Formalisation and execution of Linear Algebra: theorems and algorithms[1]

Jose Divasón

UNIVERSIDAD DE LA RIOJA

PhD Defense

Advisors: Dr. Julio Rubio

Dr. Jesús María Aransay

[1]This work has been supported by the research grant FPI-UR-12 from Universidad de La Rioja and by the project MTM2014-54151-P from Ministerio de Economía y Competitividad

Software development is error-prone

Windows

An error has occurred. To continue:

Press Enter to return to Windows, or

Press CTRL+ALT+DEL to restart your computer. If you do this,
you will lose any unsaved information in all open applications.

Error: 0E : 016F : BFF9B3D4

Press any key to continue _


Gepäckrückgabe
Baggage claim
PAGE_FAULT_IN_NONPAGED_AREA

BBC Sign in | News | Sport | Weather | Shop | Earth | Travel | Mo

NEWS

Home | Video | World | UK | Business | Tech | Science | Magazine | Entertainment & Arts

Technology

Fatal A400M crash linked to data-wipe mistake

By Leo Kelion
Technology desk editor

10 June 2015 | Technology


Do not use
Out of order

Software Verification

It is **necessary** to verify software somehow in order to minimise possible faults

Software Verification

It is **necessary** to verify software somehow in order to minimise possible faults

- Software testing is one of the major software verification techniques used in practice

Software Verification

It is **necessary** to verify software somehow in order to minimise possible faults

- Software testing is one of the major software verification techniques used in practice
- **Testing** can never be complete, **infeasible for critical systems**

Software Verification

It is **necessary** to verify software somehow in order to minimise possible faults

- Software testing is one of the major software verification techniques used in practice
- **Testing** can never be complete, **infeasible for critical systems**

> "Program testing can be used to show the presence of bugs, but never to show their absence!"
>
> — Edsger W. Dijkstra

Software Verification

It is **necessary** to verify software somehow in order to minimise possible faults

- Software testing is one of the major software verification techniques used in practice
- **Testing** can never be complete, **infeasible for critical systems**

> "Program testing can be used to show the presence of bugs, but never to show their absence!"
>
> — Edsger W. Dijkstra

- **Formal methods** refer to *"mathematically rigorous techniques and tools for the specification, design and verification of software and hardware systems"*

Formalisation of mathematics

A mathematical proof is rigorous when it has been written out as a sequence of inferences from the axioms, each inference made according to one of the stated rules

Formalisation of mathematics

A mathematical proof is rigorous when it has been written out as a sequence of inferences from the axioms, each inference made according to one of the stated rules

Tedious and requires too much effort: in 1910 Whitehead and Russell formally proved $1+1=2$ after 379 pages of work

Formalisation of mathematics

A mathematical proof is rigorous when it has been written out as a sequence of inferences from the axioms, each inference made according to one of the stated rules

Tedious and requires too much effort: in 1910 Whitehead and Russell formally proved $1 + 1 = 2$ after 379 pages of work

"My intellect never quite recovered. I have been ever since definitely less capable of dealing with difficult abstractions than I was before."

— Bertrand Russell

Formalisation of mathematics

A mathematical proof is rigorous when it has been written out as a sequence of inferences from the axioms, each inference made according to one of the stated rules

Tedious and requires too much effort: in 1910 Whitehead and Russell formally proved $1 + 1 = 2$ after 379 pages of work

"My intellect never quite recovered. I have been ever since definitely less capable of dealing with difficult abstractions than I was before."

— Bertrand Russell

- An **interactive theorem prover** is a software tool to assist with the development of formal proofs by human-machine collaboration (Isabelle, Coq, ACL2,. . .)

Formalisation of mathematics

A mathematical proof is rigorous when it has been written out as a sequence of inferences from the axioms, each inference made according to one of the stated rules
Tedious and requires too much effort: in 1910 Whitehead and Russell formally proved $1 + 1 = 2$ after 379 pages of work

"My intellect never quite recovered. I have been ever since definitely less capable of dealing with difficult abstractions than I was before."
— Bertrand Russell

- An **interactive theorem prover** is a software tool to assist with the development of formal proofs by human-machine collaboration (Isabelle, Coq, ACL2,. . .)
- For better or worse, "the machine magnifies competence, but it also magnifies incompetence. . ."

Formalisation of mathematics

A mathematical proof is rigorous when it has been written out as a sequence of inferences from the axioms, each inference made according to one of the stated rules

Tedious and requires too much effort: in 1910 Whitehead and Russell formally proved $1 + 1 = 2$ after 379 pages of work

"My intellect never quite recovered. I have been ever since definitely less capable of dealing with difficult abstractions than I was before."

— Bertrand Russell

- An **interactive theorem prover** is a software tool to assist with the development of formal proofs by human-machine collaboration (Isabelle, Coq, ACL2,. . .)
- For better or worse, "the machine magnifies competence, but it also magnifies incompetence. . ." — Lawrence C. Paulson

What

Formalisation of Linear Algebra algorithms

Why

Generation of verified algorithms usable in practice

How

- Using an interactive theorem prover
- Framework to formalise, execute, refine and connect Linear Algebra algorithms with their mathematical meaning
- Apply it to formalise four well-known algorithms and their applications

Toolkit

- Proof assistant: Isabelle (L. Paulson, T. Nipkow, M. Wenzel)
- Underlying logic: Higher-order logic (HOL) + type classes
- Additional libraries: HOL Multivariate Analysis (HMA, J. Harrison)
- Code generation infrastructure (F. Haftmann)
- Proof language: Intelligible semi-automated reasoning (Isar, M. Wenzel)
- Execution environments: GH(askell)C, PolyML (D. Matthews) and MLton

Isabelle

- Isabelle is an interactive theorem prover created by Paulson in 1986
- Worldwide user community
- Flyspeck (the formal proof of the Kepler conjecture) and seL4 (an operating-system kernel)
- Isabelle is a generic theorem prover: it has been instantiated to support different object-logics
- The most widespread object-logic supported by Isabelle is higher-order logic (HOL)

HOL = Functional Programming + Logic

HMA - Multivariate Analysis session

- Our formalisations are based on the HOL Multivariate Analysis session
- Adequate vector and matrix representation from the formalisation point of view

HMA - Multivariate Analysis session

- Our formalisations are based on the HOL Multivariate Analysis session
- Adequate vector and matrix representation from the formalisation point of view

typedef (α,β) vec = UNIV :: $((\beta$::finite$) \Rightarrow \alpha)$ set
morphisms vec−nth vec−lambda ..

- Type System vs Logic

State of the Art (January 2013)

- Isabelle/HOL has a number of Libraries that deal with Algebra and Multivariate Analysis

State of the Art (January 2013)

- Isabelle/HOL has a number of Libraries that deal with Algebra and Multivariate Analysis
- Execution was not explored (either in Isabelle or HOL Light)

State of the Art (January 2013)

- Isabelle/HOL has a number of Libraries that deal with Algebra and Multivariate Analysis
- Execution was not explored (either in Isabelle or HOL Light)
- Linear Algebra algorithms had barely been implemented

State of the Art (January 2013)

- Isabelle/HOL has a number of Libraries that deal with Algebra and Multivariate Analysis
- Execution was not explored (either in Isabelle or HOL Light)
- Linear Algebra algorithms had barely been implemented
- Example:

T. Nipkow. Gauss-Jordan Elimination for Matrices Represented as Functions. Archive of Formal Proofs (2011)

Framework

Framework to formalise, execute, refine and connect Linear Algebra algorithms with their mathematical meaning

- **Formalise:** Definition of elementary matrix operations

Framework

Framework to formalise, execute, refine and connect Linear Algebra algorithms with their mathematical meaning

- **Formalise:** Definition of elementary matrix operations

definition interchange-rows :: 'a^'n^'m⇒'m⇒'m⇒'a^'n^'m
  where interchange-rows A a b = (χ i j. if i=a then A $ b $ j else if i=b then A $ a $ j else A $ i $ j)

Framework

Framework to formalise, execute, refine and connect Linear Algebra algorithms with their mathematical meaning

- **Formalise:** Definition of elementary matrix operations

definition interchange-rows :: 'a^'n^'m⇒'m⇒'m⇒'a^'n^'m
  where interchange-rows A a b = (χ i j. if i=a then A \$ b \$ j else if i=b then A \$ a \$ j else A \$ i \$ j)

lemma interchange-rows-mat-1:
  shows interchange-rows (mat 1) a b ** A = interchange-rows A a b

Framework

Framework to formalise, execute, refine and connect Linear Algebra algorithms with their mathematical meaning

- **Formalise:** Definition of elementary matrix operations
- **Execution and refinement:** HMA matrix representation (vec) is refined to (efficient) executable representations (functions, immutable arrays). Code is exported to functional programming languages

Framework

Framework to formalise, execute, refine and connect Linear Algebra algorithms with their mathematical meaning

- **Formalise:** Definition of elementary matrix operations
- **Execution and refinement:** HMA matrix representation (vec) is refined to (efficient) executable representations (functions, immutable arrays). Code is exported to functional programming languages
- **Connection:**

Framework

Framework to formalise, execute, refine and connect Linear Algebra algorithms with their mathematical meaning

- **Formalise:** Definition of elementary matrix operations
- **Execution and refinement:** HMA matrix representation (*vec*) is refined to (efficient) executable representations (functions, immutable arrays). Code is exported to functional programming languages
- **Connection:**

lemma linear_bij_rank_eq_ncols:
  fixes f::′a::field^n::mod_type ⇒ ′a^n
  assumes linear (op ∗s) (op ∗s) f
  shows bij f ⟷ rank (matrix f) = ncols (matrix f)

Data refinement consists of replacing an abstract (probably non-executable) datatype by a more concrete (executable) one

Refinement

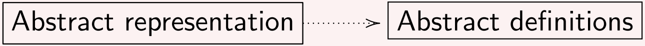
Abstract representation

Data refinement consists of replacing an abstract (probably non-executable) datatype by a more concrete (executable) one

Refinement

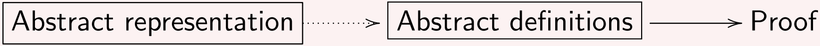
Data refinement consists of replacing an abstract (probably non-executable) datatype by a more concrete (executable) one

Refinement



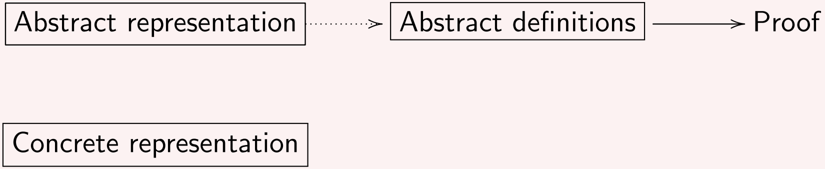
Data refinement consists of replacing an abstract (probably non-executable) datatype by a more concrete (executable) one

Refinement



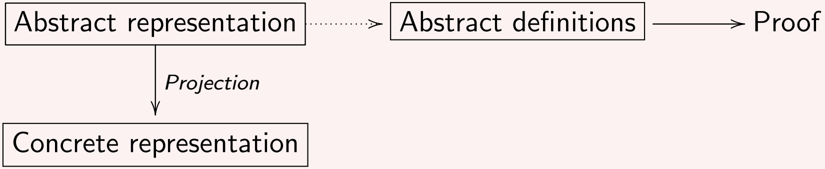
Data refinement consists of replacing an abstract (probably non-executable) datatype by a more concrete (executable) one

Refinement



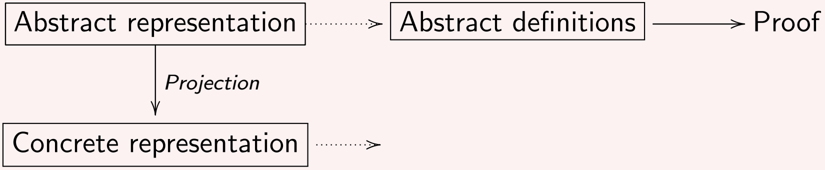
Data refinement consists of replacing an abstract (probably non-executable) datatype by a more concrete (executable) one

Refinement



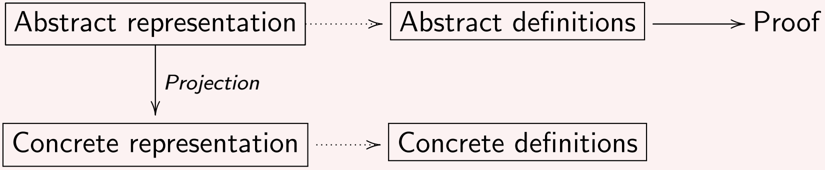
Data refinement consists of replacing an abstract (probably non-executable) datatype by a more concrete (executable) one

Refinement



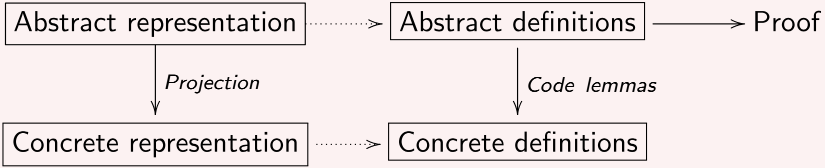
Data refinement consists of replacing an abstract (probably non-executable) datatype by a more concrete (executable) one

Refinement



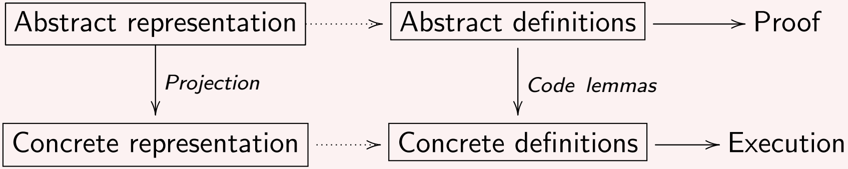
Data refinement consists of replacing an abstract (probably non-executable) datatype by a more concrete (executable) one

Refinement



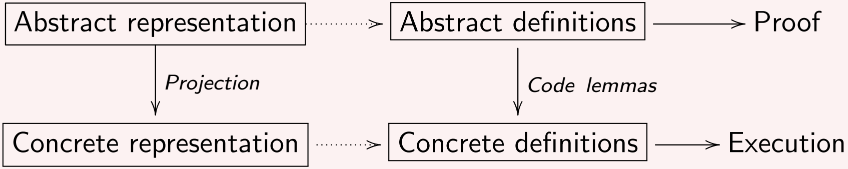
Data refinement consists of replacing an abstract (probably non-executable) datatype by a more concrete (executable) one

Refinement



Data refinement consists of replacing an abstract (probably non-executable) datatype by a more concrete (executable) one

Refinement



Two refinements have been carried out so that operations over the abstract type *vec* can be executed

1. From *vec* to *function over finite types*
2. From *vec* to *iarray*

2. From *vec* to iarray

In order to achieve better performance, a refinement has been developed using immutable arrays

- There exists a datatype in the Isabelle library called *iarray* which represents immutable arrays
- *iarray* is implemented in both SML (*Vector structure*) and Haskell (*IArray class*)
- We have refined *vec* elements and operations to *iarray* ones (proving the corresponding morphisms)

2. From *vec* to iarray

In order to achieve better performance, a refinement has been developed using immutable arrays

- There exists a datatype in the Isabelle library called *iarray* which represents immutable arrays
- *iarray* is implemented in both SML (*Vector structure*) and Haskell (*IArray class*)
- We have refined *vec* elements and operations to *iarray* ones (proving the corresponding morphisms)

Features of this refinement

1. Code can be generated to both SML and Haskell
2. Improved performance

Serialisations

- Isabelle datatypes are mapped to the corresponding implementation in the target languages
- Need to be trusted

Isabelle/HOL	SML	Haskell
iarray	*Vector.vector*	**IArray.Array**
rat	*IntInf.int / IntInf.int*	**Rational**
real	*Real.real*	**Double**
bit	**Bool.bool**	**Bool**

First part of the Fundamental Theorem of Linear Algebra

Theorem (The Rank-Nullity Theorem)

Let $\tau \in \mathcal{L}(V, W)$*, where* $\mathcal{L}(V, W)$ *is the set of linear maps between a finite-dimensional vector space* V *and a vector space* W*; then*

$$\dim V = \dim(\ker \tau) + \dim(\operatorname{im} \tau)$$

where $\ker \tau \subseteq V$ *and* $\operatorname{im} \tau \subseteq W$

First part of the Fundamental Theorem of Linear Algebra

Theorem (The Rank-Nullity Theorem)

Let $\tau \in \mathcal{L}(V, W)$*, where* $\mathcal{L}(V, W)$ *is the set of linear maps between a finite-dimensional vector space* V *and a vector space* W*; then*

$$\dim V = \dim(\ker \tau) + \dim(\operatorname{im} \tau)$$

where $\ker \tau \subseteq V$ *and* $\operatorname{im} \tau \subseteq W$

Reinterpretation with matrices

$V \cong \mathcal{F}^n$, $W \cong \mathcal{F}^m$, $\tau = A \in \mathcal{M}_{(m,n)}(\mathcal{F})$, $\operatorname{im} \tau = \mathrm{C}(A)$, $\ker \tau = \mathrm{N}(A)$

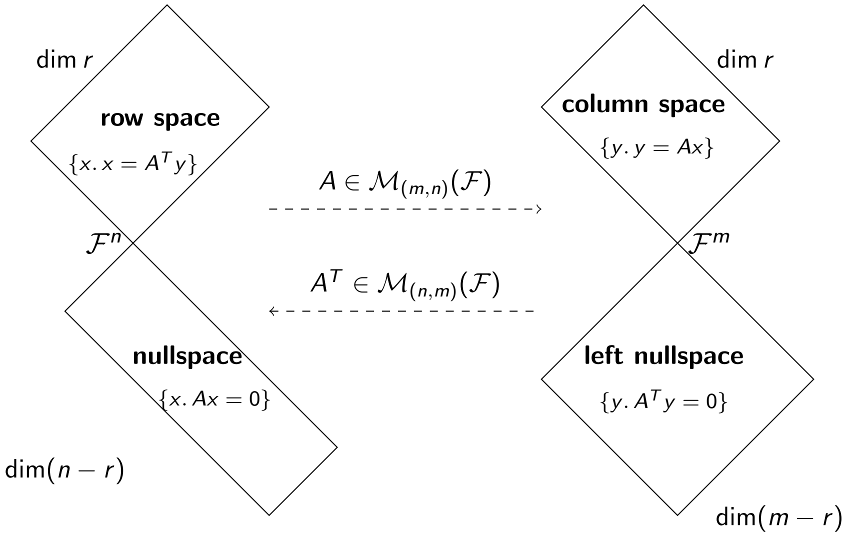


Figure : Bases of the four Fundamental subspaces

Isabelle statement

- Linear map statement

theorem rank-nullity-theorem:
shows V.dimension = V.dim {x. f x = 0} + W.dim (range f)

Isabelle statement

- Linear map statement

theorem rank-nullity-theorem:
shows V.dimension = V.dim {x. f x = 0} + W.dim (range f)

- Matrix statement

theorem rank-nullity-theorem-matrices:
fixes A::field^'cols::{wellorder}^'rows
shows ncols A = vec.dim (null-space A) + vec.dim (col-space A)

Isabelle statement

- Linear map statement

theorem rank-nullity-theorem:
shows V.dimension = V.dim {x. f x = 0} + W.dim (range f)

- Matrix statement

theorem rank-nullity-theorem-matrices:
fixes A::field^'cols::{wellorder}^'rows
shows ncols A = vec.dim (null-space A) + vec.dim (col-space A)

J. Divasón and J. Aransay. Rank-Nullity Theorem in Linear Algebra. Archive of Formal Proofs (2013)

From theorems to algorithms

- Gauss-Jordan elimination provides a direct way to compute the *reduced row echelon form (rref)* by means of *elementary row operations* over A

From theorems to algorithms

- Gauss-Jordan elimination provides a direct way to compute the *reduced row echelon form (rref)* by means of *elementary row operations* over A

Gauss-Jordan example

$$A = \begin{pmatrix} 1 & -2 & 1 & -3 & 0 \\ 3 & -6 & 2 & -7 & 0 \\ 5 & -1 & 3 & 2 & 5 \\ 0 & 7 & 4 & 5 & 1 \\ 3 & -6 & 2 & -7 & 0 \end{pmatrix} \longrightarrow A = \begin{pmatrix} 1 & 0 & 0 & 0 & 3 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 & -2 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

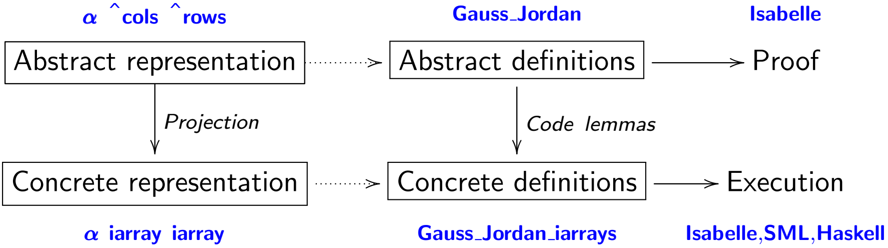
From theorems to algorithms

- Gauss-Jordan elimination provides a direct way to compute the *reduced row echelon form (rref)* by means of *elementary row operations* over A

Gauss-Jordan example

$$A = \begin{pmatrix} 1 & -2 & 1 & -3 & 0 \\ 3 & -6 & 2 & -7 & 0 \\ 5 & -1 & 3 & 2 & 5 \\ 0 & 7 & 4 & 5 & 1 \\ 3 & -6 & 2 & -7 & 0 \end{pmatrix} \longrightarrow A = \begin{pmatrix} 1 & 0 & 0 & 0 & 3 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 & -2 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\dim(\mathrm{C}(A)) = 4$$


α ^cols ^rows
Gauss_Jordan
Isabelle
Abstract representation
Abstract definitions
Proof
Projection
Code lemmas
Concrete representation
Concrete definitions
Execution
α iarray iarray
Gauss_Jordan_iarrays
Isabelle,SML,Haskell

Generalisations

From HMA and the reals to fields

lemma rank-Gauss-Jordan-real:
  fixes A::real^'n::{mod-type}^'m::{mod-type}
  shows rank A = rank (Gauss-Jordan A)
by (metis Gauss-Jordan crk-is-preserved rank-col-rank)

J. Aransay and J. Divasón. Generalizing a Mathematical Analysis library in Isabelle/HOL. Proceedings of the 7th NASA Formal Methods Symposium (NFM 2015)

Generalisations

From HMA and the reals to fields

lemma rank-Gauss-Jordan-real:
  fixes A::real^'n::{mod-type}^'m::{mod-type}
  shows rank A = rank (Gauss-Jordan A)
by (metis Gauss-Jordan crk-is-preserved rank-col-rank)

lemma rank-Gauss-Jordan:
  fixes A::'a::{field}^'n::{mod-type}^'m::{mod-type}
  shows rank A = rank (Gauss-Jordan A)
by (metis Gauss-Jordan-def invertible-Gauss-Jordan-up-to-k
    row-rank-eq-col-rank rank-def crk-is-preserved)

J. Aransay and J. Divasón. Generalizing a Mathematical Analysis library in Isabelle/HOL. Proceedings of the 7th NASA Formal Methods Symposium (NFM 2015)

The following computations can be performed by means of the Gauss-Jordan algorithm

The following computations can be performed by means of the Gauss-Jordan algorithm

Gauss-Jordan algorithm applications

- Reduced row echelon form
- Ranks
- Determinants
- Inverses
- Dimensions and bases of the null space, left null space, column space and row space
- Solution(s) of systems of linear equations

The following computations can be performed by means of the Gauss-Jordan algorithm

Gauss-Jordan algorithm applications

- Reduced row echelon form
- Ranks
- Determinants
- Inverses
- Dimensions and bases of the null space, left null space, column space and row space
- Solution(s) of systems of linear equations

J. Divasón and J. Aransay. Gauss-Jordan Algorithm and Its Applications
Archive of Formal Proofs (2014)

Ranks

$$\begin{pmatrix} 1+i & 1-i & 0 \\ 2-i & 1+3i & 7+3i \\ 3 & 2+2i & 7+3i \end{pmatrix} \in \mathcal{M}_{3\times 3}(\mathbb{C})$$

```
value "rank (list_of_list_to_matrix
            [
             [Complex 1 1, Complex 1 (-1), Complex 0 0],
             [Complex 2 (-1), Complex 1 3, Complex 7 3],
             [Complex 3 0, Complex 2 2, Complex 7 3]
            ]::complex^3^3)"
```

Proof state   Auto upda

```
"2"
  :: "nat"
```

Determinants

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \in \mathcal{M}_{3\times 3}(\mathbb{R})$$

```
value "det (list_of_list_to_matrix
           [[1,1,0],
            [0,1,1],
            [1,0,1]]::real^3^3)"
```

```
"2"
  :: "real"
```

Determinants

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \in \mathcal{M}_{3\times 3}(\mathbb{R}) \qquad A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \in \mathcal{M}_{3\times 3}(\mathbb{Z}_2)$$

```
value "det (list_of_list_to_matrix
          [[1,1,0],
           [0,1,1],
           [1,0,1]]::real^3^3)"

"2"
  :: "real"
```

```
value "det (list_of_list_to_matrix
          [[1,1,0],
           [0,1,1],
           [1,0,1]]::bit^3^3)"

"0"
  :: "bit"
```

Inverse

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \in \mathcal{M}_{3\times 3}(\mathbb{R}) \qquad inv(A) = \begin{pmatrix} 1/2 & -1/2 & 1/2 \\ 1/2 & 1/2 & -1/2 \\ -1/2 & 1/2 & 1/2 \end{pmatrix}$$

```
value "let A=(list_of_list_to_matrix
               [[1,1,0],
                [0,1,1],
                [1,0,1]]::real^3^3)
       in show_inverse (inverse_matrix A)"
```

Proof state Auto update

```
"Some [[1 / 2, - (1 / 2), 1 / 2], [1 / 2, 1 / 2, - (1 / 2)],
       [- (1 / 2), 1 / 2, 1 / 2]]"
  :: "real list list option"
```

Inverse

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \in \mathcal{M}_{3\times 3}(\mathbb{Z}_2)$$

```
value "let A=(list_of_list_to_matrix
               [[1,1,0],
                [0,1,1],
                [1,0,1]]::bit^3^3)
       in show_inverse (inverse_matrix A)"
```

```
"None"
  :: "bit list list option"
```

Bases and dimensions of fundamental subspaces

```
definition left_null_space :: "'a::{semiring_1}^'n^'m => ('a^'m) set"
  where "left_null_space A = {x. x v* A = 0}"

definition null_space :: "'a::{semiring_1}^'n^'m => ('a^'n) set"
  where "null_space A = {x. A *v x = 0}"

definition row_space :: "'a::{field}^'n^'m=>('a^'n) set"
  where "row_space A = vec.span (rows A)"

definition col_space :: "'a::{field}^'n^'m=>('a^'m) set"
  where "col_space A = vec.span (columns A)"
```

```
value "let A = (list_of_list_to_matrix
      [[ 3, 4, 0, 7],
       [ 1,-5, 2,-2],
       [-1, 4, 0, 3],
       [ 1,-1, 2, 2]]::rat^4^4)
  in vec_to_list` (basis_left_null_space A)"
```

Proof st

```
"{[- (1 / 4), - 1, - (3 / 4), 1]}"
  :: "rat list set"
```

Solving a system of linear equations

$$\begin{aligned} x + y - 4z + 10t &= 24 \\ 3x - 2y - 2z + 6t &= 15 \end{aligned}$$

Solving a system of linear equations

$$\begin{aligned} x + y - 4z + 10t &= 24 \\ 3x - 2y - 2z + 6t &= 15 \end{aligned}$$

$$\begin{pmatrix} x \\ y \\ z \\ t \end{pmatrix} = \begin{pmatrix} 63/5 \\ 57/5 \\ 0 \\ 0 \end{pmatrix} + \alpha \begin{pmatrix} 2 \\ 2 \\ 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} -26/5 \\ -24/5 \\ 0 \\ 1 \end{pmatrix}$$

Solving a system of linear equations

$$\begin{aligned} x + y - 4z + 10t &= 24 \\ 3x - 2y - 2z + 6t &= 15 \end{aligned}$$

$$\begin{pmatrix} x \\ y \\ z \\ t \end{pmatrix} = \begin{pmatrix} 63/5 \\ 57/5 \\ 0 \\ 0 \end{pmatrix} + \alpha \begin{pmatrix} 2 \\ 2 \\ 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} -26/5 \\ -24/5 \\ 0 \\ 1 \end{pmatrix}$$

```
value "let A = (list_of_list_to_matrix [[1,1,-4,10],[3,-2,-2,6]]::rat^4^2);
            b=(list_to_vec [24,15]::rat^2)
            in (print_result_solve (solve A b))"
```

Proof state | Auto update | Update | Search:

```
"Some ([63 / 5, 57 / 5, 0, 0], {[2, 2, 1, 0], [- (26 / 5), - (24 / 5), 0, 1]})"
  :: "(rat list × rat list set) option"
```

Benchmarks (using iarrays)

Size (n)	Poly/ML	GHC
100	0.04	0.36
200	0.25	2.25
300	0.85	9.09
400	2.01	17.17
500	3.90	32.56
600	6.16	56.39
800	15.96	131.73
1 000	32.08	255.84
1 200	62.33	453.57
1 400	97.16	715.87
1 600	139.70	1097.41
1 800	203.10	1609.72
2 000	284.28	2295.30

Table : Time to compute the *rref* of randomly generated $\mathbb{Z}_2$ matrices.

Imperative vs. Declarative

Imperative version	(HOL-Imp)	Verified version	(iarray)
Function	**Time perc.**	**Function**	**Time perc.**
nth.fn	29.8%	sub	33.4%
upd.fn.fn.fn	12.2%	of_fun	32.7%
IntInf.schckToInt64	12.1%	IntInf.extdFromWord64	9.3%
make.fn	8.1%	IntInf.schckToInt64	7.5%
plus_nat.fn	7.9%	row_add_iarray.fn	6.3%
...	...	...	...
Total			
9.42 seconds of CPU time		10.06 seconds of CPU time	
(0.04 seconds of GC)		(0.22 seconds of GC)	

Table : Profiling of the imperative and verified versions of Gauss-Jordan on a 600×600 matrix.

C++ vs. Verified version

Matrix sizes	**C++ version**	**Verified version**
600 × 600	01.33s.	06.16s.
1 000 × 1 000	05.94s.	32.08s.
1 200 × 1 200	10.28s.	62.33s.
1 400 × 1 400	16.62s.	97.16s.

Table : C++ vs verified version of the Gauss-Jordan algorithm.

Both programs show a cubic performance, even if the verified version is using immutable arrays

J. Aransay and J. Divasón. *Formalization and execution of Linear Algebra: from theorems to algorithms*. Proceedings of the International Symposium on Logic-Based Program Synthesis and Transformation: LOPSTR 2013

J. Aransay and J. Divasón. *Formalisation in higher-order logic and code generation to functional languages of the Gauss-Jordan algorithm*. Journal of Functional Programming. 2015

Theorem (Second Part of the Fundamental Theorem of Linear Algebra)

Given a matrix $A \in M_{(m,n)}(\mathbb{R})$

- *In* $\mathbb{R}^n$, $\mathrm{N}(A) = \mathrm{C}(A^T)^\perp$ *that is, the nullspace is the orthogonal complement of the row space*

Theorem (Second Part of the Fundamental Theorem of Linear Algebra)

Given a matrix $A \in M_{(m,n)}(\mathbb{R})$

- *In* $\mathbb{R}^n$, $\mathrm{N}(A) = \mathrm{C}(A^T)^{\perp}$ *that is, the nullspace is the orthogonal complement of the row space*
- *In* $\mathbb{R}^m$, $\mathrm{N}(A^T) = \mathrm{C}(A)^{\perp}$, *that is, the left nullspace is the orthogonal complement of the column space*

Second Part of the Fundamental Theorem of Linear Algebra

- **theorem** null-space-orthogonal-complement-row-space:
  fixes A :: real^'cols^'rows
  shows null-space A = orthogonal-complement (row-space A)

Second Part of the Fundamental Theorem of Linear Algebra

- **theorem** null-space-orthogonal-complement-row-space:
  fixes A :: real^'cols^'rows
  shows null-space A = orthogonal-complement (row-space A)
- **theorem** left-null-space-orthogonal-complement-col-space:
  fixes A :: real^'cols^'rows
  shows left-null-space A = orthogonal-complement (col-space A)

Second Part of the Fundamental Theorem of Linear Algebra

- **theorem** null-space-orthogonal-complement-row-space:
  fixes A :: real^'cols^'rows
  shows null-space A = orthogonal-complement (row-space A)
- **theorem** left-null-space-orthogonal-complement-col-space:
  fixes A :: real^'cols^'rows
  shows left-null-space A = orthogonal-complement (col-space A)

From mathematical results to algorithms

The *Gram-Schmidt process* allows us to compute the mentioned orthogonal bases

QR Decomposition

Definition (QR Decomposition)

The QR decomposition of a full column rank matrix $A \in M_{n\times m}(\mathbb{R})$ is a pair of matrices (Q, R) such that

1. $A = QR$
2. $Q \in M_{n\times m}(\mathbb{R})$ is a matrix whose columns are orthonormal vectors
3. $R \in M_{m\times m}(\mathbb{R})$ is upper triangular and invertible

QR Decomposition

Definition (QR Decomposition)

The QR decomposition of a full column rank matrix $A \in M_{n\times m}(\mathbb{R})$ is a pair of matrices (Q, R) such that

1. $A = QR$
2. $Q \in M_{n\times m}(\mathbb{R})$ is a matrix whose columns are orthonormal vectors
3. $R \in M_{m\times m}(\mathbb{R})$ is upper triangular and invertible

Algorithm

1. $Q =$ Apply Gram-Schmidt to the columns of A, normalise the vectors
2. Compute R as $R = Q^T A$

QR Decomposition

- We have formalised the previous algorithm in Isabelle, and refined it to immutable arrays
- Computations can be carried out using either floats or (for suitable inputs) symbolically
- 2700 vs. 11000 *loc.*

QR Decomposition

$$\overbrace{\begin{pmatrix} 1 & 2 & 6 \\ 9 & 4 & 2 \\ 0 & 0 & 4 \end{pmatrix}}^{\text{A}} =$$

QR Decomposition

$$\overbrace{\begin{pmatrix} 1 & 2 & 6 \\ 9 & 4 & 2 \\ 0 & 0 & 4 \end{pmatrix}}^{\text{A}} = \overbrace{\begin{pmatrix} \frac{\sqrt{82}}{82} & \frac{9\sqrt{82}}{82} & 0 \\ \frac{9\sqrt{82}}{82} & \frac{-\sqrt{82}}{82} & 0 \\ 0 & 0 & 1 \end{pmatrix}}^{\text{Q}}$$

QR Decomposition

$$\underbrace{\begin{pmatrix}1 & 2 & 6\\ 9 & 4 & 2\\ 0 & 0 & 4\end{pmatrix}}_{A} = \underbrace{\begin{pmatrix}\frac{\sqrt{82}}{82} & \frac{9\sqrt{82}}{82} & 0\\ \frac{9\sqrt{82}}{82} & \frac{-\sqrt{82}}{82} & 0\\ 0 & 0 & 1\end{pmatrix}}_{Q}\underbrace{\begin{pmatrix}\sqrt{82} & \frac{19\sqrt{82}}{41} & \frac{12\sqrt{82}}{41}\\ 0 & \frac{7\sqrt{82}}{41} & \frac{26\sqrt{82}}{41}\\ 0 & 0 & 4\end{pmatrix}}_{R}$$

QR Decomposition

$$\overbrace{\begin{pmatrix} 1 & 2 & 6 \\ 9 & 4 & 2 \\ 0 & 0 & 4 \end{pmatrix}}^{A} = \overbrace{\begin{pmatrix} \frac{\sqrt{82}}{82} & \frac{9\sqrt{82}}{82} & 0 \\ \frac{9\sqrt{82}}{82} & \frac{-\sqrt{82}}{82} & 0 \\ 0 & 0 & 1 \end{pmatrix}}^{Q} \overbrace{\begin{pmatrix} \sqrt{82} & \frac{19\sqrt{82}}{41} & \frac{12\sqrt{82}}{41} \\ 0 & \frac{7\sqrt{82}}{41} & \frac{26\sqrt{82}}{41} \\ 0 & 0 & 4 \end{pmatrix}}^{R}$$

```
value "let A = list_of_list_to_matrix
                 [[1,2,6],
                  [9,4,2],
                  [0,0,4]]::real^3^3 in
                  show_matrix (fst (QR_decomposition A))"
```

Proof state | Auto update | Update

```
"[[''1/82*sqrt(82)'', ''9/82*sqrt(82)'', ''0''],
  [''9/82*sqrt(82)'', ''-1/82*sqrt(82)'', ''0''], [''0'', ''0'', ''1'']]"
  :: "char list list list"
```

Application: Least Squares Approximation

- Let us consider a system $Ax = b$ without solution
- We can approximate the "solution" minimizing the error (least squares approximation). That is, compute $\hat{x}$ such that minimises $|| A\hat{x} - b ||$

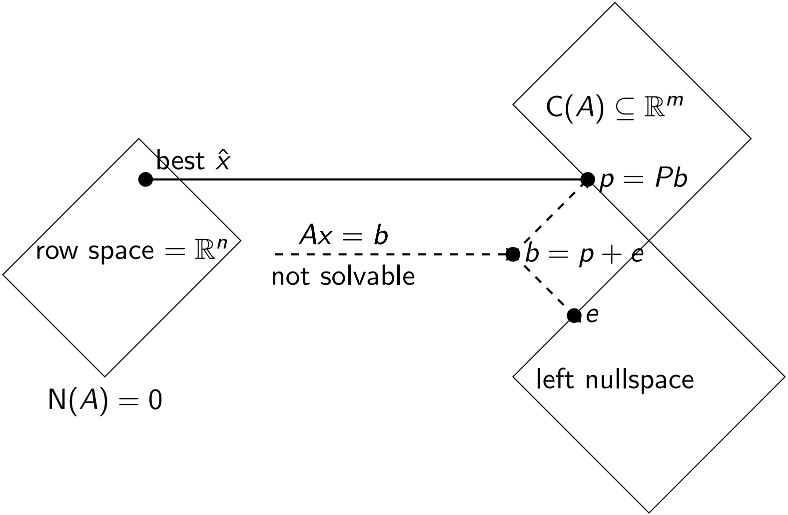


Figure : The projection $p = A\hat{x}$ is the closest point to b in $C(A)$

Application: Least Squares Approximation

- We have formalised that $\hat{x} = R^{-1}Q^{T}b$
- $\hat{x}$ can be computed symbolically, R^{-1} is computed by means of the Gauss-Jordan algorithm

Advantages over Gauss-Jordan

- Both Gauss-Jordan and *QR* can be used to compute the least squares approximation of linear systems
- *QR* has a substantial edge in precision, when applied to floating-point matrices

Example of *QR* precision over the Hilbert matrix of dimension 6

Let

$$H_6 = \begin{pmatrix} 1 & 1/2 & 1/3 & 1/4 & 1/5 & 1/6 \\ 1/2 & 1/3 & 1/4 & 1/5 & 1/6 & 1/7 \\ 1/3 & 1/4 & 1/5 & 1/6 & 1/7 & 1/8 \\ 1/4 & 1/5 & 1/6 & 1/7 & 1/8 & 1/9 \\ 1/5 & 1/6 & 1/7 & 1/8 & 1/9 & 1/10 \\ 1/6 & 1/7 & 1/8 & 1/9 & 1/10 & 1/11 \end{pmatrix}$$

and $b = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 5 \end{pmatrix}$ in $H_6 x = b$. Do note that the determinant of H_6 is $1/186313420339200000$ and its condition number greater than 10^7

Advantages over Gauss-Jordan

Comparison of the approximations to $H_6 x = b$

- 1: Least squares approximation using arbitrary precision (*QR* or Gauss-Jordan algorithm)
- 2: *QR* approximation using floating-point numbers
- 3: Gauss-Jordan approximation using floating-point numbers

1 : −13824	415170	−2907240	7754040	−8724240	3489948
2 : −13824.0	415170.0001	−2907240.0	7754040.001	−8724240.001	3489948.0
3 : −13808.6421	414731.7866	−2904277.468	7746340.301	−8715747.432	3486603.907

Benchmarks

Size (n)	**Poly/ML** (s.)
100	0.748
200	10.869
300	84.310
400	183.754

Table : Elapsed time (in seconds) to compute the QR decomposition of H_n with floating-point precision

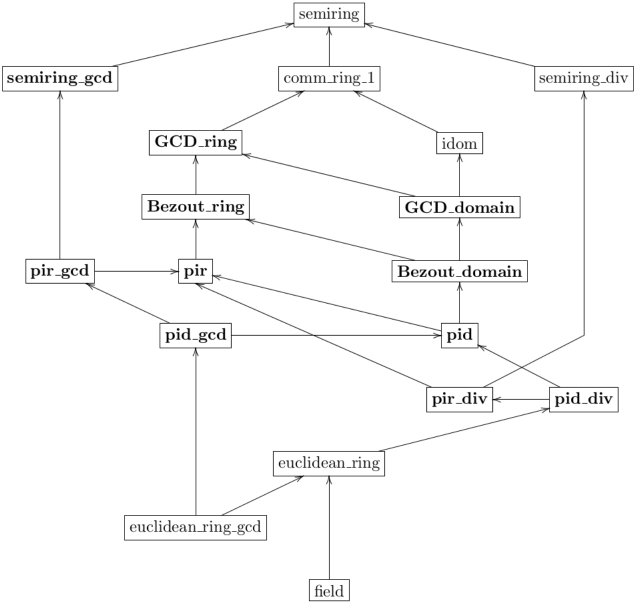
J. Divasón and J. Aransay. *QR Decomposition*. Archive of Formal Proofs. 2015

J. Aransay and J. Divasón. *A formalisation in HOL of the Fundamental Theorem of Linear Algebra and its application to the solution of the least squares problem*. Journal of Automated Reasoning. 2016

J. Aransay and J. Divasón. *Verified Computer Linear Algebra*. EACA 2016

Echelon Form

- Gauss-Jordan algorithm can only be applied to matrices whose elements belong to a field. For more general rings, a different algorithm must be used (involving *gcd*, *Bézout coefficients. . .*)
- We have formalised and refined an algorithm to compute the echelon form of a matrix over Bézout domains


semiring
semiring_gcd
comm_ring_1
semiring_div
GCD_ring
idom
Bezout_ring
GCD_domain
pir_gcd
pir
Bezout_domain
pid_gcd
pid
pir_div
pid_div
euclidean_ring
euclidean_ring_gcd
field

- We have proven the correctness of the algorithm in Bézout domains, where Bézout coefficients exist for every a, b, (*i.e.*, $\exists x\, y.\, ax + by = z$, $z \in Units$), but a computable *Bézout* function might not exist

- We have proven the correctness of the algorithm in Bézout domains, where Bézout coefficients exist for every a, b, ($i.e.$, $\exists x\, y.\, ax + by = z$, $z \in Units$), but a computable *Bézout* function might not exist
- Execution is guaranteed, at least, over Euclidean domains, where a computable *Bézout* operation exists (it might be not unique)

- We have proven the correctness of the algorithm in Bézout domains, where Bézout coefficients exist for every a, b, ($i.e.$, $\exists x\, y.\, ax + by = z$, $z \in Units$), but a computable *Bézout* function might not exist
- Execution is guaranteed, at least, over Euclidean domains, where a computable *Bézout* operation exists (it might be not unique)
- The echelon form algorithm is parametrised by a *Bézout* function

- We have proven the correctness of the algorithm in Bézout domains, where Bézout coefficients exist for every a, b, ($i.e.$, $\exists x\, y.\, ax + by = z$, $z \in Units$), but a computable *Bézout* function might not exist
- Execution is guaranteed, at least, over Euclidean domains, where a computable *Bézout* operation exists (it might be not unique)
- The echelon form algorithm is **parametrised** by a *Bézout* function

- We have proven the correctness of the algorithm in Bézout domains, where Bézout coefficients exist for every a, b, ($i.e.$, $\exists x\, y.\, ax + by = z$, $z \in Units$), but a computable *Bézout* function might not exist
- Execution is guaranteed, at least, over Euclidean domains, where a computable *Bézout* operation exists (it might be not unique)
- The echelon form algorithm is **parametrised** by a *Bézout* function
- $\mathbb{Z}$ and $\mathcal{F}[x]$ are proven to be instances of Euclidean domains

- We have proven the correctness of the algorithm in Bézout domains, where Bézout coefficients exist for every a, b, ($i.e.$, $\exists x\, y.\, ax + by = z$, $z \in Units$), but a computable *Bézout* function might not exist
- Execution is guaranteed, at least, over Euclidean domains, where a computable *Bézout* operation exists (it might be not unique)
- The echelon form algorithm is **parametrised** by a *Bézout* function
- $\mathbb{Z}$ and $\mathcal{F}[x]$ are proven to be instances of Euclidean domains
- The following computations can be carried out in Euclidean domains
  - Determinants
  - Inverses
  - Characteristic polynomial

- We have proven the correctness of the algorithm in Bézout domains, where Bézout coefficients exist for every a, b, ($i.e.$, $\exists x\, y.\, ax + by = z$, $z \in \mathit{Units}$), but a computable *Bézout* function might not exist
- Execution is guaranteed, at least, over Euclidean domains, where a computable *Bézout* operation exists (it might be not unique)
- The echelon form algorithm is **parametrised** by a *Bézout* function
- $\mathbb{Z}$ and $\mathcal{F}[x]$ are proven to be instances of Euclidean domains
- The following computations can be carried out in Euclidean domains
  - Determinants
  - Inverses
  - Characteristic polynomial
- 5000 vs. 11000 *loc*

Statement for Bézout domains:

theorem echelon-form-of-invertible:
  fixes A::'a::{bezout-domain}^'cols::{mod-type}^'rows::{mod-type}
  assumes is-bezout-ext bezout
  shows ∃P. invertible P ∧ P ** A = echelon-form-of A bezout
      ∧ echelon-form (echelon-form-of A bezout)

Statement for Bézout domains:

theorem echelon-form-of-invertible:
  fixes A::'a::{bezout-domain}^'cols::{mod-type}^'rows::{mod-type}
  assumes is-bezout-ext bezout
  shows $\exists$P. invertible P $\wedge$ P ** A = echelon-form-of A bezout
    $\wedge$ echelon-form (echelon-form-of A bezout)

Statement for Euclidean domains:

corollary echelon-form-of-euclidean-invertible:
  fixes A::'a::{euclidean-ring}^'cols::{mod-type}^'rows::{mod-type}
  shows $\exists$P. invertible P $\wedge$ P**A = (echelon-form-of A euclid-ext2)
    $\wedge$ echelon-form (echelon-form-of A euclid-ext2)

Statement for Bézout domains:

theorem echelon-form-of-invertible:
  fixes A::'a::{bezout-domain}^'cols::{mod-type}^'rows::{mod-type}
  assumes is-bezout-ext bezout
  shows $\exists$P. invertible P $\wedge$ P ** A = echelon-form-of A bezout
    $\wedge$ echelon-form (echelon-form-of A bezout)

Statement for Euclidean domains:

corollary echelon-form-of-euclidean-invertible:
  fixes A::'a::{euclidean-ring}^'cols::{mod-type}^'rows::{mod-type}
  shows $\exists$P. invertible P $\wedge$ P**A = (echelon-form-of A euclid-ext2)
    $\wedge$ echelon-form (echelon-form-of A euclid-ext2)

J. Divasón and J. Aransay. *Echelon Form*. Archive of Formal Proofs. 2015

J. Aransay and J. Divasón. *Formalisation of the Computation of the Echelon Form of a Matrix in Isabelle/HOL*. Formal Aspects of Computing. 2016

Determinant

$$A = \begin{pmatrix} -5x^2+4x+1 & x & -3x^2 \\ 4x-2 & 0 & -x+2 \\ 4x-1 & 3x & 4x^3 \end{pmatrix} \in \mathcal{M}_{3\times 3}(\mathbb{R}[x])$$

Determinant

$$A = \begin{pmatrix} -5x^2+4x+1 & x & -3x^2 \\ 4x-2 & 0 & -x+2 \\ 4x-1 & 3x & 4x^3 \end{pmatrix} \in \mathcal{M}_{3\times 3}(\mathbb{R}[x])$$

```
value "det (list_of_list_to_matrix
          [[[:1,4,-5:],[:0,1:],[:0,0,-3:]],
           [[:-2,4:],[:0:],[:2,-1:]],
           [[:-1,4:],[:0,3:],[:0,0,0,4:]]]::real poly^3^3)"
```

Proof state Auto update Update Search:

```
"[:0, - 8, - 12, 56, - 43, - 16:]"
  :: "real poly"
```

Determinant

$$A = \begin{pmatrix} -5x^2+4x+1 & x & -3x^2 \\ 4x-2 & 0 & -x+2 \\ 4x-1 & 3x & 4x^3 \end{pmatrix} \in \mathcal{M}_{3\times 3}(\mathbb{R}[x])$$

```
value "det (list_of_list_to_matrix
          [[[:1,4,-5:],[:0,1:],[:0,0,-3:]],
           [[:-2,4:],[:0:],[:2,-1:]],
           [[:-1,4:],[:0,3:],[:0,0,0,4:]]]::real poly^3^3)"
```

Proof state | Auto update | Update | Search:

```
"[:0, - 8, - 12, 56, - 43, - 16:]"
  :: "real poly"
```

$$det(A) = -16x^5 - 43x^4 + 56x^3 - 12x^2 - 8x$$

Inverse

$$A = \begin{pmatrix} 1 & -2 & 4 \\ 1 & -1 & 1 \\ 0 & 1 & -2 \end{pmatrix} \in \mathcal{M}_{3\times 3}(\mathbb{Z})$$

$$B = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \in \mathcal{M}_{3\times 3}(\mathbb{Z})$$

```
value "let A = (list_of_list_to_matrix
          [[1,-2,4],[1,-1,1],[0,1,-2]]::int^3^3)
          in show_matrix (inverse_matrix A)"

Proof state  Auto update  Update
"Some [[1, 0, 2], [2, - 2, 3], [1, - 1, 1]]"
  :: "int list list option"
```

```
value "let A = (list_of_list_to_matrix
          [[3,0,0],[0,1,0],[0,0,1]]::int^3^3)
          in show_matrix (inverse_matrix A)"

Proof state  Auto update  Update  Search:
"None"
  :: "int list list option"
```

$$inv(A) = \begin{pmatrix} 1 & 0 & 2 \\ 2 & -2 & 3 \\ 1 & -1 & 1 \end{pmatrix}$$

$$\nexists inv(B)$$

Characteristic polynomial

$$A = \begin{pmatrix} 3 & 5 & 1 \\ 2 & 1 & 3 \\ 1 & 2 & 1 \end{pmatrix} \in \mathcal{M}_{3\times 3}(\mathbb{R})$$

```
value "let A = (list_of_list_to_matrix
          [[3,5,1],[2,1,3],[1,2,1]]::real^3^3)
           in charpoly A"
```

Proof state | Auto update | Update | Search:

```
"[:7, - 10, - 5, 1:]"
  :: "real poly"
```

Characteristic polynomial

$$A = \begin{pmatrix} 3 & 5 & 1 \\ 2 & 1 & 3 \\ 1 & 2 & 1 \end{pmatrix} \in \mathcal{M}_{3\times 3}(\mathbb{R})$$

```
value "let A = (list_of_list_to_matrix
          [[3,5,1],[2,1,3],[1,2,1]]::real^3^3)
           in charpoly A"
```

```
"[:7, - 10, - 5, 1:]"
  :: "real poly"
```

$$charpoly(A) = x^3 - 5x^2 - 10x + 7$$

Hermite normal form

Definition (Hermite normal form)

A matrix H is said to be the Hermite normal form of a given matrix A with elements in a Bézout ring iff:

1. H is in echelon form;
2. the first nonzero element of a nonzero row belongs to the complete set of *nonassociates*;
3. Let h be the first nonzero element of a nonzero row; each element above h belongs to the corresponding complete set of *residues* of h;
4. There exists an invertible matrix P such that $A = PH$;

Hermite normal form

Definition (Hermite normal form)

A matrix H is said to be the Hermite normal form of a given matrix A with elements in a Bézout ring iff:

1. H is in echelon form;
2. the first nonzero element of a nonzero row belongs to the complete set of *nonassociates*;
3. Let h be the first nonzero element of a nonzero row; each element above h belongs to the corresponding complete set of *residues* of h;
4. There exists an invertible matrix P such that $A = PH$;

The Hermite normal form is unique, up to the sets of *nonassociates* and *residues*, which in our work are parameters of the *Hermite* operation.

Hermite normal form

Definition (Hermite normal form)

A matrix H is said to be the Hermite normal form of a given matrix A with elements in a Bézout ring iff:

1. H is in echelon form;
2. the first nonzero element of a nonzero row belongs to the complete set of *nonassociates*;
3. Let h be the first nonzero element of a nonzero row; each element above h belongs to the corresponding complete set of *residues* of h;
4. There exists an invertible matrix P such that $A = PH$;

The Hermite normal form is unique, up to the sets of *nonassociates* and *residues*, which in our work are **parameters** of the *Hermite* operation.

Hermite normal form

lemma Hermite-unique:
  fixes K::'a::bezout-ring-div^'n::mod-type^'n::mod-type
  assumes A = P ** H **and** A = Q ** K
  and invertible A
  and invertible P **and** invertible Q
  and Hermite associates residues H
  and Hermite associates residues K
  shows H = K

J. Divasón and J. Aransay. Hermite Normal Form. Archive of Formal Proofs. 2016

Univalent Foundations

Mathematicians' lives are about to change. Soon enough, they're going to find themselves doing mathematics at the computer, with the aid of computer proof assistants. Soon, they won't consider a theorem proven until a computer has verified it. Soon, they'll be able to collaborate freely, even with mathematicians whose skills they don't have confidence in. And soon, they'll understand the foundations of mathematics very differently.

— Vladimir Voevodsky

- Active area of research presented as a new foundation of Mathematics
- Homotopy type theory is an attempt to formally redefine the whole mathematical behaviour in a way that is both much closer to how informal mathematics is actually done and to how mathematics should be implemented to be computationally checkable.

It makes sense to implement the model in an interactive theorem prover
Approach: try to reuse as many existing Isabelle/HOL libraries as possible

A piece of Voevodsky's simplicial model

Definition

Given a simplicial set X we define $\mathbf{W}(X)$ to be the set of isomorphism classes of well-ordered morphisms $f : Y \to X$. Given a morphism $t : X' \to X$ we define $\mathbf{W}(t) : \mathbf{W}(X) \to \mathbf{W}(X')$ by $\mathbf{W}(t) = t^*$ (the pullback functor). This provides a functor $\mathbf{W} : sSet^{op} \to Set$.

Definition

$$W := \mathbf{W} \circ y^{op} : \Delta^{op} \to Set$$

where y denotes the Yoneda embedding $y : \Delta \to sSet$.

Definition

Given a simplicial set X we define $\mathbf{W}(X)$ to be the set of **isomorphism classes** of well-ordered morphisms $f : Y \to X$. Given a morphism $t : X' \to X$ we define $\mathbf{W}(t) : \mathbf{W}(X) \to \mathbf{W}(X')$ by $\mathbf{W}(t) = t^*$ (the pullback functor). This provides a functor $\mathbf{W} : sSet^{op} \to Set$.

Definition

$$W := \mathbf{W} \circ y^{op} : \Delta^{op} \to Set$$

where y denotes the Yoneda embedding $y : \Delta \to sSet$.

1. Quotient sets

Definition

Given a simplicial set X we define $\mathbf{W}(X)$ to be the set of isomorphism classes of well-ordered morphisms $f : Y \to X$. Given a morphism $t : X' \to X$ we define $\mathbf{W}(t) : \mathbf{W}(X) \to \mathbf{W}(X')$ by $\mathbf{W}(t) = t^*$ (the **pullback** functor). This provides a functor $\mathbf{W} : sSet^{op} \to Set$.

Definition

$$W := \mathbf{W} \circ y^{op} : \Delta^{op} \to Set$$

where y denotes the Yoneda embedding $y : \Delta \to sSet$.

1. Quotient sets
2. Pullback

Definition

Given a simplicial set X we define $\mathbf{W}(X)$ to be the set of isomorphism classes of well-ordered morphisms $f : Y \to X$. Given a morphism $t : X' \to X$ we define $\mathbf{W}(t) : \mathbf{W}(X) \to \mathbf{W}(X')$ by $\mathbf{W}(t) = t^*$ (the pullback **functor**). This provides a functor $\mathbf{W} : sSet^{op} \to Set$.

Definition

$$W := \mathbf{W} \circ y^{op} : \Delta^{op} \to Set$$

where y denotes the Yoneda embedding $y : \Delta \to sSet$.

1. Quotient sets
2. Pullback
3. Functors and categories

Definition

Given a simplicial set X we define $\mathbf{W}(X)$ to be the set of isomorphism classes of well-ordered morphisms $f : Y \to X$. Given a morphism $t : X' \to X$ we define $\mathbf{W}(t) : \mathbf{W}(X) \to \mathbf{W}(X')$ by $\mathbf{W}(t) = t^*$ (the pullback functor). This provides a functor $\mathbf{W} : \mathbf{sSet}^{op} \to Set$.

Definition

$$W := \mathbf{W} \circ y^{op} : \Delta^{op} \to Set$$

where y denotes the Yoneda embedding $y : \Delta \to sSet$.

1. Quotient sets
2. Pullback
3. Functors and categories
4. sSet

Definition

Given a simplicial set X we define $\mathbf{W}(X)$ to be the set of isomorphism classes of well-ordered morphisms $f : Y \to X$. Given a morphism $t : X' \to X$ we define $\mathbf{W}(t) : \mathbf{W}(X) \to \mathbf{W}(X')$ by $\mathbf{W}(t) = t^*$ (the pullback functor). This provides a functor $\mathbf{W} : sSet^{\mathbf{op}} \to Set$.

Definition

$$W := \mathbf{W} \circ y^{op} : \Delta^{op} \to Set$$

where y denotes the Yoneda embedding $y : \Delta \to sSet$.

1. Quotient sets
2. Pullback
3. Functors and categories
4. sSet
5. Op category

Definition

Given a simplicial set X we define $\mathbf{W}(X)$ to be the set of isomorphism classes of well-ordered morphisms $f : Y \to X$. Given a morphism $t : X' \to X$ we define $\mathbf{W}(t) : \mathbf{W}(X) \to \mathbf{W}(X')$ by $\mathbf{W}(t) = t^*$ (the pullback functor). This provides a functor $\mathbf{W} : sSet^{op} \to \mathbf{Set}$.

Definition

$$W := \mathbf{W} \circ y^{op} : \Delta^{op} \to Set$$

where y denotes the Yoneda embedding $y : \Delta \to sSet$.

1. Quotient sets
2. Pullback
3. Functors and categories
4. sSet
5. Op category
6. Set category

Definition

Given a simplicial set X we define $\mathbf{W}(X)$ to be the set of isomorphism classes of well-ordered morphisms $f : Y \to X$. Given a morphism $t : X' \to X$ we define $\mathbf{W}(t) : \mathbf{W}(X) \to \mathbf{W}(X')$ by $\mathbf{W}(t) = t^*$ (the pullback functor). This provides a functor $\mathbf{W} : sSet^{op} \to Set$.

Definition

$$W := \mathbf{W} \circ y^{op} : \mathbf{\Delta}^{op} \to Set$$

where y denotes the Yoneda embedding $y : \Delta \to sSet$.

1. Quotient sets
2. Pullback
3. Functors and categories
4. sSet
5. Op category
6. Set category
7. Δ category

Definition

Given a simplicial set X we define $\mathbf{W}(X)$ to be the set of isomorphism classes of well-ordered morphisms $f : Y \to X$. Given a morphism $t : X' \to X$ we define $\mathbf{W}(t) : \mathbf{W}(X) \to \mathbf{W}(X')$ by $\mathbf{W}(t) = t^*$ (the pullback functor). This provides a functor $\mathbf{W} : sSet^{op} \to Set$.

Definition

$$W := \mathbf{W} \circ y^{op} : \Delta^{op} \to Set$$

where y denotes the **Yoneda embedding** $y : \Delta \to sSet$.

1. Quotient sets
2. Pullback
3. Functors and categories
4. sSet
5. Op category
6. Set category
7. Δ category
8. Yoneda embedding

Definition

Given a simplicial set X we define $\mathbf{W}(X)$ to be the set of isomorphism classes of well-ordered morphisms $f : Y \to X$. Given a morphism $t : X' \to X$ we define $\mathbf{W}(t) : \mathbf{W}(X) \to \mathbf{W}(X')$ by $\mathbf{W}(t) = t^*$ (the pullback functor). This provides a functor $\mathbf{W} : sSet^{op} \to Set$.

Definition

$$W := \mathbf{W} \circ y^{op} : \Delta^{op} \to Set$$

where y denotes the Yoneda embedding $y : \Delta \to sSet$.

1. Quotient sets ✓
2. Pullback ✗
3. Functors and categories ✓
4. sSet ✗
5. Op category ✗
6. Set category ✓
7. Δ category ✗
8. Yoneda embedding ✗

Definition

Given a simplicial set X we define $\mathbf{W}(X)$ to be the set of isomorphism classes of well-ordered morphisms $f : Y \to X$. Given a morphism $t : X' \to X$ we define $\mathbf{W}(t) : \mathbf{W}(X) \to \mathbf{W}(X')$ by $\mathbf{W}(t) = t^*$ (the pullback functor). This provides a functor $\mathbf{W} : \mathit{osSet}^{op} \to \mathit{Set}$.

- We must show that $\mathbf{W} : \mathit{osSet}^{op} \to \mathit{Set}$ is a functor in Isabelle/HOL.
- Among other things, we have to prove that $\mathbf{W}(t) : \mathbf{W}(X) \to \mathbf{W}(X')$ is an arrow in *a* *Set*-category implemented in Isabelle/HOL.

PROBLEM

Set category in HOL

```
record 'c set-arrow =
  set-dom :: 'c set
  set-func :: 'c ⇒ 'c
  set-cod :: 'c set

definition
 set-arrow :: ['c set, 'c set-arrow] ⇒ bool where
  set-arrow U f ⟷ set-dom f ⊆ U
    ∧ set-cod f ⊆ U
    ∧ set-func f ∈ (set-dom f) → (set-cod f)
    ∧ set-func f ∈ extensional (set-dom f)
```

```
definition
  set-cat :: 'c set ⇒ ('c set, 'c set-arrow) category where
  set-cat U =
  (|
    ob = Pow U,
    ar = {f. set-arrow U f},
    dom = set-dom,
    cod = set-cod,
    id = set-id U,
    comp = set-comp
  |)
```

- The variable set U will fix the underlying type $'c$ of the category, since its objects will be subsets of U.
- In fact, this corresponds to what is sometimes called **Ens**, "the category of all sets and functions within a (variable) set U", which is a *small* category.

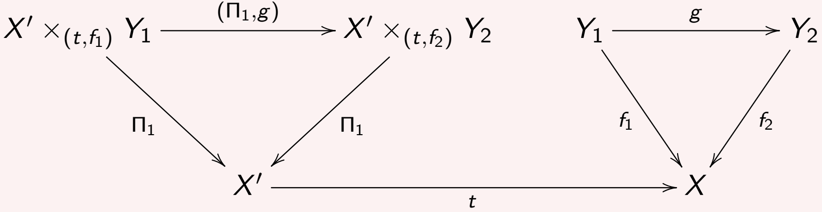
Example

Let $A = \{1, 2, 3\}$ be a set of natural numbers and $B = \{True, False\}$ a boolean set. Then, the following function would belong to the Set-category (mathematically speaking) but not to the corresponding implementation in Isabelle/HOL:

$$\begin{aligned} f : A &\longrightarrow B \\ 1 &\longrightarrow True \\ 2 &\longrightarrow True \\ 3 &\longrightarrow False \end{aligned}$$

Definition (Pullback on morphisms)

Let X', X, Y_1, Y_2 be simplicial sets, $f_1 : Y_1 \to X$ and $f_2 : Y_2 \to X$ well-ordered morphisms, $t : X' \to X$ a morphism and $g : Y_1 \to Y_2$ an isomorphism between the well-ordered morphisms f_1 and f_2. Then, the pullback on morphisms is defined as follows:



SOLUTION?

SOLUTION?

Use another logic: HOLZF (HOL + ZF)

The definition of the Set-category in Isabelle/HOLZF is the following one:

definition
SET′ :: (ZF, ZF) Category **where**
SET′ ≡ (|
  Category.Obj = {x . True} ,
  Category.Mor = {f . isZFfun f} ,
  Category.Dom = ZFfunDom ,
  Category.Cod = ZFfunCod ,
  Category.Id = λx. ZFfun x x (λx . x) ,
  Category.Comp = ZFfunComp
|)

definition SET ≡ MakeCat SET′

- Objects and arrows are of the same type
- **Products are also of type ZF**

Let Y_1, Y_2 and X be simplicial sets together with ∂_{Y_1}, s_{Y_1}, ∂_{Y_2}, s_{Y_2}, ∂_X and s_X as the corresponding face and degeneracy operators. Let $t : Y_1 \rightarrow X$ and $f : Y_2 \rightarrow X$ be morphisms. Then the following construction is a simplicial set:

$$Y_1 \times_{(t,f)} Y_2 = \{(y_1, y_2).\ y_1 \in Y_1 \wedge y_2 \in Y_2 \wedge t(y_1) = f(y_2)\}$$

$$\partial_{Y_1 \times_{(t,f)} Y_2} = (\lambda(y_1, y_2) \in Y_1 \times_{(t,f)} Y_2.\ (\partial_{Y_1}(y_1), \partial_{Y_2}(y_2))$$

$$s_{Y_1 \times_{(t,f)} Y_2} = (\lambda(y_1, y_2) \in Y_1 \times_{(t,f)} Y_2.\ (s_{Y_1}(y_1), s_{Y_2}(y_2))$$

sublocale Y1-times-Y2-tf: simplicial-set
 (λn. Sep (Y1 n |×| Y2 n) (λx. t n (Fst x) = f n (Snd x)))
 (λi n x. Opair (𝔡y1 i n (Fst x)) (𝔡y2 i n (Snd x)))
 (λi n x. Opair (sy1 i n (Fst x)) (sy2 i n (Snd x)))

- We have ported the development to Isabelle/HOLZF
- HOLZF seems to avoid the restriction

State of the art (June 2016) & Related work

Thiemann and Yamada; computation of Jordan Normal Form in Isabelle

R. Thiemann, A. Yamada. Matrices, Jordan Normal Forms, and Spectral Radius Theory. Archive of Formal Proofs. 2015

State of the art (June 2016) & Related work

Thiemann and Yamada; computation of Jordan Normal Form in Isabelle

R. Thiemann, A. Yamada. Matrices, Jordan Normal Forms, and Spectral Radius Theory. Archive of Formal Proofs. 2015

Dénès *et al*; implementation of Smith Normal Form in CoqEAL

M. Dénès, A. Mörtberg, and V. Siles. A refinement-based approach to computational algebra in COQ. Interactive Theorem Proving. 2012

State of the art (June 2016) & Related work

Thiemann and Yamada; computation of Jordan Normal Form in Isabelle

R. Thiemann, A. Yamada. Matrices, Jordan Normal Forms, and Spectral Radius Theory. Archive of Formal Proofs. 2015

Dénès *et al*; implementation of Smith Normal Form in CoqEAL

M. Dénès, A. Mörtberg, and V. Siles. A refinement-based approach to computational algebra in COQ. Interactive Theorem Proving. 2012

Gonthier; implementation of *LUP* decomposition in SSReflect

G. Gonthier. Point-Free, Set-Free Concrete Linear Algebra. Interactive Theorem Proving. 2011

Conclusions (1/2)

- Linear Algebra algorithms can be implemented in HMA (linked to mathematical results)
- Framework for implementing
- Four well-known algorithms have been formalised (almost 40000 *loc*)
- Use of parametrised algorithms
- Side-products: generalisation of HMA, ring theory, serialisations, ...

Conclusions (2/2)

- Algorithms are executable inside of Isabelle
- Better performance can be obtained thanks to code generation in SML and Haskell
- The use of immutable arrays does not pose a drawback, even in comparison to imperative programming
- The generated code is usable in practice
- HOLZF seems to be useful to formalise the simplicial model for Univalent Foundations


thank you!