Coherent and Strongly Discrete Rings in Type Theory*

Thierry Coquand, Anders Mörtberg, and Vincent Siles

Department of Computer Science and Engineering
University of Gothenburg, Sweden
{coquand,mortberg,siles}@chalmers.se

Abstract. We present a formalization of coherent and strongly discrete rings in type theory. This is a fundamental structure in constructive algebra that represents rings in which it is possible to solve linear systems of equations. These structures have been instantiated with Bézout domains (for instance $\mathbb{Z}$ and $k[x]$) and Prüfer domains (generalization of Dedekind domains) so that we get certified algorithms solving systems of equations that are applicable on these general structures. This work can be seen as basis for developing a formalized library of linear algebra over rings.

Keywords: Formalization of mathematics, Constructive algebra, Coq, SSReflect

1 Introduction

One of the fundamental operations in linear algebra is the ability to solve linear systems of equations. The concept of (strongly discrete) coherent rings abstracts over this ability which makes them an important notion in constructive algebra [14]. This makes these rings suitable as a basis for developing computational homological algebra, that is, linear algebra over rings instead of fields [3].

Another reason that these rings are important in constructive algebra is that they generalize the notion of Noetherian rings.[1] Classically any Noetherian ring is coherent but the situation in constructive mathematics is more complex and there is in fact no standard constructive definition of Noetherianity [16]. Logically, Noetherianity is expressed by a higher-order condition (it involves quantification over every ideal of the ring) while "coherent" is a simpler notion, which involves only quantification on matrices over the ring, and "strongly discrete" is a first-order notion.

One important example (aside from fields) of coherent strongly discrete rings are Bézout domains, which are a non-Noetherian generalization of principal ideal

* The research leading to these results has received funding from the European Union's 7th Framework Programme under grant agreement nr. 243847 (ForMath).

[1] Rings where all ideals are finitely generated.

domains (rings where all ideals are generated by one element). The two standard examples of Bézout domains are $\mathbb{Z}$ and $k[x]$ where k is a field. Another example of coherent strongly discrete rings are Prüfer domains with decidable divisibility which are a non-Noetherian generalization of Dedekind domains. The condition of being a Prüfer domain captures what Dedekind thought was the most important property of Dedekind domains [1], namely the ability to invert ideals (which is usually hidden in standard classical treatments of Dedekind domains). This property also has applications in control theory [18].

All our proofs and definitions are expressed in a constructive framework. While it would be possible to use classical logic in the proof of correctness of our algorithms, we feel that they are clearer and shorter in this way. It can also be argued that our definitions are better expressed in this way. For instance, we can define a coherent ring as one for which a linear system has a finite number of generators. In a classical framework, to express this in a computationally meaningful way would involve the notion of recursive functions.

All of these notions have been formalized[2] using the SSReflect extension [11] to the Coq proof assistant [5]. This work can be seen as a generalization of the previous formalization of linear algebra in the SSReflect library [10].

The main motivation behind this work is that it can be seen as a basis for a formalization of computational homological algebra. This approach is inspired by the one of homalg [3] where homological algorithms (without formalized correctness proofs) are implemented based on a notion that they call *computable* rings [2] which in fact are the same as coherent strongly discrete rings. Another source of inspiration is the work of Lombardi and Quitté [13] on constructive commutative algebra.

This paper is organized as follows: first the formalization of coherent rings is presented followed by strongly discrete rings. Next Prüfer domains are explained together with the proofs that they are both coherent and strongly discrete. This is followed by a section on how to implement a computational version of the SSReflect development. We end by a section on conclusions and further work.

2 Coherent rings

Given a ring R (in our setting commutative but it is possible to consider non-commutative rings as well [2]) one important problem to study is how to solve linear systems over R. If R is a field, then we have a nice description of the space of solution by a basis of solutions. Over an arbitrary ring R there is in general no basis.[3] But an important weaker property is that there is a finite number of

[2] Documentation and formalization can be found at:
`http://www.cse.chalmers.se/~mortberg/coherent/`

[3] For instance over the ring $R = k[X, Y, Z]$ where k is a field, the equation $pX + qY + rZ = 0$ has no basis of solutions. It can be shown that a generating system of solutions is given by $(-Y, X, 0),\ (Z, 0, -X),\ (0, -Z, Y)$.

solutions which generate all solutions. We say that the ring is *coherent* if this is the case.

More concretely, given a rectangular matrix M over R we want to find a finite number of solutions $X_1, \ldots, X_n$ of the system $MX = 0$ such that any solution is of the form $a_1X_1 + \cdots + a_nX_n$ where $a_1, \ldots, a_n \in R$. If this is possible, we say that the module of solutions of the system $MX = 0$ is finitely generated. This can be reformulated with matrices: we want to find a matrix L such that

$$MX = 0 \ \leftrightarrow \ \exists Y.X = LY$$

A ring is *coherent* if for any matrix M it is possible to compute a matrix L such that this holds. If this is the case it follows that $ML = 0$.

For this it is enough to consider the case where M has only one line. Indeed, assume that for any $1 \times n$ matrix M we can find a $n \times m$ matrix L such that $MX = 0$ iff $X = LY$ for some Y. To solve the system

$$M_1X = \cdots = M_kX = 0$$

where each M_i is a $1 \times n$ matrix first compute L_1 such that $M_1X = 0$ iff $X = LY_1$ for some Y_1. Next compute L_2 such that $M_2L_1Y_1 = 0$ iff $Y_1 = L_2Y_2$. At the end we obtain $L_1, \ldots, L_k$ such that $M_1X = \cdots = M_kX = 0$ iff X is of the form $L_1 \cdots L_kY$ and so $L_1 \cdots L_k$ provide a system of generators for the solution of the system.

Hence it is sufficient to formulate the condition for coherent rings as: For any *row* matrix M it is possible to find a matrix L such that

$$MX = 0 \ \leftrightarrow \ \exists Y.X = LY$$

Note that the notion of coherent is not stressed in classical presentations of algebra since Noetherian rings are automatically coherent, but in a computationally meaningless way. It is however fundamental, both conceptually [13, 14] and computationally. The system HOMALG [3] for instance takes this notion as the central one.

In the development, coherent rings have been implemented as in [9] using the `Canonical Structure` mechanism of Coq. In the SSReflect libraries matrices are represented by finite functions over pairs of ordinals (the indices):

```
(* 'I_n *)
Inductive ordinal (n : nat) := Ordinal m of m < n.

(* 'M[R]_(m,n) = 'M_(m,n) *)
(* 'rV[R]_m = 'M[R]_(1,m) *)
(* 'cV[R]_m = 'M[R]_(m,1) *)
Inductive matrix R m n := Matrix of {ffun 'I_m * 'I_n -> R}.
```

Hence the sizes of the matrices need to be known when implementing coherent rings. But in general the size of L cannot be predicted so we need an extra function that computes this:

```
Record mixin_of (R : ringType) : Type := Mixin {
 size_solve : forall m, 'rV[R]_m -> nat;
 solve_row : forall m (V : 'rV[R]_m), 'M[R]_(m,size_solve V);
 _ : forall m (V : 'rV[R]_m) (X : 'cV[R]_m),
  reflect (exists Y : 'cV[R]_(size_solve V), X = solve_row V *m Y)
          (V *m X == 0)
}.
```

Here `*m` denotes matrix multiplication and `V *m X == 0` is the boolean equality of matrices, so the specification says that this equality is reflected by the existence statement. An alternative to having a function computing the size would be to output a dependent pair but this has the undesired behavior that the pair has to be destructed when stating lemmas about it which in turn would mean that these lemmas would be cumbersome to use as it would not be possible to rewrite with them directly.

Using this we have implemented the algorithm for computing the generators of a system of equations:

```
Fixpoint solveMxN (m n : nat) :
  forall (M : 'M_(m,n)), 'M_(n,size_solveMxN M) :=
   match m return forall M : 'M_(m,n), 'M_(n,size_solveMxN M) with
    | S p => fun (M : 'M_(1 + _,n)) =>
      let L1 := solve_row (usubmx M)
      in L1 *m solveMxN (dsubmx M *m L1)
    | _ => fun _ => 1%:M
   end.

Lemma solveMxNP : forall m n (M : 'M[R]_(m,n)) (X : 'cV[R]_n),
  reflect (exists Y : 'cV_(size_solveMxN M), X = solveMxN M *m Y)
          (M *m X == 0).
```

In order to instantiate this structure one can of course directly give an algorithm that computes the solution of a single row system. However there is another approach that will be used in the rest of the paper that is based on the intersection of finitely generated ideals.

2.1 Ideal intersection and coherence

In the case when R is an integral domain one way to prove that R is coherent is to show that the intersection of two finitely generated ideals is again finitely generated. This amounts to given two ideals $I = (a_1, \ldots, a_n)$ and $J = (b_1, \ldots, b_m)$ compute generators $(c_1, \ldots, c_k)$ of $I \cap J$. For $I \cap J$ to be the intersection of I and J it should satisfy

$$I \cap J \subseteq I$$

$$I \cap J \subseteq J$$

$$\forall x.\ x \in I \wedge x \in J \to x \in I \cap J$$

The first two of these mean that the generators of $I \cap J$ should be possible to write as a linear combination of the generators of both I and J. The third property states that if x can be written as a linear combination of the generators of I and J then it can be written as a linear combination of the generators of $I \cap J$.

A convenient way to express this in Coq is to use strongly discrete rings, which are discussed in section 3. For now we just assume that we can find matrices V and W such that $IV = I \cap J$ and $JW = I \cap J$ (with matrix multiplication and ideals represented by row-vectors containing the generators). Using this there is an algorithm to compute generators of the solutions of a system:

$$m_1 x_1 + \cdots + m_n x_n = 0$$

The main idea is to compute generators, M_0, of the solution for $m_2 x_2 + \cdots + m_n x_n = 0$ by recursion and also compute generators $t_1, \ldots, t_p$ of $(m_1) \cap (-m_2, \ldots, -m_n)$ together with V and W such that

$$\begin{aligned}(m_1)V &= (t_1, \ldots, t_p)\\ (-m_2, \ldots, -m_n)W &= (t_1, \ldots, t_p)\end{aligned}$$

The generators of the module of solutions are then given by:

$$\begin{bmatrix} V & 0 \\ W & M_0 \end{bmatrix}$$

This has been implemented by:

```
Fixpoint solve_int m : forall (M : 'rV_m),'M_(m,size_int M) :=
  match m return forall (M : 'rV_m), 'M_(m,size_int M) with
    | S p => fun (M' : 'rV_(1 + p)) =>
             let m1 := lsubmx M' in
             let ms := rsubmx M' in
             let M0 := solve_int ms in
             let V := cap_wl m1 (-ms) in
             let W := cap_wr m1 (-ms) in
             block_mx (if m1 == 0 then delta_mx 0 0 else V) 0
                      (if m1 == 0 then 0 else W) M0
    | 0 => fun _ => 0
  end.

Lemma solve_intP : forall m (M : 'rV_m) (X : 'cV_m),
  reflect (exists Y : 'cV[R]_(size_int M), X = solve_int M *m Y)
          (M *m X == 0).
```

Here `cap_wl` computes V and `cap_wr` computes W. Note that some special care has to be taken if m_1 is zero, if this is the case we output a matrix:

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & M_0 \end{bmatrix}$$

However it would be desirable to output just

$$\begin{bmatrix} 1 & 0 \\ 0 & M_0 \end{bmatrix}$$

But this would not have the correct size. This could be solved by having a more complicated function that outputs a sum type with matrices of two different sizes. As this would give slightly more complicated proofs we decided to pad with zeroes instead. In section 5 we will discuss how to implement a more efficient algorithm, without any padding, that is more suitable for computation.

3 Strongly discrete rings

An important notion in constructive mathematics is the notion of *discrete* ring, that is, rings with decidable equality. Another important notion is *strongly discrete* rings, these are rings where membership in finitely generated ideals is decidable and if $x \in (a_1, \ldots, a_n)$ there is an algorithm computing $w_1, \ldots, w_n$ such that $x = \sum_i a_i w_i$.

Examples of such rings are multivariate polynomial rings over discrete fields (via Gröbner bases [6, 12]) and Bézout domains with explicit divisibility, that is, whenever $a \mid b$ one can compute x such that $b = xa$. We have represented strongly discrete rings in Coq as:

```
CoInductive member_spec (R : ringType) n (x : R) (I : 'rV[R]_n)
  : option 'cV[R]_n -> Type :=
| Member J of x%:M = I *m J : member_spec x I (Some J)
| NMember of (forall J, x%:M != I *m J) : member_spec x I None.

Record mixin_of (R : ringType) : Type := Mixin {
  member : forall n, R -> 'rV_n -> option 'cV_n;
  _ : forall n (x : R) (I : 'rV_n), member_spec x I (member x I)
}.
```

The structure of strongly discrete rings contains a function taking an element and a row vector (with the generators of the ideal) and return an option type with a column vector. This is `Some J` if `x` can be written as `I *m J` and if it is `None` then there should also be a proof that there cannot be any `J` satisfying `x = I *m J`. Note that the use of `CoInductive` has nothing to do with coinduction but it should be seen as a datatype without any recursion schemes (as opposed to datatypes defined using `Inductive`) on which one can do case-analysis, for more information see [11].

3.1 Ideal theory

In the development we have chosen to represent finitely generated ideals as row vectors, so an ideal in R with n generators is represented as a row matrix of type

`'rV[R]_n`. This way operations on ideals can be implemented using functions on matrices and properties can be proved using the matrix library of SSReflect.

A nice property of strongly discrete rings is that the inclusion relation of finitely generated ideals is decidable. This means that we can decide if $I \subseteq J$ and if this is the case express every generator of I as a linear combination of the generators of J. We have implemented this as the function `subid` with correctness expressed as:

```
Notation "A <= B" := (subid A B).
Notation "A == B" := ((A <= B) && (B <= A)).

Lemma subidP : forall m n (I : 'rV[R]_m) (J : 'rV[R]_n),
  reflect (exists D, I = J *m D) (I <= J)%IS.
```

Note that this is expressed using matrix multiplication, so `subidP` says that if `I <= J` then every generator of `I` can be written as a linear combination of generators of `J`.

Ideal multiplication is an example where it is convenient to represent ideals as row vectors. As the product of two finitely generated ideals is generated by all products of generators of the ideals this can be expressed compactly using matrix operations:

```
Definition mulid m n (I : 'rV_m) (J : 'rV_n) : 'rV_(m * n) :=
  mxvec (I^T *m J).

Notation "I *i J" := (mulid I J).
```

Here `mxvec` flattens a matrix of type `'M[R]_(m,n)` into a row vector of type `'rV[R]_(m * n)` and `I^T` is the transpose of `I`. By representing ideals as row vectors we get compact definitions and quite simple proofs as the theory already developed about matrices can be used when proving properties of ideal operations.

It is also convenient to specify what the intersection of I and J is: it is an ideal K such that `K <= I`, `K <= J` and `forall (x : R), member x I -> member x J -> member x K`. So in order to prove that an integral domain is coherent it suffices to give an algorithm that computes K and prove that it satisfies these three properties. The `cap_wr` and `cap_wl` functions used in `solve_with_int` can then be implemented easily by explicitly computing `D` in `subidP`.

3.2 Coherent strongly discrete rings

If a ring R is both coherent and strongly discrete it is not only possible to solve homogeneous systems $MX = 0$ but also any system $MX = A$. The algorithm for computing this is expressed by induction on the number of equations where the case of one equation follow directly from the fact that the ring is strongly discrete. In the other case the matrix looks like:

$$\begin{bmatrix} R_1 \\ M \end{bmatrix} X = \begin{bmatrix} a_1 \\ A \end{bmatrix}$$

Now compute generators G_1 for the module of system of solutions of the homogeneous system $R_1X = 0$ and also test if $a_1 \in R_1$, if this is not the case the system is not solvable otherwise get W_1 such that $R_1W_1 = a_1$. Now compute by recursion the solution S of $MG_1X = A - MW_1$ such that $MG_1S = A - MW_1$. The solution to the system is then $W_1 + G_1S$ as

$$\begin{bmatrix} R_1 \\ M \end{bmatrix} (W_1 + G_1S) = \begin{bmatrix} R_1W_1 + R_1G_1S \\ MW_1 + MG_1S \end{bmatrix} = \begin{bmatrix} a_1 \\ A \end{bmatrix}$$

This algorithm has been implemented and proved correct as the function `solve_general`. Together with `solveMxN` this constitutes the only operations used as basis in the libraries of the HOMALG project [3].

3.3 Bézout domains are strongly discrete and coherent

The first example of coherent strongly discrete rings that we studied were Bézout domains with explicit divisibility. These are integral domains where every finitely generated ideal is principal (generated by a single element). The two main examples of Bézout domains are $\mathbb{Z}$ and $k[x]$ where k is a discrete field.

Bézout domains can also be characterized as rings with a GCD operation in which there is a function computing the elements of the Bézout identity. This means that given a and b one can compute x and y such that $xa+by$ is associate[4] to $gcd(a, b)$. Based on this it is straightforward to implement a function that given a finitely generated ideal $(a_1, \ldots, a_n)$ computes g such that $(a_1, \ldots, a_n) \subseteq (g)$ and $(g) \subseteq (a_1, \ldots, a_n)$ where this g is the greatest common divisor of all the a_i. To test if $x \in (a_1, \ldots, a_n)$ in a Bézout domain first compute a principal ideal (g) and then test if $g \mid x$ and if this is the case we we can construct the witness and otherwise we know that $x \notin (a_1, \ldots, a_n)$.

For showing that Bézout domains are coherent let I and J be two finitely generated ideals and compute principal ideals such that $I = (a)$ and $J = (b)$. Now it easy to prove that $I \cap J = (lcm(a, b))$, where $lcm(a, b)$ is the lowest common multiple of a and b which is computable in our setting as any Bézout ring is a GCD domain with explicit divisibility. Hence we have now proved that $\mathbb{Z}$ and $k[x]$ are both coherent and strongly discrete which means that we can solve arbitrary systems of equations over them.

4 Prüfer domains

Another class of rings that are coherent are *Prüfer domains*. These can be seen as non-Noetherian analogues of Dedekind domains and have many different characterizations, for instance does Bourbaki list fourteen of them [4]. The one we choose here is the one in [13] that says that a Prüfer domain is an integral domain where given any x and y there exists u, v and w such that

$$ux = vy$$

[4] a and b are *associates* if $a \mid b$ and $b \mid a$ or equivalently that there exists a unit $u \in R$ such that $a = bu$.

and

$$(1-u)y = wx$$

This is implemented in Coq by:

```
Record mixin_of (R : ringType) : Type := Mixin {
  prufer: R -> R -> (R * R * R)%type;
  _ : forall x y, let: (u,v,w) := prufer x y in
      u * x = v * y /\ (1 - u) * y = w * x
}.
```

As we require that Prüfer domains have explicit divisibility it is possible to prove that they are strongly discrete which in turn means that we can use the library of ideal theory developed for strongly discrete rings when proving that they are coherent. However it would be possible to prove that Prüfer domains are coherent without assuming explicit divisibility.

The most basic examples of Prüfer domains are Bézout domains (in particular $\mathbb{Z}$ and $k[x]$). However there are many other examples, for instance if R is a Bézout domain then the ring of elements integral over R is a Prüfer domain, this gives examples from algebraic geometry like $k[x,y]/(y^2+x^4-1)$ and algebraic number theory like $\mathbb{Z}[\sqrt{-5}]$.

4.1 Principal localization matrices and strong discreteness

The key algorithm in the proof that Prüfer domains with explicit divisibility are both strongly discrete and coherent is an algorithm computing a *principal localization matrix* of an ideal [8]. This means that given a finitely generated ideal $(x_1,\ldots,x_n)$ one can compute a $n \times n$ matrix $M = (a_{ij})$ such that:

$$\sum_{i=1}^{n} a_{ii} = 1$$

and

$$\forall ijl.\ a_{lj}x_i = a_{li}x_j$$

Note that there is no constraint $n \neq 0$ which means that the first of these is a bit problematic as if $n = 0$ the sum will be empty and hence 0. To remedy this we express the property formally as:

```
Definition P1 n (M : 'M[R]_n) :=
  \big[+%R/0]_(i: 'I_n) (M i i) = (0 < n)%:R.

Definition P2 n (I : 'rV[R]_n) (M : 'M[R]_n) :=
  forall (i j l : 'I_n), (M l j) * (I 0 i) = (M l i) * (I 0 j).

Definition isPLM n (I : 'rV[R]_n) (M : 'M[R]_n) := P1 M /\ P2 I M.
```

The first statement uses an implicit coercion from booleans to rings where `false` is coerced to `0` and `true` to `1`. The algorithm computing a principal localization matrix, `plm`, is quite involved so we have omitted it from this presentation, the interested reader should have a look in the formal development and at the proofs in [8] and [13]. We have proved that this algorithm satisfies the above specification:

```
Lemma plmP : forall n (I : 'rV[R]_n), isPLM I (plm I).
```

The reason that principal localization matrices are interesting is that they give a way to compute the *inverse of a finitely generated ideal* I, this is a finitely generated ideal J such that IJ (with ideal multiplication) is principal. In fact if $I = (x_1, \ldots, x_n)$ and $M = (a_{ij})$ is its principal localization matrix then every column of M is an inverse to I. This means that we can define an algorithm for computing the inverse of ideals in Prüfer domains:

```
Definition inv_id n : 'I_n -> 'rV[R]_n -> 'rV[R]_n := match n with
  | S p => fun (i : 'I_(1 + p)%N) (I : 'rV[R]_(1 + p)%N) =>
    (col i (plm I))^T
  | _ => fun _ _ => 0
  end.

Lemma inv_idP n (I : 'rV[R]_n) i :
  (inv_id i I *i I == (I 0 i)%:M)%IS.
```

Using this it is possible to prove that Prüfer domains with explicit divisibility are strongly discrete. To compute if $x \in I$ first compute J such that $IJ = (a)$. Now $x \in I$ iff $(x) \subseteq I$ iff $xJ \subseteq (a)$. This can be decided if we can decide when an element is divisible by a.

We have used this to prove that our implementation of Prüfer domains is strongly discrete which means that the theory about ideals developed for strongly discrete rings can be used when proving that they are coherent.

4.2 Coherence

The key property of ideals in Prüfer domains for computing the intersection is that finitely generated ideals I and J satisfy:

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

This means that we can devise an algorithm for computing generators for the intersection by first computing $(I + J)^{-1}$ such that $(I + J)^{-1}(I + J) = (a)$ and then get that

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

Note the use of division here. In fact it is possible to compute the intersection without assuming division but then the algorithm is more complicated. Using this the function for computing generators of the intersection is:

```
Definition pcap (n m : nat) (I : 'rV[R]_n) (J : 'rV[R]_m) :
  'rV[R]_(pcap_size I J).+1 := match find_nonzero (I +i J) with
  | Some i => let sIJ := I +i J in
              let a := sIJ 0 i in
              let acap := inv_id i sIJ *i I *i J in
              (0 : 'M_1) +i (\row_i (odflt 0 (acap 0 i %/? a)))
  | None => 0
  end.
```

Here `%/?` is the explicit divisibility function of `R`. The reason to add 0 as a generator of the ideal is simply to have the correct size as the formalized proof that R is coherent if $I \cap J$ is computable requires that $I \cap J$ is nonempty. Also note the function `find_nonzero` which finds the first nonzero element in a row-vector. This could have been implemented using the `pick` function for picking an element satisfying a decidable predicate which is provided for all SSReflect rings. But in order to simplify the translation to an efficient version of the algorithm we avoid using it here.

To prove that `pcap` really computes the intersection we need to first prove the main property used above for finding the algorithm computing $I \cap J$:

```
Lemma pcap_id (n m : nat) (I : 'rV[R]_n) (J : 'rV[R]_m) :
  ((I +i J) *i pcap I J == I *i J)%IS.
```

Using this it is possible to prove that `pcap` compute the intersection:

```
Lemma pcap_subidl m n (I: 'rV_m) (J: 'rV_n): (pcap I J <= I)%IS.

Lemma pcap_subidr m n (I: 'rV_m) (J: 'rV_n): (pcap I J <= J)%IS.

Lemma pcap_member m n x (I : 'rV[R]_m) (J : 'rV[R]_n) :
  member x I -> member x J -> member x (pcap I J).
```

Hence we have now proved that Prüfer domains with explicit divisibility are coherent strongly discrete rings so not only can we solve homogeneous systems over them but also any linear system of equations.

4.3 Examples of Prüfer domains

As mentioned before, any Bézout domain is a Prüfer domain. The proof of this is easy:

```
Definition bezout_calc (x y: R) : (R * R * R)%type :=
  let: (g,c,d,a,b) := egcdr x y in (d * b, a * d, b * c).

Lemma bezout_calcP (x y : R) : let: (u,v,w) := bezout_calc x y in
  u * x = v * y /\ (1 - u) * y = w * x.
```

Here `egcdr` is the extended Bézout algorithm where g is the gcd of x and y, $x = ag$, $y = bg$ and $ca + db = 1$.

We have not yet formalized the proof that $\mathbb{Z}[\sqrt{-5}]$ and $k[x,y]/(y^2 - 1 + x^4)$ are Prüfer domains but we have previously implemented this in Haskell [15].

5 Computations

In the paper algorithms are presented on structures using rich dependently typed datatypes which is convenient when proving properties but for computation this is not necessary. In fact it can be more efficient to implement the algorithms on simply typed datatypes instead, a good example is matrices: As explained in section 2 they are represented using finite functions from the indices (represented using ordinals) but this representation is not suitable for computation as finite functions are represented by their graph which has to be traversed linearly each time the function is evaluated.

In order to develop more efficient versions of the algorithms we use a previously developed library where matrices are represented using lists of lists and implement the algorithms on this representation. These algorithms are then linked to the inefficient versions using translation lemmas. The methodology that we follow is summarized in [7] as:

1. Implement an abstract version of the algorithm using SSReflect structures and use the libraries to prove properties about them. Here we can use the full power of dependent types when proving correctness.
2. Refine this algorithm into an efficient one using SSReflect structures and prove that it behaves like the abstract version.
3. Translate the SSReflect structures and the efficient algorithm to the low-level data types, ensuring that they will perform the same operations as their high-level counterparts.

So far we have only presented step 1. The second step involves giving more efficient algorithms, a good example of this is the algorithms on ideals. A simple optimization that can be made is to ensure that there are no zeroes as generators in the output of the ideal operations. The goal would then be to prove that the more efficient operations generate the same ideal as the original operation. Another example is `solve_int` that can be implemented without padding with zeroes, this would then be proved to produce a set of solution of the system and then be translated to a more efficient algorithm on list based matrices.

The final step corresponds to implementing "computable" counterparts of the structures that we presented so far based on simple types. For example, computable coherent rings are implemented as:

```
Record mixin_of (R : coherentRingType)
  (CR : cstronglyDiscreteType R) : Type := Mixin {
  csize_solve : nat -> seqmatrix CR -> nat;
  csolve_row : nat -> seqmatrix CR -> seqmatrix CR;
  _ : forall n (V : 'rV[R]_n),
    seqmx_of_mx CR (solve_row V) = csolve_row n (seqmx_of_mx _ V);
  _ : forall n (V: 'rV[R]_n),
    size_solve V = csize_solve n (seqmx_of_mx _ V)
}.
```

Here `seqmatrix` is the list based representation of matrices and `seqmx_of_mx` is the translation function from SSReflect matrices to `seqmatrix`. Using this more efficient versions of the algorithms presented above can be implemented simply by changing the functions on SSReflect matrices to functions on `seqmatrix`:

```
Fixpoint csolveMxN m n (M : seqmatrix CR) : seqmatrix CR :=
  match m with
    | S p => let u := usubseqmx 1 M in
             let d := dsubseqmx 1 M in
             let G := csolve_row n u in
             let k := csize_solve n u in
             let R := mulseqmx n k d G in
        mulseqmx k (csize_solveMxN p k R) G (csolveMxN p k R)
    | _ => seqmx1 CR n
  end.

Lemma csolveMxNE : forall m n (M : 'M[R]_(m,n)),
  seqmx_of_mx _ (solveMxN M) = csolveMxN m n (seqmx_of_mx _ M).
```

The lemma states that solving the system on SSReflect matrices and then translating is the same as first translating and then compute the solution using the list based algorithm. The proof of this is straightforward as all of the functions in the algorithm have translation lemmas, so it is done by expanding definitions and translating using already implemented translation lemmas.

This way we have implemented all of the above algorithms and instances to make some computations with $\mathbb{Z}$ using the algorithms for Bézout domains. First we can compute the generators of $(2) \cap (3, 6)$:

```
Eval vm_compute in (cbcap 1 2 [::[::2]] [::[::3; 6]]).
     = [:: [:: 6]]
```

Next we can test if $6 \in (2)$:

```
Eval vm_compute in (cmember 1%N 6 [::[:: 2]]).
     = Some [:: [:: 3]]
```

It is also possible to solve the homogeneous system:

$$\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

```
Eval vm_compute in (csolveMxN 2 2 [::[:: 1;2];[::2;4]]).
     = [:: [:: 2; 0];
           [:: -1; 0]]
```

and the inhomogeneous system:

$$\begin{bmatrix} 2 & 3 \\ 4 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 4 \\ 8 \end{bmatrix}$$

```
Eval vm_compute in (csolveGeneral 2 2 [::[:: 2; 3]; [:: 4; 6]]
                                      [::[:: 4];[:: 8]]).
     = Some [:: [:: -4];
                [:: 4]]
```

The system $2x = 1$ does not have a solution in $\mathbb{Z}$:

```
Eval vm_compute in (csolveGeneral 1 1 [::[:: 2]] [::[::1]]).
     = None
```

We can also do some computations on the algorithms for Prüfer domains using $\mathbb{Z}$:

```
Eval vm_compute in (cplm 3 [::[:: 2; 3; 5]]).
     = [:: [:: 8; 12; 20];
           [:: 12; 18; 30];
          [:: -10; -15; -25]]

Eval vm_compute in (cinv_id 2 0 [:: [:: 2; 3]]).
     = [:: [:: -2; 2]]
```

The first computation computes the principal localization matrix of $(2, 3, 5)$ and the second compute the inverse of the ideal $(2, 3)$.

6 Conclusions and Further Work

In this paper we have represented in type theory interesting and mathematically nontrivial results in constructive algebra. The algorithms based on coherent and strongly discrete rings have been refined to more efficient algorithms on simple datatypes, this way we get certified mathematical algorithms that are suitable for computation. Hence can this work be seen as an example that the methodology presented in [7] is applicable on more complicated structures as well.

In the future it would be interesting to prove that multivariate polynomial rings over discrete fields are coherent and strongly discrete. This would require a formalization of Gröbner bases and the Buchberger algorithm which has already been done in Coq [17, 19]. It would be interesting to reimplement this using SSReflect and compare the complexity of the formalizations.

It would also be interesting to use this work as a basis for a library of formalized computational homological algebra inspired by the homalg project. In fact `solveMxN` and `solveGeneral` are the only operations used as a basis in homalg [2]. This formalization would involve first proving that the category of finitely presented modules over coherent strongly discrete rings form an abelian category and then use this to implement further algorithms for doing homological computations.

In SSReflect all rings are equipped with a choice operator which can be used to pick an element satisfying a decidable predicate. This could have been used more in our development, for instance in the implementation of `pcap` to find a nonzero generator of an ideal. We believe that using this feature more

would lead to simpler formal proofs, but our experience is that the use of choice complicates the implementation of efficient algorithms. As we want to be able to compute with our algorithms inside Coq we decided to have slightly more complicated proofs but easier translation to efficient algorithms.

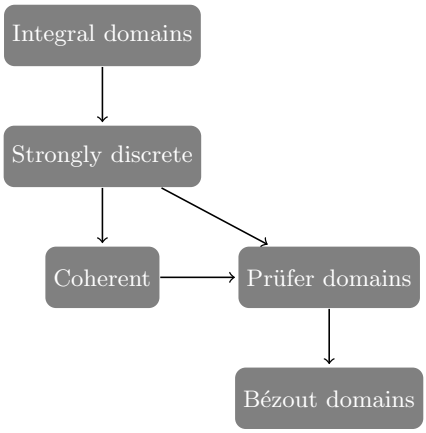

Fig. 1. The extension to the SSReflect hierarchy

In Fig. 1 the extension to the SSReflect hierarchy is presented. Integral domains are already present in the hierarchy and the extension consists of the other structures. The arrows represent that the target is an instance of the source. This presentation differ from standard presentations in constructive algebra [13, 14] as there is no need to assume that coherent rings and Prüfer domains are strongly discrete. The motivation behind this design choice is that it simplified the formalization and the examples that we are primarily interested in are all strongly discrete anyway. We actually started to formalize the notions without assuming that the rings are strongly discrete but this led to too complicated formal proofs as we could not use the library of ideal theory developed for strongly discrete rings. However, in constructive algebra ideal theory is usually developed without assuming decidable ideal membership, but in our experience, both the SSReflect library and tactics are best suited for theories with decidable functions.

A consequence of this is that it is difficult to formalize things in full generality, for instance are all rings assumed to be not only strongly discrete but also discrete. It would be more natural from the point of view of constructive mathematics to represent more general structures. However, while the use of SSReflect imposes some decidability conditions, we found that in this framework of decidable structures the notations and tactics provided are particularly elegant and well-suited.

References

1. J. Avigad. Methodology and metaphysics in the development of Dedekind's theory of ideals. The architecture of modern mathematics. Essays in history and philosophy. Oxford: Oxford University Press, 2006.
2. M. Barakat and M. Lange-Hegermann. An axiomatic setup for algorithmic homological algebra and an alternative approach to localization. *J. Algebra Appl.*, 10(2):269–293, 2011.
3. M. Barakat and D. Robertz. HOMALG – A Meta-Package for Homological Algebra. *J. Algebra Appl.*, 7(3):299–317, 2008.
4. N. Bourbaki. *Commutative algebra. Chapters 1–7.* Elements of Mathematics. Springer-Verlag, 1998.
5. Coq development team. The Coq Proof Assistant Reference Manual, version 8.3. Technical report, 2010.
6. D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties and Algorithms: An introduction to Computational Algebraic Geometry and Commutative Algebra.* Springer, 2006.
7. M. Dénès, A. Mörtberg, and V. Siles. A Refinement Based Approach to Computational Algebra in Coq. In *Proceedings Interactive Theorem Proving 2012*, volume 7406 of *Lectures Notes in Computer Science*, pages 83–98, 2012.
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. 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'09)*, volume 5674 of *LNCS*, pages 327–342, 2009.
10. G. Gonthier. Point-Free, Set-Free concrete linear algebra. In *Interactive Theorem Proving*, volume 6898 of *LNCS*, pages 103–118, 2011.
11. G. Gonthier and A. Mahboubi. A Small Scale Reflection Extension for the Coq system. Technical report, Microsoft Research INRIA, 2009.
12. H. Lombardi and H. Perdry. The Buchberger Algorithm as a Tool for Ideal Theory of Polynomial Rings in Constructive Mathematics. 1998.
13. H. Lombardi and C. Quitté. *Algèbre commutative, Méthodes constructives: Modules projectifs de type fini.* Calvage et Mounet, 2011.
14. R. Mines, F. Richman, and W. Ruitenburg. *A Course in Constructive Algebra.* Springer-Verlag, 1988.
15. A. Mörtberg. Constructive Algebra in Functional Programming and Type Theory. Master's thesis, Chalmers University of Technology, 2010.
16. H. Perdry and P. Schuster. Noetherian orders. *Mathematical. Structures in Comp. Sci.*, 21(1):111–124.
17. H. Persson. An Integrated Development of Buchberger's Algorithm in Coq. 2001.
18. A. Quadrat. The fractional representation approach to synthesis problems: An algebraic analysis viewpoint part ii: Internal stabilization. *SIAM J. Control Optim.*, 42(1):300–320, Jan. 2003.
19. L. Théry. A Certified Version of Buchberger's Algorithm. In *Proceedings of the 15th International Conference on Automated Deduction*, pages 349–364, London, UK, 1998. Springer-Verlag.