CHALMERS

$$R^q \to R^p \to M \to 0$$

Constructive Algebra in Functional Programming and Type Theory

Master of Science Thesis in the Programme Computer Science – Algorithms, Languages and Logic

ANDERS MÖRTBERG

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, May 2010

Constructive Algebra in Functional Programming and Type Theory

Anders Mörtberg

Examiner: Prof. Thierry Coquand

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover:
An exact sequence defining that the module M is finitely presented. This is related to the notion of coherent rings presented in chapter 3.

Department of Computer Science and Engineering
Göteborg, Sweden May 2010

Abstract

This thesis considers abstract algebra from a constructive point of view. The central concept of study is coherent rings − algebraic structures in which it is possible to solve homogeneous systems of linear equations. Three different algebraic theories are considered; Bézout domains, Prüfer domains and polynomial rings. The first two of these are non-Noetherian analogues of classical notions. The polynomial rings are presented from a constructive point of view with a treatment of Gröbner bases. The goal of the thesis is to study the proofs that these theories are coherent and explore how the proofs can be implemented in functional programming and type theory.

Acknowledgments

First of all I would like to thank Thierry Coquand for all help and support during the work on this thesis.

I would also like to thank Bassel Mannaa for interesting discussions and help with the implementation. The comments presented during the opposition was also very helpful.

Finally I would like to thank everyone that has read and given constructive and helpful comments on this thesis.

Contents

Chapter 1

Introduction

1.1 Background

> *"It is important to keep in mind that constructive algebra is algebra; in fact it is a generalization of algebra in that we do not assume the law of excluded middle."* [16]

Why is it that elementary algebra is so full of algorithms while advanced algebra is so full of nonconstructive arguments? Elementary algebra has factorization of polynomials, equation solving and matrix inversion. Advanced algebra on the other hand has notions such as arbitrary ideals of rings, prime and maximal ideals and Noetherian assumptions on rings [4]. For example, both the existence of maximal and prime ideals are usually proved using Zorn's lemma. Zorn's lemma relies on the axiom of choice which in turn implies the law of excluded middle [19].

Modern abstract algebra begun with the introduction of algebraic structures in the end of the 19th century. In the first half of the 20th century nonconstructive methods dominated. In 1967 Erret Bishop published a book called *Foundations of Constructive Analysis* which aimed to show that analysis could be approached constructively. This, together with increasingly powerful computers, led to a renaissance of constructive mathematics [16].

One of the main reasons to study constructive algebra is that it can give rise to new algorithms and ways to explore algebra using computers. The notion of computation is at the core of constructive mathematics. A constructive proof of the existence of a mathematical object gives a way to construct the object while a nonconstructive proof just proves the existence of such an object without necessarily giving a way to construct it. For example, a constructive proof that a polynomial can be factorized as a product of irreducible polynomials must provide the factorization while a nonconstructive proof just have to prove the existence of such a factorization without giving any *witness* of it.

Another reason to study constructive algebra is that it makes it possible to represent advanced algebra in type theory and thus to verify the correctness of mathematical proofs using computers. The reasons for this are the proofs-as-programs correspondence and the Brouwer-Heyting-Kolmogorov interpretation of intuitionistic logic which together give a way for representing mathematical propositions as types and proofs as programs. Note that it is also possible to ver-

ify classical mathematics using computers, but the point is that in constructive mathematics the proofs correspond to algorithms.

This thesis explores the question of how advanced algebra can be made constructive by considering classical structures where the assumptions of Noetherianity has been dropped. This will be defined and discussed further in the introduction to ring theory in chapter 2.

In linear algebra one of the main questions is how to solve homogeneous systems of linear equations, but in linear algebra the central notion is vector spaces which relies on the assumption that all nonzero elements has a multiplicative inverse. One main aim of this thesis is to look at what happens if this assumption is dropped. The proofs of the results should be constructive and be implemented in a functional programming language and eventually also verified in a constructive proof system.

1.2 Method

The results presented in this thesis has been implemented in the pure lazy functional programming language Haskell. The reason for using Haskell is that it has a powerful type system and the features that makes it suitable for implementing algebraic theories are mainly polymorphism and the type class system.

In order to specify the axioms of algebraic structures the automated testing tool QuickCheck [2] is used, since the axioms are natural to represent as QuickCheck properties and implementations of specific instances of the algebraic structures easily can be tested.

The final goal of the thesis is to represent the work in type theory, as an implementation in a logical proof system based on intuitionistic type theory, e.g. Agda[1] or Coq[2]. Due to time limitations this has not been done yet.

1.3 Previous work

There has been some previous work on computational algebra systems in Haskell. The HaskellForMaths[3] project by David Amos implements many important algorithms from combinatorics, group theory and commutative algebra. This project does not have a representation of algebraic structures and instead it uses the standard Haskell type classes. It also has an implementation of multivariate polynomials and the Buchberger algorithm.

A project that focuses on representing algebraic structures in Haskell is the numeric-prelude project[4]. This library contains many different structures like groups, rings, fields, modules, vector spaces, etc.

In type theory there are many examples of libraries for constructive algebra. The main interest of this project has been in implementations in Agda and Coq. The standard library of Agda contains representations of some basic algebraic structures but as far as I know there has been no larger projects in constructive algebra developed in Agda. The situation in Coq is quite different.

[1] `http://wiki.portal.chalmers.se/agda/`
[2] `http://www.lix.polytechnique.fr/coq/`
[3] `http://hackage.haskell.org/package/HaskellForMaths`
[4] `http://www.haskell.org/haskellwiki/Numeric_Prelude`

In [12] a framework for representing algebraic structures in Coq is presented. This is done as part of a project to give a formalized proof of the fundamental theorem of algebra. This is a part of the Constructive Coq Repository at Nijmegen[5] which is a large library containing formalized mathematics focusing mostly on constructive real numbers.

Another implementation of algebraic structures in Coq is the Mathematical Components project[6]. This is based on the ssreflect extension to Coq which was used in the formal proof of the Four-Color Theorem by Georges Gonthier [14]. In [11] possible ways to represent algebraic structures as part of this project is discussed together with problems related to the complexity of the representation.

Both of the references on implementations of algebraic theories in Coq has many further references to other work on representing constructive mathematics in type theory, but none of them implement neither Bézout domains nor Prüfer domains. Polynomial rings, on the other hand, has been represented in Coq together with a verified implementation of the Buchberger algorithm for computing Gröbner bases [17].

All of the major computer algebra systems like Maple, Mathematica and Matlab implement algorithms for solving systems of linear equations and computing Gröbner bases. These systems are based on less general algorithms than the algorithms presented in this thesis. Instead they focus on more specialized algorithms in order to be able to do as much optimization as possible.

The results on Bézout domains and Prüfer domains is based on the work presented in the PhD thesis of Maimouna Salou [18]. As part of it many of the proofs has been represented in the Axiom computer algebra system.

A project that studies generalized linear algebra is the homalg project[7]. It is a project that aims to translate as much homological algebra as possible into computer programs. This project is implemented using object oriented programming.

All of the web pages that has been referred to in this section has been visited in May 2010.

1.4 Outline

In chapter 2 ring theory is introduced for a reader without a background in abstract algebra. The most basic definitions that are necessary in order to read the thesis are introduced. This chapter can be read very briefly by a reader who is already familiar with ring theory. The discussions on implementation of the concepts in functional programming and type theory are probably interesting even if the reader already know the subject.

Chapter 3 presents coherent rings from a constructive point of view. Traditionally these are considered in terms of module theory but here they are described in terms of solving equations à la linear algebra.

The following two chapters discusses *Bézout domains* and *Prüfer domains* which are constructive analogies of principal ideal domains and Dedekind domains. Just as in classical mathematics where principal ideal domains are a subset of Dedekind domains are Bézout domains a subset of Prüfer domains.

[5] `http://c-corn.cs.ru.nl/`

[6] `http://www.msr-inria.inria.fr/Projects/math-components`

[7] `http://homalg.math.rwth-aachen.de/`

The high-point of these chapters are the proofs that the classes of rings are coherent and thus that it is possible to solve homogeneous systems of equations over them.

In chapter 6 polynomial rings are presented, these are rings of polynomials with coefficients from an underlying ring. The theory of Gröbner bases is presented from a constructive point of view together with the famous Buchberger algorithm used to compute these. In the end of the chapter there is a proof that these rings are also coherent.

Finally the results of the implementation are presented together with some examples and a discussion on limitations and further work.

Chapter 2

Introduction to ring theory

This chapter should serve as a short introduction to the concepts of ring theory that are necessary in order to understand the thesis. It does not claim to be a complete and thorough presentation of all concepts of basic ring theory. For a good introduction to general abstract algebra see [9] and for an introduction to some of the more advanced concepts see [1]. If the reader already has a good understanding of ring theory this chapter can be read briefly. Most important are the notes about how to define the concepts in functional programming and type theory.

2.1 Rings

The most fundamental concept of this thesis is the concept of rings. These can be defined compactly in terms of groups and monoids, but here the definition is a bit more verbose in order to give a summary of all the properties of rings in one place.

Definition 2.1. A *ring* is a set R equipped with two binary operations called *addition* and *multiplication* written $+, \bullet : R \times R \to R$ respectively. The axioms that the triple (R,+,•) must satisfy are

1. Closure under addition: $\forall a\, b \in R.\, a + b \in R$
2. Associativity of addition: $\forall a\, b\, c \in R.\, (a + b) + c = a + (b + c)$
3. Existence of additive identity: $\exists 0 \in R.\, \forall a \in R.\, 0 + a = a + 0 = a$
4. Existence of additive inverse: $\forall a \in R.\, \exists b \in R.\, a + b = b + a = 0$
5. Commutativity of addition: $\forall a\, b \in R.\, a + b = b + a$
6. Closure under multiplication: $\forall a\, b \in R.\, a \bullet b \in R$
7. Associativity of multiplication: $\forall a\, b\, c \in R.\, (a \bullet b) \bullet c = a \bullet (b \bullet c)$d
8. Existence of multiplicative identity: $\exists 1 \in R.\, \forall a \in R.\, 1 \bullet a = a \bullet 1 = a$
9. Left distributivity of multiplication over addition:

$$\forall a\, b\, c \in R.\, a \bullet (b + c) = (a \bullet b) + (a \bullet c)$$

10. Right distributivity of multiplication over addition:

$$\forall a\, b\, c \in R.\, (a + b) \bullet c = (a \bullet c) + (b \bullet c)$$

First some conventions. Multiplication is often not written explicitly, so $a \bullet b$ is written ab. The additive inverse is often written $a - b$ which means $a + (-b)$ where $(-b)$ is the additive inverse of b. The set of nonzero elements of a ring is written as R^*.

Examples of rings include $(\mathbb{Z}, +, \cdot)$ where $+$ and $\cdot$ denote the ordinary addition and multiplication for the integers. Other examples are $\mathbb{Q}, \mathbb{R}, \mathbb{C}$ with the ordinary definitions of addition and multiplication.

Note on implementation. In Haskell this can be represented as a type-class:

```
class Ring a where
  (<+>) :: a → a → a
  (<*>) :: a → a → a
  neg   :: a → a
  zero  :: a
  one   :: a
```

The ring axioms can also be represented in Haskell. The axioms are represented as functions which should be used to test that an implementation satisfies the laws. For example the property that multiplication is left distributive over addition can be specified as:

```
propLeftDist :: (Ring a, Eq a) ⇒ a → a → a → Bool
propLeftDist a b c = a <*> (b <+> c) ≡ (a <*> b) <+> (a <*> c)
```

In type theory the axioms would be possible to represent at the type level using dependent records. Then the structure would also contain the axioms and the user would have to prove that a structure satisfies them in order to construct an instance. This is better since the implementation would be proved correct and not just randomly tested.

Definition 2.2. A *commutative ring* is a ring (R,+,•) satisfying the axiom that multiplication is commutative:

$$\forall a\, b \in R.\, a \bullet b = b \bullet a$$

Note on implementation. Commutative rings can be represented as an empty type class in Haskell since they do not introduce any new operations to the structure. But since there is one more axiom that also has to be represented.

```
class Ring a ⇒ CommutativeRing a

propMulComm :: (CommutativeRing a, Eq a) ⇒ a → a → Bool
propMulComm a b = a <*> b ≡ b <*> a
```

This thesis will only consider commutative rings. All of the above examples of rings are commutative.

An example of a ring that is not commutative is the ring of $n \times n$ matrices, written $M_n(R)$, where R is the ring of the elements. Here addition and multiplication are the standard operations on matrices and it is easy to construct an example to show that matrix multiplication is non-commutative.

The previous examples have all been infinite, but there are also many finite rings. A fundamental class of finite rings are the *ring of integers modulo n*, written $\mathbb{Z}_n$[1]. This corresponds to the elements $a \in \mathbb{Z}$ in the same congruence class modulo n, for example $\mathbb{Z}_3 \simeq \{0, 1, 2\}$ since there are three congruence classes modulo 3. Addition and multiplication are defined by using the addition and multiplication of $\mathbb{Z}$ and then computing modulo n.

Note on implementation. To implement $\mathbb{Z}_n$ the power of dependent types would be desirable to have. The compiler would then be able to distinguish elements from different rings and verify that they are for instance not multiplied. The reason is that the type of $\mathbb{Z}_n$ *depends* on the value of n. It is possible to represent integers at the type level in Haskell but it is a bit cumbersome and having real dependent types is preferable.

Another example of rings is *polynomial rings*, written $R[x_1, \ldots, x_n]$ where R is the ring of the coefficients and $x_1, \ldots, x_n$ are variables. It is easy to see that these rings are commutative if R is. A concrete example of an element of $\mathbb{Z}[x, y]$ is $3x + 7y^2$. This special class of rings has many applications and are considered in more detail in chapter 6.

Definition 2.3. An *integral domain* is a commutative ring satisfying:

$$\forall a\, b \in R.\, (ab = 0 \rightarrow a = 0 \vee b = 0)$$

All of $\mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$ form integral domains with the usual definitions of addition and multiplication. For $\mathbb{Z}_n$ things is a bit more complicated. For example $\mathbb{Z}_6$ is not an integral domain since $2 \cdot 3 = 0$ modulo 6. In fact $\mathbb{Z}_n$ is an integral domain iff n is prime.

Note on implementation. Representing integral domains is a bit more difficult. Just as commutative rings they do not introduce any new operations, but how should the property be tested?

One way to do it is to test if $ab = 0$ and then check that either a or b are zero and also test the axioms for commutative rings. If $ab \neq 0$ it should only be checked that the axioms for commutative rings are satisfied.

Here type theory would be superior to functional programming, because the probability to generate a random counter example can be fairly small, since the product of most elements probably will not be zero. So having a proof that this holds would be much better.

The fact that $\mathbb{Z}_n$ is only an integral domain iff n is prime is another motivation that this structure is best captured by type theory. Using a representation of integers at the type level in Haskell it is possible to define primality testing, but this is hard and terribly slow. So having a language designed to do computation at the type level would be much better.

Definition 2.4. A *field* is an integral domain in which all nonzero elements has a multiplicative inverse:

$$\forall a \in R^*.\, \exists b \in R^*.\, ab = 1$$

[1] Often written $\mathbb{Z}/n\mathbb{Z}$.

This is often written using standard division notation, so a/b or $\frac{a}{b}$ actually means $a \bullet (b^{-1})$ where b^{-1} is the multiplicative inverse of b.

Some of the infinite rings that presented so far are fields, these are $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$. Some finite fields have also been presented. These are, just as for integral domains, $\mathbb{Z}_n$ where n is a prime number.

Note on implementation. The representation of fields is very similar to the representation of rings. The new operation that fields add is the ability to compute multiplicative inverses. This can be implemented and specified as:

```
class IntegralDomain a ⇒ Field a where
  inv :: a → a

propMulInv :: (Field a, Eq a) ⇒ a → Property
propMulInv a = a ≠ zero ⟹ inv a <*> a ≡ one
```

This is the final definition of this section. It is possible to make these definitions more fine-grained by having several intermediate structures like semi rings with and without a one or starting from monoids and groups to construct rings. The reason not to do this is simply that the concepts presented here is sufficiently complex for the following chapters. For a general abstract algebra library the approach of having more structures would be much more sensible.

2.2 Ideals

The concept of ideals is very important in commutative algebra. They are generalizations of many concepts of the integers like "even number" or "prime number". Since this thesis only consider commutative rings are all ideals *two-sided*, that is left ideals are equal to right ideals. For non-commutative rings it would have been possible to define left and right ideals instead.

Definition 2.5. For a commutative ring $(R, +, \bullet)$ and *ideal* I is a subset $I \subseteq R$ such that:

1. Closure of addition: $\forall a\, b \in I.\, a + b \in I$
2. Closure of multiplication by an element of R: $\forall a \in I.\, \forall b \in R.\, ab \in I$

In short the ideals can be described as the additive subgroups of R which are closed under multiplication by any element of R. The two canonical ideals are the zero-ideal $\{0\}$ and the whole ring R. A more interesting example is the even integers, $2\mathbb{Z}$, which form an ideal of $\mathbb{Z}$ since the addition of two even numbers is even and the result of multiplying any number with an even number is even.

This defines arbitrary ideals of rings and is not suited for constructive algebra. The interesting ideals are instead the ideals which are *finitely generated.*

Definition 2.6. An ideal I of a ring R is *finitely generated* if there exist a finite subset $X \subseteq I$ such that all elements of I can be written as a linear combination of the elements of $X = \{x_1, \ldots, x_n\}$ and R. That is:

$$\forall a \in I.\, \exists r_1, \ldots, r_n \in R.\, a = x_1 r_1 + \cdots + x_n r_n$$

The ideal generators are not written using the standard set notation but with $\langle \dots \rangle$. Examples of finitely generated ideals are both of the canonical ideals where the zero ideal is generated by $\langle 0 \rangle$ and the whole ring is generated by $\langle 1 \rangle$. The even integers are generated by $\langle 2 \rangle$, but they are also generated by $\langle 2, 4 \rangle$. So $\langle 2 \rangle$ and $\langle 2, 4 \rangle$ generate the same subset in $\mathbb{Z}$ and are thus equal. One important property of the ideals in $\mathbb{Z}$ is that they all can be generated by one element. This property of ideals have a special name.

Definition 2.7. A *principal ideal* is an ideal generated by only one element.

Rings like $\mathbb{Z}$ in which all ideals are principal are classically called *principal ideal domains*. But constructively this definition is not suitable. Instead we would only want to consider rings in which all finitely generated ideals are principal. These rings are called *Bézout domains* and are considered in chapter 4.

Note on implementation. Finitely generated ideals can be represented by its set of generators. In Haskell this can be written as:

```
data CommutativeRing a ⇒ Ideal a = Id [a]
```

In type theory it would also be possible to consider the ideals that are not finitely generated, since in type theory it would be possible to represent ideals by their logical properties.

Now some some operations on ideals and fundamental properties of ideals will be considered.

Definition 2.8. The *sum* of two ideals I and J is the set of all $x + y$ where $x \in I$ and $y \in J$.

So if $I = \langle x_1, \dots, x_n \rangle$ and $J = \langle y_1, \dots, y_m \rangle$ then

$$I + J = \langle x_1, \dots, x_n, y_1, \dots, y_m \rangle$$

Definition 2.9. The *product* of two ideals I and J is the ideal IJ generated by all products xy where $x \in I$ and $y \in J$.

The intersection of two ideals is also an ideal. But there is no general method for computing the set of generators for the intersection of two ideals in arbitrary rings. In chapter 3 it will be established that if the intersection of two finitely generated ideals is finitely generated, the ring is coherent.

In fact the ideals form a complete lattice with respect to inclusion with the sum and intersection operations. A lattice is a partially ordered set where every pair of elements have a least upper bound and a greatest lower bound. This lattice need not be distributive, that is the operators need not distribute over each other, but in chapter 5 it will be proved that one way to define *Prüfer domains* is by this property.

Now an example of ideal operations. Consider $\langle 4 \rangle$ and $\langle 6 \rangle$ in $\mathbb{Z}$, that is the sets generated by all multiples 4 and 6. The sum $\langle 4 \rangle + \langle 6 \rangle = \langle 4, 6 \rangle = \langle 2 \rangle$ since $2 = 4 \cdot (2) + 6 \cdot (-1)$. The product is $\langle 4 \rangle \langle 6 \rangle = \langle 24 \rangle$ and the intersection $\langle 4 \rangle \cap \langle 6 \rangle$ is the set generated by the lowest common multiple which will be proved in the next chapter. Thus the intersection of $\langle 4 \rangle$ and $\langle 6 \rangle$ is $\langle 12 \rangle$.

2.3 Discrete and strongly discrete rings

This section will consider some rings that are especially relevant for constructive mathematics.

Definition 2.10. A ring is called *discrete* if equality is decidable.

All of the rings studied in the thesis will be discrete. But there are many examples of rings that are not discrete. For example $\mathbb{R}$ is not discrete since it is not possible to decide if two irrational numbers are equal in finite time. Another example of rings that do not need to be discrete are formal power series rings; rings of polynomials with and infinite number of terms.

Definition 2.11. A ring is called *strongly discrete* if ideal membership is decidable.

This property is very strong. Many of the rings we have seen so far are strongly discrete, this is in fact tightly connected to whether division is decidable in the ring. In section 3.2 we will see that strong discreteness and coherence give us the possibility of solving arbitrary systems of the type $AX = B$.

2.4 Noetherian rings and Dedekind domains

This section will establish some classical notions that will play an important rôle throughout the thesis.

Definition 2.12. A ring is called *Noetherian* if every ideal is finitely generated.

This notion is not suitable for constructive mathematics since it relies on quantification over arbitrary subsets of the ring. It is not possible to formulate in first-order logic [4]. One of the goals of this thesis is to consider structures that are non-Noetherian analogues to classical notions.

Definition 2.13. A *Dedekind domain* is an integral domain in which every fractional ideal is invertible.

An ideal is invertible if there exists another ideal such that the product of the ideals is principal. Dedekind domains imply Noetherianity and is thus not suitable for constructive mathematics. Instead one consider *Prüfer domains* which are non-Noetherian analogues of Dedekind domains. These and invertible ideals will be considered in chapter 5.

Chapter 3

Coherent rings

All rings in this section are integral domains. One of the main aims of the following sections will be to prove that different rings are coherent. That means that it is possible to solve systems of equations in them.

3.1 Definition and properties

An elementary application of linear algebra is to solve systems of linear equations in many variables. But in linear algebra the central concept of study is *vector spaces* which implies that the underlying structures are fields. So when computing the solution of a system of equations one is free to use the assumption that the elements are invertible. But what happens if you drop the assumption of invertibility and just look at integral domains? This is one of the motivations to study coherence.

Definition 3.1. A ring R is *coherent* if every finitely generated ideal is finitely presented. This means that given a matrix $M \in R^{1\times n}$ there exist a matrix $L \in R^{n\times m}$ for $m \in \mathbb{N}$ such that $ML = 0$ and

$$MX = 0 \quad \leftrightarrow \quad \exists Y \in R^{m\times 1}.\ X = LY$$

This means that it is possible to compute a set of generators for solutions of equations in a coherent ring. In other words that the module of solution for $MX = 0$ is *finitely generated*.

Note on implementation. The property of coherence is quite hard to capture in Haskell. The content that can be captured is that it is possible to compute the matrix L given M such that $ML = 0$. This can be represented as:

```
class IntegralDomain a ⇒ Coherent a where
  solve :: Vector a → Matrix a

propCoherent :: (Coherent a, Eq a) ⇒ Vector a → Bool
propCoherent m = isSolution m (solve m)
```

Here isSolution just check that all elements in the product ML are zero. The logical aspects of coherence is harder to represent. But in type theory this would be possible and it would be interesting to see what this would give compared to what the Haskell approach gives.

Not only is it possible to solve equations in a coherent ring, but in fact it is possible to compute generators for any homogeneous *system* of equations in a coherent ring. The proof of this proposition is a translation of the proof of proposition 1.2 in [18].

Proposition 3.2. *In a coherent ring it is possible to solve a system $MX = 0$ where $M \in R^{m\times n}$ and $X \in R^{n\times 1}$.*

Proof. Let $M_i \in R^{1\times n}$ be the rows of M. By coherence it is possible to solve $M_1X = 0$ and get $L_1 \in R^{n\times p_1}$ such that

$$M_1X = 0 \leftrightarrow \exists Y \in R^{p_1\times 1}.\ X = L_1Y$$

Now replace X in $M_2X = 0$ by L_1Y and get $M_2L_1Y = 0$. By coherence we now obtain a new matrix $L_2 \in R^{p_1\times p_2}$ such that

$$\begin{aligned} M_1X = M_2X = 0 \quad &\leftrightarrow \quad \exists Y \in R^{p_1\times 1}.\ X = L_1Y \text{ and } M_2L_1Y = 0 \\ &\leftrightarrow \quad \exists Z \in R^{p_2\times 1}.\ X = L_1L_2Z \end{aligned}$$

By iterating this method the solution $X = L_1L_2 \dots L_mZ$ with $L_i \in R^{p_{i-1}\times p_i}$, $p_0 = n$ and $Z \in R^{p_m\times 1}$ can be computed. □

Now we will consider the intersection of finitely generated ideals in coherent rings. This gives another way of characterizing coherent rings in terms of the intersection of ideals. For the more general formulation of this in terms of modules, that is vector spaces over arbitrary rings and not just fields, see theorem 2.4 on page 82 in [16].

Proposition 3.3. *The intersection of two finitely generated ideals in a coherent ring R is finitely generated.*

Proof. Let $I = \langle a_1, \dots, a_n\rangle$ and $J = \langle b_1, \dots, b_m\rangle$ be two finitely generated ideals in R. Consider the system

$$AX - BY = 0$$

where A is the $1\times n$ matrix $(a_1, \dots, a_n)$ and B is the $1\times m$ matrix $(b_1, \dots, b_m)$. Since the ring is coherent it is possible to compute a finite number of generators $(X_1, Y_1), \dots, (X_p, Y_p)$ of the solution. This mean

$$\begin{aligned} AX_1 &= BY_1 \\ &\vdots \\ AX_p &= BY_p \end{aligned}$$

To say that $\alpha \in I \cap J$ means that $\alpha \in I \wedge \alpha \in J$. This means that there exist x_i and y_i such that $\alpha = a_1x_1 + \dots + a_nx_n$ and $\alpha = b_1y_1 + \dots + b_my_m$. Which means that $a_1x_1 + \dots + a_nx_n = b_1y_1 + \dots + b_my_m$ which is exactly what the generators above give. Thus are one set of generators for the intersection $AX_1, \dots, AX_p$ and another set of generators are $BY_1, \dots, BY_p$. □

In fact this statement can be turned around to give the other direction also. The following proposition is the most important in this section and all of the coherence proofs will rely on this.

Proposition 3.4. *If R is an integral domain such that the intersection of two finitely generated ideals is finitely generated then R is coherent.*

Proof. The proof is by induction on the length of the system to solve. First consider

$$ax = 0$$

Here the only solution is the trivial solution. Now assume that it is possible to solve a system in $n-1$ variables and consider the case with $n \geq 2$ variables:

$$a_1x_1 + \ldots + a_nx_n = 0$$

If $a_1 = 0$ one set of solutions to the system is generated by $(1, 0, \ldots, 0)$, but it is also possible to use the induction hypothesis and get the generators $v_{i2}, \ldots, v_{in}$ for the system with $x_2, \ldots, x_n$ and the solutions of the system with n unknowns are generated by $(0, v_{i2}, \ldots, v_{in})$ and $(1, 0, \ldots, 0)$.

If $a_1 \neq 0$ the set $(0, v_{i2}, \ldots, v_{in})$ of solutions can be found by the induction hypothesis again. Further, by hypothesis it is possible to find $t_1, \ldots, t_p$ such that

$$\langle a_1 \rangle \cap \langle -a_2, \ldots, -a_n \rangle = \langle t_1, \ldots t_p \rangle$$

where $t_i = a_1w_{i1} = -a_2w_{i2} - \ldots - a_nw_{in}$. So if $a_1x_1 + \ldots + a_nx_n = 0$ then $a_1x_1 = -a_2x_2 - \ldots - a_nx_n$. We also have u_i such that

$$a_1x_1 = -a_2x_2 - \ldots - a_nx_n = \sum_{i=1}^{p} u_it_i$$

This implies that $a_1x_1 = \sum_{i=1}^{p} u_it_i = \sum_{i=1}^{p} u_ia_1w_{i1}$ which by the cancellation property give that $x_1 = \sum_{i=1}^{p} u_iw_{i1}$. Similarly

$$-a_2x_2 - \ldots - a_nx_n = \sum_{i=1}^{p} u_it_i = \sum_{i=1}^{p} u_i(-a_2w_{i2} - \ldots - a_nw_{in})$$

Some reorganization gives

$$a_2\left(x_2 - \sum_{i=1}^{p} u_iw_{i2}\right) + \ldots + a_n\left(x_n - \sum_{i=1}^{p} u_iw_{in}\right) = 0$$

This gives that $(w_{i1}, \ldots, w_{in})$ and $(0, v_{i2}, \ldots, v_{in})$ generate the module of solution. □

This gives a method for proving that rings are coherent. Now the "only" thing to prove is how to compute the intersection of finitely generated ideals and then this will imply that the ring is coherent. This also shows that coherent rings can be characterized only in terms of the intersection of finitely generated ideals.

Note on implementation. One thing worth emphasizing here is the dependence on the witnesses of the intersection. That is given two finitely generated ideals $I = \langle x_1, \ldots, x_n \rangle$ and $J = \langle y_1, \ldots, y_n \rangle$ the functions that compute the intersection must also give a set of witnesses. If the intersection $I \cap J = \langle z_1, \ldots, z_l \rangle$ then the function should give a_{ij} and b_{ij} such that

$$\begin{aligned} z_k &= a_{k1}x_1 + \ldots + a_{kn}x_n \\ &= b_{k1}y_1 + \ldots + b_{km}y_m \end{aligned}$$

Note that this only gives the witnesses in one direction, that is if $x \in I \cap J$ then $x \in I$ and $x \in J$.

3.2 Coherence and strongly discrete rings

The property of strong discreteness is very strong (as indicated by the name). If the ring is strongly discrete and coherent, not only is it possible to solve systems like $MX = 0$ but it is also possible to solve general systems of the kind $MX = A$. For ideal membership to be decidable constructively there has to be a method to test if $x \in \langle x_1, \ldots, x_n \rangle$ which also should give a witness if this is the case. The witness should be a list of w_i such that $\sum_i w_i x_i = x$. In other words, it should be possible to write x as a linear combination of the w_i.

Proposition 3.5. *If R is a strongly discrete coherent integral domain then it is possible to solve arbitrary linear systems. Given $MX = A$ it is possible to compute X_0 and L such that $ML = 0$ and*

$$MX = A \quad \leftrightarrow \quad \exists Y.\, X = LY + X_0$$

Proof. The solution L to the system $MX = 0$ can be computed by proposition 3.2. The particular solution X_0 can be computed using the same method as in that proof. The base case when M has only one row is clear since R is strongly discrete. That is if $M = (x_1, \ldots, x_n)$ and $A = (a)$ then the decidability of ideal membership give that if $a \in \langle x_1, \ldots, x_n \rangle$ one get witnesses w_i such that $x_1 w_1 + \cdots + x_n w_n = a$. □

This section establishes the properties of the rings to be studied throughout the thesis. One of the goals of the rest of the thesis is to give constructive proofs that different rings are coherent. This means that in the end there will be many examples of rings in which it is possible to solve systems of linear equations.

Chapter 4

Bézout domains

This section will consider a class of integral domains called Bézout domains. These are non-Noetherian analogues of principal ideal domains. Examples of Bézout domains are $\mathbb{Z}$ and $k[x]$. The goal of this section is to give some motivation and examples of Bézout domains and then prove that they are coherent. Finally there will be some discussion on what is required for them to be strongly discrete.

4.1 Definition

One interesting class of integral domains is principal ideal domains, that are integral domains in which all ideals are principal. This means that all principal ideal domains are Noetherian, but as mentioned in chapter 2 is Noetherianity not suitable for constructive mathematics. Thus are principal ideal domains not suitable and we instead introduce Bézout domains.

Definition 4.1. An integral domain R is a *Bézout domain* iff every finitely generated ideal is principal.

Note on implementation. This definition means that it is possible to compute t such that $\langle a_1, \ldots, a_n \rangle = \langle t \rangle$. The method should also compute witnesses that $\langle a_1, \ldots, a_n \rangle \subseteq \langle t \rangle$ and $\langle a_1, \ldots, a_n \rangle \supseteq \langle t \rangle$. This can be represented in Haskell as

```
class IntegralDomain a ⇒ BezoutDomain a where
  toPrincipal :: Ideal a → (Ideal a,[a],[a])
```

Here the first list is the witness that there exists u_i such that

$$t = a_1 u_1 + \cdots + a_n u_n$$

The second list is the witness that there exists v_i for each a_i such that

$$a_i = t v_i$$

4.2 Euclidean domains

Many examples of Bézout domains are Euclidean domains. These are rings with additional structure, namely an Euclidean function. This allows a generalization of the Euclidean algorithm on integers.

Definition 4.2. An *Euclidean domain* is an integral domain R with a function $f : R^* \to \mathbb{N}$ with the property that for $a, b \in R$ there exist $q, r \in R$ such that $a = bq + r$ where either $r = 0$ or $f(r) < f(b)$.

Euclidean domains are integral domains where it is possible to perform division with remainder. Examples include $\mathbb{Z}$ with the absolute value function and the ring of polynomials $k[x]$ with the degree function.

In order to show that Euclidean domains are Bézout domains some lemmas are needed. In Euclidean domains the Euclidean algorithm for computing the greatest common divisor of two elements can be applied. There is a generalized version which computes the greatest common divisor of more than two elements.

Lemma 4.3. *The generalized greatest common divisor (ggcd) of n elements can be computed by recursively applying the algorithm for computing the gcd of two elements. More specifically:*

$$ggcd(a_1, \ldots, a_n) = gcd(a_1, gcd(a_2, \ldots\ gcd(a_{n-1}, a_n) \ldots))$$

Proof. Consider the case with $a, b, c \in R$. Let $d = ggcd(a, b, c)$ then we have that $d|b$ and $d|c$ by the definition of gcd, so $gcd(b, c) = kd$ for some k. We also have that $d|a$, so $a = jd$ for some j. Now let $u = gcd(k, j)$, by the construction of u we have that $ud|a, b, c$, but since d is the greatest common divisor u must be a unit. Thus $d = gcd(a, gcd(b, c))$. By induction the other cases follow directly. □

In Euclidean domains it is also possible to compute the extended Euclidean algorithm; given $a, b \in R$ it is possible to compute $x, y \in R$ such that $ax + by = gcd(a, b)$. This algorithm can also be generalized.

Lemma 4.4. *Given $a_1, \ldots, a_n \in R$ it is possible to compute $x_1, \ldots, x_n \in R$ such that $a_1x_1 + \cdots + a_nx_n = ggcd(a_1, \ldots, a_n)$*

Proof. Given $a, b, c \in R$ we want to compute $x, y, z \in R$ such that $ax+by+cz = ggcd(a, b, c)$. We can compute $m, n \in R$ such that $bm + cn = gcd(b, c)$. Then there are $l, k \in R$ such that $ak + gcd(b, c) \cdot l = gcd(a, gcd(b, c))$ which by lemma 4.3 is equal to $ggcd(a, b, c)$. So we get that $ak + bml + cnl = ggcd(a, b, c)$ and thus $x = k$, $y = ml$ and $z = nl$. The equations for more than three variables follow by induction. □

Now it is possible to show that all Euclidean domains are Bézout domains. This gives a rich source of examples of Bézout domains.

Proposition 4.5. *Euclidean domains are Bézout domains.*

Proof. Follow directly from lemma 4.4. Given $\langle a_1, \ldots, a_n \rangle$ we can compute $t = ggcd(a_1, \ldots, a_n)$ such that $t = a_1x_1 + \cdots + a_nx_n$ and thus we have that $\langle a_1, \ldots, a_n \rangle = \langle ggcd(a_1, \ldots, a_n) \rangle$. □

Note on implementation. The implementation of this proof will have to compute the witnesses also. One direction follow directly from the proof since the extended Euclidean algorithm is used. The other direction is also direct since division is decidable and t divides all a_i.

4.3 Coherence of Bézout domains

The coherence of Bézout domains can be proved by considering the intersection of ideals. Since all finitely generated ideals are principal it is sufficient to consider only principal ideals.

Proposition 4.6. *Given two ideals $I = \langle a \rangle$ and $J = \langle b \rangle$ the intersection is $I \cap J = \langle lcm(a, b) \rangle$. Where lcm is the lowest common multiple of a and b.*

Proof. The equality is proved by considering both inclusions.

$\subseteq$: If $f \in \langle a \rangle \cap \langle b \rangle$ then $f \in \langle a \rangle$ and $f \in \langle b \rangle$. So $a|f$ and $b|f$. But there exist a lowest common multiple of a and b, $lcm(a, b)$, such that $a|lcm(a, b)$ and $b|lcm(a, b)$ so f must be a multiple of the lowest common multiple and thus $f \in \langle lcm(a, b) \rangle$.

$\supseteq$: If $f \in \langle lcm(a, b) \rangle$ then $lcm(a, b)|f$. Since $lcm(a, b)$ is a multiple of both a and b f must be a multiple of a and b also. This means that $f \in \langle a \rangle \cap \langle b \rangle$. □

This is what we need in order to know that Bézout domains are coherent.

Theorem 4.7. *Bézout domains are coherent.*

Proof. Direct consequence of propositions 3.4 and 4.6. □

Note on implementation. The implementation of this proof relies on the computation of the witnesses. Again it is sufficient to only consider the case where the ideals are principal. Given a and b we should compute $lcm(a, b)$, u and v such that

$$\begin{aligned} lcm(a, b) &= au \\ lcm(a, b) &= bv \end{aligned}$$

using the knowledge that we can compute $gcd(a, b) = g$, u_1, u_2, v_1 and v_2 such that

$$\begin{aligned} g &= au_1 + bu_2 \\ a &= gv_1 \\ b &= gv_2 \end{aligned}$$

To compute $lcm(a, b)$ use that

$$lcm(a, b) = \frac{ab}{gcd(a, b)}$$

So $lcm(a, b)$ can be computed as

$$lcm(a, b) = gv_1v_2 = g\left(\frac{a}{g} \cdot \frac{b}{g}\right) = \frac{ab}{g} \qquad (*)$$

Now that the lowest common multiple has been computed the witnesses has to be computed. But this is easy since by $(*)$ we get that

$$u = \frac{b}{g} = v_2$$
$$v = \frac{a}{g} = v_1$$

4.4 Bézout domains and strong discreteness

Recall that a ring is called strongly discrete if ideal membership is decidable. In order to decide ideal membership for Bézout domains we need to be able to decide divisibility.

Proposition 4.8. *A Bézout domain R is strongly discrete if division is decidable in R.*

Proof. To test if $x \in \langle x_1, \ldots, x_n \rangle$ we need to find w_i such that $x = \sum w_i x_i$. Since R is a Bézout domain we can find g such that $\langle g \rangle = \langle x_1, \ldots, x_n \rangle$.

Now, since divisibility is decidable, test if $g|x$ and if this is the case we know that $x \in \langle g \rangle$. This should also give us q such that $x = qg$ and since $\langle g \rangle \subseteq \langle x_1, \ldots, x_n \rangle$ we have u_i such that $g = \sum u_i x_i$.

The witness that $x \in \langle x_1, \ldots, x_n \rangle$ can now be computed by

$$x = qg = q \sum u_i x_i = \sum (q u_i) x_i$$

and thus $w_i = qu_i$. □

This means that we can solve arbitrary linear systems $MX = A$ for Bézout domains with decidable division and in particular over $\mathbb{Z}$ and $k[x]$.

4.5 GCD domains and fields of fractions

In classical mathematics a structure that is studied is called *unique factorization domains* (UFD) which are integral domains in which all elements can be written uniquely as a product of irreducible elements. For example $\mathbb{Z}$ is an UFD by the fundamental theorem of arithmetic. But as for principal ideal domains (PIDs) this relies on Noetherianity. In classical mathematics we have the following chain of inclusions.

Euclidean domains $\subset$ PIDs $\subset$ UFDs $\subset$ Integral domains

But without the Noetherian assumption we get Bézout domains instead of PIDs. One can show that an integral domain in which any two nonzero elements have a greatest common divisor is a non-Noetherian analogue of the UFDs. These rings are called *greatest common divisor* (GCD) *domains* and complete the corresponding chain of inclusions in constructive mathematics.

Euclidean domains $\subset$ Bézout domains $\subset$ GCD domains $\subset$ Integral domains

The inclusion Bézout domains $\subset$ GCD domains is easy to see. But note that not all GCD domains are coherent.

One reason to look at GCD domains is that they provide a good setting in which to implement the *field of fractions* for an integral domain. As the name indicates it is a field in which the integral domain can be embedded.

Definition 4.9. The *field of fractions* of an integral domain R is the set of equivalence classes of pairs (a, s) where $a, s \in R$ and $s \neq 0$ under the equivalence relation:

$$(a, s) \equiv (b, t) \quad \text{iff} \quad at = bs$$

An integral domain can be embedded by the map $a \mapsto (a, 1)$. Addition and multiplication can be defined as

$$\begin{aligned} (a, s) + (b, t) &= (at + bs, st) \\ (a, s)(b, t) &= (ab, st) \end{aligned}$$

The inverse of (a, s) where $a, s \neq 0$ is (s, a). It is easy to verify that this satisfies the conditions for a field.

The equivalence classes can be viewed as fractions. Thus is the construction of fields of fractions a generalization of the construction of $\mathbb{Q}$ from $\mathbb{Z}$ to arbitrary integral domains. Another special construction is the field of fractions for $k[x]$ (k discrete field) which is called the *field of rational functions* and is denoted by $k(x)$. This is the field where the elements are fractions of polynomials in one variable which will be important in section 5.5.2 when looking at an example of a Prüfer domains that is not a Bézout domains. This can be generalized to multivariate polynomials and one then get the field of rational functions of a polynomial ring $k[x_1, \ldots, x_n]$ denoted by $k(x_1, \ldots, x_n)$.

The reason that GCD domains are a good setting for constructing the field of fractions is that it is possible to restrict the equivalence classes to those in which $gcd(a, b) = 1$. So all fractions will be in normal form. This is a generalization of the fact that $\frac{4}{2}$ can be simplified to $\frac{1}{2}$ in $\mathbb{Q}$ to arbitrary fields of fractions.

Note on implementation. To implement GCD domains just follow the usual method without forgetting the witnesses. Given nonzero $a, b \in R$ we should compute $gcd(a, b)$, x and y such that

$$\begin{aligned} a &= gcd(a, b)x \\ b &= gcd(a, b)y \\ 1 &= gcd(x, y) \end{aligned}$$

This makes it very easy to represent the field of fractions over a GCD domain. It can be represented by a pair where the second element always should be nonzero.

The reduction to normal form of a pair (a, b) works by computing $gcd(a, b)$ and if it is 1 everything is fine. If it is not equal to 1 then return (x, y).

Using this it is trivial to implement $\mathbb{Q}$ with the help of a suitable implementation of $\mathbb{Z}$ and if one also have an implementation of $k[x]$ it is trivial to implement $k(x)$. This method could be used as a basis for implementing the rational numbers in type theory. For instance this is how it is implemented in the C-CORN library mentioned in the section on previous work.

In this section we have seen that Bézout domains are coherent and given some examples of them. But the assumption that all finitely generated ideals are principal is quite strong and there are many examples of coherent rings in which this assumption does not hold. The next section will look at a superset of Bézout domains that also are coherent. These rings are called Prüfer domains and there will be some examples of Prüfer domains that are not Bézout domains for which it is not clear that it is possible to solve systems of equations over.

Chapter 5

Prüfer domains

This chapter describes another class of coherent rings called Prüfer domains. First one of the many characterizations of Prüfer domains will be presented followed by some constructions leading up to the coherence proof. Next there will be some examples of what can be done in terms of ideal arithmetic in Prüfer domains and finally there will be some examples of Prüfer domains that are not Bézout domains. This whole section follow [8, 18].

5.1 Definition

The classical definition of Prüfer domains is that they are non-Noetherian generalization of Dedekind domains. Just as Bézout domains inherit many properties from principal ideal domains Prüfer domains inherit many properties from Dedekind domains. A first observation is that Prüfer domains allow many different classifications concerning aspects like: localization, structural and arithmetical properties of ideals and polynomial rings. The classification that will be considered in this thesis is a simple first-order condition.

Definition 5.1. An integral domain R is a *Prüfer domain* iff

$$\forall x\, y.\, \exists u\, v\, w. \quad ux = vy \wedge (1-u)y = wx \quad (*)$$

Note on implementation. As for the other algebraic structures this can be represented in the Haskell type class system as

```
class IntegralDomain a ⇒ PruferDomain a where
  calcUVW :: a → a → (a,a,a)

propCalcUVW :: (PruferDomain a, Eq a) ⇒ a → a → Bool
propCalcUVW x y = u <*> x ≡ v <*> y && (one <-> u) <*> y ≡ w <*> x
  where (u,v,w) = calcUVW x y
```

In type theory this would be possible to represent as a dependent record with the properties as part of the structure.

A commutative ring satisfying $(*)$ is called *arithmetical*. In fact many properties can be proved at the level of arithmetical rings (discussed further in [8, 18]).

As mentioned above are Bézout domains a subset of Prüfer domains. The following proposition will give a way of finding examples Prüfer domains.

Proposition 5.2. *Bézout domains are Prüfer domains.*

Proof. Since we have Bézout domain we can compute g, a and b such that

$$\begin{aligned} g &= gcd(x, y) \\ x &= ag \\ y &= bg \end{aligned}$$

We can also compute c and d such that

$$ca + db = 1$$

Now let $u = db$. Then we shall find v such that

$$dbx = vy$$

Which simplifies to

$$dbag = vbg$$

Take

$$v = ad$$

Now we want to compute w such that

$$\begin{aligned} wx &= (1 - u)y \\ &= (1 - db)y \\ &= cay \\ &= cagb \end{aligned}$$

Since $x = ag$ we get that

$$w = bc$$

Now we have found u, v and w satisfying the Prüfer condition and thus the proof is complete. □

Now that we have found a source of examples of Prüfer domains we will continue to look at some useful constructions possible in Prüfer domains which will lead to the coherence proof.

5.2 Principal localization matrices

A key concept in the proof of coherence for Prüfer domains are principal localization matrices.

Definition 5.3. A *principal localization matrix* for a finitely generated ideal $\langle x_1, \ldots, x_n \rangle$ is a matrix $A = (a_{ij})$ such that

$$\begin{cases} \sum a_{ii} & = 1 \\ a_{lj} x_i & = a_{li} x_j \quad \forall i, j, l \in \{1, \ldots, n\} \end{cases}$$

Before considering how to compute the principal localization matrix in a Prüfer domain first consider how to do it for the simpler case of Bézout domains.

Proposition 5.4. *Let R be a Bézout domain and let $I = \langle x_1, \ldots, x_n \rangle$ be an ideal in R. Then I has a principal localization matrix.*

Proof. Since R is a Bézout domain we can compute g, u_i and v_i such that

$$\begin{aligned} \langle g \rangle &= \langle x_1, \ldots, x_n \rangle \\ g &= \sum_{i=1}^{n} x_i u_i \\ x_i &= g v_i \qquad \forall i \in \{1, \ldots, n\} \end{aligned}$$

Let $a_{ij} = u_i v_j$, this give for all $i, j, l \in \{1, \ldots, n\}$

$$a_{lj} x_i = u_l v_j g v_i = u_l v_i g v_j = a_{li} x_j$$

We also have

$$g = \sum_{i=1}^{n} x_i u_i = \sum_{i=1}^{n} g v_i u_i = g \sum_{i=1}^{n} a_{ii}$$

So $1 = \sum a_{ii}$ and thus (a_{ij}) is principal localization matrix for I. □

The next step is to generalize this to Prüfer domains and thus show that principal localization matrices for ideals are computable in Prüfer domains.

Proposition 5.5. *Let R be a Prüfer domain and let $I = \langle x_1, \ldots, x_n \rangle$ be a finitely generated ideal of R. Then I has a principal localization matrix.*

Proof. First an alternative equivalent condition on Prüfer domains

$$\forall x\, y.\, \exists u\, v\, w\, t. \quad u + t = 1 \wedge ux = vy \wedge wx = ty$$

Now the proof proceeds by induction on n. For the case of $n = 2$ let the matrix be

$$\begin{bmatrix} u & v \\ w & t \end{bmatrix}$$

This obviously satisfies the requirements for being a principal localization matrix. For $n > 2$ assume that it holds for $n - 1$ and thus there is a principal localization matrix $B = (b_{ij})_{1 \leq i,j \leq n-1}$ such that

$$\begin{cases} \sum_{i=1}^{n-1} b_{ii} & = 1 \\ b_{lj} x_i & = b_{li} x_j \end{cases}$$

It is possible for (x_i, x_n) where $i \in \{1, \ldots, n-1\}$ to compute (u_i, v_i, w_i, t_i) such that

$$\begin{cases} u_i x_n & = v_i x_i \\ w_i x_n & = t_i x_i \\ u_i + t_i & = 1 \end{cases}$$

Using this we will proceed by showing how a_{ii} and a_{nn}, a_{ij} where $i \neq j \in \{1, \ldots, n-1\}$ and finally a_{ni} and a_{in} can be computed using B.

- a_{ii} and a_{nn}: We have

$$1 = \sum_{i=1}^{n-1} b_{ii} = \sum_{i=1}^{n-1} b_{ii}(u_i + t_i) = \sum_{i=1}^{n-1} b_{ii}u_i + \sum_{i=1}^{n-1} b_{ii}t_i$$

  Now we can let $a_{ii} = b_{ii}u_i$ and $a_{nn} = \sum_{i=1}^{n-1} b_{ii}t_i$ and get that $\sum_{i=1}^{n} a_{ii} = 1$.

- a_{ij} where $i \neq j \in \{1, \ldots, n-1\}$: We have

$$a_{ii}x_j = b_{ii}u_i x_j = u_i(b_{ii}x_j) = u_i b_{ij} x_i$$

  This give $a_{ij} = u_i b_{ij}$.

- a_{ni}: For $i \in \{1, \ldots, n-1\}$ we have

$$a_{nn}x_i = \sum_{j=1}^{n-1} b_{jj}t_j x_i = \sum_{j=1}^{n-1} t_j(b_{jj}x_i) = \sum_{j=1}^{n-1} t_j b_{ji} x_j = \sum_{j=1}^{n-1} w_j b_{ji} x_n$$

  Now let $a_{ni} = \sum_{j=1}^{n-1} b_{ji}w_j$.

- a_{in}: Finally for $i \in \{1, \ldots, n-1\}$ we have

$$a_{ii}x_n = b_{ii}u_i x_n = b_{ii}v_i x_i$$

  Let $a_{in} = b_{ii}v_i$.

This gives a matrix (a_{ij}) satisfying $a_{ij}x_i = a_{ii}x_j$. It is necessary to verify that this indeed satisfies the properties of a principal localization matrix, that is $a_{kj}x_i = a_{ki}x_j$. Assume that it is true for $n-1$ and let i, j, k be distinct. There are then three cases to verify:

1. For $i, j, k \in \{1, \ldots, n-1\}$ we have

$$a_{kj}x_i = (u_k b_{kj})x_i = u_k(b_{ki}x_j) = (u_k b_{ki})x_j = a_{ki}x_j$$

2. When $i = n$ and $j, k \in \{1, \ldots, n-1\}$

$$a_{kj}x_n = (u_k b_{kj})x_n = (v_k x_k)b_{kj} = v_k(b_{kk}x_j) = a_{kn}x_j$$

3. Finally $k = n$ and $i, j \in \{1, \ldots, n-1\}$

$$a_{nj}x_i = \left(\sum_{l=1}^{n-1} b_{lj}w_l\right) x_i = \sum_{l=1}^{n-1} b_{li}x_j w_l = \left(\sum_{l=1}^{n-1} b_{li}w_l\right) x_j = a_{ni}x_j$$

1-3 verifies that (a_{ij}) in fact is a principal localization matrix.

□

One of the reasons to study principal localization matrices is that they can be used to invert ideals in Prüfer domains, more about this in the next section.

The following proposition describe one of the many things that can be done with ideals and principal localization matrices, see proposition 2.6 in [8] for more results.

Proposition 5.6. *Let $I = \langle x_1, \ldots, x_n \rangle$ be a finitely generated ideal in a Prüfer domain and let $A = (a_{ij})$ be a principal localization matrix for this ideal. Then the following holds:*

$$\langle x_1, \ldots, x_n \rangle \langle a_{1j}, \ldots, a_{nj} \rangle = \langle x_j \rangle$$

Proof. Recall that the result of multiplying two finitely generated ideals is the ideal generated by all products of the generators. Thus

$$\langle x_1, \ldots, x_n \rangle \langle a_{1j}, \ldots, a_{nj} \rangle = \langle x_1 a_{1j}, \ldots, x_1 a_{nj}, \ldots, x_n a_{1j}, \ldots x_n a_{nj} \rangle$$

Since (a_{ij}) is a principal localization matrix we have $a_{lj} x_i = a_{li} x_j$. So all $a_{lj} x_i$ can be rewritten to $a_{li} x_j$. This give

$$\begin{aligned} \langle x_1 a_{1j}, \ldots, x_1 a_{nj}, \ldots, x_n a_{1j}, \ldots x_n a_{nj} \rangle &= \langle x_j \rangle \langle a_{ij} \rangle \\ &= \langle x_j \rangle \end{aligned}$$

The last equality follow by $\langle a_{ij} \rangle = \langle 1 \rangle$ since $\sum_{i=1}^{n} a_{ii} = 1$. □

This result is one of the things that is needed in order to prove that Prüfer domains are coherent which is the aim of the next section.

5.3 Invertible ideals and coherence of Prüfer domains

The previous section mentioned that it is possible to invert ideals in Prüfer domains. The following definition clarifies this.

Definition 5.7. An ideal I in a ring R is called *invertible* if there exist an ideal I^{-1} in R such that II^{-1} is principal.

So how can invertible ideals be used to prove that Prüfer domains are coherent?

Lemma 5.8. *Let R be a Prüfer domain and let I and J be finitely generated ideals of R. Then the following equality hold:*

$$IJ = (I \cap J)(I + J)$$

For a proof of this see lemma 2.10 in [8].

Lemma 5.9. *Let R be a Prüfer domain. It is possible to compute the inverse of a finitely generated ideal I in R.*

Proof. This follow directly from propositions 5.5 and 5.6. Since it possible to compute a principal localization matrix for the ideal and then multiply one of the columns with the ideal and get a principal ideal. □

Now we are ready to prove the main result for Prüfer domains.

Theorem 5.10. *Prüfer domains are coherent.*

Proof. Combine lemmas 5.8 and 5.9 to get

$$IJ(I+J)^{-1} = \langle a \rangle (I \cap J)$$

Where $\langle a \rangle = (I+J)(I+J)^{-1}$. So the intersection of ideals is computable and now proposition 3.4 is applicable in order to get the desired result. □

Note on implementation. The implementation of this require some more work. The reason is that one need to "factor out" $\langle a \rangle$ from $IJ(I+J)^{-1}$. To do this algorithmically consider the following finitely generated ideal

$$\begin{aligned} I &= \langle x_1, \ldots, x_n \rangle \\ J &= \langle y_1, \ldots, y_m \rangle \end{aligned}$$

This give

$$I + J = \langle x_1, \ldots, x_n, y_1, \ldots, y_m \rangle = \langle z_1, \ldots, z_k \rangle$$

with $k = n + m$. Now compute a principal localization matrix for $I + J$ and pick the first column which give

$$(I+J)^{-1} = \langle a_{11}, \ldots, a_{k1} \rangle$$

such that

$$(I+J)(I+J)^{-1} = \langle x_1 \rangle = \langle z_1 \rangle$$

This give

$$IJ(I+J)^{-1} = \langle z_1 \rangle (I \cap J)$$

In terms of generators $J(I+J)^{-1}$ are

$$\begin{aligned} \langle y_1, \ldots, y_m \rangle \langle a_{11}, \ldots, a_{k1} \rangle &= \langle z_{n+1}, \ldots, z_k \rangle \langle a_{11}, \ldots, a_{k1} \rangle \\ &= \langle z_{n+1}a_{11}, \ldots, z_{n+1}a_{k1}, \ldots, z_k a_{11}, \ldots, z_k a_{k1} \rangle \\ &= \left\langle z_1 a_{1(n+1)}, \ldots, z_1 a_{k(n+1)}, \ldots, z_1 a_{1k}, \ldots, z_1 a_{kk} \right\rangle \\ &= \langle z_1 \rangle \left\langle a_{1(n+1)}, \ldots, a_{k(n+1)}, \ldots, a_{1k}, \ldots, a_{kk} \right\rangle \end{aligned}$$

Let

$$K = \left\langle a_{1(n+1)}, \ldots, a_{k(n+1)}, \ldots, a_{1k}, \ldots, a_{kk} \right\rangle$$

This gives

$$IK = I \cap J$$

Using this an algorithm can be implemented to compute the generators of the intersection.

Thus it is possible to solve homogeneous linear systems over any Prüfer domain. Note that for Bézout domains this gives another way of solving systems. Next it would be interesting to see some examples of Prüfer domains that are not Bézout domains, but first we will look at what Prüfer domains give us in terms of ideal arithmetic.

5.4 Ideal arithmetic

Recall the definition of a lattice. A lattice is a partially ordered set where every pair of elements has a lowest upper bound (join) and a greatest lower bound (meet). For more information on this see any basic book on algebra, for example [9]. For finitely generated ideals we have that the join of two ideals correspond to the sum of them and the meet is the intersection, so the set of ideals in a Prüfer domain form a lattice with $\subseteq$.

For ideals I, J, K in a Prüfer domain the following property hold

$$I \cap (J + K) = (I \cap J) + (I \cap K)$$

So $\cap$ distributes over $+$ and thus the set of ideals in a Prüfer domain form a distributive lattice. In fact this is one of the different ways to classify Prüfer domains that were mentioned in the beginning of this chapter. For a proof that this property is sufficient to be a Prüfer domain see theorem 1.1 in [10].

Another interesting property of the ideals in Prüfer domains is that they satisfy the cancellation property. That is given finitely generated ideals I, J, K, if

$$IJ = IK$$

then $J = K$ or $I = 0$.

5.5 Examples of Prüfer domains

The proposition to be proved will give a way to find Prüfer domains that are not Bézout domains. First some definitions are necessary.

Definition 5.11. Let B be a ring and A a subring of B. An element $x \in B$ is called *integral* over A if it satisfies an equation of the form

$$x^n + a_1 x^{n-1} + \cdots + a_1 x + a_n = 0$$

where $a_i \in A$.

The elements of B that are integral over A form a subring of B, for a proof of this see corollary 5.3 in [1]. This ring is called the *integral closure* of A in B.

This is what is necessary in order to formulate the following important result.

Proposition 5.12. *Let R be a Bézout domain and L an algebraic extension of its field of fractions K. The integral closure S of R inside L is a Prüfer domain.*

Proof. Let $x, y \in R$ be nonzero and integral over R. The element

$$s \quad = \quad \frac{x}{y}$$

is integral over K and hence satisfies an equation on the form

$$a_0 s^n + a_1 s^{n-1} + \cdots + a_n = 0 \qquad (*)$$

with $a_0, \ldots, a_n \in R$. Since R is a Bézout domain it is safe to assume that $\langle a_0, \ldots, a_n \rangle = 1$. The first step is to show that the following elements are in S, i.e. show that they are integral over R. First $a_0 s \in S$ since

$$(a_0 s)^n + a_0 a_1 (a_0 s)^{n-1} + \cdots + a_0^n a_n = 0$$

So $a_o s + a_1 \in S$ also. Now $(*)$ can be rewritten as

$$(a_o s + a_1)s^{n-1} + \cdots + a_n = 0$$

It follows that $(a_0 s + a_1)s \in S$. This process can be repeated and finally we get that

$$a_0, a_0 s, a_0 s + a_1, (a_0 s + a_1)s, (a_0 s + a_1)s + a_2, \ldots$$

are all in S. Thus we have in S

$$\langle a_0, a_0 s, a_0 s + a_1, (a_0 s + a_1)s, (a_0 s + a_1)s + a_2, \ldots \rangle = \langle a_0, \ldots, a_n \rangle = 1$$

This means that there exists $m_0, m_1, \ldots$ such that

$$m_0 a_0 + m_1 a_0 s + m_2(a_0 s + a_1) + m_3(a_0 s + a_1)s + \cdots = 1$$

The final step in the proof is to find u, v, w such that

$$\begin{cases} ux & = vy \\ (1-u)y & = wx \end{cases}$$

Let

$$u = m_0 a_0 + m_2(a_0 s + a_1) + \ldots$$

This give

$$(m_0 a_0 + m_2(a_0 s + a_1) + \ldots)x = vy$$

So we can let $v = us$. Finally we have

$$\begin{aligned} 1 - u &= m_1 a_0 s + m_3(a_0 s + a_1)s + \ldots \\ &= s(m_1 a_0 + m_3(a_0 s + a_1) + \ldots) \end{aligned}$$

This give $(1-u)y = x(m_1 a_0 + m_3(a_0 s + a_1) + \ldots)$ and thus

$$w = m_1 a_0 + m_3(a_0 s + a_1) + \ldots$$

□

Both the proposition and the proof are quite abstract. To make this more concrete we will consider two examples of rings which are not Bézout domains but can be shown to be Prüfer domains using the method just described. The first example is an extension of the integers and the second is an algebraic curve.

5.5.1 $\mathbb{Z}[\sqrt{-5}]$

This is the set where all elements are of the type $a + b\sqrt{-5}$ where $a, b \in \mathbb{Z}$. This ring is an example of an quadratic integer ring. These has been studied by number theorists for a long time, since quadratic integers are related to many important problems in number theory, for example Fermat's last theorem.

Addition and multiplication can be defined as

$$\begin{aligned} (a + b\sqrt{-5}) + (c + d\sqrt{-5}) &= (a + c) + (b + d)\sqrt{-5} \\ (a + b\sqrt{-5})(c + d\sqrt{-5}) &= (ac - 5bd) + (ad + bc)\sqrt{-5} \end{aligned}$$

It is easy to verify that this forms a ring. To see that it is not a Bézout domain we need to find a finitely generated ideal that is not principal.

Proposition 5.13. *The ideal generated by* $\langle 2, 1+\sqrt{-5}\rangle$ *in* $\mathbb{Z}[\sqrt{-5}]$ *is not principal.*

Proof. Assume that it is generated by a single element, that is $\langle x\rangle = \langle 2, 1+\sqrt{-5}\rangle$. Let $x = a+b\sqrt{-5}$ and $y = a-b\sqrt{-5}$ such that xy divide both 4 and $(1+\sqrt{-5})(1-\sqrt{-5}) = 1-(\sqrt{-5})^2 = 6$. Thus $xy|2$, but $xy = a^2+5b^2 = 2$ is impossible. This means that $xy = 1$ and thus $x = 1$ or $x = -1$.

But $\langle 1\rangle \neq \langle 2, 1+\sqrt{-5}\rangle$ which means that $\langle 2, 1+\sqrt{-5}\rangle$ cannot be generated by a single element. □

The proof that $\mathbb{Z}[\sqrt{-5}]$ is a Prüfer domain follow proposition 5.12. The reason that the proof is so verbose is that this hopefully will clarify the proof of proposition 5.12.

Proposition 5.14. $\mathbb{Z}[\sqrt{-5}]$ *is a Prüfer domain.*

Proof. Given x and y the goal is to compute u, v, w such that

$$\begin{cases} ux & = vy \\ (1-u)y & = wx \end{cases}$$

Let $a, b, c, d \in \mathbb{Z}^*$. Start with

$$\begin{aligned} s &= \frac{a+b\sqrt{-5}}{c+d\sqrt{-5}} \\ &= \frac{(a+b\sqrt{-5})(c-d\sqrt{-5})}{c^2+5d^2} \end{aligned}$$

Now s can be written on the form $s = p+q\sqrt{-5}$ where $p, q \in \mathbb{Q}$ and

$$\begin{aligned} p &= \frac{ac+5bd}{c^2+5d^2} \\ q &= \frac{bc-ad}{c^2+5d^2} \end{aligned}$$

We have

$$\begin{aligned} s = p+q\sqrt{-5} &\Rightarrow s-p = q\sqrt{-5} \\ &\Rightarrow (s-p)^2 = -5q^2 \\ &\Rightarrow s^2-2sp+p^2+5q^2 = 0 \end{aligned}$$

If we write this out with p and q we get

$$s^2 - \frac{2(ac+5bd)}{c^2+5d^2}s + \frac{a^2+5b^2}{c^2+5d^2} = 0$$

Rewrite to $a_0s^2+a_1s+a_2 = 0$ where $a_0, a_1, a_2 \in \mathbb{Z}$. We can assume that $\langle a_0, a_1, a_2\rangle = \langle 1\rangle$ since we can always divide with $ggcd(a_0, a_1, a_2)$ to get this. By the same reasoning as in the proof of proposition 5.12 we get that

$$\langle a_0, a_0s, a_0s+a_1, (a_0s+a_1)s\rangle = \langle 1\rangle$$

So we can find m_0, m_1, m_2, m_3 such that

$$m_0a_0 + m_1a_0s + m_2(a_0s + a_1) + m_3(a_0s + a_1)s = 1$$

This give

$$\begin{cases} u &= m_0a_0 + m_2(a_0s + a_1) \\ v &= m_0a_0s + m_2(a_0s + a_1)s \\ w &= m_1a_0 + m_3(a_0s + a_1) \end{cases}$$

□

Note on implementation. $\mathbb{Z}[\sqrt{-5}]$ can be represented in Haskell as a pair where the first element correspond to a and the second to b in $a + b\sqrt{-5}$. The proof that the ring is a Prüfer domain follow proposition 5.14 and is straight forward to implement now that most of the details has been worked out.

One important thing to keep in mind is that s lies in the field of fractions and thus the type of s is different from the type of, for instance, a_0. So to perform the computations some conversions has to be made. First, elements has to be embedded into the field of fractions. Then, elements has to be extracted and the extraction should only be possible if the divisor is 1. But since the proof tells us that all of $a_0s, a_0s + a_1, (a_0s + a_1)s$ lie in the integral closure of $\mathbb{Z}[\sqrt{-5}]$ this is not a problem.

5.5.2 $k[x, y]$ with $y^2 = 1 - x^4$

This ring correspond to an algebraic curve where all elements are of the form $a + b\sqrt{1 - x^4}$ with $a, b \in k[x]$.

Addition and multiplication can be defined as

$$\begin{aligned} (a + b\sqrt{1 - x^4}) + (c + d\sqrt{1 - x^4}) &= (a + c) + (b + d)\sqrt{1 - x^4} \\ (a + b\sqrt{1 - x^4})(c + d\sqrt{1 - x^4}) &= (ac + bd(1 - x^4)) + (ad + bc)\sqrt{1 - x^4} \end{aligned}$$

Again the proof that this is a Prüfer domain follow proposition 5.12.

Proposition 5.15. *$k[x, y]$ with $y^2 = 1 - x^4$ is a Prüfer domain.*

Proof. Let a, b, c, d be nonzero elements of $k[x]$. Start with

$$\begin{aligned} s &= \frac{a + b\sqrt{1 - x^4}}{c + d\sqrt{1 - x^4}} \\ &= \frac{(a + b\sqrt{1 - x^4})(c - d\sqrt{1 - x^4})}{c^2 - d^2(1 - x^4)} \\ &= \frac{ac - bd(1 - x^4) + (bc - ad)\sqrt{1 - x^4}}{c^2 - d^2(1 - x^4)} \end{aligned}$$

Now s can be written on the form $s = p + q\sqrt{1 - x^4}$ where $p, q \in k(x)$.

$$s^2 - 2\frac{(ac - bd(1 - x^4))}{c^2 - d^2(1 - x^4)}s + \frac{(ac - bd(1 - x^4))^2 - (bc - ad)^2(1 - x^4)}{(c^2 - d^2(1 - x^4))^2} = 0$$

This can be written in the form $a_0 s^2 + a_1 s + a_2 = 0$ where $a_0, a_1, a_2 \in k(x)$ and $\langle a_0, a_1, a_2 \rangle = \langle 1 \rangle$. By exactly the same reasoning as above we get

$$\begin{cases} u & = m_0 a_0 + m_2(a_0 s + a_1) \\ v & = m_0 a_0 s + m_2(a_0 s + a_1)s \\ w & = m_1 a_0 + m_3(a_0 s + a_1) \end{cases}$$

□

Note on implementation. Again this ring can be represented in Haskell as a pair where the first element correspond to a and the second to b in $a + b\sqrt{1 - x^4}$. The proof that the ring is a Prüfer domain is straight forward to implement now that most details has been worked out. Note that the field of fractions here will be the field of rational functions for $k[x]$.

This section has given two examples of rings that are not Bézout domains but Prüfer domains. Hence they are coherent and it is possible to solve systems of equations over them.

Another method for constructing Prüfer domains is considered in [6]. The method shows that if k is a discrete field and f is a polynomial in $k[x, y]$. Then the localization $R_{f'_y}$ where $R = k[x, y]/\langle f \rangle$ and f'_y is the partial derivative with respect to y is a Prüfer domain. Localization is a generalization of the construction of fields of fractions from section 4.5. This method is also relevant for proving that further algebraic curves are Prüfer domains.

Yet another method for proving that integral domains are Prüfer domains is the Gilmer-Hoffmann theorem which states that if R is an integral domain, K its field of fractions and S the integral closure of R in K. Then S is a Prüfer domain iff every element of K is a root of a polynomial in $R[x]$ where at least one of the coefficients is a unit of R. For a constructive formulation and proof of this see proposition 5.6 in [3].

5.6 Prüfer domains and strong discreteness

As for Bézout domains we have that Prüfer domains are strongly discrete if division is decidable, but not all Prüfer domains have decidable division. The reliance on divisibility is motivated by the following proposition.

Proposition 5.16. *A Prüfer domain R is strongly discrete if division is decidable in R.*

Proof. To decide if $x \in I$ compute I^{-1} such that $II^{-1} = \langle a \rangle$. We want to test if $\langle x \rangle \subseteq I$ which means that $\langle x \rangle I^{-1} \subseteq \langle a \rangle$. This can be decided if we can decide if an element is divisible by a. □

This give another proof that all Bézout domains are strongly discrete. Since all Bézout domains are Prüfer domains and have decidable division.

The next chapter will study yet another class of rings which has many different applications, these are polynomial rings. There have been a small taste of special polynomial rings in this chapter and the next chapter will consider them in general. Traditionally Dedekind domains has been of great interest in number theory while polynomial rings has been of interest in algebraic geometry.

Chapter 6

Polynomial rings

Let k be a discrete field. The set of all multivariate polynomials in n variables with coefficient in k, written $k[x_1, \ldots, x_n]$, form a commutative ring. The main aim of this section is to study this ring and eventually prove that it is coherent.

This text is mainly based on [7] and many proofs has been omitted and can be found there. But some of the most important proofs in it relies on a nonconstructive version of Dicksons lemma, so these proofs contains a gap from a constructive point of view. This is considered in greater detail in [15] and the main results of it is presented here in order to fill in the gaps.

6.1 Monomials and monomial orderings

In order to develop the theory of multivariate polynomials the theory of monomials and how to order them has to be considered. For example, what is the greatest of x^3y and x^2y^3 in $k[x, y]$? Later in the chapter we will see that many algorithms on polynomials will rely on the monomial ordering.

Definition 6.1. A *monomial* in $x_1, \ldots, x_n$ is a product of the form $x_1^{\alpha_1} \cdot \ldots \cdot x_n^{\alpha_n}$ where $\alpha_1, \ldots, \alpha_n \in \mathbb{N}$.

This can be simplified by just looking at $\alpha = (\alpha_1, \ldots, \alpha_n)$ and letting $x^\alpha = x_1^{\alpha_1} \cdot \ldots \cdot x_n^{\alpha_n}$. The total degree of a monomial α is $|\alpha| = \alpha_1 + \ldots + \alpha_n$.

Definition 6.2. A *monomial ordering* is any total relation $>$ on $\mathbb{N}^n$ satisfying

1. If $\alpha > \beta$ and $\gamma \in \mathbb{N}^n$ then $\alpha + \gamma > \beta + \gamma$.
2. $>$ is a well-ordering on $\mathbb{N}^n$.

We can now define some different monomial orderings.

Definition 6.3. *Lexicographical order*: $\alpha >_{lex} \beta$ if the left-most nonzero entry is positive in $\alpha - \beta \in \mathbb{Z}^n$.

This is a very simple ordering and it works exactly as lexicographical ordering of words. This ordering can seem a bit weird. Consider for example x and y^3z^5 with $x > y > z$, then $x >_{lex} y^3z^5$. The following ordering remedies this.

Definition 6.4. *Graded lexicographical order*: $\alpha >_{grlex} \beta$ if

$$|\alpha| > |\beta|\,, \quad \text{or} \quad |\alpha| = |\beta| \text{ and } \alpha >_{lex} \beta$$

Here the total degree of the monomials first decide which is the greatest and lexicographical ordering is used to break ties. The following ordering is a bit less intuitive, but can be more efficient when doing computations.

Definition 6.5. *Graded reverse lexicographical order*: $\alpha >_{grevlex} \beta$ if

$$|\alpha| > |\beta|\,, \quad \text{or} \quad |\alpha| = |\beta|$$

and the right-most nonzero entry in $\alpha - \beta \in \mathbb{Z}^n$ is negative.

The last two orderings are best illustrated by an example. Consider x^2z^2 and xy^2z. Since they have same total degree lexicographical ordering will have to break the tie for graded lexicographical ordering, thus $x^2z^2 >_{grlex} xy^2z$. But using graded reverse lexicographical order it will be the other way around, so $xy^2z >_{grevlex} x^2z^2$.

Note on implementation. One way to represent monomials in Haskell is as a list of integers denoting $\alpha = (\alpha_1, \ldots, \alpha_n)$. The ordering can then be represented as a phantom type in the type of monomials. The type class system can then be used to make the monomial type indexed by a certain ordering an instance of the Ord class. So for each ordering we will have a type of monomials with a special Ord instance. This is an example where the Haskell type system captures the mathematical definitions very well.

6.2 Polynomial rings

Using monomials we can now define what a polynomial is.

Definition 6.6. A *polynomial* $f \in k[x_1, \ldots, x_n]$ is a linear combination of monomials, that is

$$f = \sum_{\alpha} a_\alpha x^\alpha, \;\; a_\alpha \in k$$

For example $f = 3x^3y^2 + z^4 + \frac{1}{2}xz$ is a polynomial in $\mathbb{Q}[x, y, z]$. The total degree of a polynomial is the monomial with maximum total degree with nonzero coefficient, for example $deg(f) = 5$. It is possible to prove that addition and multiplication of polynomials can be defined in a way which satisfy the properties of commutative rings. Next some terminology.

Definition 6.7. Let $f = \sum_\alpha a_\alpha x^\alpha$ be a polynomial in $k[x_1, \ldots, x_n]$ and let $>$ be a monomial order. Then

1. The *multidegree* of f (with respect to $>$) is

$$multideg(f) = max(\alpha \in \mathbb{N}^n : a_\alpha \neq 0)$$

2. The *leading coefficient* of f is

$$LC(f) = a_{multideg(f)}$$

3. The *leading monomial* of f is

$$LM(f) = x^{multideg(f)}$$

4. The *leading term* of f is

$$LT(f) = LC(f) \cdot LM(f)$$

Let $f = 3x^3y^2 + z^4 + \frac{1}{2}xz$ be a polynomial in $\mathbb{Q}[x, y, z]$ ordered with graded lexicographical order. Then

$$\begin{aligned} multideg(f) &= (3, 2, 0) \\ LC(f) &= 3 \\ LM(f) &= x^3y^2 \\ LT(f) &= 3x^3y^2 \end{aligned}$$

As mentioned above it is possible to define have addition, subtraction and multiplication with expected behavior, but what about division? The case for multivariate polynomials is quite different from the case of univariate polynomials for which the standard division algorithm is applicable. The goal in $k[x_1, \ldots, x_n]$ is to divide a polynomial f by a list of polynomials $f_1, \ldots, f_s$ such that

$$f = q_1f_1 + \cdots + q_nf_n + r$$

with $q_1, \ldots, q_n, r \in k[x_1, \ldots, x_n]$.

Theorem 6.8. *Every $f \in k[x_1, \ldots, x_n]$ can be written as*

$$f = q_1f_1 + \cdots + q_nf_n + r$$

with $q_i, r \in k[x_1, \ldots, x_n]$ and either $r = 0$ or r is a linear combination of monomials, none of which is divisible by $LT(f_1), \ldots, LT(f_n)$. We call r the remainder on division by $f_1, \ldots, f_n$. Furthermore if $q_if_i \neq 0$ then $multideg(f) \geq multideg(q_if_i)$.

There is a standard algorithm which is given in [7, 15]. Termination is proved constructively in [15]. One important property of this is that uniqueness is not necessary. For example, division of x^2 by $[x + y, x]$ in $k[x, y]$ give

$$x^2 = (x - y)(x + y) + 0 \cdot x + y^2$$

but division of x^2 by $[x, x + y]$ give

$$x^2 = x \cdot x + 0 \cdot (x + y) + 0$$

So the order of the divisors matter!

Note on implementation. The only thing that is difficult to represent in Haskell is the order and naming of the variables. One way would be to represent this at the type level using phantom types, but this is a bit cumbersome. It would be easier to represent with dependent types since this requires that the language has types depending on values.

6.3 Properties of ideals in $k[x_1, \ldots, x_n]$

In section 4.2 it is proved that all ideals in Euclidean domains are principal and this especially hold for $k[x]$. But this does not hold for $k[x_1, \ldots, x_n]$ if $n \geq 2$.

Consider for example the ideal generated by $\langle x, y \rangle$ in $k[x, y]$. If it were to be generated by a single element then this element would have to divide both x and y. But then it must be a nonzero constant and the ideal contain no other constant than zero. Thus it is not possible for the ideal to be generated by a single element.

From here on this section deviates from [7] and is instead based on [15]. The reason for this is that [7] builds the theory leading up to the Hilbert basis theorem and the theory of Gröbner bases on a nonconstructive version of Dickson's lemma.

Definition 6.9. A poset $(E, \leq)$ is said to satisfy the *ascending chain condition* (ACC) if for every non decreasing sequence $(u_n)_{n \in \mathbb{N}}$ in E there exists $n \in \mathbb{N}$ such that $u_n = u_{n+1}$.

This definition is to be read constructively. This means that there exist an effective method for computing n. Section 3.1 in [15] contains a motivation why it is so important that ACC is read and motivated constructively and not classically.

The versions of Dickson's lemma and the Hilbert basis theorem in [7] relies on a classical interpretation of ACC and are thus not valid constructively.

Theorem 6.10. *(Constructive) Dickson's lemma: The poset finitely generated monomial ideals in $k[x_1, \ldots, x_n]$, ordered with $\subseteq$, satisfies ACC.*

This can then be used to prove the termination of the Buchberger algorithm for computing Gröbner bases. Another important application of this is in the proof of the Hilbert basis theorem.

Theorem 6.11. *(Constructive) Hilbert basis theorem: The poset of finitely generated ideals of $k[x_1, \ldots, x_n]$, ordered with $\subseteq$, satisfies ACC.*

This shows that $k[x_1, \ldots, x_n]$ satisfies the constructive version of Noetherianity. That is the set of finitely generated ideals in the ring satisfies the constructive ACC.

6.4 Gröbner bases

In this section Gröbner bases will be defined and then the Buchberger algorithm which computes a Gröbner basis for an ideal is presented.

Definition 6.12. Let $I \subset k[x_1, \ldots, x_n]$ be an ideal other than $\{0\}$. Then $LT(I)$ is the set of leading terms of elements

$$LT(I) = \{cx^\alpha : \exists f \in I \text{ with } LT(f) = cx^\alpha\}$$

and $\langle LT(I) \rangle$ is the ideal generated by the elements of $LT(I)$.

Definition 6.13. A *Gröbner basis* of an ideal $I \subseteq k[x_1, \ldots, x_n]$ is a list of generators $g_1, \ldots, g_m$ of the ideal such that for all $f \in I$ the division of f by $g_1, \ldots, g_m$ leads to a zero remainder.

Gröbner bases are the "well-behaved" ideals in $k[x_1, \ldots, x_n]$. One important property is that division no longer is dependent on the order of the divisors.

Definition 6.14. If $f, g \in k[x_1, \ldots, x_n] \backslash \{0\}$ with $multideg(f) = \alpha$ and $multideg(g) = \beta$. Let $\gamma = (\gamma_1, \ldots, \gamma_d)$ where $\gamma_i = max(\alpha_i, \beta_i)$. Now define:

- The *lowest common multiple*: $lcm(\alpha, \beta) = \gamma$
- The *S-polynomial* of f and g:

$$S[f, g] = \frac{x^\gamma}{LT(f)} f - \frac{x^\gamma}{LT(g)} g$$

Note that the S-polynomials is constructed in a way such that leading terms should be cancelled. This will be crucial to the algorithm for computing Gröbner bases. This algorithm is the famous Buchberger algorithm.

Theorem 6.15. *Let $I = \langle f_1, \ldots, f_s \rangle$ be a nonzero polynomial ideal. Then a Gröbner basis for I can be constructed in a finite number of steps by the following algorithm:*

```
Input: I = (f1,...,fs)
Output: A Gröbner basis G = (g1,...,gt) for I

G := I
REPEAT
  G' := G
  FOR each pair (p,q), p ≠ q in G' DO
    S := remainder (S_poly(p,q)) G'
    IF S ≠ 0 THEN G := G  ∪  {S}
UNTIL G = G'
```

Termination and correctness of this algorithm is proved constructively in [15]. The intuition behind the algorithm is that division with the generators of the Gröbner basis should be zero. The algorithm iteratively compute the remainder of the division of the S-polynomials with the generators of the basis. If the result of this is zero nothing need to be added, but if the result is nonzero the remainder should be added to the basis. This will force the basis to divide all elements of the ideals.

This version of the algorithm is quite naive and it generates very large bases. A simple trick to reduce the size of the Gröbner basis is given by the following lemma.

Lemma 6.16. *Let G be a Gröbner basis for the polynomial ideal I. Let $p \in G$ be a polynomial such that $LT(p) \in \langle LT(G - \{p\}) \rangle$. Then $G - \{p\}$ is also a Gröbner basis for I.*

Using this it is possible to define what a minimal Gröbner basis is.

Definition 6.17. A *minimal Gröbner basis* for a polynomial ideal I is a Gröbner basis such that

$$\forall p \in G.\ (LC(p) = 1 \wedge LT(p) \notin \langle LT(G - \{p\})\rangle)$$

By multiplying with constant terms all leading coefficients can be made 1 and by applying lemma 6.16 a minimal Gröbner basis can be constructed.

Now consider an example, we saw earlier that division of x^2 by $[x + y,\ x]$ in $k[x, y]$ give different results depending on the order, so $[x + y,\ x]$ does not form a Gröbner basis. Using the algorithm above together with the method for computing minimal Gröbner bases we get the generators $[x,\ y]$ as a Gröbner basis for the ideal. It is clear that x^2 divided by $[x,\ y]$ will not depend on the order of the divisors.

Note on implementation. In Haskell this is quite straight forward to implement. But in type theory it would require more effort since all of the proofs would have to be formalized. This has been done using a combination of Agda and Coq in [17].

Another thing worth emphasizing is that the algorithm presented is not optimized in any way, so it will be unrealistic to use it on larger examples. A lot of effort has been put into optimizing the Buchberger algorithm. One approach to optimize the algorithm is not to try all pairs of polynomials but try to find the "best" pairs, for more details see [13]. Some of the optimizations has been implemented in Haskell in the HaskellForMaths library by David Amos mentioned in the section on previous work.

6.5 Coherence of $k[x_1, \ldots, x_n]$

The proof of coherence will be proved as for Bézout domains and Prüfer domains, that is by considering the intersection of ideals. An alternative proof can be found in [15].

Definition 6.18. Let I be a finitely generated ideal in $k[x_1, \ldots, x_n]$. The *r:th elimination ideal* of I is $I_r = I \cap k[x_{r+1}, \ldots, x_n]$ where $1 \leq r \leq n$.

The computation of elimination ideals can be done by choosing the lexicographical ordering with $x_1 > \cdots > x_n$ and computing a Gröbner basis for I. Then the generators containing any of the x_l where $l < r$ are thrown away. Using this the intersection can be computed using the following trick.

Proposition 6.19. *Let I and J be ideals in $k[x_1, \ldots, x_n]$. Then*

$$I \cap J = (tI + (1 - t)J) \cap k[x_1, \ldots, x_n]$$

where t is greater than all $x_1, \ldots, x_n$.

Now coherence follow directly.

Theorem 6.20. *The polynomial ring $k[x_1, \ldots, x_n]$ is coherent.*

Proof. Direct consequence of propositions 3.4 and 6.19. □

Note on implementation. In order to be able to test that the generated ideal is in fact the intersection witnesses has to be computed.

In order to modify the algorithm to compute these some bookkeeping needs to be done. At each time an element is added to the basis the algorithm would have to store the witness that this can be written using the elements of the initial ideal.

6.6 Strong discreteness of $k[x_1, \ldots, x_n]$

This requires a version of the Buchberger algorithm that computes witnesses that the basis generate the same ideal as the first ideal. Which means that the generators for the first ideal should be able to be expressed as a linear combination of the Gröbner basis and vice-versa.

Theorem 6.21. *$k[x_1, \ldots, x_n]$ is strongly discrete, that is we can test $f \in \langle h_1, \ldots, h_m \rangle$ and find witnesses $w_1, \ldots, w_m$ such that*

$$f = w_1 h_1 + \cdots + w_m h_m$$

Proof. We want to decide if $f \in \langle h_1, \ldots, h_m \rangle$. To do this first compute a Gröbner basis $\langle g_1, \ldots, g_l \rangle$ for $\langle h_1, \ldots, h_m \rangle$. Then compute $f/[g_1, \ldots, g_l] = (qs, r)$, if $r \neq 0$ then we know that $f \notin \langle h_1, \ldots, h_m \rangle$. But if $r = 0$ we want to compute a witness that this is in fact true.

What we have is $q_1, \ldots, q_l$ such that

$$f = q_1 g_1 + \cdots + q_l g_l$$

We also have a witness that $\langle h_1, \ldots, h_m \rangle \supseteq \langle g_1, \ldots, g_l \rangle$ which consists of $a_{11}, \ldots, a_{1m}, \ldots, a_{l1}, \ldots, a_{lm}$ such that

$$\begin{aligned} g_1 &= a_{11} h_1 + \cdots + a_{1m} h_m \\ &\vdots \\ g_m &= a_{l1} h_1 + \cdots + a_{lm} h_m \end{aligned}$$

To compute the witness insert this in the above expression for f

$$\begin{aligned} f &= q_1(a_{11} h_1 + \cdots + a_{1m} h_m) + \\ &\quad \vdots \\ &\quad q_l(a_{l1} h_1 + \cdots + a_{lm} h_m) \\ &= h_1(q_1 a_{11} + \cdots + q_l a_{l1}) + \\ &\quad \vdots \\ &\quad h_m(q_1 a_{1m} + \cdots + q_l a_{lm}) \end{aligned}$$

The witnesses that $f \in \langle h_1, \ldots, h_m \rangle$ are

$$\begin{aligned} w_1 &= q_1 a_{11} + \cdots + q_l a_{l1} \\ &\vdots \\ w_m &= q_1 a_{1m} + \cdots + q_l a_{lm} \end{aligned}$$

□

This means that it is possible to solve arbitrary linear systems $MX = A$ over $k[x_1, \ldots, x_n]$.

Chapter 7

Conclusions

The aim of this chapter is to tie together all the previous chapters by looking at the result of the implementation and showing some examples of computations using it. The results are discussed and finally further work and limitations of the implementation are considered.

7.1 Implementation

The definitions and proofs in this thesis have been implemented in Haskell, the implementation can be found at `http://hackage.haskell.org/package/constructive-algebra`. This section will give a short overview of what can be done with this implementation and motivate some of the design choices. The algorithms has been implemented in the form of a library. One point of implementing it as a library is that a user easily can extend it by adding new algorithms and examples of rings.

The library is organized in three sections presented here with the most general parts first. The most general part is the level of all algebraic structures represented as type classes with QuickCheck properties to specify them. The general proofs or algorithms that hold for the different structures are also represented at this level. For example are coherent rings together with the proof that coherence implies that it is possible to solve systems of linear equations are represented at this level.

Next there is a part which contain general notions that are neither structures nor instances of the structures. Examples include ideals and the construction of fields of fractions.

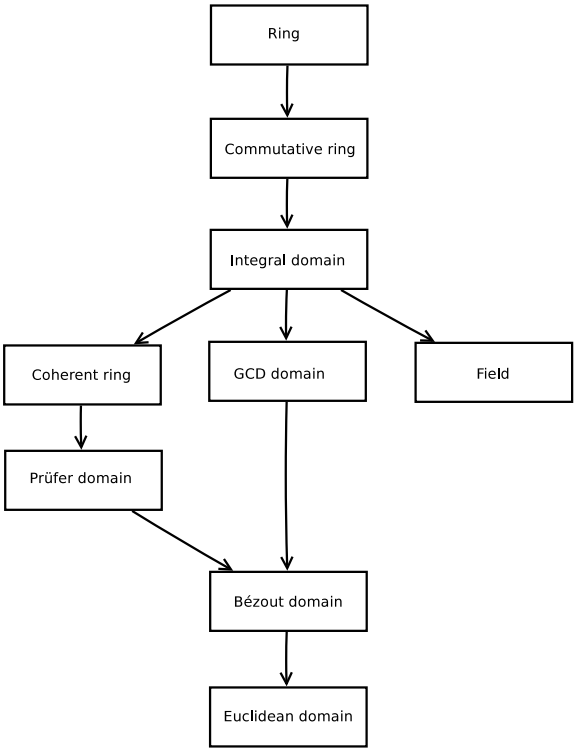
Finally there is the most concrete part which are concrete instances of the structures. The rings that has been implemented are $\mathbb{Z}$, $\mathbb{Z}_n$, $\mathbb{Z}[\sqrt{-5}]$, $\mathbb{Q}$, $k[x]$, $k(x)$, $k[x, y]$ with $y^2 = 1 - x^4$ and $k[x_1, \ldots, x_n]$. All of these has been tested to verify the different QuickCheck properties for the structures that they implement. This give some assurance that the implementation is in fact correct.

Both $\mathbb{Q}$ and $k(x)$ has been implemented using the representation of fields of fractions. Using this the implementation become more or less trivial. For example the representation of $\mathbb{Q}$ looks like

```
type Q = FieldOfFractions Z
```

This relies on that $\mathbb{Z}$ is an instance the GCD domain type class. This is no problem since $\mathbb{Z}$ is just a type synonym for the built-in arbitrary size integers of Haskell. The implementation of fields of fractions then take care of all instance declarations necessary for making it a field. It also implements equality and pretty printing functionality for the structure, given that the underlying ring is discrete and is an instance of the Show class. This illustrates the power of the Haskell type class system very well. It is easy to write general code which leads to modularity.

The hierarchy of algebraic structures is presented in figure 7.1.



Figur 7.1: Algebraic hierarchy

The following subsections will show some of the possible computations in the three different classes of rings presented in the thesis.

7.1.1 Computations in Bézout domains

The two Bézout domains to be considered in this section are $\mathbb{Z}$ and $k[x]$. Since all ideals are principal there exist a method for computing a principal ideal given an arbitrary ideal. The following example show what that computation can look like.

```
> toPrincipal (Id [4,6])
(<2>,[-1,1],[2,3])
```

```
> toPrincipal (Id [2,3])
(<1>,[-1,1],[2,3])
```

The witnesses for the first example means that

$$\begin{aligned} -1 \cdot 4 + 1 \cdot 6 &= 2 \\ 2 \cdot 2 &= 4 \\ 2 \cdot 3 &= 6 \end{aligned}$$

The intersection of ideals can also be computed in Bézout domains.

```
> Id [2] `intersectionB` Id [3]
<6>

> Id [2,3] `intersectionB` Id [3]
<3>
```

This give that Bézout domains are coherent and that it is possible to solve systems of equation in them.

```
> solveMxN (M [Vec [1,3,-2], Vec [3,5,6]])
[ 7,0]
[-3,0]
[-1,0]
```

Finally Bézout domains are also strongly discrete, so it is possible to test ideal membership in them.

```
> member 2 (Id [4,6])
Just [-1,1]

> member 3 (Id [4,6])
Nothing
```

7.1.2 Computations in Prüfer domains

The Prüfer domains to be considered in this section are $\mathbb{Z}$, $\mathbb{Z}[\sqrt{-5}]$ and $k[x,y]$ with $y^2 = 1 - x^4$. The principal localization matrices for $\langle 2,3,4\rangle$ in $\mathbb{Z}$ and $\langle x, 1-y\rangle$ in $\mathbb{Q}[x,y]$ with $y^2 = 1 - x^4$ can be computed.

```
> computePLM_PD (Id [2,3,4])
[-2,-3, -4]
[-6,-9,-12]
[ 6, 9, 12]

> computePLM_PD (Id [C (x,zero),C (one,neg one)])
[1/2+1/2*y,   1/2x^3]
[     1/2x,1/2-1/2*y]
```

The intersection of ideals can also be computed. First $\langle 2\rangle \cap \langle 1-\sqrt{-5}\rangle$ in $\mathbb{Z}[\sqrt{-5}]$ and then $\langle x\rangle \cap \langle 1-y\rangle$ in $\mathbb{Q}[x,y]$ with $y^2 = 1-x^4$ are computed.

```
> Id [ZSqrt5 (2,0)] `intersectionPD` Id [ZSqrt5 (1,-1)]
<-6,4+2*sqrt(-5)>

> Id [C (x,zero)] `intersectionPD` Id [C (one,neg one)]
<1/2x^4,1/2x-1/2x*y>
```

The above notation means that

$$\begin{aligned} \text{Id [ZSqrt5 (2,0)]} &= \langle 2\rangle \\ \text{Id [ZSqrt5 (1,-1)]} &= \langle 1-\sqrt{-5}\rangle \end{aligned}$$

in $\mathbb{Z}[\sqrt{-5}]$ and

$$\begin{aligned} \text{Id [C (x,zero)]} &= \langle x\rangle \\ \text{Id [C (one,neg one)]} &= \langle 1-y\rangle \end{aligned}$$

in $\mathbb{Q}[x,y]$ with $y^2 = 1-x^4$. It should be possible to make the input format more natural so that the user will not have to write all of the constructors.

The final result can be simplified by

$$\begin{aligned} \langle x\rangle \cap \langle 1-y\rangle &= \left\langle \frac{1}{2}x^4, \frac{1}{2}x - \frac{1}{2}xy \right\rangle \\ &= \langle x\rangle \left\langle x^3, 1-y\right\rangle \end{aligned}$$

It should be possible to construct a function that does this kind of simplifications, but this has not yet been done.

7.1.3 Computations in polynomial rings

The ring under consideration here is $\mathbb{Q}[x,y,z]$ with the lexicographical ordering on monomials. Once again the intersection of ideals is computable. The two examples are $\left\langle x^2y\right\rangle \cap \left\langle xy^2\right\rangle$ and $\left\langle x^2y, z^2\right\rangle \cap \left\langle xy^2, z\right\rangle$.

```
> Id [x^2*y] `intersectionMP` Id [x*y^2]
<x^2y^2>

> Id [x^2*y,z^2] `intersectionMP` Id [x*y^2,z]
<z^2,x^2yz,x^2y^2>
```

It is also possible to solve equations. The equation to solve is:

$$(x^3 - 2xy)t_1 + (x^2y - 2y^2 + x)t_2 + zt_3 = 0$$

```
> solve (Vec [x^3-2*x*y,x^2*y-2*y^2+x,z])
[x^2y+x-2y^2,       z,           0,0]
[   -x^3+2xy,       0,           z,0]
[          0,-x^3+2xy,-x^2y-x+2y^2,0]
```

The algorithm find four generators for the solutions, including the trivial one. The first one of these is

$$\begin{aligned} t_1 &= x^2y + x - 2y^2 \\ t_2 &= -x^3 + 2xy \\ t_3 &= 0 \end{aligned}$$

7.2 Discussion

The process of converting constructive proofs into computer programs is difficult. The reason is that many things are left out in the mathematical arguments. For example are the computation of witnesses often taken for granted and not considered at all, but it is often not completely trivial to compute them. This is the case when computing the intersection in polynomial rings, here the Buchberger algorithm has to be modified in a nontrivial manner.

The mathematical arguments also often rely on notions where the computational content is implicit. One example is the proof of proposition 5.12. Most proofs of this that I have seen are quite abstract and the steps to compute u, v, w in the definition of Prüfer domains are left out. Thus some effort has to be put into finding the computational content of the proofs even if the proofs are constructive. Note that this is not the case for all proofs, for example are the proofs that principal localization matrices are computable for Bézout domains and Prüfer domains presented in a form that is easy to convert into programs.

The difficulty of converting proofs is the reason that an implementation of the library in type theory would be good. All of the programs would then have to be proved correct and not just randomly tested. This would give an implementation that is proved correct for doing generalized linear algebra.

I am very satisfied with the choice of implementing the library in Haskell. Two of the main advantages of Haskell are polymorphism and type classes. If this would have been done in a language without polymorphism every algorithm would have to be implemented for each specific instance. But using Haskell one implementation is sufficient for all instances that implement the correct type classes.

The type class system also make the implementation elegant. The types of the programs capture the main aspects of the propositions. Just by looking at a type it is clear to what kind of structure the algorithm is applicable. Haskell is also quite compact in terms of the amount of code that need to be written, so the programs quite literally follow the mathematical proofs which make the implementation look appealing and easy to debug.

A traditional argument against Haskell is that it can be slow and that it is hard to reason about the complexity of programs due to laziness. This is not entirely true today. The GHC compiler does a very good work in optimizing programs, so the speed is not much of an issue. A possibility to make the implementation faster would be to use Haskell as a host language and then convert the programs into a more optimized mathematical language specialized in doing fast matrix computations.

The choice to use QuickCheck as the method to specify and test properties of rings proved useful. It works very well with random testing most of the time,

but sometimes the computations can get too heavy. This is the case for polynomial rings where the computation of Gröbner bases is very computationally expensive. Thus some heuristics are needed in order to reduce the space of test cases. Heuristics used in the project is to only use ideals of limited length and only have polynomials with multidegree less than some value.

The most important aspects when it comes to implementing mathematical programs are in my opinion expressability and compactness. The program should resemble the mathematical proofs as close as possible so that it is easy to read and understand. With respect to this, Haskell makes a very good language for developing mathematical programs.

7.3 Further work

The main limitations with the implementation is that most of the programs generate very large sets of generators for the solutions for systems of equations. This is not a problem for Bézout domains since they rely on that all finitely generated ideals are principal. But for Prüfer domains this becomes a problem. In rings that are both Bézout domains and Prüfer domains the methods give very different sizes of solutions. One method to solve this problem for Prüfer domains would be to implement a proof that all ideals in certain Prüfer domains can be generated by two elements, see for example [5] for more information on this. This would be both interesting and necessary for any practical implementation for solving larger examples.

The case for polynomial rings is similar. The size of the Gröbner bases can be very large and the computation very slow. For larger problems the implementation become too slow, so for a practical implementation more effort would be needed to optimize the algorithm that computes Gröbner bases. A simple solution that could increase the speed for some parts of the coherence proof is to use different orderings on the monomials in order to find one that produces small bases and fast computations.

The main focus in the future should in my opinion be to implement the whole theory of coherent rings and Prüfer domains in type theory, since this has not been done before and that it would be very interesting. After this, it would be possible to start focusing on optimizing the implementation and implementing methods to reduce the size of ideals.

Bibliography

[1] M. Atiyah and I. MacDonald. *Introduction to commutative algebra*. Addison-Wesley, 1969.

[2] Koen Claessen and John Hughes. Quickcheck: A Lightweight Tool for Random Testing of Haskell Programs. International Conference of Functional Programming, 2000.

[3] Thierry Coquand. Space of Valuations, 2008. `http://www.cse.chalmers.se/~coquand/val1.pdf`.

[4] Thierry Coquand and Henri Lombardi. A logical approach to abstract algebra. *Math. Structures Comput. Sci*, 16:885–900, 2006.

[5] Thierry Coquand, Henri Lombardi, and Claude Quitte. Generating non-Noetherian modules constructively. *Manuscripta Mathematica*, 115(4):513–520, 2004.

[6] Thierry Coquand, Henri Lombardi, and Claude Quitté. Curves and coherent Prüfer rings. 2009.

[7] David Cox, John Little, and Donal O'Shea. *Ideals, Varieties and Algorithms: An introduction to Computational Algebraic Geometry and Commutative Algebra*. Springer, 2006.

[8] L. Ducos, H. Lombardi, C. Quitté, and M. Salou. Théorie algorithmique des anneaux arithmétiques, des anneaux de Prüfer et des anneaux de Dedekind. *Journal of Algebra*, 281(2):604 – 650, 2004.

[9] John R. Durbin. *Modern Algebra: An Introduction*. Wiley, 2004.

[10] László Fuchs and Luigi Salce. *Modules ofer Non-Noetherian Domains*, volume 84 of *Mathematical Surveys and Monographs*. American Mathematical Society, 2001.

[11] François Garillot, Georges Gonthier, Assia Mahboubi, and Laurence Rideau. Packaging Mathematical Structures. In *TPHOLs '09: Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics*, pages 327–342, Berlin, Heidelberg, 2009. Springer-Verlag.

[12] Herman Geuvers, Randy Pollack, Freek Wiedijk, and Jan Zwanenburg. A Constructive Algebraic Hierarchy in Coq. *Journal of Symbolic Computation*, 34(4):271 – 286, 2002.

[13] Alessandro Giovini, Teo Mora, Gianfranco Niesi, Lorenzo Robbiano, and Carlo Traverso. "One sugar cube, please" or Selection strategies in the Buchberger algorithm. In *Proceedings of the ISSAC'91, ACM Press*, pages 49–54, 1991.

[14] Georges Gonthier. Formal Proof – The Four-Color Theorem. *Notices of the American Mathematical Society*, 55(11), 2008.

[15] Henri Lombardi and Hervé Perdry. The Buchberger Algorithm as a Tool for Ideal Theory of Polynomial Rings in Constructive Mathematics. 1998.

[16] Ray Mines, Fred Richman, and Wim Ruitenburg. *A Course in Constructive Algebra*. Springer-Verlag, 1988.

[17] Henrik Persson. An Integrated Development of Buchberger's Algorithm in Coq. 2001.

[18] Maïmouna Salou Idi Malam. *Théorie Algorithmique des Anneaux Arithmétiques, des Anneaux de Prüfer et des Anneaux de Dedekind*. PhD thesis, 2002.

[19] A.S. Troelstra and D. van Dalen. *Constructivism in Mathematics*. North-Holland, 1988. 2 volumes.