\mathbb{Q}[x]/\langle x \rangle$ since $a_i x^i \in \langle x \rangle$ for $i \geq 1$. This reveals that $\mathbb{Q}[x]/\langle x \rangle$ is more-or-less just like working in $\mathbb{Q}$. This notion of equality is known as being isomorphic and will formalized soon. Another important example is the ideal $\langle p \rangle \subset \mathbb{Z}$ when p is a prime number. It follows that $\mathbb{Z}/\langle p \rangle$ gives the integers modulo p as studied in elementary number theory. We denote here $\mathbb{Z}_m := \mathbb{Z}/\langle m \rangle$ for any integer m.

Definition 4. Let R be a ring. A *zero-divisor* $a \in R$ is a nonzero element in which there exists a nonzero $b \in R$ where $ab = 0$.

For example, $\mathbb{Q}[x]$ contains no zero-divisors. For an example of a zero-divisor, in the ring of $\mathbb{Z}_6$, observe that $2 \cdot 3 \equiv 0 \pmod{\langle 6 \rangle}$.

Definition 5. Let R_1 and R_2 be rings. A *homomorphism* is a mapping $\varphi \colon R_1 \to R_2$ that satisfies

$$\varphi(1) = 1, \qquad \varphi(a + b) = \varphi(a) + \varphi(b), \qquad \varphi(ab) = \varphi(a)\varphi(b).$$

Additionally, if φ is bijective, then φ is called an *isomorphism*; we write $R_1 \cong R_2$ and say R_1 and R_2 are isomorphic.

Given a homomorphism $\varphi \colon R_1 \to R_2$, one can prove many expected properties such as $\varphi(0) = 0$, $\varphi(a - b) = \varphi(a) - \varphi(b)$, and $\varphi(a^{-1}) = \varphi(a)^{-1}$ provided $a \in R_1^*$. A homomorphism also gives a way to create an ideal as the kernel, $\ker(\varphi) = \{a \in R_1 : \varphi(a) = 0\}$. One can show that a homomorphism is injective if and only if its kernel is trivial. Here are some examples of homomorphisms:

(i) The mapping $\pi \colon R_1 \times R_2 \to R_1$ defined as $\pi(a, b) = a$ is known as a natural projection homomorphism.

(ii) Give an ideal $I \subset R$, the homomorphism $\varphi \colon R \to R/I$ defined as $\varphi(r) = r + I$ is known as the canonical projection.

(iii) The most important isomorphism in this thesis is the Chinese remainder theorem.

Theorem 4 (CRT)**.** Let R be a ring with pairwise comaximal ideals $I_1, \ldots, I_n$. Then there is a canonical isomorphism giving $R/\prod_{i=1}^n I_i \cong \prod_{i=1}^n R/I_i$.

Proof. Let $I = \prod_{i=1}^n I_i$. The canonical isomorphism will be given by $\varphi(x+I) = (x+I_i)_{i=1..n}$. Proving that φ is a homomorphism is straightforward. What is left to show is that φ is well-defined, injective, and surjective. For well-defined, suppose $\varphi(x+I) \neq \varphi(y+I)$ but $x-y \in I$. This implies that there is some I_i where $x - y \notin I_i$. But then $x - y \notin \bigcap_{i=1}^n I_i = \prod_{i=1}^n I_i = I$ by Lemma 1. For injective, suppose $\varphi(x + I) = \varphi(y + I)$. This implies $x - y \in I_i$ for all i and so $x - y \in \bigcap_{i=1}^n I_i = I$.

For surjective, it is sufficient to show that there is an element $x_i \in R$ such that $x_i \equiv 1 \pmod{I_i}$ and $x_i \equiv 0 \pmod{I_j}$ for $j \neq i$. If we had such x_i, then for any point $(a_i + I_i)_{i=1..n}$, we could use $\sum_{i=1}^n a_i x_i$ to map to it. With that in mind, note that I_i is comaximal to $\prod_{j=1, j\neq i}^n I_j$ as proven in Lemma 1. This implies there is $b_i \in I_i$ and $x_i \in \prod_{j=1,j\neq i}^n I_j$ where $b_i + x_i = 1$. Note that $x_i = 0 \pmod{I_j}$ for $j \neq i$ and $x_i \equiv 1 - b_i \equiv 1 \pmod{I_i}$, as desired. □

This statement of Theorem 4 generalizes its analogue in elementary number theory, which asserts that if given relatively prime integers $q_1, \ldots, q_s$ and integers $u_1, \ldots, u_s$, then there exists a unique integer u where $|u| < q_1 \cdots q_s$ and $u \equiv u_i \pmod{q_i}$. This is how we will use the CRT to combine modular images. That is, consider distinct prime numbers $p_1, \ldots, p_k$. We often find ourselves in the situation where we are trying to compute some $a \in \mathbb{Z}$ and it is easier to compute its modular images b_i modulo p_i; that is, $a \equiv b_i \pmod{p_i}$. Suppose we can ensure $0 \leq a < p_1 \cdots p_k$. Then we compute multiple modular images b_i modulo p_i for $i = 1, \ldots, k$. Next, we use the CRT to retrieve b where $b \equiv b_i \pmod{p_i}$ and $|b| \leq p_1 \cdots p_k$. Since b is the unique integer satisfying these properties, it follows that $a = b$. This process is known as *recovering* a from its modular images. We will also call this entire process *Chinese remaindering*. Note that if we want to recover negative integers, we can use the symmetric range modulo m; that is, we choose integers t where $-m/2 < t \leq m/2$ as the equivalence class representatives. For example, if $m = 5$, the symmetric range has representatives $-2, -1, 0, 1, 2$. This allows us to recover any integer a where $|a| < m/2$.

We turn our attention to the important concept of the radical of an ideal and the related notion of nilpotent elements. In general, nilpotent elements should be seen as troublesome; this is exemplified in Proposition 5 below where we show that if R has a nonzero nilpotent element, then $R[x]$ contains units of nonzero degree.

Definition 6. An element $a \in R$ is *nilpotent* if $a^n = 0$ for some positive integer n.

Definition 7. Let R be a ring. The *radical of an ideal* I is $\sqrt{I} = \{a \in R : a^n \in I \text{ for some } n \in \mathbb{Z}\}$. An ideal I is *radical* if $\sqrt{I} = I$.

For example, $\sqrt{0}$ is the set of all nilpotent elements. We will define $\sqrt{R} = \sqrt{0}$ even though it does not follow the definition above. It's straightforward to show that $\sqrt{I}$ is also

an ideal. Note that computing in $R/\sqrt{I}$ as opposed to R/I essentially strips R/I of its nilpotents. It turns out for many applications zero-divisors are easier to conquer. This is true for the modular algorithm presented later, largely due to the following proposition. Its main application is it tells us that if $\sqrt{R} = 0$, then $R[x]^* = R^*$. The proposition is taken from exercise 4 of section 1.3 of [4].

Lemma 5. Let R be a ring. Suppose $a \in R^*$ and $b \in \sqrt{R}$. Then $a + b \in R^*$.

Proof. Let $b^m = 0$. Observe that $(a+b)(a^{-1} - a^{-2}b + \cdots + (-1)^{m-1}a^{-m}b^{m-1}) = 1$ since $aa^{-1} = 1$, $b^m = 0$, and all other terms cancel since expanding results in a telescoping sum. □

Proposition 6. Let R be a ring. Then $\sqrt{R[x]} = \sqrt{R}[x]$. Furthermore,

$$R[x]^* = R^* + \sqrt{R}[x]$$

where the right hand side is to be interpreted as the set of all polynomials $a_0+a_1x+\cdots+a_dx^d$ where $a_0 \in R^*$ and a_i is nilpotent for $i > 0$.

Proof. First, let $f = a_0 + a_1x + \cdots + a_nx^n \in \sqrt{R[x]}$. There is a positive integer d where $f^d = 0$. Well, the constant coefficient of f^d must also equal 0 by comparing coefficients. Therefore, $a_0 \in \sqrt{R}$. Clearly, $\sqrt{R} \subset \sqrt{R[x]}$ and so $f - a_0 \in \sqrt{R[x]}$. This implies that there is a positive integer d_1 where $(a_1x + \cdots + a_nx^n)^{d_1} = 0$. We can use similar computations to show $a_1 \in \sqrt{R}$ and hence $a_1x \in \sqrt{R[x]}$. Repeating will give $f \in \sqrt{R}[x]$. Conversely, suppose $f \in \sqrt{R}[x]$. Work by induction on $\deg(f)$. If $\deg(f) = 0$, then $f \in \sqrt{R} \subset \sqrt{R[x]}$. Next, let $f = f_0 + a_nx^n$ where $\deg(f_0) = n-1$ and so $f_0 \in \sqrt{R[x]}$ by the induction hypothesis. Since $a_n \in \sqrt{R}$, clearly $a_nx^n \in \sqrt{R[x]}$. Hence $f \in \sqrt{R[x]}$ as $\sqrt{R[x]}$ is an ideal.

For the second statement, let $f \in R^* + \sqrt{R}[x]$ so that $f = a + f_1$ where $a \in R^*$ and f_1 has no constant term with all nilpotent coefficients. Then, $f_1 \in \sqrt{R}[x] = \sqrt{R[x]}$ and so Lemma 5 gives $f \in R[x]^*$. Conversely, let $f \in R[x]^*$. Proceed by induction on $\deg(f)$. The base case with $\deg(f) = 0$ should be clear. Now, suppose f is of degree n and there is $g \in R[x]$ where $fg = 1$. Let $f = f_0 + f_1x + \cdots + f_nx^n$ and $g = g_0 + g_1x + \cdots + g_mx^m$. The equation $fg = 1$ creates the system of equations

$$\begin{aligned}
1 &= f_0g_0, \\
0 &= f_0g_1 + f_1g_0, \\
&\vdots \\
0 &= f_{n-1}g_m + f_ng_{m-1}, \\
0 &= f_ng_m.
\end{aligned}$$

This clearly gives $f_0 \in R^*$. Next, multiplying the second to last by f_n reveals $0 = f_n^2 g_{m-1}$. Then, multiplying $0 = f_{n-2}g_m + f_{n-1}g_{m-1} + f_n g_{m-2}$ by f_n^2 gives $f_n^2 g_{m-2} = 0$. Repeat this process until $f_n^m g_0 = 0$. Since g_0 is a unit, $f_n^m = 0$. Of course $f_n x^n$ would be nilpotent as well. Therefore, $f - f_n x^n$ is a unit by Lemma 5. By the induction hypothesis, $f - f_n x^n$ has the desired form and hence f does as well. □

We conclude with a simple lemma about comaximal radical ideals that will be often used in the section about the modular gcd algorithm presented later.

Lemma 7. Suppose $I_1, \ldots, I_n$ are comaximal ideals of a ring R. Then $\bigcap_{i=1}^n I_i$ is radical if and only if all I_i are radical.

Proof. ($\Longrightarrow$) Suppose some I_j is not radical. By the CRT, $R/\bigcap_{i=1}^n I_i \cong \prod_{i=1}^n R/I_i$. Since I_j is not radical, R/I_j contains a nilpotent element g_j. Well, let g map to $(0, \ldots, g_j \ldots, 0)$ using the CRT where all components are 0 besides the jth. Clearly, g would be a nonzero nilpotent as well, giving that $\bigcap_{i=1}^n I_i$ is radical.

($\Longleftarrow$) Conversely, suppose g is a nonzero nilpotent element of $R/\bigcap_{i=1}^n I_i$ and let $g \mapsto (g_1, \ldots, g_n)$. Since g is nonzero, some $g_j \neq 0$ because the CRT gives an isomorphism. If $g^k = 0$, then clearly $g_j^k = 0$ as well. Therefore, I_j is not radical. □

2.2 The Monic Euclidean Algorithm

In this section, we will assume we are working over a commutative ring R with unity where every non-zero element of the ring is either a unit or a zero-divisor, there is an algorithm that can decide if an element is a unit, and we have a method to compute inverses. This is everything we need to be able to use the monic Euclidean algorithm in $R[x]$. The purpose of the Euclidean algorithm is to compute greatest common divisors. We follow the presentation in [15].

Definition 8. Let R be a ring with elements a, b. We say a divides b and write $a \mid b$ if there exists $q \in R$ such that $aq = b$.

Definition 9. Let R be a ring and $a, b \in R[x]$. Then a *greatest common divisor* of a and b is denoted as $\gcd(a, b)$ and satisfies (i) $\gcd(a, b) \mid a$ and $\gcd(a, b) \mid b$, and (ii) any common divisor of a and b divides $\gcd(a, b)$. We use the convention that $\gcd(0, 0) = 0$.

Although greatest common divisors are not unique, as the property of being a greatest common divisor is invariant under multiplication by units, we will nevertheless write $\gcd(a, b)$ to represent an arbitrary but consistently chosen greatest common divisor.

The Euclidean algorithm works by using repeated divisions. Here, we specialize to only division by monic polynomials. That is, given $a, b \in R[x]$ with b monic, we can compute the

quotient and remainder of $a \div b$ using the following recurrences:

$$r_0 := a, \qquad q_0 = 0,$$
$$r_n := r_{n-1} - \mathrm{lc}(r_{n-1}) b x^{\deg(r_{n-1})-\deg(b)} \qquad q_n := q_{n-1} + \mathrm{lc}(r_{n-1}) x^{\deg(r_{n-1})-\deg(b)}$$

which terminate when $\deg(r_n) < \deg(b)$ or $r_n = 0$. Note that $\deg(r_n) < \deg(r_{n-1})$, so the process must end. It can be easily verified by induction that $a = bq_n + r_n$. We are making no assumptions on the uniqueness of the remainder, just the existence.

The monic Euclidean algorithm can be succinctly described as inverting leading coefficients and division by the resulting monic polynomial. Of course, if a leading coefficient can not be inverted, we can ensure that it is a zero-divisor because of our assumptions about R. If this happens, we terminate and return an error message ["ZERODIVISOR", u] where $u \in R$ is the zero-divisor.

Algorithm 1: MonicEuclideanWithErrorHandling

Input : A ring R as specified in the opening of the section, and two polynomials $a, b \in R[x]$. Assume $\deg_x(a) \geq \deg_x(b)$.
Output: Either monic $\gcd(a, b)$ or an error if a zero-divisor is encountered.

```
1  if b = 0 then
2  |  if lc(a) is a zero-divisor then return ["ZERODIVISOR", lc(a)];
3  |  return lc(a)^{-1} a
4  end
5  Set r_0 := a and r_1 := b;
6  i := 1;
7  while r_i ≠ 0 do
8  |  if lc(r_i) is a zero-divisor then return ["ZERODIVISOR", lc(r_i)];
9  |  r_i := lc(r_i)^{-1} r_i;
10 |  Set r_{i+1} as the remainder of r_{i-1} divided by r_i;
11 |  i := i + 1;
12 end
13 return r_{i-1}
```

Proposition 8. Let R be a ring and $a, b \in R[x]$ with $\deg(a) \geq \deg(b)$. Assume no zero-divisors are encountered when running the monic Euclidean algorithm on a and b. Then the output is a $\gcd(a, b)$.

Proof. Let $g = r_{i-1}$, the last nonzero remainder. We have to show that (i) $g \mid a$ and $g \mid b$, and (ii) any common divisor $d \mid a$ and $d \mid b$ also divides g. It will be useful to index the quotient q_j and leading coefficient $c_j = \mathrm{lc}(r_j)$ at the jth iteration; so $r_{j-2} = q_j r_{j-1} + c_j r_j$. With that in mind, for (i), note that $r_{i-2} = q_i r_{i-1}$, and $g \mid r_{i-2}$. This implies $g \mid q_{i-1} r_{i-2} + c_{i-1} r_{i-1} = r_{i-3}$ as well. We can repeat the above argument $i - 3$ more times to get $g \mid r_0$ and $g \mid r_1$. Clearly, $g \mid \mathrm{lc}(b) r_1 = b$ as well. This completes (i). For (ii), consider a common divisor d. Then,

$d \mid r_0 = a$ and $d \mid \mathrm{lc}(b)^{-1}b = r_1$. This implies $d \mid r_0 - q_2 r_1 = r_2$. Apply this argument $i-2$ more times to get $d \mid r_{i-1}$, as desired.

Further, the algorithm must terminate because after a finite number of iterations in the while-loop, some r_i must be of degree 0 since the $\deg(r_i) < \deg(r_{i+1})$. □

We can also make the extended monic Euclidean algorithm by introducing the recurrences

$$s_0 := 1,\ s_1 := 0 \qquad t_0 := 0,\ t_1 := 1 \tag{2.1}$$

$$s_n := s_{n-2} - \mathrm{lc}(r_{n-1})^{-1} q_n s_{n-1} \qquad t_n := t_{n-2} - \mathrm{lc}(r_{n-1})^{-1} q_n t_{n-1} \tag{2.2}$$

We have to scale the quotient q_n by $\mathrm{lc}(r_{n-1})^{-1}$ to match the scaling of r_{n-1}. It can again be easily verified that $s_n a + t_n b = r_n$ using induction. In particular, there exists $s, t \in R$ where $as + bt = \gcd(a, b)$. Of course, this is only proven for the case when the monic Euclidean algorithm does not encounter a zero-divisor. We have proven the following lemma.

Lemma 9. Let $a, b \in R[x]$ with $\deg(a) \geq \deg(b)$. Suppose no zero-divisors are encountered when running the monic Euclidean algorithm on a and b. Then there exists polynomials $s, t \in R$ where $as + bt = \gcd(a, b)$. Moreover, $\langle \gcd(a, b) \rangle = \langle a, b \rangle$.

We would like to end with a Lemma concerning gcds and direct products of rings. The proof is straightforward and follows from isomorphisms preserving divisibility, but is not commonly studied. This will be of great use in the modular gcd algorithm in chapter 4. One can generalize this result for $R \cong R_1 \times \cdots \times R_n$ easily by using induction on n.

Lemma 10. Let $R \cong R_1 \times R_2$. Let $\pi_i \colon R \to R_i$ be the natural projection homomorphisms. Let $a, b \in R$ with $\pi_i(a) = a_i$ and $\pi_i(b) = b_i$. Then

(i) If $g = \gcd(a, b)$, then $\pi_i(g)$ is a $\gcd(a_i, b_i)$.

(ii) If $g_1 = \gcd(a_1, b_1)$ and $g_2 = \gcd(a_2, b_2)$, then $g \in R$ is a $\gcd(a, b)$ where g satisfies $\pi_1(g) = g_1$ and $\pi_2(g) = g_2$.

Proof. (i) We will only prove this for $i = 1$ without loss of generality. First, $\pi_1(g) \mid a_1$ and $\pi_1(g) \mid b_1$ since homomorphisms preserve divisibility. Next, consider a common divisor $d_1 \mid a_1$ and $d_1 \mid b_1$ in R_1. Note that $1 \mid a_2$ and $1 \mid b_2$. Let $d \in R$ satisfy $\pi_1(d) = d_1$ and $\pi_2(d) = 1$. Then d must be a common divisor of a and b. Therefore, $d \mid g$ and so indeed $d_1 \mid \pi_1(g)$.

(ii) First, $g \mid a$ and $g \mid b$ since isomorphisms preserve divisibility. Next, take a common divisor $d \mid a$ and $d \mid b$. Let $d \in R$ satisfy $\pi_1(d) = d_1$ and $\pi_2(d) = d_2$. It follows that $d_i \mid a_i$ and $d_i \mid b_i$. Therefore, $d_i \mid g_i$. Using that isomorphisms preserve divisibility, we get $d \mid g$. □

2.3 Rational Reconstruction

Rational reconstruction solves the problem of recovering rational numbers from a modular image. In particular, given u and m, it finds $n, d \in \mathbb{Z}$ where $n/d \equiv u \pmod{m}$ and there is some sort of uniqueness criteria on n and d. It has a wide range of applications in computer algebra, including Encarnacion's gcd algorithm for polynomials over a number field [12] and solving linear systems over $\mathbb{Q}$. For us, it will be used multiple times in multiple algorithms, so we would like to quickly review.

With that in mind, let u and m be two integers. We can use the extended Euclidean algorithm on u and m to create a sequence of integers s_i, t_i, r_i such that

$$s_i m + t_i u = r_i, \qquad 1 \leq i \leq n+1$$

where $r_1 := m$, $r_2 := u$, and $r_{i+1} = \operatorname{rem}(r_{i-1}, r_i)$. This is done in the same way as the extended Euclidean over a ring as in the last section, but the integer division algorithm is used rather than the polynomial division algorithm. The key observation is that $t_i u \equiv r_i \pmod{m}$. In particular, if $\gcd(t_i, m) = 1$, then $t_i^{-1} \mod m$ exists and $u \equiv r_i/t_i \pmod{m}$. This is how the rational numbers $n/d \in \mathbb{Q}$ are generated. This construction also gives a uniqueness result.

Theorem 11 (Wang, Guy, Davenport, 1982)**.** Let $n, d, m, u \in \mathbb{Z}$ with $d, m > 0$ and $\gcd(n, d) = \gcd(m, d) = 1$ and $u \equiv n/d \pmod{m}$. Let $N, D \in \mathbb{Z}$ such that $N \geq n$ and $D \geq d$. If $m > 2ND$, then

(i) the modular map $\varphi\colon \mathbb{Q} \to \mathbb{Z}_m$ defined as $\varphi(\frac{a}{b}) = b^{-1}a \pmod{m}$ is injective when restricted to $\{\frac{a}{b} \in \mathbb{Q} : |\frac{a}{b}| < m,\ \gcd(b, m) = 1\}$.

(ii) on input of m and u, there exists a unique index i in the extended Euclidean algorithm satisfying $r_i/t_i = n/d$. Moreover, i is the first index such that $r_i \leq N$.

For a proof, see [25]. The restriction $\gcd(m, d) = 1$ is a valid concern and must be resolved in the algorithm that is using rational reconstruction. If, for instance, the algorithm uses multiple primes and one such prime p does not satisfy $\gcd(p, d) = 1$, then rational reconstruction will not work. The algorithm must either not use p or eventually discard p from consideration. The modular gcd algorithm in chapter 5 and inversion algorithm in chapter 6 indeed consider this.

There have been many optimizations to the main algorithm, such as Monagan's Maximal Quotient Rational Reconstruction [23]. Also see [1] for fault tolerant rational reconstruction, which we use and review in section 4.2. The base algorithm for rational reconstruction now follows in Algorithm 2 as it is depicted in [25]. Note that it requires bounds $N \geq |n|$ and $D \geq d$ on input.

Algorithm 2: RationalReconstruction

Input : Integers m, u, N, D with $m > u \geq 0$, $N, D > 0$ and $2ND < m$.

Output: Either $n/d \in \mathbb{Q}$ such that $|n| \leq N$, $0 < d \leq D$, $\gcd(n, d) = 1$, and $n/d \equiv u \pmod{m}$, or FAIL implying no such rational n/d exists.

Initialize $r_1 := m$, $r_2 := u$, and $t_1 := 0$, $t_2 := 1$;
Set $i := 2$;
while $r_i > N$ **do**
    Set $q := \lfloor r_{i-1}/r_i \rfloor$;
    Set $r_{i+1} := r_{i-1} - qr_i$;
    Set $t_{i+1} := t_{i-1} - qr_i$;
    Iterate $i := i + 1$;
end
Set $n := r_i$ and $d := t_i$;
if $d < 0$ **then** set $n := -n$ and $d := -d$;
if $d \leq D$ **and** $\gcd(n, d) = 1$ **then return** n/d;
Otherwise, **return** FAIL;

We will assume any actual calls to the rational reconstruction will use Monagan's maximal quotient rational reconstruction. Note that this version is heuristic and so doesn't require bounds on the numerator or denominator.

2.4 p-adic Representations and Hensel lifting

We would like to briefly review the p-adic representation of integers using the symmetric range. With that in mind, let p be an odd prime and n an integer. The main result we need is that we may write n in the form $n = \sum_{i=0}^{k} a_i p^i$ for some positive integer k and integers a_i where $|a_i| < p/2$. The number k is the *order* of the p-adic representation. For example, the 5-adic expansion of 13 using the symmetric range is $13 = -2 + (-2)5 + 1 \cdot 5^2$. Further, we will need this form to be unique. Once we have proven the uniqueness and existence for integers, it is easy to extend this result to that of polynomials.

Proposition 12. Let $p > 2$ be a prime number. Every integer $n \in \mathbb{Z}$ has a unique p-adic representation with finite order if the symmetric range is used.

Proof. Existence: First, consider only $n \geq 0$. If $n \leq p/2$, then the p-adic representation of n clearly exists. Otherwise, divide n by p in the symmetric range to achieve $n = pq_0 + r_0$ with $|r_0| < p/2$. We claim $|q_0| < |n|$. If $q_0 = 0$, we are done, so assume $|q_0| \geq 1$. Now, suppose

$|n| \leq |q_0|$ to the contrary and observe that

$$\begin{aligned}
|q_0| \geq |n| &= |pq_0 + r_0| = |pq_0 - (-r_0)| \\
&\geq |pq_0| - |-r_0| \\
&> p|q_0| - \frac{p}{2} \\
&= \frac{p}{2}|q_0| + \frac{p}{2}|q_0| - \frac{p}{2} \\
&\geq \frac{p}{2}|q_0| + \frac{p}{2} - \frac{p}{2} \\
&= \frac{p}{2}|q_0|
\end{aligned}$$

which is a contradiction since $p > 2$. Also, if $q_0 < 0$, then $n < p0 + r_0 = r_0 < p/2$ which falls into a case that is already handled. In particular, we may assume that $0 \leq q_0 < n$ and so we may repeat the same argument on q_0 giving $n = (pq_1 + r_1)p + r_0$. There are only a finite amount of integers between 0 and n, so this process must terminate after k iterations with $q_k = 0$. This gives

$$n = r_k p^k + r_{k-1} p^{k-1} + \cdots + r_1 p + r_0,$$

as desired. If $n < 0$, then find the p-adic representation for $-n$ and flip the signs of the remainders.

Uniqueness: Suppose n had two p-adic representations:

$$n = a_0 + a_1 p + \cdots + a_k p^k = b_0 + b_1 p + \cdots + b_m p^m$$

where m, k are positive integers and $a_i, b_i \in \mathbb{Z}$ with $|a_i|, |b_i| < p/2$. Reduce this equation modulo p to get $a_0 = b_0$ by size constraints. After that, reduce modulo p^2 to get $a_1 p \equiv b_1 p \pmod{p^2}$ so that $a_1 \equiv b_1 \pmod{p}$. By size constraints again, $a_1 = b_1$. This process can be repeated to get all $a_i = b_i$ for $i \leq \max\{m, k\}$. Thus, uniqueness is shown. □

We can also give a p-adic representation to polynomials in $\mathbb{Z}[z_1, \ldots, z_n]$. That is, let $f = \sum_{i=0}^{k} c_i \mathbf{z}_i$ where $\mathbf{z}_i$ are monomials and $c_i \in \mathbb{Z}$. Then each c_i has a unique p-adic representation: $c_i = \sum_{j=0}^{m_i} a_{ij} p^j$. Let $m = \max_{i=1..k}\{m_i\}$ and define $a_{ij} = 0$ for $i > m_{ij}$. Expanding out and grouping in terms of powers of p gives a p-adic representation of f:

$$f = \sum_{i=0}^{k} c_i \mathbf{z}_i = \sum_{i=0}^{k} \Big(\sum_{j=0}^{m} a_{ij} p^j \Big) \mathbf{z}_i = \sum_{j=0}^{m} \Big(\sum_{i=0}^{k} a_{ij} \mathbf{z}_i \Big) p^j.$$

The integer m would be the order of f with respect to its p-adic representation.

Proposition 13. Let $p > 2$ be a prime number. Every polynomial $f \in \mathbb{Z}[z_1, \ldots, z_n]$ has a unique p-adic representation.

Proof. Existence follows from the preceding paragraph. The uniqueness result can be proven the same way as the integer case. □

Our main use of the p-adic representation is in Hensel lifting. Let $f \in \mathbb{Z}[x]$ and $a_1, b_1 \in \mathbb{Z}_p[x]$ satisfying $f \equiv a_1 b_1 \pmod{p}$ and $\gcd(a_1, b_1) = 1$. Hensel lifting gives a construction that lifts this factorization to a factorization mod p^j for any $j > 1$; that is, it constructs unique polynomials a_j, b_j such that $f \equiv a_j b_j \pmod{p^j}$ with $a_j \equiv a_1 \pmod{p}$ and $b_j \equiv b_1 \pmod{p}$ for all j. This is done via the Hensel construction:

1) Assume there are polynomials $a_{j-1}, b_{j-1} \in \mathbb{Z}_{p^{j-1}}[x]$ where $f \equiv a_{j-1} b_{j-1} \pmod{p^{j-1}}$.

2) We will compute the next term in p-adic representation of a_j and b_j. That is, we will compute $u, v \in \mathbb{Z}[x]$ satisfying $a_j = a_{j-1} + up^{j-1}$ and $b_j = b_{j-1} + vp^{j-1}$ where u and v have integer coefficients bounded by $p/2$, $\deg(u) < \deg(a_j)$, and $\deg(v) < \deg(b_j)$.

3) We also want to satisfy $f \equiv a_j b_j \pmod{p^j}$. This gives

$$f \equiv a_j b_j \equiv (a_{j-1} + up^{j-1})(b_{j-1} + vp^{j-1}) \equiv a_{j-1} b_{j-1} + (a_{j-1} v + u b_{j-1}) p^{j-1} \pmod{p^j}.$$

4) Subtract $a_{j-1} b_{j-1}$ on both sides and divide through by p^{j-1} to get

$$\frac{f - a_{j-1} b_{j-1}}{p^{j-1}} \equiv a_{j-1} v + u b_{j-1} \pmod{p}.$$

5) Let $c_j = \frac{f - a_{j-1} b_{j-1}}{p^{j-1}}$. Note that $a_{j-1} \equiv a_1 \pmod{p}$ and $b_{j-1} \equiv b_1 \pmod{p}$. So, it's sufficient to be able to solve the Diophantine equation $c \equiv a_1 v + u b_1 \pmod{p}$ for u and v in $\mathbb{Z}_p[x]$ satisfying the requirements from step 1.

Well, Theorem 2.6 in [14] states that this result exists when $\gcd(a_1, b_1) \equiv 1 \pmod{p}$ and is unique when it satisfies the degree constraints in step 2. We generalize it in section 4.1 as Lemma 25.

Hensel lifting is used often in computer algebra. One of its main applications is for factorization of polynomials over $\mathbb{Z}$. See chapter 8 of [14] for details.

Starting with an initial factorization mod p allows us to use the construction above to lift to as high p-adic order as we like. Please see section 6.5 of [14] for more details of this process. We will generalize the Hensel construction in section 4.1 to a $R[x]$ where R is a quotient ring over $\mathbb{Q}$.

2.5 Fields and Extensions

Definition 10. A *field* is a ring where every nonzero element has a multiplicative inverse.

For example, $\mathbb{Q}$ and $\mathbb{Z}_p$ (where p is a prime number) are fields. The *characteristic* m of a field is the smallest positive integer where $\sum_{i=1}^{m} 1 = 0$, or if no such m exists, the characteristic is 0. The field $\mathbb{Q}$ is of characteristic 0 and $\mathbb{Z}_p$ is of characteristic p. It is not hard to prove that the characteristic is either 0 or a prime number.

Definition 11. Let k and F be fields with a homomorphism $\varphi\colon k \to F$. Then $k \to F$ is a *field extension*. The *degree* of F over k is the dimension of F as a k-vector space. If the degree is a finite number m, we say $k \to F$ is a finite extension and write $[F : k] = m$.

If there is a tower of finite extensions $k \to F \to L$, one can show that $[L : k] = [L : F][F : k]$ by taking products of pairs of elements from each corresponding basis.

The most important field extension in this paper will be $k \to k[z]/\langle t(z)\rangle$ where $t(z)$ is an irreducible polynomial; recall that a polynomial is irreducible if whenever $t(z) = a(z)b(z)$, either $a(z)$ or $b(z)$ is a unit. It is not immediately obvious that $k[z]/\langle t(z)\rangle$ is a field. This follows since any equivalence class modulo $t(z)$ can be represented by a polynomial $f(z)$ of smaller degree than $t(z)$. By Lemma 9, $\langle t(z), f(z)\rangle = \langle \gcd(t(z), f(z)\rangle$. Well as long as $f(z) \not\equiv 0 \pmod{t(z)}$, then $\gcd(t(z), f(z)) = 1$ because of the degree constraints of $f(z)$ and irreducibility of $t(z)$. It is easy to see from here that $f(z)$ would be unit modulo $t(z)$ as long as $f(z) \not\equiv 0 \pmod{t(z)}$.

Consider a finite field extension $k \to F$. Let $\alpha \in F$ be a nonzero element. Since $[F : k]$ is finite, the set $\{1, \alpha, \alpha^2, \dots\}$ must have a linear dependence in it. That is, there exists $c_0, \dots, c_m$ with $c_m \neq 0$ and $c_0 + c_1\alpha + \cdots + c_m\alpha^m = 0$. If we choose m minimally, then $t(z) = c_0 + c_1 z + \cdots + c_m z^m$ is the polynomial of lowest degree over k with α as a root. If in addition $c_m = 1$, then $t(x)$ is called the *minimal polynomial* of α over k. We define $k(\alpha) := k[z]/t(z)$. When $k = \mathbb{Q}$ and α is contained in some finite extension $\mathbb{Q} \to F$, we call $\mathbb{Q}(\alpha)$ an algebraic number field. For example, $\mathbb{Q}(\sqrt{2}) \cong \mathbb{Q}(z)/\langle z^2 - 2\rangle$ is an algebraic number field defined by the minimal polynomial $z^2 - 2$.

Once we have an extension of the form $\mathbb{Q} \to \mathbb{Q}(\alpha_1)$, we can extend it further to $\mathbb{Q} \to \mathbb{Q}(\alpha_1) \to \mathbb{Q}(\alpha_1, \alpha_2)$ where $\mathbb{Q}(\alpha_1, \alpha_2) := \mathbb{Q}(\alpha_1)(\alpha_2)$. This is equivalent to working in $\mathbb{Q}[z_1, z_2]/\langle t_1(z_1), t_2(z_1, z_2)\rangle$ where $t_1(z_1)$ is the minimal polynomial of α_1 over $\mathbb{Q}$ and $t_2(z_1, z_2)$ is the minimal polynomial of α_2 over $\mathbb{Q}(\alpha_1)$. Note that the theorem of the primitive element (see Theorem 5.4.1 of [9]) tells us that $\mathbb{Q}(\alpha_1, \alpha_2) \cong \mathbb{Q}(\beta)$ for some $\beta \in \mathbb{Q}(\alpha_1, \alpha_2)$. Here, we do not use such a primitive element and instead work over multiple extensions. However, this does justify us calling $\mathbb{Q}(\alpha_1, \alpha_2)$ an algebraic number field.

We can repeat the above construction for n extensions:

$$\mathbb{Q}(\alpha_1, \dots, \alpha_n) \cong \mathbb{Q}[z_1, \dots, z_n]/\langle t_1(z_1), \dots, t_n(z_1, \dots, z_n)\rangle$$

where $t_k(z_1, \dots, z_k)$ is the minimal polynomial of α_k over $\mathbb{Q}(\alpha_1, \dots, \alpha_{k-1})$ when z_i is evaluated at α_i for $i < k$. These sets $\{t_1(z_1), \dots, t_n(z_1, \dots, z_n)\}$ obtained in this fashion are examples of *triangular sets*, the topic to which the remainder of this thesis is dedicated.

Chapter 3

Triangular Sets

This section details the important concept of a triangular set. We start with definitions and examples while recalling our motivation from algebraic number fields. We continue with results about counting field multiplications of arithmetic operations modulo triangular sets; including a new result about multiplication which we show is a practical improvement over the currently used methods. The last section is dedicated to radical triangular sets. We give a structure theorem with significant corollaries and conclude with an algorithm for determining when T is radical over $\mathbb{Q}$, but is not when reduced modulo a prime number p.

3.1 Definitions and Examples

We begin with some notation. All computations will be done in the ring $k[z_1, \ldots, z_n]$ endowed with the monomial ordering $z_i < z_{i+1}$ and k a field. Let $f \in k[z_1, \ldots, z_n]$ be non-constant. The *main variable* $\operatorname{mvar}(f)$ of f is the largest variable with nonzero degree in f, and the *main degree* of f is $\operatorname{mdeg}(f) = \deg_{\operatorname{mvar}(f)}(f)$. This notation was taken from [2].

As noted in the previous section, triangular sets will be of key interest in this paper. Further, they are to be viewed as a generalization of an algebraic number field with multiple extensions. For this reason, we impose extra structure than is standard:

Definition 12. A *triangular set* T is an ordered set of non-constant polynomials in $k[z_1, \ldots, z_n]$ with distinct main variables. Further:

(i) $|T| = n$,
(ii) $T = \{t_1, \ldots, t_n\}$ where $\operatorname{mvar}(t_i) = z_i$,
(iii) t_i is monic with respect to z_i, and
(iv) $\deg_{z_j}(t_i) < \operatorname{mdeg}(t_j)$ for $j < i$.

The degree of T is $\prod_{i=1}^{n} \operatorname{mdeg}(t_i)$. Also, $T = \emptyset$ is a triangular set.

Condition (i) states there are no unused variables. We call such a T zero-dimensional, this can be shown to be equivalent to $\mathbf{V}(T)$ being finite, see Proposition 8 of section 5.3 of [7]. Condition (ii) gives a standard notation that will be used throughout this paper.

Conditions (iii) and (iv) relates the definition to that of minimal polynomials. Condition (iv) is commonly referred to as a reduced triangular set as seen in [2]. The degree of T is akin to the degree of an extension; that is, they are both equal to the dimension over k as a k-vector space. These extra assumptions were also used in [20].

Example 1. The polynomials $\{z_1^3+4z_1, z_2^2+(z_1+1)z_2+4\}$ form a triangular set. However, $\{z_2^2+(z_1+1)z_2+4\}$ would not since there is no polynomial with z_1 as a main variable. Also, $\{t_1 = z_1^3+4z_1,\ t_2 = z_2^2+z_1^4z_2+3\}$ is not because $\deg_{z_1}(t_2) = 4 > \mathrm{mdeg}(t_1)$.

Given a triangular set T, we define $T_i = \{t_1, \ldots, t_i\}$ and $T_0 = \emptyset$. For example, let $T = \{z_1^3+1,\ z_2^3+2,\ z_3^3+3\}$. Then, $T_3 = T$, $T_2 = \{z_1^3+1, z_2^3+2\}$, $T_1 = \{z_1^3+1\}$. Since each t_i is monic in its main variable, $k[z_1, \ldots, z_k] \cap \langle T\rangle = \langle T_k\rangle$. This is known as an elimination ideal, see chapter 3 of [7] for more details.

The presence of zero-divisors presents many unforeseen difficulties that the following examples illustrate.

Example 2. It is possible for a monic polynomial to factor as two polynomials with zero-divisors as leading coefficients. Consider the triangular set $T = \{z_1^4+3z_1^2+2, z_2^3-z_2\}$. There are some obvious zero-divisors modulo T which come from the factorizations $t_1 = (z_1^2+1)(z_1^2+2)$ and $t_2 = (z_2-1)(z_2+1)z_2$, but a not so obvious one is

$$z_2^3 - z_2 = \left((z_1^2+2)z_2^2 - 1\right)\left((z_1^2+1)z_2^3 + z_2\right). \tag{3.1}$$

Factoring $z_2^3 - z_2 = (z_2^2-1)z_2$ is nicer because it creates a splitting $\langle T\rangle = \langle t_1, z_2^2-1\rangle \cap \langle t_1, z_2\rangle$ of triangular sets where the factorization in (3.1) would not. This greatly enhances the complexity of handling zero-divisors. Equation (3.1) also shows that the degree formula for the product of two polynomials does not hold in this setting.

Example 3. Another difficulty is that denominators in the factors of a polynomial $a(x) \in R[x]$ may not appear in the denominators of $a(x)$. For instance, let $t_1(z_1) = z_1^2 - 5$. Then,

$$x^2 + x - 1 = (x - \tfrac{1}{2}z_1 + \tfrac{1}{2})(x + \tfrac{1}{2}z_1 + \tfrac{1}{2}).$$

In the special case when R is an algebraic number field, the denominator of any factor $f(x)$ of $a(x)$ ($\mathrm{denom}(f) = 2$ in this example) must divide the defect d of the R, as noted in [12]. It is known that the discriminant Δ of the defining polynomial of R is a multiple of d, usually, much larger than d ($\Delta = 20$ in this example). Although one could try to generalize the discriminant to the case $n > 1$, we handle fractions with a different approach. In particular, Lemma 32 in section 5.1 gives a characterization of primes appearing in denominators of monic factors of monic polynomials modulo a radical triangular set. Also, see [26] for more details considering defects and discriminants.

3.2 Arithmetic Modulo Triangular Sets

To start, we prove a tight bound on the number of field multiplications it takes to multiply two polynomials and reduce modulo a triangular set. We assume the inputs are reduced. We will need this later when doing an asymptotic analysis of the modular algorithms.

Let δ be the degree of a triangular set T with n variables. To multiply two polynomials modulo a triangular set, the obvious approach is to multiply out the polynomials and then reduce. The reduction step involves doing divisions by the polynomials in the triangular set. The way these divisions are carried out leaves a large impact on the total number of operations. We would like to illustrate by describing the classical approach as outlined in [20]. We will assume a and b are reduced and dense in all variables. Let $d_i = \text{mdeg}(t_i)$ for all i. First, view a and b as polynomials in z_n with coefficients modulo T_{n-1}. Multiplying ab modulo T_{n-1} involves recursively multiplying all pairs of coefficients from a and b and reducing modulo T_{n-1}. There are d_n^2 such pairings and the process yields a polynomial c with $\deg_{z_n}(c) = 2(d_n - 1)$ and coefficients that are reduced with respect to T_{n-1}. Next, we have to divide c by t_n. If one uses the high school division algorithm, this involves scaling d_n coefficients of t_n for $\deg_{z_n}(c) - d_n + 1$ iterations for a total of $d_n(d_n - 1)$ recursive multiplications modulo T_{n-1}.

Let $M(n)$ be the number of field multiplications used during a multiplication of a and b modulo T. The algorithm described above does $d_n^2 + d_n(d_n - 1)$ multiplications modulo T_{n-1} each costing $M(n-1)$ field multiplications. This gives a recurrence

$$M(n) \leq (d_n^2 + d_n(d_n - 1))M(n-1)$$

If there are no extensions, it takes a single field multiplication so that $M(0) = 1$. It is straightforward to solve this to get $M(n) = O(2^n\delta^2)$. This is as stated in [20] for the classical multiplication algorithm. We show that it can in be done in $O(\delta^2)$ field multiplications in Proposition 14. It should be noted that one normally assumes $\text{mdeg}(t_i) \geq 2$ since extensions by linear polynomials are trivial. With that in mind, the classical multiplication algorithm is $O(\delta^3)$. We mention this because it should be clear that our result does not turn an exponential-time algorithm to a quadratic one, but rather cubic to quadratic.

We would like to note that we have done an actual field multiplication count (in our code) and got the exact same result in the dense case as the proposition states. The key idea of the optimization is to do as little recursive reductions as possible and was originally done by Monagan for the ring $\mathbb{Z}_n$ with n too big for a single machine word, see [22].

Proposition 14. Let $M(n)$ be the number of field multiplications required to multiply $a, b \in k[z_1, \ldots, z_n]/T$ and reduce by the triangular set T. Let $\text{mdeg}(t_i) = d_i$ and define

$\delta_1 = d_1$, $\delta_2 = d_1 d_2$, and so on ending with $\delta_n = d_1 d_2 \cdots d_n = \delta$. Then

$$M(n) \leq \delta_n^2 + \sum_{k=1}^{n} \delta_k^2 \frac{d_k - 1}{d_k} \prod_{j=k+1}^{n} (2d_j - 1) \tag{3.2}$$

which is exact in the dense case. Further, $M(n) \leq 3\delta^2$.

Proof. Let $D(n)$ be the number of field multiplications it takes to reduce a polynomial of degree $2(d_j - 1)$ in each corresponding variable by T_n. It is assumed that $D(n)$ works by first reducing by t_1, then reducing by t_2 modulo T_1, etc. Well, multiplying ab will always take δ_n^2 multiplications before reducing, and this is true whether or not these multiplications are done recursively or at the on-set; in particular, it follows that $M(n) = \delta_n^2 + D(n)$. We proceed by describing a division algorithm to divide $c = ab$ by t_n modulo T_{n-1}. Let $c = c_0 + c_1 z_n + \cdots + c_{2(d_n-1)} z_n^{2(d_n-1)}$ and $t_n = p_0 + p_1 z_n + \cdots + z_{d_n}^{d_n}$. We can compute the quotient $q = q_0 + \cdots + q_{d_n-2} z_n^{d_n-2}$ and remainder $r = r_0 + \cdots + r_{d_n-1} z_n^{d_n-1}$ via the linear system generated by $r = c - t_n q$,

$$\begin{aligned}
q_{d_n-2} &= c_{2d_n-2}, \\
q_{d_n-3} &= c_{2d_n-3} - q_{d_n-2} p_{d_n-1}, \\
q_{d_n-4} &= c_{2d_n-4} - q_{d_n-3} p_{d_n-1} - q_{d_n-2} p_{d_n-2}, \\
&\vdots \\
q_0 &= c_{d_n} - q_1 p_{d_n-1} - \cdots - q_{d_n-2} p_2, \\
r_{d_n-1} &= c_{d_n-1} - q_0 p_{d_n-1} - q_1 p_{d_n-2} - \cdots - q_{d_n-2} p_1, \\
r_{d_n-2} &= c_{d_n-2} - q_0 p_{d_n-2} - q_1 p_{d_n-3} - \cdots - q_{d_n-3} p_0, \\
&\vdots \\
r_1 &= c_1 - q_0 p_1 - q_1 p_0, \\
r_0 &= c_0 - q_0 p_0.
\end{aligned}$$

We will outline a method to solve the equations above. The key idea is to compute the entire right-hand-side of the equations before reduction. First, set $q_{d_n-2} = c_{2d_n-2}$ and reduce by T_{n-1}. Then, multiply $q_{d_n-2} p_{d_n-2}$ over k and subtract it from c_{2d_n-3}, and then reduce by T_{n-1} to obtain q_{d_n-3}. It should be clear how to generalize this result and compute all q_k. Next, to get r_k, simply multiply the corresponding $q_i p_j$ over k and end by reducing the result of the sum by T_{n-1}. This reveals we only have to do a single reduction per equation and each reduction is of a polynomial of degree at most $2(d_j - 1)$ in the corresponding variable; this takes at most $(2d_n - 1)D(n-1)$ field multiplications. Multiplying each $q_i p_j$

will take δ_{n-1}^2 field multiplications each. This takes

$$(1 + 2 + \cdots + (d_n - 2))\delta_{n-1}^2 = \binom{d_n - 1}{2}\delta_{n-1}^2$$

field multiplications for computing all q_i in the top $d_n - 1$ rows, and

$$((1 + 2 + \cdots + (d_n - 1) + (d_n - 1))\delta_{n-1}^2 = \left(\binom{d_n}{2} + d_n - 1\right)\delta_{n-1}^2$$

field multiplications for computing all r_i in the bottom d_n rows. Overall,

$$\begin{aligned} D(n) &= (2d_n - 1)D(n-1) + \Big(\frac{(d_n - 1)(d_n - 2)}{2} + \frac{(d_n(d_n - 1)}{2} + d_n - 1\Big)\delta_{n-1}^2 \\ &= (2d_n - 1)D(n-1) + d_n(d_n - 1)\delta_{n-1}^2. \end{aligned}$$

If there are no extensions, it takes 0 multiplications to reduce; so we may use $D(0) = 0$ as our initial condition. The solution can be found most easily using Maple's `rsolve` command and some algebraic simplification. The command is

```
rsolve({M(n) = (2*d[n]-1)*M(n-1) + d[n]*(d[n]-1)*del(n-1)^2, del(n)=del(n-1)*d[n],
M(0)=0, del(1)=d[1]}, {M(n), del(n)});
```

For the lighter bound, we claim $D(n) \leq 2\delta_n^2$. To prove this, proceed by induction on n. The base case $n = 0$ follows from $D(0) = 0 \leq 2 = 2\delta_0$. Next,

$$\begin{aligned} D(n) &= (2d_n - 1)D(n-1) + d_n(d_n - 1)\delta_{n-1}^2 \\ &\leq (2d_n - 1)2\delta_{n-1}^2 + d_n(d_n - 1)\delta_{n-1}^2 \\ &= 4d_n\delta_{n-1}^2 - 2\delta_{n-1}^2 + d_n(d_n - 1)\delta_{n-1}^2 \\ &= 4d_n\delta_{n-1}^2 - 2\delta_{n-1}^2 + d_n^2\delta_{n-1}^2 - d_n\delta_{n-1}^2 \\ &= 3d_n\delta_{n-1}^2 - 2\delta_{n-1}^2 + \delta_n^2 \\ &= (3d_n - 2)\delta_{n-1}^2 + \delta_n^2. \end{aligned}$$

To finish, note that $3d_n - 2 \leq d_n^2$ which follows from $d_n^2 - 3d_n + 2 = (d_n - 2)(d_n - 1) \geq 0$ for all $d_n \in \mathbb{Z}$. Thus, $D(n) \leq d_n^2\delta_{n-1}^2 + \delta_n^2 = 2\delta_n^2$ and indeed $M(n) \leq \delta_n^2 + D(n) \leq 3\delta_n^2$. □

In [20], Li et al prove that multiplication can be done in $O(4^n\delta\log(\delta)\log(\log(\delta)))$ field operations. Their method computes the coefficients by lifting modulo the ideal $\langle x_n\rangle$ using Newton-iteration, and computes the coefficients of $x_1, \ldots, x_{n-1}$ recursively. However, the algorithm outlined in the proof above is what we are actually using, so it is what we should be counting. It should also be noted that the constant in their algorithm is much larger than the one from ours, so one would expect ours to perform better for smaller inputs. Comparing these two quantities is not obvious. To aid the reader we compare our bound

(3.1) with theirs in the table below. Because they do not give an explicit constant, we use 3 since their proofs ensure it is smaller. The table considers extensions of degree $\delta = d^n$ with $\text{mdeg}(t_i) = d$. We give the smallest value of n such that our bound exceeds theirs.

d	n	$\delta = d^n$
5	29	186264514923095703125
6	14	78364164096
7	10	282475249
8	8	16777216
9	6	531441
10	5	100000
12	4	20736
16	3	4096
28	2	784
115	1	115

Table 3.1: The first column is the main degree of each t_i, the second is smallest number of extensions where our bound exceeds the bound given in [20], the third is the degree of this extension $\delta = d^n$. Values of d that are omitted have the same value of n as the largest shown predecessor. For $d < 5$, our bound is always smaller. For $d \geq 115$ their bound is always smaller.

Of course the bounds do not say it all. The algorithm in [20] does a lot of extra processing (such as evaluations/interpolations, reversing coefficients of a polynomial, truncating power series) so their constant is much larger than 3, which we used. Also, it should be noted that one can not expect to ever efficiently compute over extensions of degree larger than 10^6 with modern computers; view our timing tests in section 5.3 and 6.2 for evidence of this.

Next, we would like to go over a field multiplication count for the other arithmetic operations assuming classical quadratic multiplication: division, inversion, and GCDs via the Euclidean algorithm. We will not get an exact count as in Proposition 14, instead just focus on asymptotics. We will need these when analyzing the modular gcd algorithm and the inversion algorithm later. We will be using the extended Euclidean algorithm for computing inverses here, and will only need this result when the field is $\mathbb{Z}_p$. When using the Euclidean algorithm, we need to assume no zero-divisors are encountered.

Proposition 15. Let $T \subset k[z_1, \ldots, z_n]$ be a triangular set and $R = k[z_1, \ldots, z_n]/T$. Let $a, b \in R[x]$ with $\deg(a) \geq \deg(b)$ and b monic. Then the remainder and quotient of $a \div b$ can be computed in $O(\deg(b)(\deg(a) - \deg(b) + 1)\delta^2)$ field multiplications.

Proof. The standard division works by multiplying the coefficients of b modulo T by an element of R for at most $\deg(a) - \deg(b) + 1$ iterations. There are $\deg(b)$ coefficients of b not including the leading coefficient; note that we ignore $\text{lc}(b)$ since we are assuming b is monic. This implies that we need to do $\deg(b)(\deg(a) - \deg(b) + 1)$ ring multiplications. We can do ring multiplications in $O(\delta^2)$ field multiplications by Proposition 14, giving the result. $\square$

Proposition 16. Let $T \subset k[z_1, \ldots, z_n]$ be a triangular set and $R = k[z_1, \ldots, z_n]/T$. Assume inverses in k can be computed in a $O(1)$ field multiplications. Let $a \in R$. Then a^{-1} can be computed in $O(\delta^2)$ field multiplications. We will be using the extended Euclidean algorithm in a and t_n modulo T_{n-1} to compute a^{-1} and we will assume no zero-divisors are encountered.

Proof. Work by induction on n. Note that our assumption on k satisfies the base case $n = 0$. Next, let $\delta_m = \prod_{i=1}^{m} \deg(t_i)$ as in Proposition 14. Let $I(n)$ be the number of field multiplications it takes to compute the inverse of an element with n variables. Then the first step of the Euclidean algorithm is to invert $\mathrm{lc}(a)$. After that, we would have to invert the leading coefficient of the remainder of $t_n \div a$. Since the worst case is the degree of each successive remainder going down by 1, this will take a total of at most $\deg(a) = \deg(t_n) - 1$ recursive inversions. By Proposition 15, this will take $O((\deg(t_n) - 1)\delta_{n-1}^2)$ field multiplications, the next remainder will take $O((\deg(t_n) - 2)\delta_{n-1})$ field multiplications, and so on. In total,

$$I(n) = \deg(t_n)I(n-1) + \sum_{j=1}^{\deg(t_n)-1} O(j\delta_{n-1}) = \deg(t_n)I(n-1) + O(\deg(t_n)^2\delta_{n-1}).$$

Note that we also have to multiply through by the inverse of the leading coefficient at each step. This will take $O(\deg(t_n)^2\delta_{n-1})$ over all steps as well.

Now, the induction hypothesis states $I(n-1) = O(\delta_{n-1}^2)$. So,

$$I(n) = \deg(t_n)I(n-1) + O(\delta^2) = \deg(t_n)O(\delta_{n-1}^2) + O(\delta^2) = O(\delta^2),$$

completing the inductive step. We have not counted the extra multiplications in the extended Euclidean algorithm, but this does not impact the asymptotics; see Theorem 3.11 of [13]. □

Proposition 17. Let $T \subset k[z_1, \ldots, z_n]$ be a triangular set and $R = k[z_1, \ldots, z_n]/T$. Let $a, b \in R[x]$ with $\deg(a) \geq \deg(b)$. Then running the Euclidean algorithm on a and b takes $O(d_a d_b \delta^2)$ field multiplications assuming no zero-divisors are encountered.

Proof. Let $d_a = \deg(a)$ and $d_b = \deg(b)$. We will have to perform at most d_b remainders to complete the Euclidean algorithm. This implies we need to invert d_b leading coefficients as well as $\mathrm{lc}(b)$. This accounts for $O(d_b\delta^2)$ field multiplications. Multiplying through by the leading coefficients will cost $O(d_b^2\delta^2) \leq O(d_a d_b \delta^2)$ field multiplications since each remainder has degree $\leq d_b$ and there are d_b of them. Next, computing all but the first remainder cost a total of $O(d_b^2\delta^2)$ field multiplications since each remainder has $\leq d_b$ degree and there are d_b of them in the worst case. Finally, the first remainder costs $O(d_b(d_a - d_b + 1)\delta^2)$ field multiplications. Thus, the entire cost is $O(d_a d_b \delta^2 + d_b(d_a - d_b + 1)\delta^2) = O(d_a d_b \delta^2)$. □

3.3 Radical Triangular Sets

We turn our attention now to radical triangular sets. We start with a useful structure theorem and some corollaries. Theorem 18 is well-known in the literature, but we give a new proof. The more standard proof is based on a concept known as associated primes to an ideal, see Proposition 4.7 in [17]. Our proof is a good example of how we view triangular sets from an algebraic number theory perspective.

Theorem 18. Let $T \subseteq k[z_1, \ldots, z_n]$ be a zero-dimensional triangular set and $R = k[z_1, \ldots, z_n]/T$. Then R is isomorphic to a direct product of fields if and only if $\langle T \rangle$ is radical.

Proof. ($\Longleftarrow$) Use induction on the number of variables n. If $n = 1$, let $t_1 = a_1 a_2 \cdots a_m$ be the monic irreducible factorization of t_1. Since $\langle t_1 \rangle$ is radical, t_1 must be square-free. By the CRT, $k[z_1]/\langle t_1 \rangle \equiv \prod_{i=1}^{m} k[z_1]/\langle a_i \rangle$. Since each a_i is irreducible, $k[z_1]/\langle a_i \rangle$ is a field and so $k[z_1]/\langle t_1 \rangle$ is isomorphic to a product of fields.

We proceed with the inductive step. I claim $\langle T_{n-1} \rangle$ is radical. To see this, suppose $g^k \in \langle T_{n-1} \rangle$. Then, $g^k \in \langle T \rangle$ which implies $g \in \langle T \rangle$ by $\langle T \rangle$ being radical. Also, $g \in k[z_1, \ldots, z_{n-1}]$ by assumption. Therefore, $g \in \langle T_{n-1} \rangle$. So, $k[z_1, \ldots, z_{n-1}]/T_{n-1} \cong \prod F_i$ where F_i are fields by the induction hypothesis. We can extend this isomorphism to

$$(k[z_1, \ldots, z_{n-1}]/T_{n-1})[z_n] \cong (\prod F_i)[z_n] = \prod F_i[z_n].$$

Observe that

$$R = k[z_1, \ldots, z_n]/T = (k[z_1, \ldots, z_{n-1}]/T_{n-1})[z_n]/\langle t_n \rangle \cong (\prod F_i[z_n])/\langle t_n \rangle \cong \prod F_i[z_n]/\langle t_n \rangle$$

where we are assuming that t_n is reduced appropriately in F_i. I claim that t_n is square-free and nonzero in $F_i[z_n]$. Being nonzero simply follows from t_n being monic. Next, if t_n was not square-free in $F_i[z_n]$, there would be some $g^k \mid t_n$ in F_i. But this would imply $F_i[z_n]/t_n$ has a nilpotent element, contradicting that T is radical. Thus, each $F_i[z_n]/t_n$ will be isomorphic to a product of fields by the base case. This completes the inductive step and this direction of the proof.

($\Longrightarrow$) Since $k[z_1, \ldots, z_n]/\langle T \rangle \cong \prod F_i$ is isomorphic to a direct product of fields, it will have no nilpotent elements. Thus, $\langle T \rangle$ is radical. □

One important reason that we restrict the modular gcd algorithm to radical triangular sets is that it is sufficient for gcds to exist for any two polynomials; this is proven in Corollary 19. Next, we prove a result in Corollary 20 about the extended Euclidean representation modulo a radical triangular set. It should be noted that it works even if $\mathrm{lc}(g)$ is a zero-divisor, or (more generally) if the Euclidean algorithm would encounter a zero-divisor. This shows it is more powerful than the extended Euclidean algorithm. After that we prove that all nonzero elements are either units or zero-divisors and a version of the division algorithm.

Corollary 19. Let $T \subset k[z_1, \ldots, z_n]$ be a radical, zero-dimensional triangular set and $R = k[z_1, \ldots, z_n]/T$. Let $a, b \in R[x]$. Then a greatest common divisor of a and b exists.

Proof. Let $R[x] \cong \prod F_i[x]$ where each F_i is a field. Let $a \mapsto (a_i)_i$ and $b \mapsto (b_i)_i$. Since $F_i[x]$ is a Euclidean domain, $g_i = \gcd(a_i, b_i)$ certainly exists. Well, the polynomial $g \mapsto (g_i)_i$ is a gcd of a and b in $R[x]$ by Lemma 10. □

Corollary 20 (Extended Euclidean Representation)**.** Let $T \subset k[z_1, \ldots, z_n]$ be a radical, zero-dimensional triangular set and $R = k[z_1, \ldots, z_n]/T$. Let $a, b \in R[x]$ with $g = \gcd(a, b)$. Then, there exists polynomial $A, B \in R[x]$ such that $aA + bB = g$.

Proof. Note that $R \cong \prod F_i$ where F_i is a field, and we can extend this to $R[x] \cong \prod F_i[x]$. Let $a \mapsto (a_i)_i$ and $b \mapsto (b_i)_i$ under this isomorphism. Define $h_i = \gcd(a_i, b_i)$ in $F_i[x]$. By the extended Euclidean algorithm, there exists $A_i, B_i \in F_i[x]$ such that $a_i A_i + b_i B_i = h_i$. Let $h, A, B \in R[x]$ be the unique polynomials such that $h \mapsto (h_i)_i$ and $A \mapsto (A_i)_i$ and $B \mapsto (B_i)_i$ so that $aA + bB = h$. Since $h \mid g$, we can multiply through by the quotient to write g as a linear combination of a and b. □

Corollary 21. Let T be a radical triangular set of $k[z_1, \ldots, z_n]$. Then every nonzero element in R is either a zero-divisor or a unit.

Proof. Since T is a radical triangular set, so is T_{n-1}. In particular, $g = \gcd(a, t_n) \pmod{T_{n-1}}$ exists. Note that there exists $\overline{a} \in R$ where $g\overline{a} = a$. If g is a zero-divisor with cofactor h, then $ah = g\overline{a}h = 0 \pmod{T}$. If g is a unit with $gh = 1$, then Corollary 20 gives polynomials $A, B \in R$ where $aA + t_n B = g \pmod{T_{n-1}}$ and so $a(hA) + t_n(Bh) = 1 \pmod{T_{n-1}}$. Modding out by t_n gives $a(hA) = 1 \pmod{T}$. Therefore, it is enough to show that g is either a unit or zero-divisor.

As $g \mid t_n$, there exists $q \in R$ where $gq = t_n \pmod{T_{n-1}}$. Well, this gives g is a zero-divisor unless $q \equiv 0 \pmod{T}$. This in turn would imply $t_n \mid q \pmod{T_{n-1}}$, and hence there exists $Q \in R$ where $gt_nQ = t_n \pmod{T_{n-1}}$. However, t_n is monic and so can not be a zero-divisor modulo T_{n-1} which means $gQ - 1 = 0 \pmod{T_{n-1}}$. Therefore, g is a unit and a is either a unit or zero-divisor. □

Corollary 21 can also be proven by noting that R is a finite-dimensional algebra over a field. In particular, the radical assumption is not actually required. It can also be seen trivially in the radical case by examining elements in a product of fields. However, our proof reveals the connection between gcds and zero-divisors.

Corollary 22 (Division Algorithm)**.** Let $T \subset k[z_1, \ldots, z_n]$ be a radical, zero-dimensional triangular set and $R = k[z_1, \ldots, z_n]/T$. Let $a, b \in R[x]$ with $\deg(a) \geq \deg(b) \geq 1$. Suppose b is not a zero-divisor. Then there exists a quotient q and remainder r satisfying $a = bq + r$ and $\deg(r) < \deg(b)$. The remainder r is unique if and only if $\mathrm{lc}(b)$ is a unit.

Proof. Note that $R \cong \prod F_i$ is isomorphic to a product of fields, and we can extend this to $R[x] \cong \prod F_i[x]$. Let $a \mapsto (a_i)_i$ and $b \mapsto (b_i)_i$ under this isomorphism. Note that all $b_i \neq 0$ or else b would a zero-divisor. Apply the division algorithm over a field to get q_i, $r_i \in F_i[x]$ with $a_i = q_i b_i + r_i$ and $r_i = 0$ or $\deg(r_i) < \deg(b_i)$. The element $r \mapsto (r_i)_i$ will have degree less than b since all of its images do. This shows existence.

For uniqueness, first let $\mathrm{lc}(b)$ be a unit. Suppose $a = bq_1 + r_1$ and $a = bq_2 + r_2$. Then, $b(q_1 - q_2) = r_2 - r_1$. Since $\mathrm{lc}(b)$ is a unit, $\deg(b(q_1 - q_2)) \geq \deg(b)$ unless $q_1 - q_2 = 0$. Clearly, $\deg(r_2 - r_1) < \deg(b)$ which leaves $q_1 - q_2 = 0$ as the only possibility. It follows that $r_1 = r_2$. Conversely, suppose $\mathrm{lc}(b)$ is not a unit, and hence a zero-divisor by Corollary 21. Let $v \in R$ be such that with $\mathrm{lc}(b)v = 0$. Note that $\deg(bv) < \deg(b)$. Given a remainder r and quotient q, algebraic manipulation gives

$$a = bq + r = (q - v)b + bv + r.$$

Since $\deg(bv + r) < \deg(b)$, this gives two distinct remainders as long as $bv \neq 0$, which must be the case as b is not a zero-divisor. □

We now move to the special cases when $k = \mathbb{Q}$ and $k = \mathbb{Z}_p$. To motivate, suppose we are working with a radical triangular set $T \subset \mathbb{Q}[z_1, \ldots, z_n]$. The modular algorithm presented later will work by turning a problem over $\mathbb{Q}$ into problems modulo multiple prime numbers p_i. However, because we would like to use the powerful corollaries above, we need $T \mod p_i$ to still be radical. This does not always occur. For example, the triangular set $\{z_1^2 - 3\}$ is radical over $\mathbb{Q}$, but is not over $\mathbb{Z}_3$ or $\mathbb{Z}_2$. This motivates the definition of a radical prime.

Definition 13. Let $T \subset \mathbb{Q}[z_1, \ldots, z_n]$ be a radical triangular set. A prime number p is a radical prime with respect to T if p does not appear as a denominator of any of the polynomials in T, and if $T \mod p \subset \mathbb{Z}_p[z_1, \ldots, z_n]$ remains radical.

If there were an infinite family of nonradical primes, it would present a problem for the algorithm. We prove this can not happen. This has also been proven with quantitative bounds in section 3 of [10]. We need the following lemma, which is a restatement of Corollary 7.3 of [17], but we give an original proof that uses Theorem 18. It also serves as the main idea of our algorithm for testing if a prime is radical; see IsRadicalPrime below.

Lemma 23. Let $T \subset k[z_1, \ldots, z_n]$ be a triangular set where k is a finite field or of characteristic 0. Then T is radical if and only if $\gcd(t_i, t_i') = 1 \pmod{T_{i-1}}$ for all i.

Proof. ($\Longrightarrow$) Let $j \leq n$ be a positive integer. Note that T_{j-1} is radical so

$$k[z_1, \ldots, z_j]/T_{j-1} \cong \prod F_i[z_j]$$

for some finite field extensions $k \to F_i$. Let $t_j \mapsto (t_{ji})$ under this isomorphism. I claim t_{ji} is square-free over F_i. If it were not, then $F_i[z_j]/\langle t_{ji} \rangle$ would contain a nilpotent element and

so $k[z_1, \ldots, z_j]/T_j$ would as well, contradicting that T_j is radical. Since k is a finite field or of characteristic 0, so is each F_i as $k \to F_i$ is finite. For the sake of contradiction, let $g \in F_i[z_n]$ be an irreducible factor of both t_{ji} and t'_{ji}. Then we may write $t = ga$ for some $a \in F_i[z_n]$. Applying the derivative operator gives $t' = a'g + g'a$. Since $g \mid t'$, we get $g \mid g'a$. If F_i is of characteristic 0, then $\gcd(g, g') = 1$ because g is irreducible. This gives $g \mid a$, but then $g^2 \mid t_{ji}$, contradicting that t_{ji} is square-free. Now suppose F_i is a finite field and so has prime characteristic p. We can apply the same proof unless $g' = 0$. This can only happen if g is of the form $g = g_0 + g_1 z_j^p + \cdots + g_m z_j^{pm}$ for some positive integer m. Note that the Frobenius mapping $\varphi\colon F_i \to F_i$ where $\varphi(x) = x^p$ is a homomorphism. Further, it is injective since F_i is a field and surjective because F_i is a finite field. Thus, there exists $h_l \in F_i$ where $h_l^p = g_l$. In particular, $g = (h_0 + h_1 z_j + \cdots + h_m z_j^m)^p$, contradicting that g is irreducible. Thus, the only possible case is $\gcd(t_{ji}, t'_{ji}) = 1$ over F_i.

($\Longleftarrow$) Proceed by induction on n. If $n = 1$, then $T = \{t_1(z_1)\}$. If t_1 and t'_1 are relatively prime, then t_1 is square-free and so T is radical. Now, the induction hypothesis asserts T_{n-1} is radical, and so $k[z_1, \ldots, z_{n-1}/T_{n-1} \cong \prod F_i$ where F_i are finite extensions of k. Let $t_n \mapsto (t_{ni})$ under this isomorphism. Let $g_i = \gcd(t_{ni}, t'_{ni})$ over F_i and $g \in R$ satisfy $g \mapsto (g_i)$ under the isomorphism. Then, g would be a common divisor of $\gcd(t_n, t'_n) = 1$ and so would be a unit. Since homomorphisms preserve units, g_i is a unit and we may assume $g_i = 1$ without loss. It follows that t_{ni} is square-free over F_i by the base case. Then, $F_i[z_n]/t_{ni}$ contains no nilpotent elements, and so $k[z_1, \ldots, z_n]/T \cong \prod F_i[z_n]/t_{ni}$ contains no nilpotents as well. □

Lemma 23 also gives an algorithm for testing if a prime p is radical. It may not always output `True` or `False` as Lemma 23 uses a gcd computation, which, if computed using the monic Euclidean algorithm over a ring, may encounter a zero-divisor. If this happens, we output the zero-divisor with an error message. This case is caught later in the modular gcd and inversion algorithms, of which IsRadicalPrime is a subroutine.

Algorithm 3: IsRadicalPrime

Input : A zero-dimensional, radical triangular set $T \subset \mathbb{Q}[z_1, \ldots, z_n]$ and a prime number p that does not divide any denominator of any coefficient of any $t_i \in T$.

Output: A boolean indicating if T remains radical modulo p, or a zero-divisor.

```
for i = 1, ..., n do
    dt := ∂/∂z_i T[i];
    g := gcd(T[i], dt) over Z_p[z_1, ..., z_i]/T_{i-1};
    if g = ["ZERODIVISOR", u] then return ["ZERODIVISOR", u];
    if g ≠ 1 then return False;
end
return True;
```

Lastly, we prove that all but finitely many primes are radical. This is important since our modular gcd and inversion algorithms will skip nonradical primes.

Theorem 24. Let $T \subset \mathbb{Q}[z_1, \ldots, z_n]$ be a radical, zero-dimensional triangular set. All but finitely many primes are radical primes.

Proof. By Lemma 23, $\gcd(t_i, t_i') = 1$. By the extended Euclidean representation (Corollary 20), there exist polynomials $A_i, B_i \in (\mathbb{Q}[z_1, \ldots, z_{i-1}]/T_{i-1})[z_i]$ where $A_i t_i + B_i t_i' = 1 \pmod{T_{i-1}}$. Take any prime p that does not divide the denominator of any A_i, B_i, t_i, t_i'. This means one can reduce this equation modulo p and so $A_i t_i + B_i t_i' \pmod{T_{i-1}, p}$. This implies $\gcd(t_i, t_i') = 1 \pmod{T_{i-1}, p}$ and so T remains radical modulo p by Lemma 23. There is only a finite number of primes that divide the denominator of any of these polynomials. $\square$

Chapter 4

Resolving Zero-Divisors

The point of this section is to go over the ways we have to handle zero-divisors. We give two methods: one based on Hensel lifting and one based on fault tolerant rational reconstruction. Suppose we encounter a zero-divisor w modulo a prime p. We call the process of bringing this to a zero-divisor u over $\mathbb{Q}$ *resolving* w.

4.1 Hensel Lifting

We turn our attention to lifting a factorization $f = ab \pmod{T, p}$ for $a, b, f \in R[x]$ where $R = \mathbb{Q}[z_1, \ldots, z_n]/T$. This will generalize the Hensel lifting technique outlined in section 2.4. Instead of lifting p-adic representations, we just lift to a specified bound B; that is, until $p^j \geq B$. Then, we apply rational reconstruction to get a polynomial over $\mathbb{Q}$ since factors may have fractions.

A general factorization will not be liftable; certain conditions are necessary for existence and uniqueness of each lifting step. For one, we will need $\gcd(a, b) = 1 \pmod{p}$ (as is required in the case with no extensions) to satisfy existence. Further, we will need both a and b to be monic to satisfy uniqueness. The following lemma gives a uniqueness criterion for the extended Euclidean representation. It generalizes Theorem 2.6 in [14] from $k[x]$ to $R[x]$ and follows the same proof. Note that a and b being monic is essential.

Lemma 25. Let $T \subset k[z_1, \ldots, z_n]$ be a radical triangular set and $R = k[z_1, \ldots, z_n]/T$. Let $a, b \in R[x]$ be monic and relatively prime. Then, for any polynomial $c \in R[x]$, there exist unique polynomials $\sigma, \tau \in R[x]$ such that

$$a\sigma + b\tau = c, \quad \deg(\sigma) < \deg(b).$$

Proof. Existence: The extended Euclidean representation (Corollary 20) gives $A, B \in R[x]$ where $aA + bB = 1$. Multiplying through $1 = Aa + Bb$ by c gives $a(cA) + b(cB) = c$. Dividing cA by b, which we can do since b is monic, gives $cA = qb + r$ with $r = 0$ or $\deg(r) < \deg(b)$.

Define $\sigma = r$ and $\tau = cB + qa$. Observe that

$$a\sigma + b\tau = ar + b(cB + qa) = ar + bcB + abq = a(r + bq) + bcB = acA + bcB = c(aA + bB) = c$$

thus σ and τ satisfy the conditions of the Lemma.

Uniqueness: Suppose both pairs σ_1, τ_1 and σ_2, τ_2 satisfy $a\sigma_i + b\tau_i = c$ with the desired degree constraint. This yields

$$(\sigma_1 - \sigma_2)a = b(\tau_2 - \tau_1).$$

Since $\gcd(a, b) = 1$ the extended Euclidean representation (Corollary 20) reveals there are $A, B \in R[x]$ where $Aa + Bb = 1$. Multiplying through by $\sigma_1 - \sigma_2$ reveals that $b \mid \sigma_1 - \sigma_2$. However, since b is monic and $\deg(\sigma_1 - \sigma_2) < \deg(b)$, this is only possible if $\sigma_1 - \sigma_2 = 0$. Thus $0 = b(\tau_2 - \tau_1)$. Next, since b is not a zero-divisor (because it is monic), this can only happen if $\tau_2 - \tau_1 = 0$ as well. □

We are particularly interested in trying to factor t_n modulo T_{n-1} because encountering a zero-divisor may lead to such a factorization; that is, if w is a zero-divisor with main variable z_n, we can write $u = \gcd(t_n, w)$ and then $t_n = uv \mod \langle T_{n-1} \rangle$ by the division algorithm. As long as T is radical, the next lemma shows we automatically get $\gcd(u, v) = 1$.

Lemma 26. Let $T \subset k[z_1, \ldots, z_n]$ be a radical, zero-dimensional triangular set. Suppose $t_n \equiv uv \pmod{T_{n-1}}$. Then, $\gcd(u, v) = 1 \pmod{T_{n-1}}$.

Proof. Let g be a common divisor of u and v over T_{n-1} so that $u = \overline{u}g \pmod{T_{n-1}}$ and $v = \overline{v}g \pmod{T_{n-1}}$. Note that $t_n \equiv \overline{uv}g^2 \pmod{T_{n-1}}$. This would imply $(\overline{uv}g)^2 \equiv 0 \pmod{T}$; that is, $\overline{uv}g$ is a nilpotent element. However, since nilpotent elements do not exist modulo a radical ideal, $\overline{uv}g \equiv 0 \pmod{T}$. This would imply $\overline{uv}g \equiv qt_n \pmod{T_{n-1}}$ for some polynomial q. Then,

$$(gq - 1)t_n \equiv gqt_n - t_n \equiv g\overline{uv}g - t_n \equiv 0 \pmod{T_{n-1}}.$$

Since t_n is monic in z_n, it can not be a zero-divisor modulo T_{n-1}. Therefore, $gq - 1 \equiv 0 \pmod{T_{n-1}}$. Thus, g is a unit modulo T_{n-1} and so indeed $\gcd(u, v) = 1 \pmod{T_{n-1}}$. □

Finally, the next proposition shows that lifting is possible. The proof given is simply the Hensel construction.

Proposition 27. Let $T \subset \mathbb{Z}_p[z_1, \ldots, z_n]$ be a zero-dimensional radical triangular set with p a prime number. Suppose $t_n \equiv u_0 v_0 \pmod{T_{n-1}, p}$ where u_0 and v_0 are monic. Then, there exist unique monic polynomials $u_j, v_j \in \mathbb{Z}_{p^j}[z_1, \ldots, z_n]$ such that $t_n \equiv u_j v_j \pmod{T_{n-1}, p^j}$ and $u_j \equiv u_0 \mod \pmod{T_{n-1}, p}$ and $v_j \equiv v_0 \mod \pmod{T_{n-1}, p}$ for all $j \geq 1$.

Proof. (by induction on j): The base case is clear. For the inductive step, we want to be able to write $u_j = u_{j-1} + p^{j-1}a \pmod{T_{n-1}, p^j}$ and $v_j = v_{j-1} + p^{j-1}b \pmod{T_{n-1}, p^j}$ satisfying

$$t_n \equiv u_j v_j \pmod{T_{n-1}, p^j}.$$

Multiplying out u_j, v_j gives

$$t_n \equiv u_j v_j \equiv u_{j-1}v_{j-1} + p^{j-1}(av_{j-1} + bu_{j-1}) \pmod{T_{n-1}, p^j}.$$

Subtracting $u_{j-1}v_{j-1}$ on both sides and dividing through by p^{j-1} gives

$$\frac{t_n - u_{j-1}v_{j-1}}{p^{j-1}} \equiv av_0 + bu_0 \pmod{T_{n-1}, p}.$$

Let $c = \frac{t_n - u_{j-1}v_{j-1}}{p^{j-1}}$. By Lemma 25 using T_{n-1} as the triangular set and z_n as the indeterminate, there exists unique polynomials σ, τ such that $u_0\sigma + v_0\tau \equiv c \pmod{T_{n-1}, p}$ with $\deg(\sigma) < \deg(v_0)$ and $\deg(\tau) < \deg(u_0)$ since certainly $\deg(c) = \deg(t_n - u_{j-1}v_{j-1}) < \deg(t_n) = \deg(u_0) + \deg(v_0)$. Set $a = \tau$ and $b = \sigma$. Because of these degree constraints, $u_j = u_{j-1} + ap^{j-1}$ has the same leading coefficient as u_{j-1} and hence u_0; in particular u_j is monic. Similarly, v_j is monic as well. By uniqueness of σ and τ, we get uniqueness of u_j and v_j. □

What follows is a formal presentation of the Hensel construction. The algorithm HenselLift takes input $u_0, v_0, f \in R/\langle p\rangle[x]$ where u_0, v_0 are monic and $f = u_0v_0 \pmod{p}$. It also requires a bound B that is used to notify termination of the Hensel construction and output FAIL. A crucial part of the Hensel construction is solving the Diophantine equation $\sigma u_0 + \tau v_0 = c \pmod{T, p}$. This is done using the extended Euclidean algorithm and Lemma 25. It is possible that a zero-divisor is encountered in this process. This has to be accounted for. Therefore, we allow the HenselLift algorithm to also output ["ZERODIVISOR", u] if it encounters a zero-divisor $u \in R/\langle p\rangle$. See Algorithm 4 for pseudo-code.

In general the input f will have fractions thus the error e in our Hensel lifting algorithm will also have fractions and hence it can never become 0, which is why we have to use rational reconstruction in line 5.

The standard implementation of Hensel lifting requires a bound on the coefficients of the factors of the polynomial $f \in R[x]$. For the base case $n = 0$ where $R[x] = \mathbb{Q}[x]$ one can use the Mignotte bound (see [13]). For the case $n = 1$ Weinberger and Rothschild [26] give a bound but note that it is large. We do not know of any bounds for the general case $n > 1$ and hypothesize that they would be bad. Therefore a more "engineering"-esque approach is needed. Since we do not know whether the input zero-divisor a_0 is the image of a monic factor of f, we repeat the Hensel lifting each time a zero-divisor is encountered in our modular GCD algorithm, first using a bound of 2^{60}, then 2^{120}, then 2^{240} and so

Algorithm 4: HenselLift

Input : A zero-dimensional radical triangular set $T \subset \mathbb{Q}[z_1, \ldots, z_n]$, a radical prime p, polynomials $f \in R[x]$ and $a_0, b_0 \in R/\langle p \rangle[x]$ where $R = \mathbb{Q}[z_1, \ldots, z_n]/T$, and a bound B. Further, assume $f \equiv a_0 b_0 \pmod{p}$ and $\gcd(a_0, b_0) = 1$.

Output: Either polynomials $a, b \in R[x]$ where $f = ab$, FAIL if the bound B is reached, or ["ZERODIVISOR", w] if a zero-divisor $w \in R/\langle p \rangle$ is encountered.

```
1  Solve sa_0 + tb_0 = 1 using the monic extended Euclidean algorithm for s, t ∈ R/⟨p⟩[x];
2  if a zero-divisor w is encountered then return ["ZERODIVISOR", w];
3  Initialize u = a_0, v = b_0 and lift u and v from R/⟨p⟩ to R;
4  for i = 1, 2, ... do
5  |  Set a :=RationalReconstuction(u (mod p^i));
6  |  if a ≠ FAIL, and a|f then return a, f/a;
7  |  if p^i > 2B then return FAIL;
8  |  Compute e := f − uv as polynomials over ℚ;
9  |  Set c := (e/p^i) mod p ;
10 |  Solve σa_0 + τb_0 = c for σ, τ ∈ R/⟨p⟩[x] using sa_0 + tb_0 = 1;
11 |  Lift σ and τ from R/⟨p⟩ to R and set u := u + τp^i and v := v + σp^i;
12 end
```

on, until the coefficients of any monic factor of f can be recovered using rational number reconstruction.

The prime application of Hensel lifting will be as a solution to the zero-divisor problem. This is the goal Algorithm 5, HandleZeroDivisorHensel. The algorithm assumes a zero-divisor modulo a prime p has been encountered by another algorithm. It attempts to lift this zero-divisor using HenselLift. If HenselLift encounters a new zero-divisor w, it recursively calls HandleZeroDivisorHensel(w). If lifting fails (i.e., a bound is reached), it instructs the algorithm using it to pick a new prime, and a bigger bound will be used for the next zero-divisor. If lifting succeeds in finding a factorization $t_n = uv \pmod{T_{n-1}}$ over $\mathbb{Q}$, then the algorithm using it works recursively on new triangular sets $T^{(u)}$ and $T^{(v)}$ where t_n is replaced by u and v, respectively. Note that $T^{(u)}$ and $T^{(v)}$ are comaximal by Lemma 26 and hence both $T^{(u)}$ and $T^{(v)}$ are radical by Lemma 7.

It should be made clear that this is not the first case of using p-adic lifting techniques on triangular sets. In particular, lifting the triangular decomposition of a regular chain has been used by Dahan, Maza, Schost, Wu, Xie in [10].

4.2 Fault Tolerant Rational Reconstruction

When trying to recover a polynomial from $\mathbb{Z}_p$ to $\mathbb{Q}$, the techniques often employed in computer algebra are Chinese remaindering and Hensel lifting. Using Chinese remaindering here is not straightforward. To motivate, consider the following example:

Algorithm 5: HandleZeroDivisorHensel

Input : A zero-dimensional radical triangular set $T \subset \mathbb{Z}_p[z_1, \ldots, z_n]$ modulo a prime p and a zero-divisor $u_0 \in R$ where $R = \mathbb{Z}_p[z_1, \ldots, z_n]/T$. Assume $\text{mvar}(u) = n$.

Output: A message indicating the next steps that should be carried out, including any important parameters;

1 Set v_0 :=Quotient$(t_n, u_0) \pmod{T_{n-1}, p}$;
2 **if** $v_0 =$ ["ZERODIVISOR", w] **then return** HandleZeroDivisorHensel(w);
3 **if** the global variable B is unassigned **then** set $B := 2^{60}$ **else** set $B := B^2$;
4 Set u, v :=HenselLift(t_n, u_0, v_0, B);
5 **if** $u =$ ["ZERODIVISOR", w] **then return** HandleZeroDivisorHensel(w);
6 **else if** $u = FAIL$ **then return** *FAIL*. This indicates that a new prime or bigger bound is needed;
7 **else return** u and v;

Example 4. Consider the triangular set $T = z^2 + 14z + 24$ and polynomials $a = x^4 + x^3 + (z+3)\,x^2 + (z+4)\,x + 3\,z + 1$ and $b = x^2 + x + z$. The remainder of a divided by b modulo T is $(z+1)x + 1$. Here, $z+1$ is not a zero-divisor since $z^2 + 14z + 24 = (z+2)(z+12)$. But, if we are working modulo 11, it becomes a zero-divisor. Any attempt at combining zero-divisors using Chinese remaindering would fail if $p = 11$ was one of the chosen primes.

Abbott's new algorithm Fault Tolerant Rational Reconstruction in [1] circumvents this problem. It can still find the desired value if there are enough correct images. In particular, we use the heuristic algorithm HRR given in [1] which requires a 2-to-1 lucky to unlucky prime ratio. We would like to illustrate the algorithm with an example.

Example 5. Suppose we are trying to use the CRT and rational reconstruction to recover $f = \frac{5}{3}x^2 - \frac{14}{9}x - \frac{59}{18}y^3 - \frac{3}{2}y^2 + \frac{8}{9}y - \frac{5}{6}z^2$ from multiple modular images. Suppose the primes $71, 73, 79, 83, 89, 97, 101$ were used and all but 71 computed correct modular images of f. Let us say 71 computed $51x^2 + (54y+21)x + 23y^2 + 53yz + 43 \pmod{71}$ instead. Assuming x is the main variable, we cannot use degree constraints to determine that 71 is an unlucky prime. No matter how many primes are chosen, the CRT and rational reconstruction will never recover f if we include this unlucky prime. However, using the HRR algorithm from [1] circumvents this. We will demonstrate this with the following Maple session which uses Monagan's RECDEN package. The `phirpoly(f,p)` command reduces f mod a prime p.

```
> f;
```

$$-\frac{59}{18}y^3 + \frac{5}{3}x^2 - \frac{3}{2}x^2 - \frac{5}{6}z^2 - \frac{14}{9}x + \frac{8}{9}y$$

```
> m;
```

$$[71, 73, 79, 83, 89, 97, 101]$$

```
> f_p := [seq(phirpoly(f,m[i]), i = 1..7)]:
> f_p[1] := rpoly(51*x^2 + (54*y + 21)*x + 23*y^2 +53*y*z + 43, [x,y,z], 71);
```

$$f_p[1] := \left((51x^2 + (54y+21)\right)x + 23y^2 + 53yz + 43 \mod 71$$

```
> a := ichremrpoly(f_p):
> hrrrpoly(a);
```

$$-\frac{59}{18}y^3 + \frac{5}{3}x^2 - \frac{3}{2}x^2 - \frac{5}{6}z^2 - \frac{14}{9}x + \frac{8}{9}y$$

In regards to our use, recall the example from the introduction where $z_1 - z_2$ and $z_1 + z_2$ are zero-divisors, and the EA was run on inputs that results in the polynomial remainder sequence r_1, r_2 where $\text{lc}(r_1) = z_1 + 18z_2$ and $\text{lc}(r_2) = z_1 - z_2$. For any prime p besides 17, the monic Euclidean algorithm would terminate with the zero-divisor $z_1 - z_2 \pmod p$. When $p = 17$, we instead terminate with $z_1 + z_2 \pmod{17}$ since $\text{lc}(r_1)$ is a zero-divisor modulo 17. Normally, we couldn't successfully combine these zero-divisors into one over $\mathbb{Q}$ using the CRT and RR. However, HRR allows us to correctly combine these to a zero-divisor $\mathbb{Q}$; it would output $z_1 + z_2$ once enough primes are used.

A similar technique for using bad primes in rational reconstruction was developed by Boehm et al [3], but Abbott's is better suited for our application since we require the heuristic algorithm HRR that he developed. Pseudo-code for the HRR algorithm in Abbott's paper follows. It is the rational reconstruction algorithm as presented in section 2.3 with a couple key differences. First, it completes the Euclidean algorithm over $\mathbb{Z}$ and finds the maximal quotient $q_{\max}$ with respect to absolute values along with $s_{\max}$ and $t_{\max}$ that occur at the same index in the integer remainder sequence; this is an idea developed in [23]. Second, it returns $u + M\frac{s_{\max}}{t_{\max}}$ rather than just $r_i/t_i = \frac{ut_i + Ms_i}{t_i}$. This is the crucial step that allows it to use bad and unlucky primes.

Algorithm 6: HRR

Input : Positive integers u and M.
Output: A fraction N/D where $\gcd(N, D) = 1$, $D > 0$, and $N/D \equiv u \pmod M$.

1 Set $A_{\text{crit}} := 10^6$, $q_{\max} := 0$, and $i := 2$;
2 **if** $\gcd(u, M)^2 > A_{crit}M$ **then return** 0;
3 Set $s_1 := 1$, $s_2 := 0$, $t_1 := 0$, $t_2 := 1$;
4 Set $r_1 := M$, $r_2 := u$;
5 **while** $r_i \neq 0$ **do**
6 | Set $q := \lfloor r_{i-1}/r_i \rfloor$;
7 | Set $r_{i+1} := r_{i-1} - r_i q$, $t_{i+1} := t_{i-1} - t_i q$, and $s_{i+1} := s_{i-1} - s_i q$;
8 | Set $i := i + 1$;
9 | **if** $|q| \geq |q_{max}|$ **then** set $s_{\max} := s_i$, $t_{\max} := t_i$, and $q_{\max} := q$;
10 **end**
11 **if** $q_{max} < A_{crit}$ **then return** *FAIL*;
12 **return** $u + M\frac{s_{max}}{t_{max}}$

Algorithm HRR uses a *convinceness criteria* in the form of a value A_{crit}. Abbott notes that the value 10^6 has worked well for him in practice, so we use that here as well. Note that when the algorithm does not output failure, it returns $u + M\frac{s_{\max}}{t_{\max}}$. This does not imply $N = t_{\max}u + M$ and $D = t_{\max}$ are the integer outputs for numerator and denominator. In fact,

cancellation in these values is a major result of Abbott when bad primes are encountered. Please see [1] for further details.

An algorithm for resolving zero-divisors using Abbott's HRR algorithm is now straightforward. If a zero-divisor is encountered, combine it with any past zero-divisors if any, and try using the HRR algorithm on the result. After that, if a zero-divisor is successfully lifted to $\mathbb{Q}$, split the triangular sets. The difference between handling zero-divisors with Hensel lifting and HRR is that Hensel lifting only needs 1 prime while HRR will require multiple.

Algorithm 7: HandleZeroDivisorHRR

Input : A zero-dimensional, radical triangular set $T \subset \mathbb{Z}_p[z_1, \ldots, z_n]$ and a zero-divisor $u \in R$ where $R = \mathbb{Z}_p[z_1, \ldots, z_n]/T$. Assume $\text{mvar}(u) = n$.

Output: A message indicating the next steps that should be carried out, including any important parameters;

Use CRT to combine u with previous zero-divisors (if any);
Set $w = \text{HRR}(u)$; `// Note, HRR works on the coefficients of` u.
if $w =$ *FAIL* **then return** *FAIL*. This indicates that more primes are needed;
if $w \mid t_n$ *over* $\mathbb{Q}$ **then**
    return w *and* t_n/w. This indicates that a splitting should be done;
end
return *FAIL*. This indicates that more primes are needed;

Chapter 5

A Modular GCD Algorithm

The main content of this section is to fully present and show the correctness of our modular gcd algorithm. We start with the variant using Hensel lifting and treat fault tolerant rational reconstruction as a small alteration (as it is). In the process of proving the algorithm returns correct output, we prove a lemma about the prime numbers that may appear in the denominator of a monic factor of a monic polynomial modulo a radical triangular set. This is a generalization of known results from algebraic number theory, see Theorem 3.2 from [12] for instance. We conclude with timing tests and an asymptotic analysis.

5.1 Overview with Hensel lifting

First, we consider what happens when a zero-divisor is encountered. In particular, suppose a zero-divisor w over $\mathbb{Q}$ is found while running the modular algorithm. It will be used to factor $t_k = uv \pmod{T_{k-1}}$ where u and v are monic with main variable z_k; it is possible that $u = w$ or $v = w$, but is not always the case. From here, the algorithm proceeds to split T into $T^{(u)}$ and $T^{(v)}$ where t_k is replaced with u in $T^{(u)}$ and v in $T^{(v)}$. Of course t_i is reduced for $i > k$ as well. The algorithm then continues recursively. Once the recursive calls are finished, we could use the CRT to combine gcds into a single gcd, but this would be very time consuming. Instead, we just return both gcds along with their associated triangular sets. This approach is similar to Hubert's in [17] which she calls a pseudo-gcd. Here, we refer to this as a component-wise gcd, or c-gcd for short:

Definition 14. Let R be a commutative ring with unity such that $R \cong \prod_{i=1}^{r} R_i$ and $a, b \in R[x]$. Let $\pi_i \colon R \to R_i$ be the natural projections. A component-wise gcd of a and b is a tuple $(g_1, \ldots, g_r) \in \prod_{i=1}^{r} R_i[x]$ where each $g_i = \gcd(\pi_i(a), \pi_i(b))$ and $\operatorname{lc}(g_i)$ is a unit.

The modular algorithm's goal will be to compute c-gcd(a, b) given $a, b \in R[x]$ where $R = \mathbb{Q}[z_1, \ldots, z_n]/T$ and $T \subset \mathbb{Q}[z_1, \ldots, z_n]$ is a radical triangular set. As with all modular gcd algorithms, it is possible that some primes are unlucky. We also prove this only happens for a finite number of cases.

Definition 15. Let $T \subset \mathbb{Q}[z_1, \ldots, z_n]$ be a radical triangular set, and $R = \mathbb{Q}[z_1, \ldots, z_n]/T$. Let $a, b \in R[x]$ and $g = \text{c-gcd}(a, b)$. A prime number p is an *unlucky prime* if g does not remain a componentwise greatest common divisor of a and b modulo p. Additionally, a prime is *bad* if it divides any denominator in T, any denominator in a or b, or $\text{lc}(a)\text{lc}(b)$ vanishes mod p.

For example, define $T = \{t_1 = z^2 + \frac{1}{3}\}$, $a = x^4 + 4x^3 + 3x^2 + (z+3)x + 3z + 9$, and $b = 3x^3 + 9x^2 - x - 3,$. Then, $\gcd(a, b) = x + 3 \pmod{T}$. Note that 3 would be a bad prime since 3 divides a denominator in t_1 and $\text{lc}(b)$ vanishes modulo 3. Also, 5 and 2 would be unlucky primes since $\gcd(a, b) = x^2 + (3z+3)x + 4z \pmod{T, 5}$ and $\gcd(a, b) = x^2 + (z+1)x + z \pmod{T, 2}$.

Theorem 28. Let $T \subset \mathbb{Q}[z_1, \ldots, z_n]$ be a radical triangular set, and $R = \mathbb{Q}[z_1, \ldots, z_n]/T$. Let $a, b \in R[x]$ and $g = \text{c-gcd}(a, b)$. Only finitely many primes are unlucky.

Proof. Let $R[x] \cong \prod R_i[x]$ where $g = (g_i)$ and $g_i = \gcd(a_i, b_i) \in R_i[x]$ is monic or 0. Let $a \mapsto (a_i)$ and $b \mapsto (b_i)$. If $g_i = 0$, then $a_i = 0$ and $b_i = 0$ and no primes are unlucky since, $\gcd(0, 0) \equiv 0$. Suppose $g_i = \gcd(a_i, b_i)$ is nonzero and monic. Let $\overline{a}_i$ and $\overline{b}_i$ be the quotients so that $a_i = g_i\overline{a}_i$ and $b_i = g_i\overline{b}_i$. I claim $\gcd(\overline{a}_i, \overline{b}_i) = 1$. To show this, consider a common divisor f of $\overline{a}_i$ and $\overline{b}_i$. Note that $fg_i \mid a_i$ and $fg_i \mid b_i$. Since $g_i = \gcd(a_i, b_i)$, it follows that $fg_i \mid g_i$; so there exists $q \in R_i[x]$ where $fg_iq = g_i$. Rewrite this equation as $(fq - 1)g_i = 0$. Well, g_i is monic in x, and so can not be a zero-divisor. This implies $fq - 1 = 0$ and so indeed f is a unit. Thus, $\gcd(\overline{a}_i, \overline{b}_i) = 1$. By the extended Euclidean representation (Corollary 20), there exists $A_i, B_i \in R_i[x]$ where $\overline{a}_iA_i + \overline{b}_iB_i = 1$.

Let p be a prime where p does not divide any of the denominators in $a_i, \overline{a}_i, A_i, b_i, \overline{b}_i, B_i, g_i$. Then, we can reduce the equations

$$\overline{a}_iA_i + \overline{b}_iB_i = 1 \pmod{p}, \tag{5.1}$$

$$a_i = g_i\overline{a}_i \pmod{p}, \qquad b_i = g_i\overline{b}_i \pmod{p}. \tag{5.2}$$

We will now show that $g_i = \gcd(a_i, b_i) \pmod{p}$. By (5.2), we get g_i is a common divisor of a_i and b_i modulo p. Consider a common divisor c of a_i and b_i modulo p. Multiplying equation (5.1) through by g_i gives $a_iA_i + b_iB_i = g_i \pmod{p}$. Clearly, $c \mid g_i$ modulo p. Thus, g_i is indeed a greatest common divisor of a_i and b_i modulo p. As there are finitely many primes that can divide the denominators of fractions in the polynomials $a_i, \overline{a}_i, A_i, b_i, \overline{b}_i, B_i, g_i$, there are indeed finitely many unlucky primes. □

The crux of ModularC-GCD is an algorithm to compute $\gcd(a, b)$ for two polynomials $a, b \in (\mathbb{Z}_p[z_1, \ldots, z_n]/T)[x]$. The algorithm we will be using for this is MonicEuclideanC-GCD. It is a variant of the monic Euclidean algorithm from section 2.2. For computing inverses, the extended Euclidean algorithm can be used; modifying MonicEuclideanC-GCD to do this is straightforward and is done using the recurrences (2.1) and (2.2).

Algorithm 8: MonicEuclideanC-GCD

Input : A zero-dimensional, radical triangular set $T \subset k[z_1, \ldots, z_n]$ and two polynomials $a, b \in R[x]$ where $R = k[z_1, \ldots, z_n]/T$. Assume $\deg(a) \geq \deg(b) \geq 0$.

Output: Either monic $\gcd(a, b) \pmod{T}$ or a zero-divisor.

```
1  if b = 0 then
2  |  Compute s := lc(a)^{-1} (mod T_{n-1}) using the EEA;
3  |  if s =["ZERODIVISOR", u] then return ["ZERODIVISOR", u];
4  |  return s × a
5  end
6  Initialize r_0 := a, r_1 := b and i := 1;
7  while r_i ≠ 0 do
8  |  Compute s := lc(r_i)^{-1} (mod T_{n-1}) using the EEA;
9  |  if s =["ZERODIVISOR", u] then  return ["ZERODIVISOR", u] else
   |  r_i := s × r_i;
10 |  Let r_{i+1} be the remainder of r_{i-1} divided by r_i;
11 |  i = i + 1;
12 end
13 return r_{i-1}
```

A short discussion about the zero-divisors that may appear is warranted. To compute an inverse, the modular algorithm will be using the extended Euclidean algorithm. The first step would be to invert a leading coefficient u of some polynomial. This requires a recursive call to ExtendedEuclideanC-GCD$(u, t_k) \pmod{T_{k-1}}$ where $z_k = \text{mvar}(u)$. If u is not monic, then it would again attempt to invert $\text{lc}(u)$. Because of the recursive nature, it will keep inverting leading coefficients until it succeeds or a monic zero-divisor is found. The main point is that we may assume that the zero-divisors encountered are monic.

We would like to give a high level overview of the algorithm since looking at pseudo-code is not always the best way to understand it. Please see Algorithm 9 for pseudo-code. The inputs are $a, b \in R[x]$ where T is a radical triangular set and $R = \mathbb{Q}[z_1, \ldots, z_n]/T$,

1. Pick a new prime p that is not bad.
2. Test if p is a radical prime.
   - 2.1 If a zero-divisor is encountered, resolve it using HandleZeroDivisorHensel.
   - 2.2 If p is not radical, go back to step 1. Otherwise, continue as p is a radical prime.
3. Use the monic Euclidean algorithm to compute $g_p = \gcd(a, b) \pmod{p}$.
   - 3.1 If a zero-divisor is encountered, resolve it using HandleZeroDivisorHensel.
   - 3.2 Combine all gcds computed modulo primes of lowest degree using Chinese remaindering and rational reconstruction into a polynomial h over $\mathbb{Q}$.
   - 3.3 Test if $h \mid a$ and $h \mid b$. If the division test succeeds, return h. Otherwise, we need more primes, so go back to step 1.

Example 6. This example illustrates how the IsRadical function can run into a zero-divisor. Consider $T = \{z_1^2 - 1, z_2^3 + 9z_2^2 + \frac{3z_1+51}{2}z_2 - \frac{53z_1+3}{2}\}$. We will be running the algorithm over $\mathbb{Q}$ to illustrate. First, it would determine that $T_1 = \{z_1^2 - 1\}$ is radical. Now, when it is running the Euclidean algorithm on $t_2 = z_2^3 + 9z_2^2 + \frac{3z_1+51}{2}z_2 - \frac{53z_1+3}{2}$ and $t_2' = 3z_2^2 + 18z_2 + \frac{3z_1+51}{2}$, the first remainder would be $(z_1 - 1)z_2 - 28z_1 - 27$. However, $z_1 - 1$ is a zero-divisor, so the algorithm would output ["ZERODIVISOR", $z_1 - 1$]. This same zero-divisor will show up for every odd prime (2 appears in the denominator of t_2 and so should not be considered). This explains why we can not just simply pick a new prime in ModularC-GCD if IsRadical encounters a zero-divisor. We instead have to resolve the zero-divisor when encountered modulo a prime using Hensel lifting or fault tolerant rational reconstruction.

In lines 23-32 of ModularC-GCD, we use heuristics to determine if a prime p is lucky or unlucky. This method was developed by Brown in [5] and used in every modular gcd algorithm henceforth. To justify our use of them in this ring, we need a uniqueness criterion for gcds to show that what we are combining is the modular image of the same unique polynomial. This assumes all units are of zero degree in x, which is verified by in Proposition 3; if there were a unit of positive degree, we have no way of knowing if a gcd modulo a prime were multiplied by it.

Lemma 29. Let $R = k[z_1, \ldots, z_n]/T$ where T is a radical zero-dimensional triangular set and let $a, b \in R[x]$. If $g = \gcd(a, b)$ is monic, any other $\gcd(a, b)$ has the same degree. In particular, g is the unique monic $\gcd(a, b)$.

Proof. Let h be a $\gcd(a, b)$. Then, $h \mid g$ and $g \mid h$. This implies the existence of u, v where $hu = g$ and $gv = h$. Basic algebraic manipulation gives $g(uv - 1) = 0$. Since g is monic, it can not be a zero-divisor. Therefore, v is a unit and so $\deg_x(v) = 0$. Thus, $\deg_x(h) = \deg_x(g)$. In particular, if h were monic, then $v = 1$ by comparing leading coefficients of $gv = h$. □

Now that all algorithms have been given, we give a proof of correctness for ModularC-GCD. First, we show that a finite number of zero-divisors can be encountered. This ensures that the algorithm terminates. After that, we prove a lemma about the primes that may occur in a monic factorization modulo the triangular set; note this is nontrivial by example 3. This a key step in the proof that the returned value of ModularC-GCD is correct. The proof will require the concept of localization, the formal process of including denominators in a ring; see section 1.2 of [4] for details. For notation purposes, we let S be a set of prime numbers and define R_S as the localization of R with respect to S. Note that when $R = \mathbb{Q}[z_1, \ldots, z_n]/T$, it is required that any prime dividing any $\text{den}(t_i)$ must be included in S for R_S to be a ring. We will also need the concept of resultants and iterated resultants.

Algorithm 9: ModularC-GCD

Input : A zero-dimensional, radical triangular set $T \subset \mathbb{Q}[z_1, \ldots, z_n]$ and two polynomials $a, b \in R[x]$ where $R = \mathbb{Q}[z_1, \ldots, z_n]/T$. Assume $\deg(a) \geq \deg(b) \geq 0$.

Output: A tuple consisting of comaximal triangular sets $T^{(i)}$ such that $T = \bigcap T^{(i)}$ and $g_i = \gcd(a, b) \mod \langle T^{(i)} \rangle$ where $g_i = 0$ or $\mathrm{lc}(g_i)$ is a unit.

Initialize $dg := \deg(b)$, $M = 1$;
Main Loop: Pick a prime p that is not bad; `// See definition 15.`
Test if p is a radical prime, $N :=$ isRadicalPrime(T, p);
if $N =$ ["ZERODIVISOR", u] **then**
    $K :=$ HandleZeroDivisorHensel(u);
    if $K =$ FAIL **then** Pick a new prime, go to Main Loop;
    else if K is a factorization $t_k = wv \pmod{T_{k-1}}$ **then**
        Create triangular sets $T^{(w)}$ and $T^{(v)}$ where t_k is replaced by w and v, respectively;
        return ModularC-GCD$(a, b) \pmod{T^{(w)}}$, ModularC-GCD$(a, b) \pmod{T^{(v)}}$
    end
else if $N =$ False **then**
    Pick a new prime: Go to Main Loop;
end
Set $g := \gcd(a, b) \mod \langle T, p \rangle$ using algorithm MonicEuclideanC-GCD;
if $g =$ ["ZERODIVISOR", u] **then**
    $K :=$ HandleZeroDivisorHensel(u);
    if $K =$ FAIL **then** Pick a new prime: Go to Main Loop;
    else if K is a factorization $t_k = wv \pmod{T_{k-1}}$ **then**
        Create triangular sets $T^{(w)}$ and $T^{(v)}$ where t_k is replaced by w and v, respectively;
        return ModularC-GCD$(a, b) \pmod{T^{(w)}}$, ModularC-GCD$(a, b) \pmod{T^{(v)}}$
    end
else
    if $\deg(g) = dg$ **then**
        The chosen prime seems to be lucky;
        Use CRT to combine g with other gcds (if any), store the result in G and set $M := M \times p$;
    else if $\deg(g) > dg$ **then**
        The chosen prime was unlucky, discard g;
        Pick a new prime: Go to Main Loop;
    else if $\deg(g) < dg$ **then**
        All previous primes were unlucky, discard G;
        Set $G := g$, $M := p$, and $dg := \deg(g)$;
    end
    Set $h :=$ RationalReconstruction$(G \pmod{M})$;
    if $h \neq$ FAIL and $h \mid a$ and $h \mid b$ **then return** h;
    Pick a new prime: Go to Main Loop;
end

Definition 16. Let $a, b \in R[x]$ where R is a ring. Let $a = a_0 + a_1 x + \cdots + a_n x^n$ and $b = b_0 + b_1 x + \cdots + b_m x^m$. Then the resultant of a and b with respect to x is

$$\operatorname{res}_x(a,b) := \det\Big(\begin{bmatrix} a_n & & & b_m & & \\ a_{n-1} & \ddots & & b_{m-1} & \ddots & \\ \vdots & & a_n & \vdots & & b_m \\ a_0 & & a_{n-1} & b_0 & & b_{m-1} \\ & \ddots & \vdots & & \ddots & \vdots \\ & & a_0 & & & b_0 \end{bmatrix}\Big)$$

where the entries that are blank are all zero and there are m columns of coefficients from a and n columns of coefficients from b.

Resultants are intimitely related to gcds when R is a unique factorization domain. For instance, see page 288 of [14] for the result $\deg_x(\gcd(a,b)) = 0$ if and only if $\operatorname{res}_x(a,b) \neq 0$ for $a, b \in R[x]$.

Definition 17. Given a triangular set T of $k[z_1, \ldots, z_n]$, the iterated resultant of $f \in k[z_1, \ldots, z_n]$ with respect to T is

$$\operatorname{iterres}(f, \{t_1\}) = \operatorname{res}_{z_1}(f, t_1), \qquad \operatorname{iterres}(f, T) = \operatorname{iterres}(\operatorname{res}_{z_n}(f, t_n), T_{n-1}).$$

In the language of polynomial system solving, the iterated resultant of $f \in k[z_1, \ldots, z_n]$ with respect to T is first eliminating z_n, then eliminating z_{n-1}, and so on; the word "eliminate" is used because $\operatorname{res}_x(a,b)$ no longer has any term with x in it while $\operatorname{res}_x(a,b) \in \langle a, b\rangle$, see Chapter 4 of [8] for more details. The main point is that $\operatorname{iterres}(f, T) \in k$.

One important property of iterated resultants is that if $f, T \in R'[x] \subset R[x]$ where R' is a subring, then there exist $A, B_1, \ldots, B_n \in R'[x]$ where $Af + B_1 t_1 + \cdots + B_n t_n = \operatorname{iterres}(f, T)$. This follows from the same proof as given in Theorem 7.1 of [14] using induction on n. Another interesting property is that $\operatorname{iterres}(f, T) = 0$ if and only if f is a zero-divisor for $f \notin \langle T\rangle$, see Lemma 4 of [6]. Let $\alpha \in R$ and $m(x) = \operatorname{iterres}(x - \alpha, T)$. Then $\deg_x(m(x)) = \delta$ where δ is the degree of T. This can be proven easily using the definition and induction on n.

Proposition 30. Let $R = \mathbb{Q}[z_1, \ldots, z_n]/T$ where T is a radical zero-dimensional triangular set. Let $a, b \in R[x]$. A finite number of zero-divisors are encountered when running ModularC-GCD(a, b).

Proof. We use induction on the degree of the extension $\delta = d_1 \cdots d_n$ where $d_i = \operatorname{mdeg}(t_i)$. If $\delta = 1$, then $R = \mathbb{Q}$ so no zero-divisors occur.

First, there are a finite number of non-radical primes. So we may assume that T remains radical modulo any chosen prime.

Second, consider (theoretically) running the monic Euclidean algorithm over $\mathbb{Q}$ where we split the triangular set if a zero-divisor is encountered. In this process, a finite number of primes divide either denominators or leading coefficients; so we may assume the algorithm is not choosing these primes without loss of generality; in particular, the monic Euclidean algorithm over $\mathbb{Q}$ consists of a finite amount of divisions and the monic Euclidean algorithm over $\mathbb{Z}_p$ (for any chosen prime p) consists of the modular image of the same divisions. This relies on Corollary 22 since we only divide by monic polynomials.

Now, suppose a prime p is chosen by the algorithm and a zero-divisor u_p is encountered modulo p at some point of the algorithm. This implies $\gcd(u_p, t_k) \not\equiv 1 \pmod{T_{k-1}, p}$. We may assume that $u_p = \gcd(u_p, t_k) \pmod{T_{k-1}, p}$ and that u_p is monic; this is because the monic Euclidean algorithm will only output such zero-divisors. If u_p lifts to a zero-divisor over $\mathbb{Q}$, the algorithm constructs two triangular sets, each with degree smaller than δ. So by induction, a finite number of zero-divisors occur in each recursive call. Now, suppose lifting fails. This implies there is some polynomial u over $\mathbb{Q}$ that reduces to u_p modulo p and appears in the theoretical run of the Euclidean algorithm over $\mathbb{Q}$. Note that $\gcd(u, t_k) = 1 \pmod{T_{k-1}}$ over $\mathbb{Q}$ since we are assuming the lifting failed. By Theorem 28, this happens for only a finite amount of primes. Thus, a finite number of zero-divisors are encountered. □

We will also make multiple uses of this result by Encarnacion to prove Lemma 32. It can be found as Theorem 3.2 in [12].

Theorem 31 (Encarnacion, 1995)**.** Let α be an algebraic number with minimal polynomial $t \in \mathbb{Q}[z]$. Let $f \in \mathbb{Z}[\alpha][x]$. If u is a monic divisor of f, then u is in the coset $\frac{1}{rd}\mathbb{Z}[\alpha][x]$ where d^2 divides the discriminant $\Delta(t)$, and $r = \operatorname{res}_z(\operatorname{lc}(f), t)$ where r is computed by first doing the substitution $\alpha \mapsto z$.

Lemma 32. Let T be a radical, zero-dimensional triangular set over $\mathbb{Q}$, $R = \mathbb{Q}[z_1, \ldots, z_n]/T$, and $F = \mathbb{Z}[z_1, \ldots, z_n]$. Suppose $f, u \in R[x]$ are monic such that $u \mid f$. Let

$$S = \{\text{prime numbers } p \in \mathbb{Z} : p \text{ is a nonradical prime with respect to } T, \text{ or } p \mid \operatorname{den}(f)\}.$$

Then, $u \in F_S[x]/T$. Recall that any $p \mid \operatorname{denom}(t_i)$ would be included in S by the definition of radical prime. In particular, the primes appearing in denominators of a monic factor of f are either nonradical primes or divisors of $\operatorname{den}(f)$.

Proof. Proceed by induction on n. Consider the base case $n = 1$. Let $t_1 = a_1 a_2 \cdots a_s$ be the monic irreducible factorization. Note that a_i, a_j are relatively prime since t_1 is square-free and $a_1, a_2 \in F_S$ by Gauss's lemma (since S contains any primes dividing $\operatorname{den}(t_1)$, see Proposition 5 of section 9.3 in [11] for Gauss's Lemma). Let $u_i = u \mod a_i$ and $f_i = f \mod a_i$. Now, clear f_i of any denominator so it has an integer leading coefficient. We can apply Theorem 31 noting that $r = \operatorname{lc}(f_i)^{\deg_{z_1}(a_i)}$; this gives $u_i \in \frac{1}{r\Delta(a_i)}Z[\alpha][x]$. Make f_i monic again for the remainder of this proof. This shows $\operatorname{den}(u_i)$ consists of primes dividing

the discriminant $\Delta(a_i)$ or $\text{den}(f_i)$. Note that any prime $p \mid \Delta(a_i)$ would force a_i, and hence t_1, to not be square-free modulo p. This would imply p is nonradical and so is contained in S; in particular, $u_i \in F_S/\langle a_i \rangle\ [x]$.

The last concern is if combining $(u_1, u_2, \ldots, u_s) \mapsto u$ using the CRT introduces a new prime p into the denominator. We prove this can only happen if p is nonradical. It is sufficient to show that combining two extensions is enough since we can simply combine two at a time until the list is exhausted; so we will show this for a_1 and a_2. Now, consider the resultant $r = \text{res}_{z_1}(a_1, a_2)$. There are polynomials $A, B \in F_S$ where $Aa_1 + Ba_2 = r$. Note that any prime $p \mid r$ forces $\gcd(a_1, a_2) \neq 1 \pmod{p}$ and so t_1 would not be square-free mod p; in particular, $A/r, B/r \in F_S$. Now, let $v = (A/r)a_1u_2 + (B/r)a_2u_1$. Note that

$$v \mod a_1 = (B/r)a_2u_1 = (1 - (A/r)a_1)u_1 = u_1.$$

Similarly, $v \mod a_2 = u_2$. Since the CRT gives an isomorphism, $u = v$ and indeed $u \in F_S/T\ [x]$. This completes the base case.

For the inductive step, we will generalize each step used in the base case. Instead of just factoring t_1, we decompose T as a product of comaximal triangular sets known as its triangular decomposition. We will end up working with fields defined by multiple extensions and use the theorem of the primitive element to simplify to the single extension case. Finally, for the combining, iterated resultants are used instead of resultants.

With that in mind, start by decomposing T into its triangular decomposition, which can be done in the the following way:

1. Factor $t_1 = a_1a_2 \cdots a_{s_1}$ into relatively prime monic irreducibles over $\mathbb{Q}$ as in the base case. This gives $\mathbb{Q}[z_1]/T_1$ is isomorphic to the product of fields $\prod_i \mathbb{Q}[z_1]/a_i$. By Gauss's lemma, a prime dividing the $\text{den}(a_i)$ must also divide $\text{den}(f)$. In particular, $a_i \in F_S$.

2. We can factor the image of $t_2^{(i)}$ over each $\mathbb{Q}[z_1]/a_i$ into monic relatively prime irreducibles $t_2^{(i)} = b_1^{(i)}b_2^{(i)} \cdots b_{s_2}^{(i)}$. Note that changing rings from $\mathbb{Q}[z_1]/t_1$ to $\mathbb{Q}[z_1]/a_i$ only involves division by a_i, and hence the only primes introduced into denominators can come from $\text{den}(a_i)$.

3. By the induction hypothesis, any prime p dividing $\text{den}(b_j^{(i)})$ is either not a radical prime of the triangular set $\{a_i\}$ or comes from $\text{den}(t_2^{(i)})$. If $\{a_i\}$ is not radical modulo p, then neither is $\{t_1\}$, clearly.

4. Use this to decompose $k[z_1, z_2]/T_2$ into fields $\mathbb{Q}[z_1, z_2]/\langle a_i, b_j^{(i)} \rangle$ where $a_i \in F_S$ and $b_j^{(i)} \in F_S/\langle a_i \rangle$.

5. Repeat to get $\mathbb{Q}[z_1, \ldots, z_n]/T \cong \prod \mathbb{Q}[z_1, \ldots, z_n]/T^{(i)}$ where each $\mathbb{Q}[z_1, \ldots, z_n]/T^{(i)}$ is a field and $T^{(i)} \subset F_S$ using the induction hypothesis.

Let $f^{(i)} = f \mod T^{(i)}$ and similarly $u^{(i)} = u \mod T^{(i)}$. By the construction above, these reductions can only introduce primes in S into denominators.

Note that $\mathbb{Q}[z_1, \ldots, z_n]/T^{(i)}$ is an algebraic number field. Using the theorem of the primitive element, one can write $\mathbb{Q}[z_1, \ldots, z_n]/T^{(i)} = \mathbb{Q}(\alpha)$ for some $\alpha \in \mathbb{Q}[z_1, \ldots, z_n]/T^{(i)}$. Further, we may assume $\alpha = \lambda_1 z_1 + \cdots + \lambda_n z_n$ for integers λ_i using the same proof as in page 119 of Theorem 5.4.1 in [9] (since the only property Cox used of the base field was that it is infinite and the defining polynomials of the extension are square-free). Next, we claim the minimal polynomial $m_{\alpha,\mathbb{Q}}(z)$ equals $m(z) := \text{iterres}(z - \alpha, T^{(i)})$. This can be verified easily: $m(\alpha) = 0$ clearly, and $\deg_z(\text{iterres}(z - \alpha, T^{(i)}))$ is bounded above by the degree of $T^{(i)}$. Well, the fields $\mathbb{Q}(\alpha)$ and $\mathbb{Q}[z_1, \ldots, z_n]/T^{(i)}$ are equal and so have the same degree over $\mathbb{Q}$, which implies $\deg(m_{\alpha,\mathbb{Q}})$ equals the degree of $\mathbb{Q}[z_1, \ldots, z_n]/T^{(i)}$. Because both are monic, uniqueness of the minimal polynomial gives $m(z) = m_{\alpha,\mathbb{Q}}(z)$. This result was used in the single extension case by Trager in [24]. Also, $\text{iterres}(z - \alpha, T^{(i)})$ is computed without introducing any prime into denominators other than those in the defining polynomials of $T^{(i)}$. We now prove that there are mappings $\mathbb{Q}(\alpha) \longleftrightarrow \mathbb{Q}[z_1, \ldots, z_n]/T^{(i)}$ that only introduce primes that are contained in S. First, $\alpha \mapsto \lambda_1 z_1 + \cdots + \lambda_n z_n$ clearly gives us one such map. For the other direction, note that $t_1^{(i)}$ certainly has a root in $\mathbb{Q}(\alpha)$. This implies $(z_1 - b_1(\alpha)) \mid t_1^{(i)}$. In particular, the base case guarantees $z_1 - b_1(\alpha)$ only has primes in the denominator that are contained in S. We will use $z_1 \mapsto b_1(\alpha)$. For $t_2^{(i)}$, first substitute $z_1 = b_1(\alpha)$ and then use the same idea as with $t_1^{(i)}$ to get a map $z_2 \mapsto b_2(\alpha)$. Repeat $n-2$ more times to get maps for each z_i that is a function of α that does not introduce any primes into denominators besides those in S. This shows that only primes in S can be introduced into the denominator of $u^{(i)}$ when being mapped between the rings $\mathbb{Q}(\alpha)[x]$ and $\mathbb{Q}[z_1, \ldots, z_n]/T^{(i)}\ [x]$. Now, using 31 again shows that $u^{(i)} \in F/T^{(i)}\ [x]$.

Of course $\text{den}(u^{(i)}) \neq \text{den}(u)$. It remains to show that going from $\prod \mathbb{Q}[z_1, \ldots, z_n]/T^{(i)}$ to $\mathbb{Q}[z_1, \ldots, z_n]/T$ only introduces primes in the denominators that are divisors of $\text{den}(f)$ or nonradical. This will follow from using iterated resultants similarly to the resultants in the base case. Suppose we are trying to combine $T^{(i)}$ and $T^{(j)}$ where all $t_k^{(i)} = t_k^{(j)}$ besides $t_n^{(i)} \neq t_n^{(j)}$. Now, perform the iterated resultant and write

$$r = \text{iterres}(\text{res}(t_n^{(i)}, t_n^{(j)}), T_{n-1}^{(i)}) = At_n^{(i)} + Bt_n^{(j)} \pmod{T_{n-1}^{(i)}}$$

with $A, B \in F_S/T_{n-1}^{(i)}\ [z_n]$ since $t_n^{(i)}, t_n^{(j)}$ are contained in $F_S/T_{n-1}^{(i)}[x]$ by construction. Well, any prime p that divides r would have the property of $\gcd(t_n^{(i)}, t_n^{(j)}) \neq 1 \pmod{p}$. Hence t_n would not be square-free and so T would not be radical mod p. Thus, after recovering all splittings into the ring $\mathbb{Q}[z_1, \ldots, z_n]/T\ [x]$, we indeed get $u \in F_S/T\ [x]$. □

Lemma 32 also confirms our use of rational reconstruction. Recall that the rational reconstruction algorithm takes u, M as input and finds $n, d \in \mathbb{Z}$ where $n/d \equiv u \pmod{M}$ with $\gcd(d, M) = 1$ and $\gcd(n, d) = 1$. The condition $\gcd(d, M) = 1$ has to be enforced by

the algorithm using rational reconstruction. Lemma 32 indeed enforces this as it shows only bad and nonradical primes (which the algorithm checks for and disregards) are used in the modular gcd computations. It is also a vital step in proving the modular c-gcd algorithm returns correct output.

Theorem 33. Let $R = \mathbb{Q}[z_1, \ldots, z_n]/T$ where T is a radical zero-dimensional triangular set and let $a, b \in R[x]$. The modular algorithm using Hensel lifting to handle zero-divisors outputs a correct c-gcd if run on a and b.

Proof. It is enough to prove this for a single component of the decomposition. For ease of notation, let $T \subset R$ be the triangular set associated to this component. Let h be the monic polynomial returned from the modular algorithm modulo T (at line 34) and define $g = \gcd(a, b) \pmod{T}$ over $\mathbb{Q}$.

First, we may assume that b is monic. If $\mathrm{lc}_x(b)$ is a unit, divide through by its inverse and this does not change $\gcd(a, b)$. If $\mathrm{lc}_x(b)$ is a zero-divisor, the EA mod p would catch it and cause a splitting, contradicting that the EA mod p did not encounter a zero-divisor in this component of the c-gcd. Since h passed the trial division in step 34, it follows that $h \mid g$ and hence $\deg(h) \leq \deg(g)$ since h is monic.

Suppose $\mathrm{lc}(g)$ is invertible. If so, make g monic without loss of generality. Let p be a prime used to compute h. Since g is monic and divides b which is also monic, any prime appearing in $\mathrm{den}(g)$ is either nonradical or a divisor of $\mathrm{den}(b)$ by Lemma 32. In particular, since the prime p was used successfully to compute h, it can not occur in the denominator of g. So, we may reduce g modulo p. Let $\overline{f}$ denote the reduction of a polynomial $f \in R[x]$ mod p. Since $\overline{g} \mid \overline{a}$ and $\overline{g} \mid \overline{b}$, it follows that $\overline{g} \mid \overline{h}$ and so $\deg(g) \leq \deg(h)$. Since $h \mid g$, they have the same degree, and both are monic, it must be that $h = g$ and so indeed h is a greatest common divisor of a and b.

Suppose $\mathrm{lc}(g)$ was a zero-divisor and that $\mathrm{mvar}(\mathrm{lc}(g)) = z_n$ without loss. Inspect $\mathrm{lc}_{z_n}(\mathrm{lc}(g))$; if this is a unit, make it monic. If it is a zero-divisor, inspect $\mathrm{lc}_{z_{n-1}}(\mathrm{lc}_{z_n}(g)))$. Continue until $u = \mathrm{lc}_{z_{k+1}}(\cdots(\mathrm{lc}_{z_n}(\mathrm{lc}_x(g))\cdots)$ is a monic zero-divisor. Further, if $\gcd(u, t_k) \neq u$, then $u/\gcd(u, t_k)$ is a unit and so we can divide through by it to ensure $\gcd(u, t_k) = u$. Let $t_k = uv \pmod{T_{k-1}}$ be a monic factorization. Note that Lemma 32 guarantees that the same factorization $\overline{uv} = \overline{t_k} \pmod{T_{k-1}, p}$ occurs modulo p. Hence, we can split T into triangular sets $T^{(u)}$ and $T^{(v)}$ where t_k is replaced by u and v, respectively, and this same splitting occurs modulo p.

Let $g_u = g \mod T^{(u)}$ and $g_v = g \mod T^{(v)}$ and similarly for other relevant polynomials. By Lemma 10 $\overline{h_u}$ is still a gcd of $\overline{a_u}$ and $\overline{b_u}$ and g_u for a_u and b_u. Now, we consider both triangular sets $T^{(u)}$ and $T^{(v)}$. First, in $T^{(v)}$, u is invertible otherwise T would not be radical. So, multiply g_v by u^{-1} so that $\mathrm{lc}_{z_{k+1}}(\cdots(\mathrm{lc}_{z_n}(\mathrm{lc}_x(g))\cdots) = 1$. Reinspect $w = \mathrm{lc}_{z_{k+2}}(\cdots(\mathrm{lc}_{z_n}(\mathrm{lc}_x(g_v)))\cdots)$. If w is not a zero-divisor, multiply through by its inverse and repeat until a zero-divisor is encountered as a leading coefficient. Do the same computations

to find another splitting and be in the same situation as that of u in T. Otherwise, in $T^{(u)}$, $u = 0$ and so $\mathrm{lc}_{z_{k+1}}(\cdots(\mathrm{lc}_{z_n}(\mathrm{lc}_x(g_u))\cdots)$ has changed; if it is invertible, multiply through by its inverse until a monic zero-divisor is found in the leading coefficient chain. We again wind up in the situation with a monic factorization of t_j that is reducible modulo p.

The process described in the last paragraph must terminate with a splitting in which the image of g is monic since $\mathrm{lc}_x(g)$ has finite degree in each variable. We have already shown that the image of h would be an associate of the image in g in this case. Since being a gcd persists through isomorphisms in the sense of Lemma 10, this gives indeed that h is a $\gcd(a, b)$ modulo T over $\mathbb{Q}$, as desired. □

5.2 Overview with FTRR

We now show how to make a modular algorithm that uses Abbott's Fault Tolerant Rational Reconstruction to handle zero-divisors. The only change to ModularC-GCD that requires change is that HandleZeroDivisorHRR is used in place of HandleZeroDivisorHensel. Proving this variant of the modular algorithm works is based mostly on the idea of the Euclidean algorithm over $\mathbb{Q}$ agreeing with the Euclidean algorithm over $\mathbb{Z}_p$ for all but finitely many primes. We give a proof of this.

Lemma 34. Let T be a radical triangular set over $\mathbb{Q}$ in n variables. Let $R = \mathbb{Q}[z_1, \ldots, z_n]/T$. Running the Euclidean algorithm on two polynomials in $R[x]$ agrees with the Euclidean algorithm in $R/\langle p\rangle[x]$ for all but finitely many primes p.

Proof. Note that the Euclidean algorithm consists of a finite amount of divisions; so it enough to show that a single division over $\mathbb{Q}$ agrees with a single division modulo all but finitely many primes. In symbols, let $a, b \in R[x]$ with $\deg_x(b) \leq \deg_x(a)$ and let r be the remainder when a is divided by b.

Work by induction on n, the number of extensions. Consider the base case $n = 0$. Let $a = bq + r$ and p be a prime number. As long as p does not divide any of the relevant denominators, this equation can be reduced modulo p. Further, if $p \nmid \mathrm{lc}(b)$, then $\deg(r \bmod p) < \deg(b \bmod p)$. Since there are finitely many primes that cause these issues, the base case is satisfied.

Now consider the nth case. In the process of dividing a by b over $\mathbb{Q}$, numerous smaller degree divisions must be done, in particular, they all only use $n - 1$ or less extensions. Use the induction hypothesis here to rule out a set of finitely many primes S. Take a prime number $p \notin S$. If a zero-divisor is encountered in one of the smaller divisions, the algorithm would terminate both over $\mathbb{Q}$ and $\mathbb{Z}_p$ with the same zero-divisor after reduction modulo p. So suppose no zero-divisors are encountered in any of these smaller divisions. In particular, $\mathrm{lc}(b)$ is a unit over $\mathbb{Q}$ and $\mathbb{Z}_p$. Next, $\deg_x(b \bmod p) < \deg_x(b)$ is only true for primes $p \mid \mathrm{lc}(b)$, which happens for finitely many primes. We may safely disregard these. After

that, the uniqueness of remainders in Corollary 22 establishes that the remainders are the same after reduction by p. □

Lemma 34 establishes that if $g = \text{EuclideanC-GCD}(a, b)$ over $\mathbb{Q}$, then $g_p = \text{EuclideanC-GCD}(a, b)$ over $\mathbb{Z}_p$ satisfies $g \equiv g_p \pmod{p}$ for all but finitely many primes p; whether g is a c-gcd of a and b or a zero-divisor. This is the exact scenario where the fault tolerant rational reconstruction applies. Thus, we have shown that the modular c-gcd algorithm works when using fault tolerant rational reconstruction to handle zero-divisors. We should still show that the output is an actual gcd, but the proof is the same as Theorem 33.

Theorem 35. Let $R = \mathbb{Q}[z_1, \ldots, z_n]/T$ where T is a radical zero-dimensional triangular set and let $a, b \in R[x]$. The modular algorithm using fault tolerant rational reconstruction to handle zero-divisors outputs a correct c-gcd if run on a and b.

5.3 Implementation and Timing Results

We have implemented algorithm `ModularC-GCD` with both methods of resolving zero-divisors as presented above using Maple's RECDEN package which uses a recursive dense data structure for polynomials with extensions. Details can be found in Monagan and van Hoeij's paper [15]. The reader may find our Maple code for our software there together with several examples and their output at `http://www.cecm.sfu.ca/CAG/code/MODGCD`.

The remainder of this section will be used to compare our algorithm with the `RegularGcd` algorithm (see [19]) which is in the `RegularChains` package of Maple. Algorithm `RegularGcd` computes a subresultant polynomial remainder sequence and outputs the last non-zero element of the sequence. We highlight three differences between the output of `RegularGcd` and `ModularC-GCD`.

1. The algorithms may compute different decompositions of the input triangular set.

2. `RegularGcd` returns the last non-zero subresultant but not reduced modulo T; it often returns a gcd g with $\deg_{z_i}(g) > \text{mdeg}(t_i)$. To compute the reduced version, the procedure `NormalForm` is required. `ModularC-GCD` uses the CRT and rational reconstruction on images of the c-gcd modulo multiple primes, so it computes the reduced version of the c-gcd automatically.

3. `RegularGcd` computes gcds up to units, and for some inputs the units can be large. `ModularC-GCD` computes the monic gcd which may have large fractions.

Example 7. We would like to illustrate the differences with an example provided by an anonymous referee of a paper that this thesis is based upon. Let

$$
\begin{aligned}
T &= \{x^3 - x, \quad y^2 - \tfrac{3}{2}yx^2 - \tfrac{3}{2}yx + y + 2x^2 - 2\},\\
a &= z^2 - \tfrac{8}{3}zyx^2 + 3zyx - \tfrac{7}{3}zy - \tfrac{1}{3}zx^2 + 3zx - \tfrac{5}{3}z + \tfrac{25}{6}yx^2 - \tfrac{13}{2}yx + \tfrac{10}{3}y + \tfrac{16}{3}x^2 - 2x - \tfrac{10}{3},\\
b &= z^2 + \tfrac{29}{12}zyx^2 + \tfrac{7}{4}zyx - \tfrac{11}{3}zy - \tfrac{8}{3}zx^2 + 3zx + \tfrac{2}{3}z + \tfrac{67}{12}yx^2 - \tfrac{11}{4}yx - \tfrac{13}{3}y - \tfrac{13}{3}x^2 - 2x + \tfrac{19}{3}.
\end{aligned}
$$

When we run our algorithm to compute c-gcd$(a, b) \pmod{T}$, it returns

$$
\begin{aligned}
z^2 + (3x - 2)z - 2x + 2 \quad &(\text{mod } y, x^2 - 1),\\
z + \tfrac{1}{2}x - \tfrac{3}{2} \quad &(\text{mod } y - \tfrac{3}{2}x - \tfrac{1}{2}, x^2 - 1),\\
z + 5 \quad &(\text{mod } y + 2, x),\\
1 \quad &(\text{mod } y - 1, x).
\end{aligned}
$$

The same example using `RegularGcd` returns

$$
\begin{aligned}
(-96\,y + 168)\,z - 552\,y + 696 \quad &(\text{mod } y + 2, x),\\
154368\,y^3 - 117504\,y^2 - 559872\,y + 585216 \quad &(\text{mod } y - 1, x),\\
z^2 + (\tfrac{2}{3} - \tfrac{8}{3}x^2 + 3x)z \quad &(\text{mod } y, x - 1),\\
(366x^2 - 90x - 96)yz + (102x^2 + 270x - 552)y \quad &(\text{mod } y - 2, x - 1),\\
z^2 + (\tfrac{2}{3} - \tfrac{8}{3}x^2 + 3x)z + \tfrac{19}{13} - \tfrac{13}{3}x^2 - 2x \quad &(\text{mod } y, x + 1),\\
(366x^2 - 90x - 96)yz + (102x^2 + 270x - 552)y \quad &(\text{mod } y + 1, x + 1).
\end{aligned}
$$

As can be seen, our algorithm only decomposes T into 4 triangular sets while `RegularGcd` decomposes T into 6. Further, it is easy to notice that each component in our output is reduced, while the output of `RegularGcd` is not. Applying the `NormalForm` command to reduce the output of `RegularGcd` returns

$$
\begin{aligned}
360z + 1800 \quad &(\text{mod } y + 2, x), & 62208 \quad &(\text{mod } y - 1, x),\\
z^2 + z \quad &(\text{mod } y, x - 1), & 360z - 360 \quad &(\text{mod } y - 2, x - 1),\\
z^2 - 5z + 4 \quad &(\text{mod } y, x + 1), & -360z + 720 \quad &(\text{mod } y + 1, x + 1).
\end{aligned}
$$

Notice that `RegularGcd` also circumvents fractions. In general, the output of our algorithm deals with smaller numbers. This can certainly be seen as an advantage for a user.

Finally, we would like to conclude with some timing tests which show the power of using a modular GCD algorithm that recovers the monic c-gcd from images modulo primes using rational reconstruction.

We first construct random triangular sets where each t_i is monic in z_i and dense in $z_1, \ldots, z_{i-1}$ with random two digit coefficients. We then generate $a, b, g \in R[x]$ with degrees $6, 5$, and 4, respectively. Then, compute c-gcd(A, B) where $A = ag$ and $B = bg$. Maple code for generating the test inputs is included on our website.

extension		ModularC-GCD			`RegularGcd`		
n	degrees	time	divide	#primes	time real	cpu	#terms
1	[4]	0.013	0.006	3	0.064	0.064	170
2	[2, 2]	0.029	0.022	3	0.241	0.346	720
2	[3, 3]	0.184	0.138	17	1.73	4.433	2645
3	[2, 2, 2]	0.218	0.204	9	10.372	29.357	8640
2	[4, 4]	0.512	0.391	33	12.349	40.705	5780
4	[2, 2, 2, 2]	1.403	1.132	33	401.439	758.942	103680
3	[3, 3, 3]	2.755	1.893	65	413.54	1307.46	60835
3	[4, 2, 4]	1.695	1.233	33	39.327	86.088	19860
1	[64]	6.738	5.607	65	43.963	160.021	3470
2	[8, 8]	13.321	11.386	129	1437.76	5251.05	30420
3	[4, 4, 4]	17.065	14.093	129	7185.85	22591.4	196520

Table 5.1: The first column is the number of algebraic variables, the second is the degree of the extensions, the third is the CPU time it took to compute c-gcd of the inputs for ModularC-GCD, the fourth is the CPU time in ModularC-Gcd spent doing trial divisions over $\mathbb{Q}$, the fifth is the number of primes needed to recover g, the sixth is the real time it took for `RegularGcd` to do the same computation, the seventh is the total CPU time it took for `RegularGcd` and the last is the number of terms in the unnormalized gcd output by `RegularGcd`. All times are in seconds.

In the previous data-set, g is not created as a monic polynomial in x, but ModularC-GCD computes the monic gcd(A, B). Since lc(g) is a random polynomial, its inverse in R will likely have very large rational coefficients, and so additional primes have to be used to recover the monic gcd. This brings us to an important advantage of our algorithm: it is output-sensitive. In Table 2 below g is a monic degree 4 polynomial with a and b still of degree 6 and 5. Notice that our algorithm finishes much faster than the earlier computation, while `RegularGcd` takes about the same amount of time. This happens because the coefficients of subresultants of A and B are always large no matter how small the coefficients of gcd(A, B) are. Note that the timing tests are only done with using Hensel lifting to handle zero-divisors, but both approaches will have the same running time since zero-divisors are rarely encountered when generating random polynomials.

Let $d_a = \deg_x a$, $d_b = \deg_x b$ with $d_a \geq d_b$ and let $d_g = \deg_x g$. In Table 3 below we increased d_a and d_b from 6 and 5 in Table 1 to 9 and 8 leaving the degree of g at 4. By increasing d_b we increase the number of steps in the Euclidean algorithm which causes an expression swell in `RegularGcd` in the size of the integer coefficients and the degree of each $z_1, \ldots, z_n$, that is, the expression swell is $(n+1)$ dimensional. The number of multiplications in R that the monic Euclidean algorithm does is at most $(d_a - d_b + 2)(d_g + d_b)$ for the first

extension		ModularC-GCD			`RegularGcd`		
n	degrees	time	divide	#primes	time real	cpu	#terms
1	[4]	0.01	0.006	2	0.065	0.065	170
2	[2, 2]	0.02	0.016	2	0.238	0.329	715
2	[3, 3]	0.048	0.041	2	1.771	4.412	2630
3	[2, 2, 2]	0.05	0.041	2	11.293	31.766	8465
2	[4, 4]	0.077	0.068	2	11.521	36.854	5750
4	[2, 2, 2, 2]	0.117	0.097	2	321.859	431.368	99670
3	[3, 3, 3]	0.222	0.201	2	508.465	1615.28	57645
3	[4, 2, 4]	0.05	0.032	2	34.358	71.351	16230
1	[64]	0.304	0.282	2	27.55	98.354	3450
2	[8, 8]	0.482	0.455	2	1628.7	5979.51	29505
3	[4, 4, 4]	0.525	0.477	2	2989.18	4751.04	192825

Table 5.2: The columns are the same as for Table 5.1. Here, the gcd of the inputs has much smaller coefficients than in Table 5.1

division and $\sum_{i=d_g}^{d_g+d_b-1} 2i = d_b(d_b+2d_g-1)$ for the remaining divisions. The trial divisions of A by g and B by g cost $d_a d_g$ and $d_b d_g$ multiplications in R respectively. Increasing d_a, d_b, d_g from $6, 5, 4$ in Table 1 to $9, 8, 4$ increases the number of multiplications in R in the monic Euclidean algorithm from 87 to 156 and from $24 + 20 = 44$ to $36 + 32 = 68$ for the trial divisions but the monic gcd remains unchanged. Comparing Table 1 and Table 3 the reader can see that the increase in ModularC-GCD is less than a factor of 2.

extension		ModularC-GCD			`RegularGcd`		
n	degrees	time	divide	#primes	time real	cpu	#terms
1	[4]	0.021	0.011	5	0.124	0.13	260
2	[2, 2]	0.043	0.031	5	0.968	1.912	1620
2	[3, 3]	0.214	0.163	17	10.517	34.513	6125
3	[2, 2, 2]	0.287	0.204	9	64.997	173.53	29160
2	[4, 4]	0.638	0.427	33	67.413	245.789	13520
4	[2, 2, 2, 2]	2.05	1.613	33	2725.13	3528.41	524880
3	[3, 3, 3]	3.35	2.731	33	3704.61	11924.0	214375
3	[4, 2, 4]	2.399	1.793	33	334.201	869.116	68940
1	[64]	10.097	8.584	65	171.726	658.518	5360
2	[8, 8]	21.890	18.086	129	10418.4	38554.9	72000
3	[4, 4, 4]	37.007	31.369	129	> 50000	–	–

Table 5.3: The columns are the same as for Table 5.1. Here, the degree of the input polynomials are raised to 9 and 8 while their gcd still has degree 4.

5.4 Asymptotic Analysis

We will do an asymptotic analysis for the modular c-gcd algorithm that uses Hensel lifting to handle zero-divisors. The running time of the algorithm is dominated by running the Euclidean algorithm modulo multiple primes and the division test. This is verified in the previous section's timing results. Because of this, we will only consider the running time based on these two parts of the algorithm. Also, the expected case is that no zero-divisors are encountered. Further, not encountering a zero-divisor is arguably the worst case scenario. This is because if a zero-divisor is successfully lifted to $\mathbb{Q}$, then the degree of each component will smaller. Therefore, reduction by the triangular set takes less operations. This can also be seen in the timing tests by observing the running time with degrees $[4, 4, 4]$ and $[4, 2, 4]$ in Table 5.1 gives a ratio of about 10.

Now, suppose M primes are needed to successfully compute $g = \gcd(a, b)$. Since there are only finitely many unlucky primes, we assume the algorithm doesn't encounter any of these. This implies we need M runs of the Euclidean algorithm modulo primes. This part takes a total of $O(M \deg(a) \deg(b)\delta^2)$ field multiplications modulo primes by Proposition 17. We could find a bound on M, but we do not think it is worthwhile because our algorithm is output sensitive and any bound will be bad since it has to handle the worst case. Next, the implementation of the algorithm does not perform the division test at after each prime. We have coded it so that division is only tested $O(\log(M))$ times. Each division takes $O(\deg(g) \deg(b)\delta^2)$ operations over $\mathbb{Q}$ for a total of $O(\log(M) \deg(g) \deg(b)\delta^2)$ multiplications in $\mathbb{Q}$. Because we're assuming no unlucky primes are encountered, this is an expected case analysis.

We would like to discuss an optimization for the division test. It does not avoid the worst case, but it does improve the expected case. Suppose rational reconstruction successfully outputs a polynomial $h \in \mathbb{Q}[z_1, \ldots, z_n]/T\ [x]$. Instead of going straight into the division test, we can make use of a check prime. That is, we pick one more prime p where p is not bad or radical. Then, compute $g = \gcd(a, b) \pmod{p}$. If a zero-divisor is encountered in the radical test or in the computation of g, we pick a new check prime. Next, we check if $h \equiv g \pmod{p}$. If it is, we proceed to the division test. If it is not we go back to the main loop and pick more primes starting with p. More rigorously, we replace lines 33-35 of ModularC-GCD with the following pseudo-code.

This optimization only performs the division test once in the expected case. Since there are finitely many unlucky primes by Theorem 28, the algorithm expects to always pick a lucky prime. Therefore, the only time the division test can be needlessly performed in the expected case, is if not enough primes are picked to exceed the bounded needed by rational reconstruction. The use of a check prime supersedes this since the check prime is expected to be lucky as well. Thus, we have the following theorem.

```
Set h := RationalReconstruction(G (mod M));
if h ≠ FAIL then
    Check-Prime Loop: Pick a new prime p that is not bad or radical;
    if a zero-divisor is encountered then pick a new prime, go to Check-Prime Loop;
    Compute g := gcd(a, b) (mod p);
    if a zero-divisor is encountered then pick a new prime, go to Check-Prime Loop;
    if g ≢ h (mod p) then pick a new prime, go to Main Loop;
    if h | a and h | b then return h;
end
Pick a new prime: Go to Main Loop;
```

Theorem 36. The ModularC-GCD algorithm performs $O(M \deg(a) \deg(b) \delta^2)$ operations in $\mathbb{Z}_p$. Additionally, it uses $O(\deg(g) \deg(b) \delta^2)$ operations over $\mathbb{Q}$ in the expected case, and $O(\log(M) \deg(g) \deg(b) \delta^2)$ operations in the worst case.

Chapter 6

An Inversion Algorithm

We would like to present a second application of the use of Hensel lifting for resolving zero-divisors: the inversion problem. The inversion problem takes as input a zero-dimensional triangular set $T \subset \mathbb{Q}[z_1, \ldots, z_n]$ and $a \in \mathbb{Q}[z_1, \ldots, z_n]/T$ and computes a^{-1} or determines a^{-1} does not exist. This can be done using the `Inverse` command in the `RegularChains` library of Maple. We present a new algorithm that is based on the Newton iteration in section 6.1. We conclude with time tests in section 6.2.

6.1 Overview and Analysis

We would like to start with motivating the use of Newton iteration. With that in mind, computing a^{-1} can be phrased as computing a root of the equation $f(x) = a - \frac{1}{x}$. Then this is a fixed-point of $g(x) = x - \frac{f(x)}{f'(x)}$ as long as $f'(a^{-1}) \neq 0$. Therefore, we can use the Newton fixed-point iteration scheme $x_{n+1} := x_n - \frac{f(x_n)}{f'(x_n)}$. Well, $f'(x) = \frac{1}{x^2}$ and so $x_{n+1} := 2x_n - ax_n^2$. It is well known that Newton iteration is quadratically convergent, which is verified in the succeeding lemma.

Lemma 37. Let $T \subset \mathbb{Q}[z_1, \ldots, z_n]$ be a triangular set and $a \in R = \mathbb{Q}[z_1, \ldots, z_n]/T$. Suppose $x_k \in \mathbb{Z}_{p^k}[z_1, \ldots, z_n]$ satisfies $ax_k \equiv 1 \pmod{T, p^k}$. Then the next term in the Newton iteration scheme $x_{k+1} := 2x_k - ax_k^2$ satisfies $ax_{k+1} = 1 \pmod{T, p^{2k}}$.

Proof. First, note that we can write $1 - ax_k \equiv fp^k \pmod{T}$ for some $f \in R$. Then,

$$1 - ax_{k+1} = 1 - a(2x_k - ax_k^2) = 1 - 2ax_k - (ax_k)^2 = (1 - ax_k)^2 = f^2p^{2k} \pmod{T}.$$

Therefore, $1 - ax_{k+1} \equiv 0 \pmod{T, p^{2k}}$. □

This leads us to an inversion algorithm. Let $a \in \mathbb{Q}[z_1, \ldots, z_n]/T$ be the input. We first show how we determine non-invertibility. For this purpose, suppose $a \in \mathbb{Q}[z_1, \ldots, z_n]/T$ is a zero-divisor and $\text{mvar}(a) = z_i$. Then $\gcd(a, t_i) \pmod{T_{i-1}}$ is nontrivial by Corollary 21. Therefore, we can use a modular gcd algorithm similar to ModularC-GCD to compute the

c-gcd$(a, t_i) \pmod{T_{i-1}}$. Otherwise, a is invertible, so we can compute the inverse modulo a lucky prime p, and using the Newton iteration in Lemma 37 and rational reconstruction, then lift it to $\mathbb{Q}$. If the rational reconstruction returns a polynomial $b \in R$, we of course still have to test if $ab \equiv 1 \pmod{T}$.

A complication occurs if a zero-divisor is encountered in the EEA of a and t_i. If this happens, we simply use the HandleZeroDivisorHensel algorithm. One could also use FTRR for this purpose, but we have not pursued this. If a zero-divisor is encountered and lifts to $\mathbb{Q}$, we split the triangular set and compute a^{-1} modulo each. We give pseudo-code below.

One weakness of our algorithm is that we have only succeeded in proving it works when the triangular set is radical. This only comes into play since we do a modular gcd computation when a is not invertible. The algorithm in [21] works more generally for radical and nonradical ideals. We would like to note that for actually computing the inverse the radical assumption is not required. So if the user knows that the inverse exists by some other means, they can still use our algorithm. The timing tests that follow show that it is worthwhile.

Theorem 38. Let $T \subset \mathbb{Q}[z_1, \ldots, z_n]$ be a radical zero-dimensional triangular set. Let $a \in \mathbb{Q}[z_1, \ldots, z_n]/T$. Then running Inversion$(a)$ terminates with correct output.

Proof. First, suppose the input a was a zero-divisor over $\mathbb{Q}$. This implies the $\gcd(a, t_n) \pmod{T_{n-1}}$ is nontrivial over $\mathbb{Q}$, and so $\gcd(a, t_n) \pmod{T_{n-1}, p}$ must be nontrivial for any chosen prime p as well. Then, using the same proof as in the proof of modularC-GCD Theorem 33, we will successfully recover c-gcd$(a, t_n) \pmod{T_{n-1}}$.

Otherwise, suppose a is a unit. By Theorem 28, $1 = \gcd(a, t_n) \pmod{T_{n-1}, p}$ for all but finitely many primes p. We may suppose the algorithm picks these primes without loss of generality. Using the same proof as Proposition 30, we may conclude that $1 = \gcd(a, t_n) \pmod{T_{n-1}, p}$ is successfully computed by the EA mod p. From here, the algorithm computes a^{-1} with higher and higher accuracy. We can compute a^{-1} over $\mathbb{Q}$ using linear algebra and so the integers appearing in the rational numbers of its coefficients must be bounded. This bound will be reached by Lemma 37, and once this bound has been reached, the rational reconstruction will succeed. Since the inverse is unique, (over $\mathbb{Q}$ and over $\mathbb{Z}_{p^k}$ for any positive integer k), the algorithm will successfully compute a^{-1} over $\mathbb{Q}$. □

We now justify the use of rational reconstruction for the invertible case. Let a and T be the inputs and suppose the algorithm is using the prime number p. Recall that the algorithm checks if p is a bad prime so a and T are reducible by p and $a \not\equiv 0 \pmod{T, p}$. The only concern is if p appears in the denominator of the rational coefficients of a^{-1}. Suppose p^k is the largest power of p appearing as such. Then, $p^k a^{-1}$ is reducible by p. Notice that $a(p^k a^{-1}) = p^k$ and so a becomes a zero-divisor modulo p. This shows that p is an unlucky prime; i.e., $\gcd(a, t_i) \neq 1 \pmod{T, p}$ while $\gcd(a, t_i) = 1 \pmod{T}$. The algorithm would

Algorithm 10: Inversion

Input : A radical, zero-dimensional triangular set $T \subset \mathbb{Q}[z_1, \ldots, z_n]$ and an element $a \in R$ with $\text{mvar}(a) = z_n$.

Output: a^{-1} or a message indicating that a is a zero-divisor.

Initialize $dg := \deg(b)$, $M = 1$;
Main Loop: Pick a nonbad prime p; `// See definition 15.`
Test if p is a radical prime, $N :=$ isRadicalPrime(T, p);
if $N =$ ["ZERODIVISOR", u] **then**
    $H :=$ HandleZeroDivisorHensel(u);
    if $H =$ FAIL **then** pick a new prime, go to Main Loop;
    else if H is a factorization $t_k = wv \pmod{T_{k-1}}$ **then**
        Create triangular sets $T^{(w)}$ and $T^{(v)}$ where t_k is replaced by w and v;
        return Inversion$(a, b) \pmod{T^{(w)}}$, Inversion$(a, b) \pmod{T^{(v)}}$
    end
else if $N =$ False **then** Pick a new prime: Go to Main Loop ;
Use the EEA on t_n and a to compute $g = \gcd(a, t_n) \pmod{T_{n-1}, p}$ and b where $ab \equiv 1 \pmod{T, p}$;
if $b =$["ZERO-DIVISOR", u] **then**
    Call $H =$ HandleZeroDivisor(u);
    if $H =$ FAIL **then** pick a new prime, go to Main Loop;
    else if H is a factorization $t_k = wv \pmod{T_{k-1}}$ **then**
        Create triangular sets $T^{(w)}$ and $T^{(v)}$ where t_k is replaced by w and v;
        return Inversion$(a) \pmod{T^{(w)}}$, Inversion$(a) \pmod{T^{(v)}}$
    end
else if $g \neq 1 \pmod{T_{n-1}, p}$ **then**
    if $\deg(g) = dg$ **then**
        The chosen prime seems to be lucky;
        Use CRT to combine g with other gcds (if any) and store the result in G;
        Set $M := M \times p$;
    else if $\deg(g) > dg$ **then**
        The chosen prime was unlucky, discard g and pick a new prime. Go to Main Loop;
    else if $\deg(g) < dg$ **then**
        All previous primes were unlucky, discard G;
        Set $G := g$, $M := p$, and $dg := \deg(g)$;
    end
    Set $h :=$ RationalReconstruction$(G \pmod{M})$;
    if $h \neq$ FAIL **and** $h \mid a$ **and** $h \mid t_n$ **then return** "a is not a unit.";
    Pick a new prime: Go to Main Loop;
else if $g = 1 \pmod{T_{n-1}, p}$ **then**
    Set $k := 1$;
    while true **do** `// Newton iteration: see Lemma 37.`
        Set $h :=$ RationalReconstruction$(b \pmod{p^k})$;
        if $h \neq$ FAIL **and** $ah \equiv 1 \pmod{T}$ **then return** h;
        Set $k := 2k$ and $b := 2b - ab^2 \pmod{T, p^k}$;
    end
end

catch this before the inversion loop is started. Thus, rational reconstruction can be safely used.

Lastly, we would like to give a quick asymptotic analysis. The analysis for noninvertible elements is the same as ModularC-GCD since we were using the same algorithm, so we will focus on the case where the input is an invertible element. With that in mind, the algorithm's running time will be dominated by the multiplication test over $\mathbb{Q}$ and the iterations over $\mathbb{Z}_{p^k}$, so this is what we count.

Proposition 39. Let $a \in \mathbb{Q}[z_1, \ldots, z_n]/T$ be an invertible element. Suppose the numerators of the fractions in a^{-1} are bounded by N add the denominators by D. The Inversion algorithm computes a^{-1} in $O(\log_2(\log_p(ND))\delta^2)$ multiplications in $\mathbb{Z}_{p^k}$ where k is varying powers of 2, and $O(\log_2(\log_p(ND))\delta^2)$ multiplications in $\mathbb{Q}$ in the worst case and $O(\delta^2)$ multiplications in $\mathbb{Q}$ in the expected case.

Proof. Note that the recurrence $x_{k+1} := 2x_k - ax_k^2$ will take 2 ring multiplications. These will cost $O(\delta^2)$ multiplications each by Proposition 14. This has to be iterated until rational reconstruction does not fail and returns the correct fractions; that is, until $p^{2^k} \geq 2ND$. Thus, $O(\log_2(\log_p(ND))$ iterations are required. In total, $O(\log_2(\log_p(ND))\delta^2)$ multiplications are done in $\mathbb{Z}_{p^k}$ and similarly for over $\mathbb{Q}$. For the expected case, we can make use of check primes as in ModularC-GCD to only have to do the multiplication test in line 38 once. □

6.2 Implementation and Timing Results

We have implemented algorithm `Inversion` as presented above using Maple's RECDEN package, as in the ModularC-GCD implementation. The reader may find our Maple code for our software there together with several examples and their output at

`http://www.cecm.sfu.ca/CAG/code/MODGCD`

The remainder of this section will be used to compare our algorithm with the `Inverse` procedure in the `RegularChains` package of Maple. The algorithm that `Inverse` uses works by using an extended subresultant polynomial remainder sequence algorithm if there is more than 1 extension. If there is only 1, it uses a basic modular extended greatest common divisor algorithm. It is coded this way because one needs to only use $\text{gcdex}(a, t_1) \pmod{T_0}$ to compute a^{-1} if there is only 1 extension. Because of this, their procedure outperforms ours in the single extension case. However, multiple extensions are the main point of interest in this thesis and by those wishing to compute modulo triangular sets, so this should not be viewed as a weakness.

Note that for testing invertibility `Inverse` uses the same algorithm as `RegularGcd` and `Inversion` uses the same algorithm as `ModularC-GCD`. Because of this, we do not bother

making timing tests for this case. Please refer to tables 5.1, 5.2, 5.3 for the results that would be found.

Finally, we would like to conclude with some timing tests for computing inverses. We first construct random triangular sets where each t_i is monic in z_i and dense in $z_1, \ldots, z_{i-1}$ with random two digit coefficients. We then generate $a \in R$ that is dense in all variables again with two digit coefficients. Because of the random generation, it is likely that a is invertible, so we compute a^{-1} using our `Inversion` procedure. We then compute a^{-1} using the `Inverse` command. Maple code for generating the test inputs is included the appendix.

n	degree	Inversion	Inverse
1	[64]	0.181	0.068
1	[128]	1.32	0.237
1	[300]	17.174	3.748
2	[2, 64]	4.288	61.834
3	[5, 5, 5]	9.747	14.676
4	[3, 3, 3, 3]	6.536	112.987
3	[2, 2, 2]	0.013	0.026
4	[2, 2, 2, 2]	0.086	0.126
5	[2, 2, 2, 2, 2]	0.586	28.637
6	[2, 2, 2, 2, 2, 2]	8.197	> 10000

Table 6.1: The first column is the number of algebraic variables, the second is the degree of the extensions, the third is the CPU time it took to compute the inverse of the inputs for Inversion, the fourth is the CPU time it took to compute the inverse of the inputs for `Inverse`. All times are in seconds.

The time tests confirm that `Inverse` is faster for a single extension. However, as soon as there is a second extension, `Inversion` is faster. This can be seen succinctly by viewing the time tests for extensions of degree [128] and $[2, 64]$ in Table 6.1. Similarly to the time tests for gcd computation, the `RegularChains` package is very bad at handling multiple extensions.

Chapter 7

Conclusion and Future Work

In summary, creating algorithms for computation modulo triangular sets is difficult because of zero-divisors. We have applied two new techniques to resolve this difficulty: fault tolerant rational reconstruction and Hensel lifting. We have developed two applications using these techniques, namely, a modular gcd algorithm and an inversion algorithm. Using timing tests, we have shown that our approach gives a practical improvement over the algorithms used in Maple's `RegularChains` package. There is plenty of room for improvement with our algorithms that should be discussed. For the modular gcd algorithm:

1. We could perhaps avoid the radical prime test as done in the algebraic number field case in [15]. This would not be a large gain as the radical prime test takes a small fraction of the running time.

2. The division test is a large bottleneck of the algorithm and should be the first place to optimize. We have attempted to create a modular division algorithm for this; However, it did not present a large gain, if any at all, unless we were willing to give up a proof of correctness in the algorithm. The difficulty comes in bounding the rational coefficients of $\gcd(a, b)$ for $a, b \in R[x]$ where $R = \mathbb{Q}[z_1, \ldots, z_n]/T$. We have researched bounds on these, but they are far from tight.

3. The algorithm only works with univariate polynomials. The obvious way to handle multivariate polynomials would be to use evaluations and interpolation as is done by Brown over $\mathbb{Q}$ with no extensions, see [5]. This would require proving results about uniqueness of interpolation over products of fields. We could also make use of sparse interpolation techniques here, see [16] and [27].

For the inversion algorithm:

1. We can likely ignore the radical prime test. We could not construct a proof that does not require it though.

2. Similarly to the modular gcd algorithm, testing if $ab = 1 \pmod{T}$ is a large bottleneck. We are not sure how to get around this besides proving a tight bound on the

rational numbers appearing in a^{-1}. However, no bound will be tight. We randomly generated a dense triangular set $T = \{t_1, t_2\}$ where $\text{mdeg}(t_1) = \text{mdeg}(t_2) = 3$ and integer coefficients only consisting of two digits. We then randomly generated a dense element modulo T again with with only 2 digit coefficients. Well, a^{-1} is dense and consists of fractions with 46 digits in the numerator and denominator. This is the worst case that the bound would have to cover for. However, the same bound on $(a^{-1})^{-1}$ would be a gross overestimation. Because these examples are so easy to construct, we think it is best to do the multiplication test.

3. An optimization could be to use a modular algorithm to compute a^{-1} rather than a lifting algorithm based on Newton iteration. This would certainly help the univariate case. It could be worth it to code it as a special case, similarly to the `Inverse` function.

A major flaw of our algorithms when comparing them to those of Maza, Schost, Li in the `RegularChains` package of Maple is in terms of robustness. In particular, their algorithms do not require monic inputs and do much more than just the two applications we have laid out. However, we have shown our algorithms are much faster. Because of this, the technique of using Hensel lifting for modular algorithms is one worth pursuing and should allow for more modular algorithms to be created. For instance, a modular algorithm for more resultant computation (that uses the Euclidean algorithm) in $R[x]$ would be useful. Also, matrix computations deserve some attention. The problem of computing matrix inverses over triangular sets has been explored in [21] without the modular approach. Both matrix inversion and linear system solving could benefit from a modular approach.

Bibliography

[1] John Abbott. Fault-tolerant modular reconstruction of rational numbers. Journal of Symbolic Computation, Volume 80, pages 707 – 718. May-June 2017.

[2] P. Aubry, D. Lazard, and M. Moreno Maza. On the theories of triangular sets. J. Symb. Comp., **28**: 105 – 124, 1999.

[3] Janko Boehm, Wolfram Decker, Claus Fieker and Gerhard Pfister. The use of bad primes in rational reconstruction. Math Comp **84**. 3013–3027. 2015.

[4] S. Bosch. Algebraic Geometry and Commutative Algebra. Springer-Verlag London. 2013.

[5] W. S. Brown. On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors. J. ACM **18**: 478 – 504. 1971.

[6] C. Chen, F. Lemaire, O. Golubitsky, M. Moreno Maza and W. Pan. Comprehensive Triangular Decomposition Proceedings of CASC 2007: Computer Algebra in Scientific Computing, pages 73–101, Lecture Notes in Computer Science, vol. **4770**, Springer-Verlag, 2007.

[7] D. Cox, J. Little, D. O'Shea. Ideals, Varieties and Algorithms. Springer-Verlag, 1991.

[8] D. Cox, J. Little, D. O'Shea. Using Algebraic Geometry. Springer-Verlag, 1998.

[9] D. Cox. Galois Theory. Wiley-Interscience, 2004.

[10] X. Dahan, M. Moreno Maza, E. Schost, W. Wu and Y. Xie. Lifting Techniques for Triangular Decompositions. Proceedings of ISSAC'05, Beijing, China, ACM Press, 2005.

[11] David Dummit, Richard M. Foote. Abstract Algebra. Englewood Cliffs, N.J. Prentice Hall, 1991.

[12] M. J. Encarnacion. Computing GCDs of Polynomials over Algebraic Number Fields, J. Symb. Comp. **20**: 299 – 313, 1995.

[13] J. von zur Gathen and J. Gerhard, *Modern Computer Algebra*, 3rd ed., Cambridge University Press, 2013.

[14] K. O. Geddes, S. R. Czapor, and G. Labahn. Algorithms for Computer Algebra. Kluwer, 1992.

[15] Mark van Hoeij and Michael Monagan, A modular GCD algorithm over number fields presented with multiple extensions. Proceedings of ISSAC '02, ACM Press, pp. 109 – 116. 2002.

[16] Jiaxiong Hu and Michael Monagan, A Fast Parallel Sparse Polynomial GCD Algorithm. Proceedings of ISSAC '16, ACM Press, pp. 271–278. 2016.

[17] E. Hubert. Notes on Triangular Sets and Triangulation-Decomposition Algorithms I: Polynomial Systems. In Symbolic and Numerical Scientific Computing edited by F. Winkler and U. Langer. Lecture Notes in Computer Science 2630, pp. 1 – 39. 2003.

[18] L. Langemyr, S. McCallum. The Computation of Polynomial GCDâĂŹs over an Algebraic Number Field, J. Symbolic Computation **8**, pp. 429 – 448. 1989.

[19] Xin Li, Marc Moreno Maza, and Wei Pan. Computations Modulo Regular Chains. Proceedsings of ISSAC '09, pp. 239–246, 2009. See also https://arxiv.org/pdf/0903.3690.

[20] X. Li, M. Moreno Maza, and E Schost. Fast Arithmetic for Triangular Sets: from Theory to Practice. Journal of Symbolic Computation, **44**(7): 891-907, 2009.

[21] Marc Moreno Maza, Eric Schost, Paul Vrbik. Inversion Modulo Zero-Dimensional Regular Chains. Proceedings of Computer Algebra in Scientific Computing (CASC 2012), Springer Verlag, LNCS **6885**, pages 224-235, 2012.

[22] M. B. Monagan. In-place arithmetic for polynomials over $\mathbf{Z}_n$.
Proceedings of DISCO '92, Springer-Verlag LNCS, **721**, pp. 22–34, 1993.

[23] M. B. Monagan. Maximal Quotient Rational Reconstruction: An Almost Optimal Algorithm for Rational Reconstruction. *Proceedings of ISSAC '2004*, ACM Press, pp. 243–249, 2004.

[24] Barry Trager. Algebraic Factoring and Rational Function Integration. SYMSAC '76 Proceedings of the third ACM symposium on Symbolic and algebraic computation. Pages 219–226

[25] Paul S. Wang, M.J.T. Guy, and J.H. Davenport. P-adic reconstruction of rational numbers. ACM SIGSAM Bulletin **16**(2): 2–3, 1982.

[26] P.J. Weinberger and L.P. Rothschild. Factoring Polynomials over Algebraic Number Fields. *ACM Trans. on Math. Soft.* **2**(4): 335–350, 1976.

[27] Richard Zippel, Probabilistic Algorithms for Spare Polynomials. Proceedings of EUROSAM '79, Springer Lecture Notes on Computer Science **72**, pp 216–226. 1979.

Appendix A

Code for Time Tests

```
#### These are for the inversion algorithm timetests ####
############################################################

reduce_by_tset := proc(f,rT)
# Input :: f := a polynomial in the recden format.
#          T := a triangular set in the recden format.
#
# This method reduces f modulo T. It will work over Q or Z/p.
local F,t;
    F := f;
    if nops(rT) >= 1 then
        for t in rT do
            F := phirpoly(F,t);
        od;
    fi;
    return F;
end proc:

randpoly_dense := proc(X,d)
# Create a random dense polynomial f with deg(f,X[i]) < d[i].
# The integer coefficients will have two decimal digits.
local n;
    n := nops(X);
    if n = 1 then
        return randpoly(X[1],degree=d[1]-1,dense);
    else
        return randpoly(X[n], coeffs=proc() randpoly_dense(X[1..n-1], d[2..n]) end proc,
                    dense, degree=d[1]-1);
    fi;
end proc:
```

```
construct_triangular_set := proc(T, Z)
# Input :: T := A triangular set with variables in Z.
#          Z := A list of variables.
#
# It is assumed that T is of the form t1(Z[-1]), t2(Z[-1],Z[-2]), etc, and that
# the polynomials in T are in Maple's format.
#
# This method also turns T into a reduced triangular set.
local rT,i,n;
    n := nops(Z);
    rT := [rpoly(T[1],Z[-1]), seq(1..n-1)];
    for i from 2 to nops(Z) do
        rT[i] := rpoly(T[i], Z[n-i+1..n]);
        rT[i] := reduce_by_tset(rT[i], rT[1..i-1]);
    od;
    return rT;
end proc:



randdensetriset := proc(X,d)
# Create a random triangular set:
#     T[1] will be in X[-1] with degree d[1].
#     T[2] will be in X[-1],X[-2] with degree d[2].
#     etc.
#
# All polies will be dense and monic in their main variable.
local n,T,R,i;
    n := nops(d);
    T := [seq(1..n)];
    for i from 1 to n do
        T[i] := X[n-i+1]^d[i] + randpoly_dense(X[n-i+1..n], d[1..i]);
    od;
    return construct_triangular_set(T,X);
end proc:
```

```
# N is the list of extension degrees.
# For example, [3,3] is two extensions each of degree 3.
N := [[4], [8], [16], [32], [64], [128], [256], [512], [3,3], [3,3,3],
      [3,3,3,3], [2,2], [2,2,2], [2,2,2,2], [2,2,2,2,2]];

k := 1;
for d in N do
    # Create triangular set.
    n := nops(d);
    X := [seq(Z[i], i = 1..n)]:
    T := randdensetriset(X, d):


    # Create element to be inverted.
    f := randpoly_dense(X, d);
    F := reduce_by_tset(rpoly(f, X), T):


    # Perform our procedure.
    st1 := time();
    H := [inversionrpoly(F)]:
    et1 := time();


    # Perform RegularChains procedure.
    R := PolynomialRing(X):
    rc := Chain([seq(rpoly(T[i]),i=1..nops(T))],Empty(R),R):

    st2 := time();
    G := Inverse(f,rc,R):
    et2 := time();

    # Store times in a table along with important parameters.
    times_ours[k] := [n,d,et1-st1]:
    times_maple[k] := [n,d,et2-st2]:
    k := k + 1;
od:
```

```
##########################################
#### These are for the cgcd time tests ####
##########################################

randpolyn := proc(X,D) local i,C,f,n;
# Creates a randomly generated dense polynomial.
   n := nops(X);
   if n=1 then C := rand(1..99); f := add(C()*X[1]^i,i=0..D[1]-1);
   else f := add( randpolyn(X[2..-1],D[2..-1])*X[1]^i,i=0..D[1]-1);
   fi;
end:

randtriset := proc(X,D)
# Creates a randomly generated dense triangular set.
   T := [];
   n := nops(X);
   Y := [seq(X[n-i],i=0..n-1)];
   for i from 1 to n do
       x := Y[i];
       d := D[i];
       T := [op(T),x^d+(randpolyn(Y[1..i],D[1..i]))];
   od;
   return construct_triangular_set(T,X);
end:

construct_triangular_set := proc(T, Z)
# Input :: T := A triangular set with variables in Z.
#          Z := A list of variables.
#
# It is assumed that T is of the form t1(Z[-1]), t2(Z[-1],Z[-2]), etc, and that
# the polynomials in T are in Maple's format.
#
# This method also turns T into a reduced triangular set.
local rT,i,n;
    n := nops(Z);
    rT := [rpoly(T[1],Z[-1]), seq(1..n-1)];
    for i from 2 to nops(Z) do
        rT[i] := rpoly(T[i], [seq(Z[j],j=n-i+1..n,1)],  [seq(T[j],j=1..i-1,1)]);
    od;
    return rT;
end proc:
```

```
# N is the list of extension degrees.
# For example, [3,3] is two extensions each of degree 3.
N := [ [4], [2,2], [3,3], [2,2,2], [4,4], [2,2,2,2], [3,3,3], [4,2,4], [8,8], [4,4,4] ];

k := 1;
for d in N do
    n := nops(d);
    X := [x,seq(Z[i], i = 1..n)]:
    T := randtriset(X[2..], d):

    a := add( randpolyn(X[2..],d)*x^i, i=0..6 );
    b := add( randpolyn(X[2..],d)*x^i, i=0..5 );
    g := add( randpolyn(X[2..],d)*x^i, i=0..4 );

    A := reduce_by_tset(rpoly(a*g, X), T):
    B := reduce_by_tset(rpoly(b*g, X), T):

    st_cgcd := time();
    NPRIMES := 0; DIVTIME := 0;
    H := [cgcd(A,B,2^30)]:
    et_cgcd := time();

    MAX := 0;
    for h in H do MAX := max(MAX,maxratrpoly(h)) od;

    R := PolynomialRing(X):
    rc := Chain([seq(rpoly(T[i]),i=1..nops(T))],Empty(R),R):

    A1 := rpoly(A):
    B1 := rpoly(B):

    st_reg := time();
    G := RegularGcd(A1,B1,x,rc,R):
    et1_reg := time();
    for h in G do NormalForm(h[1],h[2],R): od:
    et2_reg := time();
    MAXG := 0;
    for h in G do MAXG := max(MAXG, nops(expand(h[1])) ); od;

    times_cgcd[k] := [d,et_cgcd-st_cgcd,nops(H),NPRIMES,ilog10(MAX+1)+1,DIVTIME]:
    times_reggcd[k] := [d,et1_reg-st_reg,nops(G),MAXG]:
od:
```