Constructive Algebra in Type Theory

Anders Mörtberg – mortberg@chalmers.se

September 4, 2012

$$\begin{pmatrix} 0 & 2 \\ 1 & 0 \end{pmatrix}^T \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 2 \\ 1 & 0 \end{pmatrix}^T \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$$

Matlab 2009b

```
A=[ 0 2 ; 1 0 ];
b=[ 2 ; 0 ];
A'\b
```

Matlab 2009b

```
A=[ 0 2 ; 1 0 ];
b=[ 2 ; 0 ];
A'\b

ans =
0
1
```

$$\begin{pmatrix} 0 \\ 2 \end{pmatrix} \neq \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Matlab 2009b

- *"The bug seems to be new in release R2009b and applies to Linux and Windows 32 and 64 bit versions."* `http://www.netlib.org/na-digest-html/09/v09n48.html`

- *A serious bug in Matlab2009b?* `http://www.walkingrandomly.com/?p=1964`

Solutions?

- Testing
  - User/Usability
  - Automated, for example using QuickCheck
- Formal specification and verification

Solutions?

- Testing
  - User/Usability
  - Automated, for example using QuickCheck
- **Formal specification and verification**

ForMath project

- ForMath – Formalization of Mathematics[1]
- Linear and commutative algebra
  - Algebraic structures, vector spaces, matrices, polynomials...
- Algebraic topology
  - Homology groups, homological algebra, applications in image analysis...
- Real number computation and numerical analysis

[1] `http://wiki.portal.chalmers.se/cse/pmwiki.php/ForMath/`

How?

- Coq – Interactive proof assistant based on intuitionistic type theory
- SSReflect – Small Scale Reflection extension

SSREFLECT?

- Fine grained automation
  - Concise proof style
- Examples: Four color theorem, Decidability of ACF, Feit-Thompson theorem (part of the classification of finite simple groups)...
- Large library: $\sim$ 14000 proofs and 4000 definitions
  - Polynomials, matrices, big operators, algebraic hierarchy, etc...

Paper 1: A Refinement-Based Approach to Computational Algebra in Coq

(jww. Maxime Dénès and Vincent Siles)

Introduction

- Goal: Verification and implementation of efficient computer algebra algorithms
- Problem: Tension between proof-oriented definitions and efficient implementations

Refinement-Based approach

In the paper we:

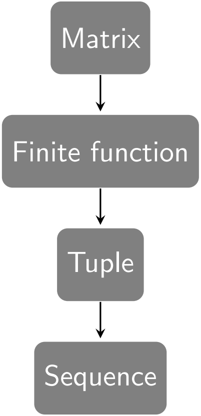
- Describe a *methodology* to verify and implement efficient algorithms
- Apply it to a few examples:
  - Strassen's fast matrix product
  - Matrix rank computation
  - Karatsuba's algorithm for polynomial multiplication
  - Multivariate GCD
- Abstract to make our approach composable

SSReflect matrices

```
Inductive matrix R m n :=
  Matrix of {ffun 'I_m * 'I_n -> R}.
```

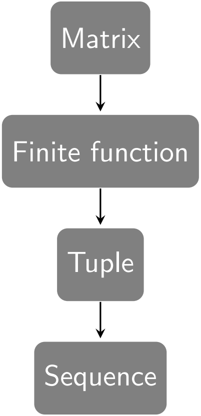
SSReflect matrices

```
Inductive matrix R m n :=
  Matrix of {ffun 'I_m * 'I_n -> R}.
```



SSReflect matrices

```
Inductive matrix R m n :=
  Matrix of {ffun 'I_m * 'I_n -> R}.
```



- Fine-grained architecture
- Proof-oriented design
- Not suited for computation

Objective

- Define concrete and executable representations and operations on matrices, using a relaxed datatype (unsized lists)
- Devise a way to link them with the theory in SSReflect
- Still be able to use convenient tools to reason about the algorithms we implement

Methodology

```
'M[R]_(m,n)
```

```
seq (seq R)
```

Methodology

$$
\begin{tikzcd}
\texttt{'M[R]\_(m,n)} \arrow[d, "\texttt{seqmx\_of\_mx}"] \\
\texttt{seq (seq R)}
\end{tikzcd}
$$

Methodology

`'M[R]_(m,n)`

$+$, $\times$, $\cdot^T$, ...

$\downarrow$ `seqmx_of_mx`

`seq (seq R)`

`addseqmx`,

`mulseqmx`,

`trseqmx`, ...

List-based representation of matrices

```
Variable R : ringType.
Definition seqmatrix := seq (seq R).
```

List-based representation of matrices

```
Variable R : ringType.
Definition seqmatrix := seq (seq R).

Definition seqmx_of_mx (M : 'M_(m,n)) : seqmatrix :=
  map (fun i => map (fun j => M i j) (enum 'I_n))
     (enum 'I_m).
```

List-based representation of matrices

```
Variable R : ringType.
Definition seqmatrix := seq (seq R).

Definition seqmx_of_mx (M : 'M_(m,n)) : seqmatrix :=
  map (fun i => map (fun j => M i j) (enum 'I_n))
    (enum 'I_m).

Lemma seqmx_eqP (M N : 'M_(m,n)) :
  reflect (M = N) (seqmx_of_mx M == seqmx_of_mx N).
```

List-based representation of matrices

```
Variable R : ringType.
Definition seqmatrix := seq (seq R).

Definition seqmx_of_mx (M : 'M_(m,n)) : seqmatrix :=
  map (fun i => map (fun j => M i j) (enum 'I_n))
     (enum 'I_m).

Lemma seqmx_eqP (M N : 'M_(m,n)) :
  reflect (M = N) (seqmx_of_mx M == seqmx_of_mx N).

Definition addseqmx (M N : seqmatrix) : seqmatrix :=
  zipwith (zipwith (fun x y => x + y)) M N.

Lemma addseqmxE (M N : 'M[R]_(m,n)) :
  seqmx_of_mx (M + N) =
    addseqmx (seqmx_of_mx M) (seqmx_of_mx N).
```

Methodology

'M[R]_(m,n)

$+, \times, \cdot^{T}, \ldots$

$\Big\downarrow$ seqmx_of_mx

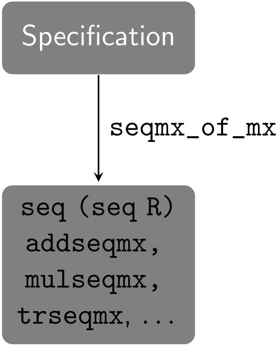
seq (seq R)

addseqmx,

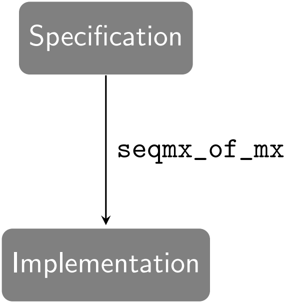
mulseqmx,

trseqmx, ...

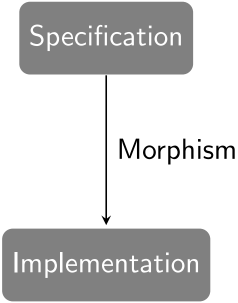
Methodology



Methodology



Methodology



Polynomials

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

Morphism: `polyseq`

Karatsuba

- Naive product:

$$(aX^k + b)(cX^k + d) = acX^{2k} + (ad + bc)X^k + bd$$

- Karatsuba's algorithm:

$$(aX^k+b)(cX^k+d) = acX^{2k}+((a + b)(c + d)-ac-bd)X^k+bd$$

Karatsuba

- Naive product:

$$(aX^k + b)(cX^k + d) = acX^{2k} + (ad + bc)X^k + bd$$

- Karatsuba's algorithm:

$$(aX^k+b)(cX^k+d) = acX^{2k}+((a+b)(c+d)-ac-bd)X^k+bd$$

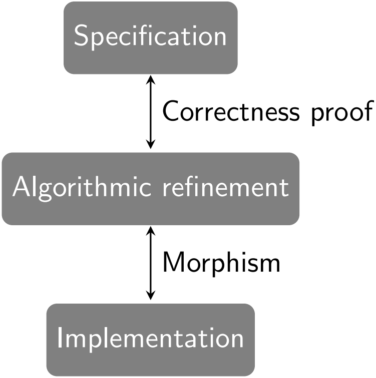
- Complexity: $\mathcal{O}(n^{\log_2 3})$

Refinements

Specification

Implementation

Refinements



Karatsuba

```
Fixpoint karatsuba_rec n p q := match n with
 | n'.+1 => let m := minn (size p)./2 (size q)./2 in
            let (a,b) := splitp m p in
            let (c,d) := splitp m q in
            let ac := karatsuba_rec n' a c in
            let bd := karatsuba_rec n' b d in
   ...
   ac * 'X^(2 * m) + (abcd - ac - bd) * 'X^m + bd

Definition karatsuba p q := let (p1,q1) := ... in
  karatsuba_rec (size p1) p1 q1.


Lemma karatsubaE p q : karatsuba p q = p * q.
```

Karatsuba

```
Fixpoint karatsuba_rec_seq n p q := match n with
 | n'.+1 => let m := minn (size p)./2 (size q)./2 in
            let (a,b) := splitp_seq m p in
            let (c,d) := splitp_seq m q in
            let ac := karatsuba_rec_seq n' a c in
            let bd := karatsuba_rec_seq n' b d in
   ...
   add_seq (add_seq (shift (2 * m) ac) (shift m
     (sub_seq (sub_seq abcd ac) bd))) bd

Definition karatsuba_seq p q := let (p1,q1) := ... in
  karatsuba_rec_seq (size p1) p1 q1.

Lemma karatsubaE p q : karatsuba p q = p * q.
```

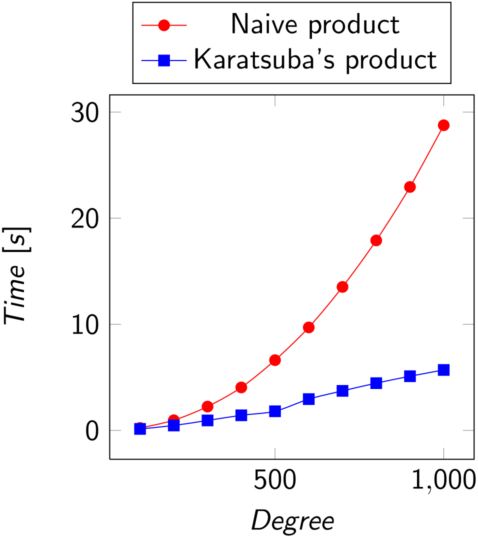
Karatsuba

```
Fixpoint karatsuba_rec_seq n p q := match n with
 | n'.+1 => let m := minn (size p)./2 (size q)./2 in
            let (a,b) := splitp_seq m p in
            let (c,d) := splitp_seq m q in
            let ac := karatsuba_rec_seq n' a c in
            let bd := karatsuba_rec_seq n' b d in
   ...
   add_seq (add_seq (shift (2 * m) ac) (shift m
     (sub_seq (sub_seq abcd ac) bd))) bd

Definition karatsuba_seq p q := let (p1,q1) := ... in
  karatsuba_rec_seq (size p1) p1 q1.

Lemma karatsuba_seqE :
  {morph polyseq : p q / karatsuba p q >->
                         karatsuba_seq p q}.
```

Benchmarks



Naive product
Karatsuba's product
30
20
10
0
500
1,000
Time [s]
Degree

Conclusion

- Software engineering methodology to organize execution-oriented and properties-oriented definitions
- The usual pitfall: trying to prove correctness on a concrete implementation
- Concise correctness proofs
- Problems of realistic sizes can already be treated (product of 4096×4096 dense matrices)

Paper 2:
A Formal Proof of Sasaki-Murao Algorithm

(jww. Thierry Coquand and Vincent Siles)

Introduction

- Determinants: Basic operation in linear algebra
- Goal: Efficient algorithm for computing determinant over any commutative ring (not necessarily with division)

Naive algorithm

- Laplace expansion: Express the determinant in terms of determinants of submatrices

$$
\begin{aligned}
\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} &= a \begin{vmatrix} e & f \\ h & i \end{vmatrix} - b \begin{vmatrix} d & f \\ g & i \end{vmatrix} + c \begin{vmatrix} d & e \\ g & h \end{vmatrix} \\
&= a(ei - hf) - b(di - fg) + c(dg - eg) \\
&= aei - ahf - bdi + bfg + cdg - ceg
\end{aligned}
$$

- Not suitable for computation (factorial complexity)

Gaussian elimination

- Gaussian elimination: Convert the matrix into triangular form using elementary operations
- Polynomial time algorithm
- Relies on division – Is there a polynomial time algorithm not using division?

Division free Gaussian algorithm

- We want an operation doing:

$$\begin{pmatrix} a & l \\ c & M \end{pmatrix} \rightsquigarrow \begin{pmatrix} a & l \\ 0 & M' \end{pmatrix}$$

Division free Gaussian algorithm

- We want an operation doing:

$$\begin{pmatrix} a & l \\ c & M \end{pmatrix} \rightsquigarrow \begin{pmatrix} a & l \\ 0 & M' \end{pmatrix}$$

- We can do:

$$\begin{pmatrix} a & l \\ c & M \end{pmatrix} \rightsquigarrow \begin{pmatrix} a & l \\ ac & aM \end{pmatrix} \rightsquigarrow \begin{pmatrix} a & l \\ 0 & aM - cl \end{pmatrix}$$

- Problems:
  - Computes $a^n \cdot det$
  - $a = 0$?

Sasaki-Murao algorithm

- Apply the algorithm to $x \cdot Id - M$
- Compute on $R[x]$ with pseudo-division
- Put $x = 0$ in the result

Sasaki-Murao algorithm

```
dvd_step :: R[x] -> Matrix R[x] -> Matrix R[x]
dvd_step g M = mapM (\x -> g | x) M

sasaki_rec :: R[x] -> Matrix R[x] -> R[x]
sasaki_rec g M = case M of
  Empty -> g
  Cons a l c M ->
     let M' = a * M - c * l in
     sasaki_rec a (dvd_step g M')

sasaki :: Matrix R -> R[x]
sasaki M = sasaki_rec 1 (x * Id - M)
```

Sasaki-Murao algorithm

- Very simple algorithm!
- No problem with 0 (x along the diagonal)
- Get characteristic polynomial for free
- Works for any commutative ring
- Complicated correctness proof – relies on Sylvester identities
- Paper: Simpler proof and Coq formalization

Paper 3: Coherent and Strongly Discrete Rings in Type Theory

(jww. Thierry Coquand and Vincent Siles)

Introduction

- Goal: Algorithms for solving (in)homogeneous systems of equations over commutative rings
- Application: Basis for library on homological algebra – HOMALG project[2]

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

Coherent rings

For every row matrix $M \in R^{1\times m}$ there exists $L \in R^{m\times n}$ such that

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

L generate the module of solutions for $MX = 0$

Coherent rings

Theorem: Possible to solve $MX = 0$ where $M \in R^{m \times n}$

Constructive proof $\Rightarrow$ algorithm computing generators of solutions

Coherent rings

Examples of coherent rings:

- Fields – Gaussian elimination
- Bézout domains – $\mathbb{Z}$, $\mathbb{Q}[x]$, ...
- Prüfer domains – $\mathbb{Z}[\sqrt{-5}]$, $\mathbb{Q}[x, y]/(y^2 - 1 + x^4)$, ...
- Polynomial rings – $k[x_1, \ldots, x_n]$ via Gröbner bases
- ...

Coherent rings

Examples of coherent rings:

- Fields – Gaussian elimination
- **Bézout domains** – $\mathbb{Z}$, $\mathbb{Q}[x]$, $\ldots$
- **Prüfer domains** – $\mathbb{Z}[\sqrt{-5}]$, $\mathbb{Q}[x, y]/(y^2 - 1 + x^4)$, $\ldots$
- Polynomial rings – $k[x_1, \ldots, x_n]$ via Gröbner bases
- ...

Strongly discrete rings

There exists an algorithm testing if $x \in I$ for finitely generated ideal I and if this is the case produce a witness.

Strongly discrete rings

There exists an algorithm testing if $x \in I$ for finitely generated ideal I and if this is the case produce a witness.

That is, if $I = (x_1, \ldots, x_n)$ compute $w_1, \ldots, w_n$ such that

$$x = x_1 w_1 + \cdots + x_n w_n$$

Coherent strongly discrete rings

Theorem: For coherent strongly discrete rings it is possible to solve inhomogeneous systems of equations

$$MX = A$$

Formalization

- Formalized using the refinement-based approach
- Structures instantiated with Bézout domains and Prüfer domains – Get certified algorithms for solving systems of equations over $\mathbb{Z}$, $\mathbb{Q}[x]$, $\mathbb{Z}[\sqrt{-5}]$, ...

Paper 4:
Towards a Certified Computation of Homology Groups for Digital Images

(jww. Jónathan Heras, Maxime Dénès, Gadea Mata, María Poza and Vincent Siles)

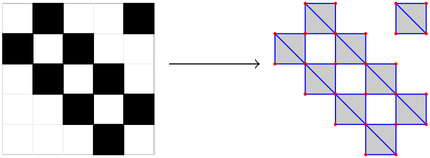
Homology

- Important tool in algebraic topology
- Algebraic method for computing topological properties (number of connected components, holes, cavities...)

Homology groups from digital images

$$
\begin{tikzcd}
\text{Digital Image} \arrow[d] \\
\text{Simplicial Complex} \arrow[d] \\
\text{Boundary matrices} \arrow[d] \\
\text{Homology} \arrow[uuu, bend left]
\end{tikzcd}
$$

Digital image



Boundary matrices and homology

- Boundary matrices (with only 0 and 1):

$$\ldots \xrightarrow{M_2} Faces \xrightarrow{M_1} Edges \xrightarrow{M_0} Nodes \xrightarrow{0} 0$$

- Homology groups:

$$H_0 = ker(0)/im(M_0)$$
$$H_1 = ker(M_0)/im(M_1)$$

Topological information

- $dim_{\mathbb{Z}_2}(H_0) =$ Number of connected components
- $dim_{\mathbb{Z}_2}(H_1) =$ Number of holes
- Computation: Rank of matrices over a field

Summary

- Methodology to verify and implement efficient algorithms
- CoqEAL[3] – The Coq Effective Algebra Library
- Already: Strassen, matrix rank, Karatsuba, multivariate GCD, Sasaki-Murao, general algorithms to solve systems of equations, homology of digital images...
- Future work: Gröbner bases, computational homological algebra, multivariate GCD based on subresultants...
- Future work: More modular design and automation

[3] http://www-sop.inria.fr/members/Maxime.Denes/coqeal/

Conclusion: Solutions?

- Testing
  - User/Usability
  - Automated, for example using QuickCheck
- Formal specification and verification

Conclusion: Solutions?

- Testing
  - User/Usability
  - Automated, for example using QuickCheck
- Formal specification and verification

"Beware of bugs in the above code; I have only proved it correct, not tried it." – Donald Knuth

Thank you!

This work has been partially funded by the FORMATH project, nr. 243847, of the FET program within the 7th Framework program of the European Commission

Extra slides

Effective structures

Problem: How do we compose effective structures?
For example, how to define computable matrices over computable polynomials?

```
Variable R : ringType.
Definition addseqmx (M N : seq (seq R)) :=
  zipwith (zipwith (fun x y => x + y)) M N.
```

If we instantiate R to abstract polynomials, we loose executability

Effective structures

We define concrete structures reflecting usual algebraic structures:

```
Record trans_struct (A B : Type) : Type := Trans {
  trans : A -> B;
  _ : injective trans
}.
Record mixin_of (V : zmodType) (T : Type) : Type :=
 Mixin {
  zero : T;
  opp : T -> T;
  add : T -> T -> T;
  tstruct : trans_struct V T;
  _ : (trans tstruct) 0 = zero;
  _ : {morph (trans tstruct) : x / -x >-> opp x};
  _ : {morph (trans tstruct) : x y / x+y >-> add x y}
}.
```

Sasaki-Murao algorithm

```
data Matrix a = Empty | Cons a [a] [a] (Matrix a)

|1 2 3|
|4 5 6|
|7 8 9|
   =
Cons 1 [2,3] [4,7]
  (Cons 5 [6] [8] (Cons 9 [] [] Empty))
```

Coherent rings

- Standard definition: Every ideal is finitely presented, i.e. exists exact sequence:

$$R^m \xrightarrow{\psi} R^n \xrightarrow{\varphi} I \to 0$$

Future work

- Polynomial rings – $k[x_1, \ldots, x_n]$ via Gröbner bases
- Implement library of homological algebra – HOMALG project[4]

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

Formalization

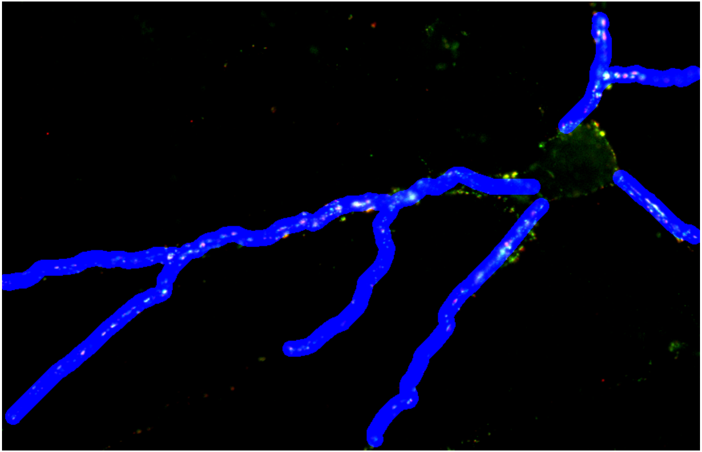
$$
\begin{tikzcd}
\text{Digital Image} \arrow[d] \\
\text{Simplicial Complex} \arrow[d] \\
\text{Boundary matrices} \arrow[d] \\
\text{Homology} \arrow[uuu, bend left]
\end{tikzcd}
$$

Application: Counting synapses

- Count the number of synapses in digital images
- Applications in biomedical engineering

Application: Counting synapses

- Synapses are the points of connection between neurons
- Relevance: Computational capabilities of the brain
- An automated and reliable method is necessary



Counting synapses

- Compute the number of connected components ($dim(H_0)$)
- Formalized in COQ:
  - Digital images
  - Simplicial complexes
  - Boundary matrices
  - Rank computation