Partial Report for Deliverable Nr. 2.2

Linear Algebra over a Principal Ideal Domain

Smith Normal Form

Responsible: Thierry Coquand, coquand@chalmers.se
Written by: Cyril Cohen, cohen@crans.org
and Anders Mörtberg, mortberg@chalmers.se
Site: University of Gothenburg, Sweden

Deliverable Date: February 2011
Forecast Date: August 2011

Abstract

This report presents a formalization of the theory of rings with explicit divisibility using the SSReflect extension to the Coq system. The goal of this work is to be able to formalize and prove correct the Smith normal form algorithm for computing homology groups of simplicial complexes from algebraic topology.

1 Introduction

This report discusses ongoing work to formalize the theory of rings with explicit divisibility in the Coq system with the SSReflect extension. One important application of this is the formalization of the Smith normal form algorithm which is a generalization of Gauss' elimination algorithm to principal ideal domains instead of fields. Our motivation for studying this algorithm is mainly for computing homology groups of simplicial complexes in algebraic topology. This has applications in analysis of digital images [2] for example. The first step in formalizing this is to have a definition of principal ideal domains in type theory, as presented in this document.

The results presented here will also be used in the development of D2.3: linear algebra over a coherent strongly discrete ring.

2 Rings with explicit divisibility

From here on all rings are discrete integral domains. The standard examples for all of the rings presented here are $\mathbb{Z}$ and $k[x]$ where k is a field. This section is loosely based on the presentation of divisibility in discrete domains of [6].

Definition 1 A ring R has *explicit divisibility* if it has a divisibility test that produces witnesses.

That is, given a and b we can check if $a \mid b$ and if this is the case get x such that $b = xa$. Two elements $a, b \in R$ are *associated* if $a \mid b$ and $b \mid a$, as we have cancellation this is equivalent to $b = ua$ for some unit u. Note that this give an equivalence relation.

One important property is that if R is a ring with explicit divisibility then $R[x]$ also is. This follow from the standard long division algorithm for polynomials over a field which can be modified in such a way that it uses the explicit divisibility relation instead. If the algorithm at some point needs to divide an element by an element that doesn't divide it then the algorithm terminates with the result that the polynomial doesn't divide the other.

An interesting class of rings with explicit divisibility are GCD rings, these are non-Noetherian analogues of unique factorization domains.

Definition 2 A *GCD ring* R is a ring with explicit divisibility in which every pair of elements has a greatest common divisor, that is, for $a, b \in R$ there is $gcd(a, b)$ such that $gcd(a, b) \mid a$, $gcd(a, b) \mid b$ and $\forall g$, $(g \mid a) \wedge (g \mid b) \Rightarrow g \mid gcd(a, b)$.

Note first that we make no restriction on a and b, so these can both be zero, if this is the case then the greatest common divisor is zero. This makes sense as zero is the maximum element for the divisibility relation. Note also that R is assumed to be an ring with explicit divisibility so the fact that $gcd(a, b) \mid a$ means that we can find x such that $a = x \cdot gcd(a, b)$. By Euclid's algorithm we know that both $\mathbb{Z}$ and $k[x]$ are GCD rings.

With the above definition the greatest common divisor of two elements is not unique, e.g. the greatest common divisor of 2 and 3 in $\mathbb{Z}$ is either 1 or -1. But if we consider equality up to multiplication by units, i.e. up to associatedness, then the greatest common divisor is unique. So from here on equality should be read as equality up to associatedness.

Two elements $a, b \in R$ are *coprime*, written $a \perp b$, if their greatest common divisor is 1 (or equivalently a unit). For example 2 and 3 are coprime. One important lemma which is sometimes called *Euclid's lemma*[1] is that if $a \mid bc$ and $a \perp b$ then $a \mid c$. This can be used to prove that an irreducible element in a GCD ring is prime.

We now turn our attention to the polynomial ring $R[x]$ over a GCD ring R. The greatest common divisor of the coefficients of $p \in R[x]$ is called the *content* of p and is written $cont(p)$. A polynomial is *primitive* if its content is 1.

Theorem 1 Let R be a GCD ring and $p, q \in R[x]$, then $cont(pq) = cont(p)cont(q)$.

This result is called *Gauss lemma* and one corollary is that the product of two primitive polynomials is primitive.

Theorem 2 Let R be a GCD ring and $p \in R[x]$, then we can find a primitive q such that $p = cont(p) \cdot q$.

Using these two theorems one can prove constructively that if R is a GCD ring then $R[x]$ is GCD ring. This can be done in the same manner as in the proof found in [4] that if R is a Unique Factorization Domain (abbreviated UFD) then $R[x]$ is an UFD as well. This is

[1] It seems like this result is called *Gauss' lemma* in France, but we will use Euclid's lemma as Gauss lemma will denote something else later on.

interesting since it gives an alternative proof to the one presented in [6] which does not rely on the field of fractions of $R[x]$. So this alternative proof should be more suitable to formalize in type theory as it does not rely on the field of fractions. Formalizing this theorem would give an algorithm for computing the greatest common divisor of two elements in for example $\mathbb{Z}[x_1, \ldots, x_n]$ or $k[x_1, \ldots, x_n]$.

Another class of rings with explicit divisibility are Bézout rings.

Definition 3 A *Bézout ring* is a ring R with explicit divisibility such that for any two elements $a, b \in R$ there is $x, y \in R$ such that $ax + by = gcd(a, b)$.

Clearly any Bézout ring is a GCD ring. We also know that both $\mathbb{Z}$ and $k[x]$ can be proved to be Bézout rings using the extended version of Euclid's algorithm.

Just as GCD rings are non-Noetherian analogues of the classical notion of UFD, Bézout rings are the non-Noetherian analogues of principal ideal domains. Bézout rings can hence also be characterized as rings in which every finitely generated ideal is principal.

The final class of rings with explicit divisibility that we shall consider in this section are Euclidean rings.

Definition 4 An *Euclidean ring* is a ring a R with a Euclidean norm $\mathcal{N} : R \to \mathbb{N}$ such that for any $a \in R$ and nonzero $b \in R$ we have $\mathcal{N}(a) \leqslant \mathcal{N}(ab)$. Further, for any $a \in R$ and nonzero $b \in R$ we can find $q, r \in R$ such that $a = bq + r$ and either $r = 0$ or $\mathcal{N}(r) < \mathcal{N}(b)$.

The extended version of Euclid's algorithm can be implemented for Euclidean rings and hence any Euclidean ring is both a Bézout and a GCD ring. Both $\mathbb{Z}$ and $k[x]$ are examples of Euclidean rings: take the absolute value function for $\mathbb{Z}$ and the size of the underlying list as explained in section 3.1 of [1] for $k[x]$ and then the standard division algorithm can be used to compute q and r.

3 Smith normal form

As mentioned in the introduction the Smith normal form algorithm is a generalization of Gauss' algorithm for solving equations over any principal ideal domain and not just over a field.

Before introducing the Smith normal form of a matrix we need to discuss principal ideal domains in constructive mathematics. The reason is that we cannot use Bézout rings as it is still an open problem to show that the algorithm is correct over them [5]. So what we need is a suitable constructive approximation of the Noetherian property. The proper definition in this case is that the strict divisibility relation is well-founded.

Definition 5 A *constructive principal ideal domain* is a Bézout ring in which the strict divisibility relation is well-founded. a divides b *strictly* if $a \mid b$ but $b \nmid a$.

This is the right setting in order to specify the Smith normal form algorithm.

Theorem 3 Let R be a constructive principal ideal domain and $M \in R^{m\times n}$. Then it is possible to compute invertible $S \in R^{m\times m}$ and $T \in R^{n\times n}$ such that

$$SMT = \begin{pmatrix} \alpha_1 & 0 & 0 & & \cdots & & 0 \\ 0 & \alpha_2 & 0 & & \cdots & & 0 \\ 0 & 0 & \ddots & & & & 0 \\ \vdots & & & \alpha_r & & & \vdots \\ & & & & 0 & & \\ & & & & & \ddots & \\ 0 & & & \cdots & & & 0 \end{pmatrix}$$

and $\alpha_i \mid \alpha_{i+1}$ for $1 \leqslant i < r$.

As mentioned in the introduction this algorithm can be used for computing homology groups of simplicial complexes in algebraic topology.

4 Formalization

In this section we will discuss the formalization of the theory of rings with explicit divisibility using the SSReflect extension to the Coq system.

The algebraic structures have been formalized in the same manner as described in [3] using coercions and canonical structures. The development here extends the existing algebraic hierarchy of SSReflect and the relationship between the structures and their place in the existing library is shown in figure 1 below. In the figure IntegralDomain is already present in the SSReflect hierarchy and the other boxes indicate how our development extends the existing hierarchy.

We will now discuss the formalization of the new structures starting with the definition of rings with explicit divisibility. This is called DvdRing in the implementation and is represented as:

```
CoInductive div_spec (R : ringType) (a b :R) : option R -> Type :=
| DivDvd x of a = x * b : div_spec a b (Some x)
| DivNDvd of (forall x, a != x * b) : div_spec a b None.

Module DvdRing.
Record mixin_of (R : ringType) : Type := Mixin {
  div : R -> R -> option R;
  _ : forall a b, div_spec a b (div a b)
}.
End DvdRing.
```

So for a ring to be a DvdRing it needs to have a function `div` that returns an option type, such that if `div a b = None` then $a \nmid b$ and if `div a b = Some x` then this x is the witness that $a \mid b$. The specification for DvdRings says this, but it also requires the user to prove that if $a \nmid b$ then there is no x such that $a = xb$.

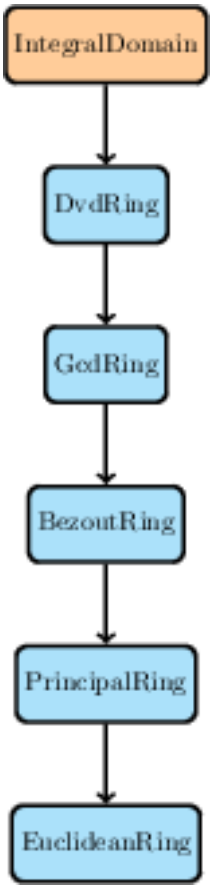

Figure 1: The extension to the existing algebraic hierarchy

This has been used to prove many results about divisibility. We have also implemented the notion of associatedness mentioned above and proved properties about it. Sadly this is not a rewritable relation and hence we can't use the `rewrite` tactic when reasoning about elements that are associates.

We have also implemented a proof that if R is a DvdRing then $R[x]$ also is. This was done by implementing a version of long division for polynomials which uses the `div` function and terminates with `None` if it reaches a `None` at some point in the computation.

Next we have the GCDRing structure which is implemented as:

```
Module GcdRing.
Record mixin_of (R : dvdRingType) : Type := Mixin {
  gcdr : R -> R -> R;
  _ : forall a b g, g %| gcdr a b = (g %| a) && (g %| b)
}.
End GcdRing.
```

Here `%|` is notation for the divisibility function utilizes that there is a coercion from `option` to `bool` in SSReflect libraries.

For a ring to be a GcdRing it need to have a gcd function and satisfy the property above. This property is sufficient as it implicitly give that $gcd(a, b) \mid a$ and $gcd(a, b) \mid b$ as divisibility is reflexive.

We have formalized the notion of being coprime and proved Euclid's lemma and that the prime elements are the same as the irreducible elements in a GCDRing. We have also implemented the notion of content of polynomials over a GCDRing and proved Gauss lemma. This proof is quite involved and we have found a slightly different proof from the one in [6] which is more suitable for formalization.

The BezoutRing structure looks like:

```
CoInductive bspec (R : gcdRingType) (a b : R) : R * R -> Type :=
  BezoutSpec x y of gcdr a b %= x * a + y * b : bspec a b (x, y).

Module BezoutRing.
Record mixin_of (R : gcdRingType) : Type := Mixin {
  bezout : R -> R -> (R * R);
  _ : forall a b, bspec a b (bezout a b)
}.
End BezoutRing.
```

Here `%=` denotes associatedness. The ring is assumed to be a GCDRing and the user should provide a function that computes a pair (x, y) such that $ax + by = gcd(a, b)$. We have proved that it is sufficient to instantiate this structure in order to get a GCDRing.

Finally we have the EuclideanRing structure that represents Euclidean rings

```
CoInductive edivr_spec (R : ringType)
  (g : R -> nat) (a b : R) : R * R -> Type :=
  EdivrSpec q r of a = q * b + r & (b != 0) ==> (g r < g b)
  : edivr_spec g a b (q, r).

Module EuclideanRing.
Record mixin_of (R : ringType) : Type := Mixin {
  enorm : R -> nat;
  ediv : R -> R -> (R * R);
  _ : forall a b, a != 0 -> enorm b <= enorm (a * b);
  _ : forall a b, edivr_spec enorm a b (ediv a b)
}.
End EuclideanRing.
```

This structure contains the Euclidean norm and the Euclidean division function together with their specifications. The theory for proving that Euclidean rings are Bézout and GCD rings is quite interesting, this involves the implementation and correctness proof of the extended Euclid's algorithm and the ordinary Euclid's algorithm. All of this has been implemented and proved correct, so we would only need to instantiate the EuclideanRing structure and

then we would get the Bézout-, GCD- and DvdRing structures for free. But we are not forced to use the default implementations; maybe there is a more efficient algorithm for a specific structure and we may then want to implement this and not use the general predefined version.

As we mentioned in the beginning the two standard examples of these rings are $\mathbb{Z}$ and $k[x]$ where k is a field. We have instantiated the EuclideanRing structure with two implementations of $\mathbb{Z}$, one unary and one binary. The unary one is already part of the SSReflect library, so proving that it satisfies the axioms of EuclideanRing was straight-forward. We have tested to compute with it and it works fine, but it is quite inefficient as it is a unary representation:

```
> Time Eval compute in (gcdr 11466%:Z 1428).
     = Posz 42
     : DvdRing.Pack (GcdRing.class zint_gcdType) zint_gcdType
Finished transaction in 1. secs (0.97327u,0.003333s)

> Time Eval compute in (gcdr 114662%:Z 14282).
[Segmentation fault]
```

The reason we get a segmentation fault is that it uses too much memory. The more efficient binary implementation of integers was slightly more difficult to prove that it is an instance of the EuclideanRing structure as we had to use the standard Coq implementation of $\mathbb{Z}$ so we couldn't use all of the facilities of SSReflect in the proofs. But in the end it turned out to be significantly faster:

```
> Time Eval compute in (gcdr 11466 1428)%Z.
     = 42%Z
     : DvdRing.Pack (GcdRing.class Z_gcdType) Z_gcdType
Finished transaction in 0. secs (0.339978u,0.s)

> Time Eval compute in (gcdr 114662 14282)%Z.
     = 2%Z
     : DvdRing.Pack (GcdRing.class Z_gcdType) Z_gcdType
Finished transaction in 5. secs (4.973009u,0.066663s)
```

Note that the last computation actually terminates and doesn't use too much memory. Also note that the first computation is faster than in the previous test.

Implementing the proof that $k[x]$ is an EuclideanRing also required some work. One possibility would have been to use the pseudo-division algorithm that is already in the SSReflect libraries but we decided not to do this as it would be unnecessarily inefficient, as it has to compute many irrelevant things. So instead we implemented the algorithm from scratch. The correctness proof is similar to the proof that $R[x]$ is a ring with explicit divisibility if R is. So we can compute for example greatest common divisors of elements of $k[x]$ where k is one of the fields in the SSReflect library like $\mathbb{Q}$ or $\mathbb{Z}_p$ where p is a prime.

In order to be able to start formalizing the theory about Smith normal forms we implemented constructive principal ideal domains, called PriRings in the implementation. This was done by implementing a predicate for strict divisibility and specifying that it is well

founded.

```
Definition sdvdr (R : dvdRingType) (x y : R) :=
  (x %| y) && ~~(y %| x).

Module PriRing.
Record mixin_of (R : dvdRingType) : Type := Mixin {
  _ : well_founded (@sdvdr R)
}.
End PriRing.
```

So to prove that EuclideanRings are PriRings we had to prove that strict divisibility is well founded. The `well_founded` predicate in Coq is expressed using accessibility predicates and hence we had to do reasoning about this in order to prove what we wanted. This has been done and hence we will be able to compute the Smith normal form for matrices of coefficients from $\mathbb{Z}$ and $k[x]$ once we have formalized the algorithm.

The Smith normal form algorithm has been implemented in the functional programming language Haskell. The implementation is based on the algebraic hierarchy presented in [7]. We have begun converting this implementation to Coq and started proving its correctness using SSReflect.

5 Conclusion and further work

In this report we have presented a formalization of the theory of rings with explicit divisibility. This gives formalized algorithms for computing for example greatest common divisors of elements in $\mathbb{Z}$ and $k[x]$ where k is $\mathbb{Q}$ or $\mathbb{Z}_p$ where p is prime. We can also perform certified long division in polynomial rings over these rings.

In order to finish the deliverable we need to convert the Haskell implementation of the Smith normal form algorithm into Coq and prove it correct. This will most likely require quite a lot of work but at least all the preliminary work has been done and we are now in the correct setting to start the formalization. The next step would then be to do some experiments with it and compute some homology groups for real examples from algebraic topology.

It would also be interesting to finish the proof that $R[x]$ is a GCD ring if R is and use this to compute greatest common divisors over multivariate polynomial rings. For doing this we would probably need to do a lot of reasoning about associated elements and as this relation is not rewritable we wouldn't be able to use the `rewrite` tactic which may result in very long proofs. In order to remedy this C. Cohen has explored if it would be possible to use quotient types or setoids in order to be able to rewrite with the relation but so far no satisfactory solution has been found.

References

[1] C. Cohen and A. Mahboubi. A formal quantifier elimination for algebraically closed fields. In *Proceedings of the 10th ASIC and 9th MKM international conference, and 17th Calculemus conference on Intelligent computer mathematics*, AISC'10/MKM'10/Calculemus'10, pages 189–203, Berlin, Heidelberg, 2010. Springer-Verlag.

[2] ForMath La Rioja node. From Digital Images to Simplicial Complexes: A Report. Technical report, Departamento de Matemticas y Computacin, Universidad de La Rioja, 2011.

[3] F. Garillot, G. Gonthier, A. Mahboubi, and L. Rideau. Packaging mathematical structures. In *Proceedings 22nd International Conference on Theorem Proving in Higher Order Logics (TPHOLs'2009)*, volume 5674 of *Lecture Notes in Computer Science*, pages 327–342, 2009.

[4] D. E. Knuth. *The art of computer programming, volume 2: seminumerical algorithms.* Addison-Wesley, 1981.

[5] D. Lorenzini. On Bzout Domains. `http://www.math.uga.edu/~lorenz/Bezout.pdf`.

[6] R. Mines, F. Richman, and W. Ruitenburg. *A Course in Constructive Algebra.* Springer-Verlag, 1988.

[7] A. Mörtberg. Constructive Algebra in Functional Programming and Type Theory. Master's thesis, Chalmers University of Technology, 2010. `http://web.student.chalmers.se/~mortberg/master/MSc-Thesis.pdf`.