A formal proof of Sasaki-Murao algorithm

THIERRY COQUAND
and
ANDERS MÖRTBERG
and
VINCENT SILES
University of Gothenburg

The Sasaki-Murao algorithm computes the determinant of any square matrix over a commutative ring in polynomial time. The algorithm itself can be written as a short and simple functional program, but its correctness involves nontrivial mathematics. We here represent this algorithm in Type Theory with a new correctness proof, using the Coq proof assistant and the SSReflect extension.

Introduction

The goal of this note is to give a presentation of a formal proof of the Sasaki-Murao algorithm [SM82]. This is an elegant algorithm for computing the determinant of a square matrix over an arbitrary commutative ring in polynomial time. Usual presentations of this algorithm are quite complex, and rely on some Sylvester identities [AL04]. We believe that the proof we shall present, which was obtained by formalizing this algorithm in Type Theory (more precisely in the SSReflect [GM10] extension to Coq [Tea10]) is simpler. It does not rely on Sylvester identities and indeed gives a proof of some of them as corollaries. It provides also a good example of how one can use a library of formalized mathematical results to prove formally a computer algebra program. The whole formalization can be found at [MS12].

1. SASAKI-MURAO ALGORITHM

1.1 Matrices

For any $n \in \mathbb{N}$, we define $I_n = \{i \in \mathbb{N} \mid i < n\}$ (with $I_0 = \emptyset$). If R is a set, a $m \times n$ matrix of elements of the set R is a function $I_m \times I_n \to R$. We can also view any such matrix as a family of elements (m_{ij}) for $i \in I_m$ and $j \in I_n$.

If M is a $m \times n$ matrix, f a function of type $I_p \to I_m$ and g a function of type $I_q \to I_n$, we define the $p \times q$ sub-matrix[1] $M(f, g)$ by

$$M(f, g)(i, j) = M(f\ i, g\ j)$$

[1]In the usual definition of sub-matrix, only some lines and columns are removed, which would be enough for the following proofs. But our more general definition make the Coq formalization easier to achieve.

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

We often use the following operation on finite maps: if $f : I_p \to I_m$, we defined $f^+ : I_{1+p} \to I_{1+m}$ such that

$$\begin{aligned} f^+0 &= 0 \\ f^+(1+x) &= 1 + (f\ x) \end{aligned}$$

If R is a ring, let 1_n be the $n \times n$ identity matrix. We can also define addition and multiplication of matrices as usual. We can decompose a non-empty $m \times n$ matrix M in four components:

—the top-left element m_{00}, which is an element of R

—the top-right line vector $L = m_{01}, m_{02}, \ldots, m_{0(n-1)}$

—the bottom-left column vector $C = m_{10}, m_{20}, \ldots, m_{(m-1)0}$

—the bottom-right $(m-1) \times (n-1)$ matrix $N_{ij} = m_{(1+i,1+j)}$

$$\left(\begin{array}{c|c} m_{00} & L \\ \hline & \\ C & N \end{array}\right)$$

With this decomposition, we define the central operation of our algorithm, which defines a $(m-1) \times (n-1)$ matrix:

$$M' = m_{00}N - CL$$

This operation $M \longmapsto M'$ transforms a $m \times n$ matrix into a $(m-1) \times (n-1)$ matrix is crucial in the Sasaki-Murao algorithm. In the special case where $m = n = 2$ the matrix M' (of size 1×1) can be identified with the determinant of M.

Lemma 1.1.1. *For any $m \times n$ matrix M, for any map $f : I_p \to I_{m-1}$ and any map $g : I_q \to I_{n-1}$, we have the following identity:*

$$M'(f, g) = M(f^+, g^+)'$$

Proof. This lemma is easy to prove once one has realized two facts:

(1) Selecting a sub-matrix commutes with most of the basic operations about matrices. In particular, $(M - N)(f, g) = M(f, g) - N(f, g)$, $(aM)(f, g) = aM(f, g)$. For multiplication, we have $(MN)(f, g) = M(f, id)N(id, g)$ where id is the identity function.

(2) For any matrix M described as a block $(r\ L\ C\ N)$, we have that $M(f^+, g^+)$ is the block $(r\ L(id, g)\ C(f, id)\ N(f, g))$

From this two observations, we then have:

$$\begin{aligned} M'(f, g) &= (rN - Cl)(f, g) \\ &= rN(f, g) - C(f, id)L(id, g) \end{aligned}$$

and

$$M(f^+, g^+)' = rN(f, g) - C(f, id)L(id, g)$$

So, we can conclude that $M'(f, g) = M(f^+, g^+)'$. □

The block decomposition suggests the following possible representation of matrices in a functional language using the data type (where $[R]$ is the type of lists over the type R, using HASKELL notation):

$$Mat\ R\ ::=\ Empty \mid Mat\ R\ [R]\ [R]\ (Mat\ R)$$

So a matrix M is either the empty matrix or a compound matrix $Mat\ m\ L\ C\ N$. It is direct, using this representation, to define the operations of addition, multiplication on matrices, and the operation M' on non-empty matrices. From this representation, we can also compute other standard views of a $m \times n$ matrix, such as a list of lines $l_1, \dots, l_m$ or as a list of columns $c_1, \dots, c_n$.

If M is a square $n \times n$ matrix over a ring R we write $|M|$ the determinant of M. A *k-minor* of M is a determinant $|M(f, g)|$ for any strictly increasing maps $f : I_k \to I_n$ and $g : I_k \to I_n$. A leading principal minor of M is a determinant $|M(f, f)|$ where f is the inclusion of I_k into I_n.

1.2 The algorithm

We present Sasaki-Murao algorithm using functional programming notations. This algorithm computes in polynomial time, not only the determinant of a matrix, but also its characteristic polynomial. We assume that we have a representation of polynomials over the ring R and that we are given an operation p/q on $R[X]$ which should be the quotient of p by q when q is a *monic* polynomial. This operation is directly extended to an operation M/q of type $Mat\ R[X] \to R[X] \to Mat\ R[X]$. We define then an auxiliary function $\phi\ a\ M$ of type $R[X] \to Mat\ R[X] \to R[X]$. The definition is:

$$\begin{aligned} \phi\ a\ Empty &= a \\ \phi\ a\ (Mat\ m\ L\ C\ N) &= \phi\ m\ ((mN - CL)/a) \end{aligned}$$

From now on, we assume R to be a commutative ring.

The proof relies on the notion of *regular* element of a ring: a *regular* element of R is an element a such that $ax = 0$ implies $x = 0$. An alternative (and equivalent) definition is to say that multiplication by a is injective or that a can be cancelled from $ax = ay$ giving $x = y$.

THEOREM 1.2.1. *Let P be a square matrix of elements of $R[X]$. If all leading principal minors of P are monic, then $\phi\ 1\ P$ is the determinant of P. In particular, if $P = X1_n - M$ for some square matrix M of elements in R, $\phi\ 1\ P$ is the* characteristic polynomial *of M.*

This gives a remarkably simple (and polynomial time [AL04]) algorithm for computing the characteristic polynomial $\chi_M(X)$ of a matrix M. The determinant of M is then $\chi_{-M}(0)$.

2. CORRECTNESS PROOF

We first start to prove some auxiliary lemmas:

LEMMA 2.0.2. *If M is a $n \times n$ matrix, $n > 0$ then we have*

$$m_{00}^{n-1}|M| = m_{00}|M'|.$$

In particular, if m_{00} is regular and $n > 1$, then we have

$$m_{00}^{n-2}|M| = |M'|.$$

Proof. Let us view the matrix M as a list of lines $l_0, \ldots, l_{n-1}$ and let N_1 be the matrix $l_0,\ m_{00}l_1, \ldots,\ m_{00}l_{n-1}$. The matrix N_1 is computed from M by multiplying all of its lines (except the first one) by m_{00}. By the properties of the determinant, we can assert that $|N_1| = m_{00}^{n-1}|M|$.

Let N_2 be the matrix $l_0,\ m_{00}l_1 - m_{10}l_0,\ \ldots,\ m_{00}l_{n-1} - m_{(n-1)0}l_0$. The matrix N_2 is computed from N_1 by subtracting a multiple of l_0 from every line except l_0:

$$m_{00}l_{1+i} \leftarrow m_{00}l_{1+i} - m_{(1+i)0}l_0.$$

By the properties of the determinant, we can assert that $|N_2| = |N_1|$.

Using the definition of the previous section, we can also view the matrix M as the block matrix $(m_{00}\ L\ C\ N)$, and then the matrix N_2 is the block matrix $(m_{00}\ L\ 0\ M')$. Hence we have $|N_2| = m_{00}|M'|$. From this equality, we can now prove that

$$m_{00}^{n-1}|M| = |N_1| = |N_2| = m_{00}|M'|.$$

If m_{00} is regular and $n > 2$, this equality simplifies to $m_{00}^{n-2}|M| = |M'|$ □

Corollary 2.0.3. *Let M be a $n \times n$ matrix with $n > 0$. If f and g are two strictly increasing maps from I_k to I_{n-1}, then $|M'(f,g)| = m_{00}^{k-1}|M(f^+,g^+)|$ if m_{00} is regular.*

Proof. Using Lemma 1.1.1, we know that $M'(f,g) = M(f^+,g^+)'$, so this corollary follows from Lemma 2.0.2. □

Let a be an element of R and M a $n \times n$ matrix. We say that a and M are related if and only if

(1) a is regular
(2) a^k divides each $k+1$ minor of M
(3) each principal minor of M is regular

Lemma 2.0.4. *Let a be a regular element of R and M a $n \times n$ matrix, with $n > 0$. If a and M are related, then a divides every element of M'. Furthermore if $aN = M'$ then m_{00} and N are related and if $n > 1$*

$$m_{00}^{n-2}|M| = a^{n-1}|N|$$

Proof. Let us start by stating two simple facts: m_{00} is a 1×1 principal minor of M and for all i, j, ${M'}_{ij}$ is a 2×2 minor of M. Therefore, since a and M are related, m_{00} is regular and a divides all the ${M'}_{ij}$ (by having $k = 1$), so a divides M'.

Let us write $M' = aN$, we now need to show that m_{00} and N are related, and if $n > 1$,

$$m_{00}^{n-2}|M| = a^{n-1}|N|$$

Let us consider two strictly increasing maps $f : I_k \to I_{n-1},\ g : I_l \to I_{n-1}$, we have $|M'(f,g)| = u^{k-1}|M(f^+,g^+)|$ by Corollary 2.0.3. From the definition of related, we also know that a^k divides $|M(f^+, g^+)|$. Since $M' = aN$ we have $|M'(f,g)| = a^k|N(f,g)|$. If we write $ba^k = |M(f^+, g^+)|$, we have that $ba^k u^{k-1} = a^k|N(f,g)|$. Since a is regular, this equality implies $bu^{k-1} = |N(f,g)|$, and we see that u^{k-1} divides each k minor of N. This also shows that $|N(f,g)|$ is regular whenever $|M(f^+, g^+)|$ is regular. In particular, each principal minor of N is regular. Finally, since $|M'| = a^{n-1}|N|$ we have $m_{00}^{n-2}|M| = a^{n-1}|N|$ by Lemma 2.0.2. □

Since any monic polynomial is also a regular element of the ring of polynomials, Theorem 1.2.1 follows directly from Lemma 2.0.4 by performing a straightforward induction over the size n. In the case where P is $X1_n - M$ for some square matrix M over R, we can use the fact that any principal minor of $X1_n - M$ is the characteristic polynomial of a smaller matrix, and thus is always monic. In the end, the second part of the conclusion follows directly for the first: $\phi\ 1\ (X1_n - M) = \chi_M(X)$.

Now, we explain how to derive some Sylvester equalities from Lemma 2.0.4. If we look at the computation of $\phi\ 1\ P$ we get a chain of equalities

$$\phi\ 1\ P = \phi\ u_1\ P_1 = \phi\ u_2\ P_2 = \cdots = \phi\ u_{n-1}\ P_{n-1}$$

and we have that u_k is the $k : th$ leading principal minor of P, while P_k is the $(n-k) \times (n-k)$ matrix

$$P_k(i,j) = |P(f_{i,k}, f_{j,k})|$$

where $f_{i,k}(l) = l$ if $l < k$ and $f_{i,k}(k) = i + k$. (We have $P_0 = P$.) Lemma 2.0.4 shows that we have for $k < l$

$$|P_k|u_l^{n-l-1} = |P_l|u_k^{n-k-1}$$

This is a Sylvester equality for the matrix $P = X1_n - M$. If we evaluate this identity at $X = 0$, we get the corresponding Sylvester equality for the M matrix over an arbitrary commutative ring.

3. REPRESENTATION IN TYPE THEORY

The original functional program is easily described in Type Theory, since it is an extension of simply typed λ-calculus:

```
Variable R : ringType.
Variable CR : cringType R.

Definition cpoly := seq CR. (* polynomials are lists *)

Inductive Matrix : Type :=
 | eM (* the empty matrix *)
 | cM of CR & seq CR & seq CR & Matrix.

Definition ex_dvd_step d (M : Matrix cpoly) :=
 mapM (fun x => divp_seq x d) M.

(* main "\phi" function of the algorithm *)
Fixpoint exBareiss_rec (n : nat) (g : cpoly) (M : Matrix cpoly)
```

```
  {struct n} : cpoly := match n,M with
    | _,eM => g
    | 0,_ => g
    | S p, cM a l c M =>
      let M' := subM (multEM a M) (mults c l) in
      let M'' := ex_dvd_step g M' in
        exBareiss_rec p a M''
end.

(* This function computes det M for a matrix of polynomials *)
Definition exBareiss (n : nat) (M : Matrix cpoly) : cpoly :=
  exBareiss_rec n 1 M.

(* Applied to xI - M, this gives another definition of the
   characteristic polynomial *)
Definition ex_char_poly_alt (n : nat) (M : Matrix CR) :=
  exBareiss n (ex_char_poly_mx n M).

(* The determinant is the constant part of the char poly *)
Definition ex_bdet (n : nat) (M : Matrix CR) :=
  nth (zero CR) (ex_char_poly_alt n (oppM M)) 0.
```

The `Matrix` type allows to define “ill-shaped” matrices since there are no links between the size of the blocks. When proving correctness of the algorithm, we have to be careful and only consider *valid* inputs.

As we previously said, this is a simple functional program, but its correctness involves nontrivial mathematics. We choose to use the SSReflect library to formalize the proof because it already contains many results that we need. The main scheme is to translate this program using SSReflect data types, prove its correctness and then prove that both implementations output the same results on valid inputs following the methodology presented in [DMS12b].

First, here is a description of the SSReflect data types we need:

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

Variable R : ringType.

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

```
(* {poly R} *)
Record polynomial := Polynomial {
   polyseq :> seq R;
   _ : last 1 polyseq != 0
}.
```

Here dependent types are used to express well-formedness. For example, polynomials are encoded as lists (of their coefficients) with a proof that the last one is not zero. With this restriction, we are sure that one list exactly represent a unique polynomial. Matrices are described as finite functions over the finite sets of indexes.

With this definition, it is easy to define the sub-matrix $M(f, g)$ along with minors:

```
(* M(f,g) *)
Definition submatrix m n p q (f : 'I_p -> 'I_m) (g : 'I_q -> 'I_n)
  (A : 'M[R]_(m,n)) : 'M[R]_(p,q) :=
  \matrix_(i < p, j < q) A (f i) (g j).

Definition minor m n p (f : 'I_p -> 'I_m) (g : 'I_p -> 'I_n)
  (A : 'M[R]_(m,n)) : R := \det (submatrix f g A).
```

Using SSReflect notations and types, we can now write the steps of the functional program (where rdivp is the pseudo-division operation [Knu81] of $R[X]$):

```
Definition dvd_step (m n : nat) (d : {poly R})
   (M: 'M[{poly R}]_(m,n)) : 'M[{poly R}]_(m,n) :=
   map_mx (fun x => rdivp x d) M.

(* main "\phi" function of the algorithm *)
Fixpoint Bareiss_rec m a : 'M[{poly R}]_(1 + m) -> {poly R} :=
  match m return 'M[_]_(1 + m) -> {poly R} with
    | S p => fun (M : 'M[_]_(1 + _)) =>
      let d := M 0 0 in (* up left *)
      let l := ursubmx M in (* up right *)
      let c := dlsubmx M in (* down left *)
      let N := drsubmx M in (* down right *)
      let M' := d *: N - c *m l in
      let M'' := dvd_step a M' in
        Bareiss_rec d M''
    | _ => fun M => M 0 0
  end.

Definition Bareiss (n : nat) (M : 'M[{poly R}]_(1 + n)) :=
  Bareiss_rec 1 M.

Definition char_poly_alt n (M : 'M[R]_(1 + n)) :=
  Bareiss (char_poly_mx M).

Definition bdet n (M : 'M[R]_(1 + n)) :=
 (char_poly_alt (-M))`_0.
```

The main achievement of this paper is the formalized proof of correctness (detailed in the previous section) of this program:

```
Lemma BareissE : forall n (M : 'M[{poly R}]_(1 + n)),
 (forall p (h h' : p.+1 <= 1 + n), monic (pminor h h' M)) ->
  Bareiss M = \det M.

Lemma char_poly_altE : forall n (M : 'M[R]_(1 + n)),
  char_poly_alt M = char_poly M.

Lemma bdetE n (M : 'M[R]_(1 + n)) : bdet M = \det M.
```

Now we want to prove that the original functional program is correct. Both implementations are very close to each other, so to prove the correctness of the `ex_bdet` program, we just have to show that it computes the same result than `bdet` on similar (valid) inputs. This is one of the advantages of formalizing correctness of program in Type Theory: one can express the program *and* its correctness in the same language!

```
Lemma exBareiss_recE :
 forall n (g : {poly R}) (M : 'M[{poly R}]_(1 + n)),
  trans (Bareiss_rec g M) = exBareiss_rec (1+n) (trans g) (trans M).

Lemma exBareissE : forall n (M : 'M[{poly R}]_(1 + n)),
  trans (Bareiss M) = exBareiss (1 + n) (trans M).

Lemma ex_char_poly_mxE : forall n (M : 'M[R]_n),
  trans (char_poly_mx M) = ex_char_poly_mx n (trans M).

Lemma ex_detE : forall n (M : 'M[R]_(1 + n)),
  trans (bdet M) = ex_bdet (1 + n) (trans M).
```

To link the two implementations, we rely on CoqEAL [DMS12a], a library built on top of SSReflect libraries that we are currently developing. It allows to mirror the main algebraic hierarchy of SSReflect with more concrete data types (e.g. here we mirror the matrix type `'M[R]_(m,n)` by the concrete type `Matrix CR`, assuming `CR` mirrors `R`) in order to prove the correctness of functional programs using the whole power of SSReflect libraries.

This process is done in the same manner as in [GGMR09] using the *canonical structure* mechanism of Coq to overload the `trans` function, which can then be uniformly called on elements of the ring, polynomials or matrices. This function links the SSReflect structures to the one we use for the functional program description, ensuring that the correctness properties are translated the program that we actually run in practice.

We can easily prove that translating a SSReflect matrix into a `Matrix` always lead to a "valid" `Matrix`, and there is a bijection between SSReflect matrices and "valid" matrices, so we are sure that our program computes the correct determinant for all valid inputs.

In the end, the correctness of `ex_bdet` is proved using the lemmas `bdetE` and `ex_bdetE`, stating that for any valid input, `ex_bdet` outputs the determinant of the matrix:

```
Lemma ex_bdet_correct (n : nat) (M : 'M[R]_(1 + n)) :
  trans (\det M) = ex_bdet (1 + n) (trans M).
```

4. CONCLUSIONS AND BENCHMARKS

In this paper the formalization of a polynomial time algorithm for computing the determinant over any commutative ring has been presented. In order to be able to do the formalization in a convenient way a new correctness proof more suitable for formalization has been found. The formalized algorithm has also been refined to a more efficient version on simple types, following the methodology of [DMS12b]. This work can be seen as an indication that this methodology works well on more complicated examples involving many different computable structures, in this case matrices of polynomials.

We have tested the implementation on randomly generated matrices with $\mathbb{Z}$ coefficients:

```
(* Random 3x3 matrix *)
Definition M3 :=
  cM 10%Z [:: (-42%Z); 13%Z] [:: (-34)%Z; 77%Z]
     (cM 15%Z [:: 76%Z] [:: 98%Z]
         (cM 49%Z [::] [::] (@eM _ _))).

Time Eval vm_compute in ex_bdet 3 M3.
     = (-441217)%Z
  Finished transaction in 0. secs (0.006667u,0.s)

Definition M10 := (* Random 10x10 matrix *).

Time Eval vm_compute in ex_bdet 10 M10.
     = (-406683286186860)%Z
  Finished transaction in 1. secs (1.316581u,0.s)

Definition M20 := (* Random 20x20 matrix *).

Time Eval vm_compute in ex_bdet 20 M20.
     = 75728050107481969127694371861%Z
  Finished transaction in 63. secs (62.825904u,0.016666s)
```

This indicates that the implementation is indeed quite efficient, we believe that the slow-down of the last computation is due to the fact that the size of the determinant is so large and that the intermediate arithmetic operations has to be done on very big numbers. We have verified this by extracting the function to Haskell and the determinant of the 20×20 matrix can then be computed in `0.273` seconds. The main reasons for this is that the Haskell program has been compiled and have an efficient implementation of arithmetic operations for large numbers.

The main contribution of this paper is to express Sasaki-Murao algorithm as a functional program and show that, in this form, it admits a quite simple proof of correctness, arguably simpler than proofs available in the literature [AL04] (for instance, we don't have to rely on Sylvester identities). For doing this proof formally, it is furthermore convenient and elegant to have a single formalism (here type theory) in which we can write the algorithm, its specification and its proof of correctness.

References

[AL04] J. Abdeljaoued and H. Lombardi. *Méthodes matricielles - Introduction à la complexité algébrique.* Springer, 2004.

[DMS12a] M. Dénès, A. Mörtberg, and V. Siles. CoqEAL, the Coq Effective Algebra Library, 2012. `http://www-sop.inria.fr/members/Maxime.Denes/coqeal`.

[DMS12b] Maxime Dénès, Anders Mörtberg, and Vincent Siles. A refinement based approach to computational algebra in Coq. In *Interactive Theorem Proving*, volume 7406 of *LNCS*, pages 83–98, 2012.

[GGMR09] 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.

[GM10] Georges Gonthier and Assia Mahboubi. An introduction to small scale reflection in Coq. *Journal of Formalized Reasoning*, 3(2):95–152, 2010.

[Knu81] Donald E. Knuth. *The art of computer programming, volume 2: seminumerical algorithms.* Addison-Wesley, 1981.

[MS12] A. Mörtberg and V. Siles. Formalization of Bareiss/Sasaki-Murao algorithm, 2012. `http://www.cse.chalmers.se/~siles/coq/formalisation.html`.

[SM82] Tateaki Sasaki and Hirokazu Murao. Efficient Gaussian Elimination Method for Symbolic Determinants and Linear Systems. *ACM Trans. Math. Softw.*, 8(3):277–289, September 1982.

[Tea10] Coq Development Team. The Coq Proof Assistant Reference Manual, version 8.3. Technical report, 2010.