Recent Work

ZhengPu Shi

March 1, 2023

Outline

1. **CoqMatrix Library**
   - Introduction to CoqMatrix
   - Example usage of CoqMatrix
   - Publish a Coq library to the Coq Package Index
   - Implementation details
     - Setoid Equality instead of Leibniz Equality
     - General mathematical properties and algebra structures
     - Hierarchical matrix element types
     - Hierarchical matrix theory
   - Performance of the extracted OCaml program
2. **My Research Plan**

About the CoqMatrix Library

This is an open-source formal matrix library implemented in Coq that supports multiple models[1]at the same time.

Supported models

DepList (DL)   DepPair (DP)   DepRec (DR)   NatFun (NF)
FinFun (FF)   SafeNatFun (SF)   . . .

In addition to having formally verifying common operations and properties of matrices, it also includes the following features:

Features of CoqMatrix

- supports a unified interface for multiple models under one framework;
- supports **conversion** between all models, and initially constructs the isomorphism between these structures;
- supports the development of **vector** theory;
- fully supports **Setoid equality** instead of **Leibniz equality**;
- hierarchical design for matrix element types;
- hierarchical design for matrix theory.

[1]Shi, Z., Chen G. Integration of Multiple Formal Matrix Models in Coq. SETTA 2022

The framework of the formal matrix theory

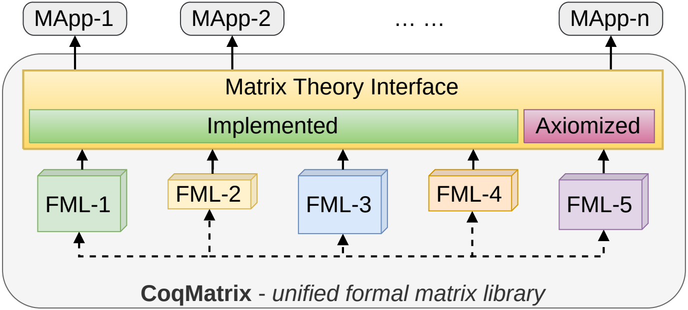

Figure: Relationship between FMLs, MApps, and CoqMatrix.

FML: Formal matrix library   **MApp**: Matrix application

Implemented: the matrix theory implemented by at least one model.

Axiomed: the new matrix theory, not yet formalized by any models, but exists now.

The framework of the formal vector theory

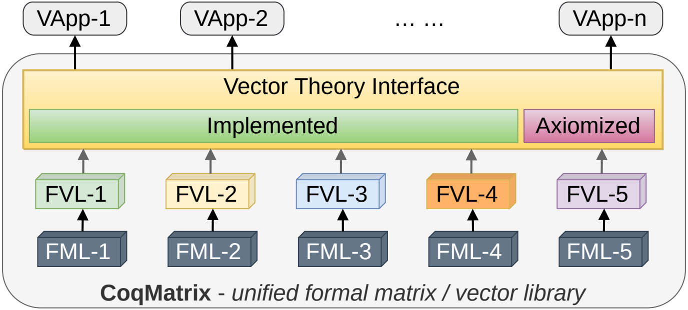

Figure: Relationship between FVLs, VApps, and CoqMatrix.

FVL: Formal vector library **VApp**: Vector application

Installation & Basic usage

Installation

- The easist way is to install the stable version using OPAM

```
$ opam repo add coq-released https://coq.inria.fr/opam/released
$ opam install coq-matrix
```

- Use the project on github to get the latest version and contribute to development

```
$ git clone https://github.com/zhengpushi/CoqMatrix.git
$ cd CoqMatrix
$ make -j    # it takes < 25 seconds
$ make install
```

Now, we can use this matrix library with the logical-name “CoqMatrix”

Basic Usage

```
(* You should import the matrix library on different element types, such as Nat/Z/Q/Qc/R. *)
From CoqMatrix Require Import MatrixNat.
(* Now, all the types, functions, lemmas, notations are available *)

(* Or, import the vector library on different element types *)
From CoqMatrix Require Import VectorZ.

(* Or, use matrix library on your favorite model *)
From CoqMatrix Require Import MatrixQ.
Import MatrixQ_DL. (* DL/DP/DR/NF/FF/SF *)
(* Now, the matrix model is switched to DepList (DL) *)
```

Outline

Example usage of CoqMatrix

```
(* import matrix library on Q (rational number) *)
From CoqMatrix Require Import MatrixQ.

(* matrix type *)
Check mat.                (* : nat -> nat -> Type *)

(* construct a matrix with a list *)
Example m1 : mat 2 3 := l2m [[1;2;3];[4;5;6]].

(* construct a matrix with a function *)
Example m3 : mat 2 3 :=
  mk_mat
    (fun i j =>
       match i, j with
       | 0%nat, 0%nat => 1
       | 1%nat, 2%nat => 2
       | _, _ => 0
       end).

(** show content *)
Compute (m2l m3).         (* = [[1; 0; 0]; [0; 0; 2]] *)
```

Example usage of CoqMatrix (cont.)

```
(* construct a matrix with every elements *)
Example m2 : mat 2 2 := mk_mat_2_2 1 2 3 4.

(** transpose a matrix *)
Compute m2l (m2\T).        (* = [[1; 3]; [2; 4]] *)

(** matrix addition *)
Compute m2l (m2 + m2).     (* = [[2; 4]; [6; 8]] *)

(** matrix scalar multiplication *)
Compute m2l (3 c* m2).     (* = [[3; 6]; [9; 12]] *)

(** matrix multiplication *)
Compute m2l (m2 * m2).     (* = [[7; 10]; [15; 22]] *)

(** identity matrix *)
Compute m2l (mat1 3).      (* = [[1; 0; 0]; [0; 1; 0]; [0; 0; 1]] *)
```

More examples, see online **README.md** or Coq.

Outline

1. **CoqMatrix Library**
   - Introduction to CoqMatrix
   - Example usage of CoqMatrix
   - Publish a Coq library to the Coq Package Index
   - Implementation details
     - Setoid Equality instead of Leibniz Equality
     - General mathematical properties and algebra structures
     - Hierarchical matrix element types
     - Hierarchical matrix theory
   - Performance of the extracted OCaml program
2. **My Research Plan**

Publish a Coq library to the Coq Package Index

Reference: `https://coq.inria.fr/opam-packaging.html`

Prepare your package

- contain a Makefile providing two commands: **make, make install.**
- an archive which must be available to download, such as **https://github.com/<you>/foo/archive/1.0.0.tar.gz**

Prepare your local opam archive of Coq

- clone the repository

```
$ git clone https://github.com/coq/opam-coq-archive -o upstream
```

- create sub-directory named as follows:

```
$ mkdir -p released/packages/coq-foo/coq-foo.1.0.0
```

- create a text file "opam" in this directory, follows the template
- commit the new "opam" file

Publish a Coq library to the Coq Package Index (cont.)

Test your package locally

- lint your opam file

```
$ opam lint --check-upstream released/packages/coq-foo/coq-foo.1.0.0/opam
```

- test your package

```
$ opam repo add test ./released
$ opam install -v coq-foo
```

Submit your package

- push your change to your personal fork

```
$ git push origin coq-foo.1.0.0
```

- visit the GitHub page of your fork, click the **new pull request** button, and submit.
- the **CI** in GitHub will be triggered, be sure no errors in any toochains, and wait for merge by manager.

Search your package from Coq Package Index

https://coq.inria.fr/opam/www/

Outline

Setoid Equality instead of Leibniz Equality

What is the differences?

Leibniz equality is the standard way to define equality in Coq.

```
Inductive eq {A : Type} (x : A) : A → Prop := eq_refl : eq x x.
Infix "=" := eq.
```

Setoid equality defines equality in terms of an equivalence relationship.

```
Variable A : Type.   Variable Aeq : A → A → Prop.
Context {Equiv_Aeq : Equivalence A Aeq}.   Infix "==" := Aeq.
```

Why is such a heavy approach needed?

1. setoid equality is a general case of leibniz equality;
2. to represent equality on some special matrix element types, such as rational numbers, residue classes, functions, and matrices themselves.

How to implement it?

1. use "==" instead of "=", from element to list, list of lists, matrix, and vector.
2. prove "operations respect the equality relation" to enable rewrite in Coq, e.g., **Lemma madd_meq_mor** : **Proper** (**meq** $==>$ **meq** $==>$ **meq**) **madd**, means that, $\forall\ r\ c\ (m_1\ m_2\ m_3\ m_4 : mat\ r\ c), m_1 == m_2 \rightarrow m_3 == m_4 \rightarrow m_1 + m_3 == m_2 + m_4.$

Outline

General mathematical properties and algebra structures

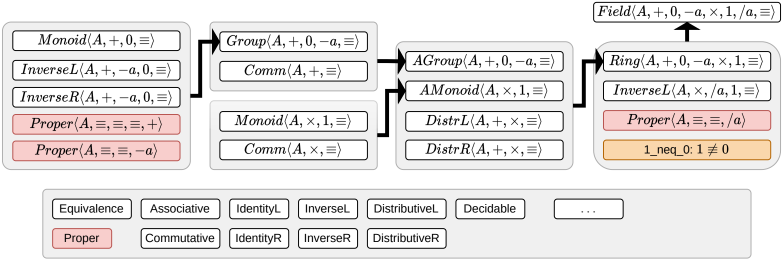

Figure: Hierarchical mathematical structures using typeclass.

Advantages of using typeclass in Coq to organize the definitions and propositions:

- the structure corresponds to mathematics concepts and can be easily combined;
- less code.

Details about **typeclass** can be refer to "*A Tutorial on typeclasses in Coq*" in vol4 of SF.

General mathematical properties and algebra structures (cont.)

Declare property or structure in Coq

```
Class Associative {A : Type} (Aop : A -> A -> A) (Aeq : A -> A -> Prop) := {
  associative : forall a b c, Aeq (Aop (Aop a b) c) (Aop a (Aop b c));
}.

Class Monoid {A : Type} (Aop : A -> A -> A) (Aeq : A -> A -> Prop) (A0 : A) := {
  monoidAaddProper :> Proper (Aeq ==> Aeq ==> Aeq) Aadd;
  monoidEquiv :> Equivalence Aeq;
  monoidAssoc :> Associative Aadd Aeq;
  monoidIdL :> IdentityL Aadd A0 Aeq;
  monoidIdR :> IdentityR Aadd A0 Aeq;
}.
```

Here, syntax :> assures that a *Monoid* object also has the type on the right side.

Demo usage of monoid structure

```
Context `{M:Monoid}. (* automatically declared A,Aadd,... too *)
Infix "+" := Aop. Infix "==" := Aeq.

Goal ∀a b c : A, (a + b) + c + A0 == a + (b + c).
Proof.
  intros.
  rewrite monoidIdR. apply associative.
Qed.
```

We can observed that, the declaration of a structure only needs few code.

Outline

Hierarchical matrix element types

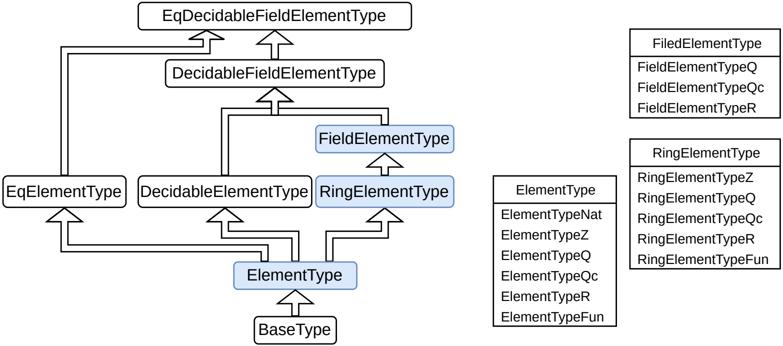

Figure: Interface of the hierarchical matrix element types using Module & Module Type

Hierarchical matrix element types (cont.)

Abstract matrix element types in Coq

```
(** Type of element *)
Module Type ElementType.
  Parameter A : Type.
  Parameter A0 : A.
  Parameter Aeq : relation A.
  Axiom Equiv_Aeq : Equivalence Aeq.
End ElementType.

(** Type of element with ring structure *)
Module Type RingElementType <: ElementType.
  Include ElementType.

  Parameter A1 : A.
  Parameter Aadd Amul : A → A → A.
  Parameter Aopp : A → A.
  Notation Asub := λ x y ⇒ Aadd x (Aopp y))

  Axiom Aadd_aeq_mor : Proper (Aeq ==> Aeq ==> Aeq) (Aadd).
  Axiom Aopp_aeq_mor : Proper (Aeq ==> Aeq) (Aopp).
  Axiom Amul_aeq_mor : Proper (Aeq ==> Aeq ==> Aeq) (Amul).
  Axiom Ring_thy : ring_theory A0 A1 Aadd Amul Asub Aopp Aeq.
End RingElementType.
```

Outline

Hierarchical matrix theory

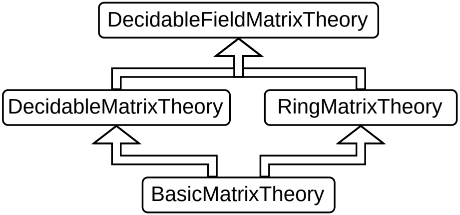

Figure: Interface of the hierarchical matrix theory using Module & Module Type.

Hierarchical matrix theory (cont.)

Basic Matrix Theory

```
Module Type BasicMatrixTheory (E : ElementType).
  Export E.
  Parameter mat : nat → nat → Type.
  Parameter meq : ∀ {r c}, mat r c → mat r c → Prop.
  Axiom meq_equiv : ∀ {r c}, Equivalence (meq (r:=r) (c:=c)).
      Infix "==" := Aeq : A_scope.    Infix "==" := meq : mat_scope.
  Parameter mnth : ∀ {r c}, mat r c → nat → nat → A.
  Axiom meq_iff_mnth: ∀ {r c} (m1 m2: mat r c), m1 == m2 →
  ∀ ri ci, ri < r → ci < c → (mnth m1 ri ci == mnth m2 ri ci)).
  (** ** Convert between list list and matrix *)
  Parameter l2m : ∀ {r c} (dl: list (list A)), mat r c.
  Parameter m2l : ∀ {r c}, mat r c → list (list A).
  (** ** Specific matrix *)
  Parameter mk_mat_3_3 : ∀ a1 a2 a3 b1 b2 b3 c1 c2 c3: A, mat 3 3.
  (** ** Matrix transposition *)
  Parameter mtrans : ∀ {r c} (m: mat r c), mat c r.
  Axiom mtrans_trans : ∀ {r c} (m: mat r c), mtrans (mtrans m) == m.
  (** ** Mapping of matrix *)
  Parameter mmap : ∀ {r c} (f: A → A) (m: mat r c), mat r c.
  Parameter mmap2 : ∀ {r c} (f: A → A → A) (m1 m2: mat r c), mat r c.
End BasicMatrixTheory.
```

Hierarchical matrix theory (cont.)

Ring Matrix Theory

```
Module Type RingMatrixTheory (E:RingElementType) <: BasicMatrixTheory E.
  Include (BasicMatrixTheory E).
  Parameter mat0: ∀ {r c}, mat r c.  Parameter mat1: ∀ {n}, mat n n.
  Parameter madd msub: ∀ {r c}, mat r c → mat r c → mat r c.
  Parameter mopp: ∀ {r c}, mat r c → mat r c.
  Parameter mcmul: ∀ {r c}, A → mat r c → mat r c.
  Parameter mmul: ∀ {r c s}, mat r c → mat c s → mat r s.
      Infix " + " := madd.   Infix " ×c " := mcmul.   Infix " × " := mmul.
      Infix " − " := msub.   Notation " − m" := (mopp m).
  Axiom madd_comm: ∀ m1 m2, m1 + m2 == m2 + m1.
  Axiom madd_assoc: ∀ m1 m2 m3, (m1 + m2) + m3 == m1 + (m2 + m3).
  Axiom madd_0_l: ∀ m, mat0 + m == m.
  Axiom mopp_opp: ∀ m, - - m == m.
  Axiom madd_opp: ∀ m, m + (- m) == mat0.
  Axiom msub_comm: ∀ m1 m2, m1 - m2 == - (m2 - m1).
  Axiom msub_assoc: ∀ m1 m2 m3, (m1 - m2) - m3 == m1 - (m2 + m3).
  Axiom msub_0_l: ∀ m, mat0 - m == - m.
  Axiom mcmul_assoc: ∀ a b m, a ×c (b ×c m) == (a * b) ×c m.
  Axiom mcmul_perm : ∀ a b m, a ×c (b ×c m) == b ×c (a ×c m).
  Axiom mcmulAddDistrL: ∀ a m1 m2, a ×c (m1 + m2) == (a ×c m1) + (a ×c m2).
  Axiom mcmul_1_l: ∀ m, A1 ×c m == m
  Axiom mmulAddDistrL: ∀ m1 m2 m3, m1 ×(m2 + m3) == m1 × m2 + m1 × m3.
  Axiom mmul_assoc: ∀ m1 m2 m3, (m1 × m2) × m3 == m1 × (m2 × m3).
  Axiom mmul_0_l: ∀ m, mat0 × m == mat0.
  Axiom mmul_1_l: ∀ m, mat1 × m == m.
End RingMatrixTheory.
```

Outline

Performance of the extracted OCaml program

To evaluate the performance differences of these models, we designed an experiment.

The steps of the experiment

1. For some model **M**, set the initial matrix size $n = 50$ and go to the step (2).
2. Generate two sets DL_A and DL_B of dlist type of size $n \times n$ with random data.
3. Calls the function **l2m** on the model **M** to construct the matrices $A_{n\times n}$ and $B_{n\times n}$ on the model with DL_A and DL_B, separately.
4. Calls the function **mmul** on the model **M** to perform **Matrix Multiplication** $C = A \times B$.
5. Calls the function **m2l** on the model **M** to convert the matrix C into DL_C of dlist type;
6. Calculate the elapsed time from steps (2) to (5); if the time exceeds one minute then exit the loop.
7. Change the size of the matrix to $n = n * 120\%$;
8. Jump to (2) to continue the loop.

Thus, the performance differences is determined by step (3) to step (5), that is:

$$\textbf{l2m} + \textbf{mmul} + \textbf{m2l}$$

Performance of the extracted OCaml program (cont.)

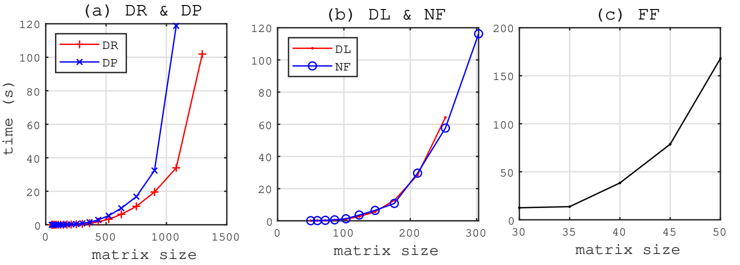

Figure: Performance comparison of OCaml programs extracted from multiple matrix models in Coq when performing matrix multiplication

Result of the experiment

- Figure (a) shows, the performance of **DR** and **DP** are at the same level, and the size of the matrix they can process within 1 minute is about 1000×1000.
- Figure (b) shows, the performance of **DL** and **NF** are similiar, and the size of the matrix they can process within 1 minute is about 250×250.
- Figure (c) is the case of the **FF** model, which cannot handle a matrix of size 50×50 within 1 minute.
- In short, when performing the same operations here, the performance of each model from high to low is as follows: **DR** $\geq$ **DP** $>$ **NF** $\geq$ **DL** $>$ **FF**.

Outline

My Research Plan

- My current research work:
  1. FV of Quaternion, for coordinate sub-system in FCS.
  2. FV of Inversion Matrix Algorithm with Determinant and Adjoint Matrix, which is useful for symbol matrix.
  3. FV of Quantity Calculus System, which support calculus with unit.
- My further research plan:
  1. FV of Control Theory, which is the basis of Automatic Controls and Flight Control Systems, and heavily use matrix theory. The related concepts are:
     1. State Space Description
     2. Stability Analysis.
  2. FV of Complex Variable Function
     1. Laplace transform
     2. Transfer Function

Thank you!

Q&A