A Toolbox for Complex Linear Algebra in HOL Light

Sanaz Khan-Afshar, Vincent Aravantinos, Osman Hasan, and Sofiène Tahar

Hardware Verification Group
Department of Computer and Electrical Engineering
Concordia University
Montréal, Québec, Canada
{s_khanaf,vincent,o_hasan,tahar}@ece.concordia.ca

Technical Report
February 2014

Abstract

Linear algebra is a mathematical theory which is essential to both pure and applied mathematics. While many theorem provers support linear spaces, they usually consider only real vector spaces or provide very abstract formalizations which are useful for the formalization of mathematics but lack many notions required for applied mathematics and therefore many other fields, e.g., computer science, engineering, physics, economics, natural sciences, etc. In this paper, we propose a new formalization of complex linear algebras whose objective is to be a toolbox for various engineering applications.

1 Introduction

Linear algebra is essential to many fields of applied mathematics. In engineering, it has applications in mechanics [12], signal processing [10], control [16], etc. Many of these applications require not only real vector spaces but also *complex* vector spaces. We do not know of any formalization which answer our goals in a satisfactory way. In Coq, we know of [14] which provides a general, axiomatic, formalization of linear algebras. However, this work lacks many notions that are useful for applications (e.g., hermitians, self-adjoints) and actually corresponds to the formalization of a classic linear algebra textbook. Even though this is extremely useful for the formalization of mathematics, this does not answer most of the engineering needs that we mentioned earlier. [5] presents a formalization of *real* vector spaces in PVS and applies it to control theory. This work lacks the support of infinite-dimension and of complex vectors. In HOL Light, even though the amount of proved theorems in [4] is impressive, it suffers from the same problems for engineering applications. The work we present in this paper is an extension of [9]: it is more general, covers more concepts, and provides many tools which improve the usability as a toolbox instead of just a list of theorems.

In our formalization, finite-dimension vector spaces are implemented by vectors (in the sense of HOL Light datatypes) and infinite-dimension ones by function spaces [9]. We show some of the formalized definitions and related properties of complex vectors, which are oriented towards an engineering use. We also provide some tactics allowing to automate some common theorem proving tasks. The rest of the paper is organized as follows: Section 2 presents our formalizations of complex vector spaces, in finite dimension. Section 3 makes use of higher-order rewriting to derive some properties of finite-dimension spaces from the ones of infinite-dimension spaces. In Section 4 we conclude the paper.

All the source codes corresponding to the work presented in this paper, including the application, are available at: `http://hvg.ece.concordia.ca/projects/optics/CXVEC.zip`.

2 Finite Dimension Complex Vector Spaces

For finite dimension vector spaces, we make use of the HOL Light built-in type of *vector*s. Note that this type does not denote, in general, elements of a vector space (contrarily to what their name suggests). Namely, vectors in HOL-Light are basically lists whose length is explicitly encoded in the type. Whereas a vector space $\mathtt{V}$ over field $\mathtt{F}$ is a set [15] with two operations, vector addition $(+ : \mathtt{V} \times \mathtt{V} \to \mathtt{V})$ and scalar multiplication $(\cdot : \mathtt{F} \times \mathtt{V} \to \mathtt{V})$, satisfying eight axioms, specified in Table 1. Therefore, we first define addition and scalar multiplication over $\mathbb{C}^{\mathtt{N}}$ as well as

the identity element of addition:

Definition 2.1 (Arithmetics over $\texttt{complex}^{\texttt{N}}$)

$\vdash$ $\texttt{cvector\_add} = \texttt{vector\_map2}\ (+ : \mathbb{C} \to \mathbb{C})$
$\vdash$ $\texttt{cvector\_mul}\ (\texttt{a} : \mathbb{C}) = \texttt{vector\_map}\ ((* : \mathbb{C} \to \mathbb{C})\ \texttt{a})$

where `vector_map` and `vector_map2` are defined as follows:

Definition 2.2 (Unary and binary componentwise operators)

$\vdash\ \forall(\texttt{f} : \texttt{A} \to \texttt{B})(\texttt{v} : \texttt{A}^{\texttt{N}}).\ \texttt{vector\_map f v} =\ \texttt{lambda i.}\ \texttt{f}(\texttt{v\$i}) : \texttt{B}^{\texttt{N}}$

$\vdash\ \forall(\texttt{f} : \texttt{A} \to \texttt{B} \to \texttt{C})(\texttt{v1} : \texttt{A}^{\texttt{N}})(\texttt{v2} : \texttt{B}^{\texttt{N}}).$
$\texttt{vector\_map2 f v1 v2} =\ \texttt{lambda i.}\ \texttt{f}(\texttt{v1\$i})(\texttt{v2\$i}) : \texttt{C}^{\texttt{N}}$

where `v$i` returns the $\texttt{i}^{th}$ component of `v` and `lambda i. f(v$i)` applies the unary operator `f` on each component of `v` and returns the vector which has `f(v$1)` as its first component, `f(v$2)` as its second component, etc.

Property	**Formalization**
Addition associativity	$\vdash\ \forall\ \texttt{u v w.}\ \texttt{u} + \texttt{v} + \texttt{w} = (\texttt{u} + \texttt{v}) + \texttt{w}$
Addition commutativity	$\vdash\ \forall\ \texttt{u v.}\ \texttt{u} + \texttt{v} = \texttt{v} + \texttt{u}$
Addition unit	$\vdash\ \forall\ \texttt{u.}\ \texttt{u} + \texttt{cvector\_zero} = \texttt{u}$
Addition inverse	$\vdash\ \forall\ \texttt{u.}\ \texttt{u} + (-\ -\ \texttt{u}) = \texttt{cvector\_zero}$
Vector distributivity	$\vdash\ \forall\ \texttt{a u v.}\ \texttt{a}\ \%\ (\texttt{u} + \texttt{v}) = \texttt{a}\ \%\ \texttt{u} + \texttt{a}\ \%\ \texttt{v}$
Scalar distributivity	$\vdash\ \forall\ \texttt{a b u.}\ (\texttt{a} + \texttt{b})\ \%\ \texttt{u} = \texttt{a}\ \%\ \texttt{u} + \texttt{b}\ \%\ \texttt{u}$
Mul. associativity	$\vdash\ \forall\ \texttt{a b u.}\ \texttt{a}\ \%\ \texttt{b}\ \%\ \texttt{u} = (\texttt{a} * \texttt{b})\ \%\ \texttt{u}$
Scalar mul. unit	$\vdash\ \forall\ \texttt{u.}\ \texttt{Cx}(\&1)\ \%\ \texttt{u} = \texttt{u}$

Table 1: Vector Space Axioms for Complex Vectors

We developed a tactic, called `CVECTOR_ARITH_TAC`, which is majorly adapted from the tactic `VECTOR_ARITH_TAC` [3] and is able to prove simple arithmetics properties of complex vectors in an automated way. Using this tactic we prove all eight axioms of vector spaces, indicated in Table 1, plus many other basic but useful facts. The symbol (%) and $(--)$, in Table 1, are overloaded by scalar multiplication and complex vector negation, respectively, and `cvector_zero` is complex null vector. Both `cvector_zero` and negation are formalized using componentwise operators. Having all the eight axioms of linear space formalized, we have the complex vector space formalized.

Next, we address notions of the *dot product* and the *cross product*, which are extensively useful in engineering, e.g., mechanics and electromagnetics. The dot product is defined for two complex vectors u, v of dimension n as $\sum_{i=1}^{n} \overline{u}_i v_i$, where

$\overline{x}$ denotes the complex conjugate of x. Note that the dot product is a complex number, and that the fact that the conjugate is used is an important difference with the real case. This is defined in HOL Light as follows:

Definition 2.3 (Complex vector dot product)
$\vdash\ \forall\ (\mathtt{u} : \mathbb{C}^{\mathtt{N}})\ (\mathtt{v} : \mathbb{C}^{\mathtt{N}}).\ \mathtt{u} \cdot \mathtt{v} = \mathtt{vsum}\ (\mathtt{1..dimindex(:N)})\ (\lambda \mathtt{i}.\ \mathtt{cnj}\ (\mathtt{u\$i}) * \mathtt{v\$i})$

where `cnj` denotes the complex conjugate in HOL Light. From dot product, many interesting notions can be defined. Here we present norm, orthogonality and collinearity, which is defined as follows:

Definition 2.4 (Norm, orthogonality and collinearity of complex vectors)

$\vdash \forall \mathtt{u}.\ \mathtt{cnorm\ u} = \mathtt{sqrt}\ (\mathtt{real\_of\_complex}\ (\mathtt{u\ cdot\ u}))$
$\vdash \forall \mathtt{x\ y}.\ \mathtt{orthogonal\ x\ y} \Leftrightarrow \mathtt{x\ cdot\ y} = \mathtt{Cx}(\&0)$
$\vdash \forall \mathtt{x\ y}.\ \mathtt{collinear\_cvectors\ x\ y}\ \Leftrightarrow\ \exists \mathtt{a}.\ (\mathtt{y} = \mathtt{a}\%\mathtt{x})\ \vee\ (\mathtt{x} = \mathtt{a}\%\mathtt{y})$

where `real_of_complex` is a function casting a complex number with no imaginary part into a real number. Many basic properties of dot product, norm, orthogonality and collinearity are proved, e.g., symmetry of orthogonality, preservation of orthogonality by scalar multiplication, etc. We also proved some more involved theorems, e.g., the Pythagorean theorem:

Theorem 2.5 (Pythagorean Theorem)
$\vdash \forall \mathtt{x\ y}.\ \mathtt{orthogonal\ x\ y} \Leftrightarrow\ \mathtt{cnorm2}(\mathtt{x} + \mathtt{y}) = \mathtt{cnorm2\ x} + \mathtt{cnorm2\ y}$

or existence of an orthogonal decomposition for any vector w.r.t. another one:

Theorem 2.6 (Decomposition)
$\vdash\ \forall \mathtt{u\ v}.\ \mathtt{let\ cunit\_v} = \mathtt{inv}\ (\mathtt{Cx}(\mathtt{cnorm\ v}))\ \mathtt{in}$
$\qquad \mathtt{orthogonal}\ (\mathtt{u} - (\mathtt{u\ cdot\ cunit\_v})\%\mathtt{cunit\_v})\ \mathtt{v}$

These two theorems play a crucial role in practice. They are also very useful when proving the Cauchy-Schwarz Inequality, which has itself many applications, e.g., in the error analysis of many engineering systems:

Theorem 2.7 (Cauchy-Schwarz Inequality)
$\vdash\ \forall \mathtt{u\ v}.\ \mathtt{norm}\ (\mathtt{u\ cdot\ v})\ \leq\ \mathtt{cnorm\ u} * \mathtt{cnorm\ v}$

where `norm` denotes the norm of a complex number. In addition to the notions presented here, many other concepts were defined, and some related theorems were proved, We refer to the implementation [1] for details.

The cross product between two vectors u and v of $\mathbb{C}^3$ is classically defined as $(u_2v_3 - u_3v_2, u_3v_1 - u_1v_3, u_1v_2 - u_2v_1)$. Note that, contrarily to the dot product, the outcome is again a vector of $\mathbb{C}^3$. This is formalized as follows:

Definition 2.8 (Complex cross product)
$\vdash\ \forall \mathtt{u\ v} : \mathbb{C}^3.\ \mathtt{u} \times \mathtt{v} : \mathbb{C}^3 =$
$\mathtt{vector}\ [\mathtt{u\$2} * \mathtt{v\$3} - \mathtt{u\$3} * \mathtt{v\$2};\ \mathtt{u\$3} * \mathtt{v\$1} - \mathtt{u\$1} * \mathtt{v\$3};\ \mathtt{u\$1} * \mathtt{v\$2} - \mathtt{u\$2} * \mathtt{v\$1}]$

where `vector` is a HOL Light function taking a list as input and returning a vector. In Table 2, we present some of the properties of cross product.

Property	**Formalization**
Left zero	$\vdash\ \forall\, \mathtt{u}.\ \mathtt{cvector\_zero} \times \mathtt{u} = \mathtt{cvector\_zero}$
Right zero	$\vdash \forall\, \mathtt{u}.\ \mathtt{u} \times \mathtt{cvector\_zero} = \mathtt{cvector\_zero}$
Irreflexivity	$\vdash\ \forall\, \mathtt{u}.\ \mathtt{u} \times \mathtt{u} = \mathtt{cvector\_zero}$
Asymmetry	$\vdash\ \forall\, \mathtt{u\ v}.\ -(\mathtt{u} \times \mathtt{v}) = \mathtt{v} \times \mathtt{u}$
Left-distributivity over addition	$\vdash\ \forall\, \mathtt{u\ v\ w}.\ (\mathtt{u} + \mathtt{z}) \times \mathtt{w} = \mathtt{u} \times \mathtt{w} + \mathtt{z} \times \mathtt{w}$
Right-distributivity over addition	$\vdash\ \forall\, \mathtt{u\ v\ w}.\ \mathtt{u} \times (\mathtt{v} + \mathtt{w}) = \mathtt{u} \times \mathtt{v} + \mathtt{u} \times \mathtt{w}$
Left-distributivity over scalar mul.	$\vdash\ \forall\, \mathtt{a\ u\ v}.\ (\mathtt{a}\%\mathtt{u}) \times \mathtt{v} = \mathtt{a}\%(\mathtt{u} \times \mathtt{v})$
Right-distributivity over scalar mul.	$\vdash\ \forall \mathtt{a\ u\ v}.\ \mathtt{u} \times (\mathtt{a}\%\mathtt{v}) = \mathtt{a}\%(\mathtt{u} \times \mathtt{v})$

Table 2: Cross Product Properties

One of major properties of cross product between two vector is that $u \times v$ is perpendicular to u and v, therefore it is normal to the plane containing them. This property is used in particular as a criterion to check the collinearity of two vectors, as follows:

Theorem 2.9 (Cross product and collinearity)
$\vdash\ \forall\, \mathtt{x} : \mathbb{C}^3\ \mathtt{y}.\ \mathtt{x} \times \mathtt{y} = \mathtt{cvector\_zero} \Leftrightarrow \mathtt{collinear\_cvectors\ x\ y}$

We also define many basic notions of linear algebras like, e.g., the canonical basis of the vector space, i.e., the set of vectors $(1, 0, 0, 0, \dots)$, $(0, 1, 0, 0, \dots)$, $(0, 0, 1, 0, \dots)$, etc. This is done as follows:

Definition 2.10 $\vdash\ \forall \mathtt{k}.\ \mathtt{cbasis\ k} = \lambda \mathtt{i}.\ \mathtt{if\ i} = \mathtt{k\ then\ 1\ else\ 0}$

With this definition `cbasis 1` represents $(1, 0, 0, \dots)$, `cbasis 2` represents $(0, 1, 0, \dots)$, etc. Finally, an essential notion of finite-dimension vector spaces is the one of *matrix*. Matrices are essentially defined as vectors of vectors: a $M \times N$ matrix is formalized by a value of type $\mathbb{C}^{\mathtt{N}^{\mathtt{M}}}$. Several arithmetic notions can then be formalized over matrices:

Definition 2.11 (Complex Matrix Arithmetic)
$\vdash\ \mathtt{m}_1 +_{\mathbb{C}^{\mathtt{N}^{\mathtt{M}}}} \mathtt{m}_2 = \mathtt{lambda\ i.\ m}_1\mathtt{\$i} +_{\mathbb{C}^{\mathtt{N}}} \mathtt{m}_2\mathtt{\$i}$
$\vdash\ \mathtt{a}\ \%_{\mathbb{C}^{\mathtt{N}^{\mathtt{M}}}}\ \mathtt{m} = \mathtt{lambda\ i.\ a}\ \%_{\mathbb{C}^{\mathtt{N}}}\ \mathtt{m\$i}$
$\vdash\ \mathtt{m}_1 * * \mathtt{m}_2 = \mathtt{lambda\ i\ j.\ vsum}\ (\mathtt{1..dimindex(:N)})\ (\lambda \mathtt{k.\ m}_1\mathtt{\$i\$k}\ *\ \mathtt{m}_2\mathtt{\$k\$j})$

where `vsum s f` denotes $\sum_{\texttt{x}\in\texttt{s}}$ `f x`, `dimindex s` is the number of elements of `s`, and (: `N`) is the set of all inhabitants of the type `N` (the "universe" of `N`, also written `UNIV`). Note that the indices providing the type information of operators, e.g., $+_{\mathbb{C}^{\texttt{M}}}$, are for readability purposes only: the overload mechanism of HOL Light allows one to use the same symbol for various types. We will use them, every time we introduce a new domain to a function.

Here, we conclude by the definition of matrix-vector multiplication, which makes the connection between matrices and vectors, essentially used in applications:

Definition 2.12 (Matrix-Vector Multiplication)
$\vdash$ `m ** v = lambda i. vsum (1..dimindex(: N)) (λj. m$i$j * v$j)`

In [9], we present the formalization of complex vector with infinite dimension, using the concept of linear operators. In next section, not only we derive properties of complex matrices from the properties of linear operators, we connect our formalization of finite and infinite complex vectors by deriving the properties of the former from the latter.

3 Deriving Complex Vectors Properties

Complex vector spaces can be seen as a particular case of complex function spaces, presented in [9]: indeed, if the type `A` in $\texttt{A} \to \mathbb{C}$ is instantiated with a finite type, say a type with three inhabitants `one`, `two`, `three`, then functions mapping each of `one`, `two`, `three` to a complex number can be seen as 3-dimension vectors. For instance the function $\{\texttt{one} \mapsto 1 + \texttt{ii}, \texttt{two} \mapsto \texttt{ii}, \texttt{three} \mapsto 2\}$, (where `ii` denotes the HOL Light counterpart of the complex number j) can be seen as the vector $(1 + j, j, 2)$. So, for a type `t` with n inhabitants, there is a bijection between complex-valued functions of domain `t` and complex vectors of dimension n.

This bijection is actually at the heart of vector representation in HOL Light, as introduced by Harrison [4], defining the type constructor (`A`)`finite_image` which has a finite set of inhabitants whatever is `A`. In addition, if `A` is already finite and has n elements then (`A`)`finite_image` also has n elements. Harrison also defines two inverse operations: $\vdash$ `mk_cart` : ((`A`)`finite_image` $\to$ `B`) $\to$ $\texttt{B}^{\texttt{A}}$ and $\vdash$ `dest_cart` : $\texttt{B}^{\texttt{A}} \to$ ((`A`)`finite_image` $\to$ `B`) where $\texttt{B}^{\texttt{A}}$ is the type of vectors with as many coordinates as `A` has elements.

For complex vectors, we can even go further by looking at how *operations* translate from one type to the other. For instance, by using the above inverse functions, we can transfer `cfun` addition to the type $\mathbb{C}^{\texttt{N}}$ as follows: `v` $+_{\mathbb{C}^{\texttt{N}}}$ `w = mk_cart (dest_cart v` $+_{\texttt{cfun}}$ `dest_cart w)`. This definition basically takes two vectors as input, transforms them into values of type `cfun`, computes their `cfun`-addition,

and transforms back the result into a vector. This operation can actually be easily seen to be equivalent to Definition 2.1. Therefore, not only we have a bijection between the types (N)finite_image $\to \mathbb{C}$ and $\mathbb{C}^{\mathtt{N}}$, but also an *isomorphism* between the structures $((\mathtt{N})\mathtt{finite\_image} \to \mathbb{C}, +_{\mathtt{cfun}})$ and $(\mathbb{C}^N, +_{\mathbb{C}^{\mathtt{N}}})$.

We now show how to use these observations to derive *automatically* properties from the type cfun to the type $\mathbb{C}^{\mathtt{N}}$.

3.1 Arithmetic properties

We provide in our formalization an Ocaml function cfun_to_cvector which takes as input a theorem stating some fact about cfun arithmetic, and returns its counterpart $\mathbb{C}^{\mathtt{N}}$ theorem. For example, the theorem CFUN_ADD_RDISTRIB states the right distributivity of cfun scalar multiplication over addition:

Theorem 3.1 (CFUN_ADD_RDISTRIB)
$\vdash \forall \mathtt{a\ b\ f}.\ (\mathtt{a} + \mathtt{b})\ \%\ \mathtt{f} = \mathtt{a}\ \%\ \mathtt{f} + \mathtt{b}\ \%\ \mathtt{f}$

Then the following Ocaml function call returns automatically the counterpart theorem for complex vectors:

```
# cfun_to_cvector CFUN_ADD_RDISTRIB;;
val it : thm = ∀a b v:ℂ^N. (a + b) % v = a % v + b % v
```

In practice, the theorems dealing with complex function arithmetic are stored in a database (a simple association list mapping names with theorems). Then the function cfun_to_cvector is called on all the theorems of this database and the result is automatically assigned to a theorem with the same name except that the prefix CFUN_ is replaced by CVECTOR_. This entails a systematic transfer of new cfun theorems to $\mathbb{C}^{\mathtt{N}}$, ensuring a naming uniformity across both types. For instance the above theorem CFUN_ADD_RDISTRIB yields the theorem CVECTOR_ADD_RDISTRIB. Given *th*, the theorem we want to port to cfun, the implementation of cfun_to_cvector follows the following steps:

1. Instantiate cfun (i.e., $\mathtt{A} \to \mathbb{C}$) with $(\mathtt{N})\mathtt{finite\_image} \to \mathbb{C}$ (application of the HOL Light primitive rule of type instantiation);

2. Replace every variable f of type $(\mathtt{N})\mathtt{finite\_image} \to \mathbb{C}$ by dest_cart v, where v is of type $\mathbb{C}^{\mathtt{N}}$. This is done by (higher-order) rewriting with the following theorem proved easily by using the bijectivity of dest_cart:

   Theorem 3.2 (FORALL_CFUN) $\vdash \forall \mathtt{P}.\ (\forall \mathtt{f}.\ \mathtt{P\ f}) \Leftrightarrow (\forall \mathtt{v}.\ \mathtt{P\ (dest\_cart\ v)})$

   Example 1 *If th is CFUN_ADD_RDISTRIB, then we obtain the theorem:*
   $\vdash \forall \mathtt{a\ b\ v}.\ (\mathtt{a} + \mathtt{b})\ \%\ \mathtt{dest\_cart\ v} = \mathtt{a}\ \%\ \mathtt{dest\_cart\ v} + \mathtt{b}\ \%\ \mathtt{dest\_cart\ v}$

3. "Pull" every occurrence of `dest_cart` from atomic to outermost positions. This is done by rewriting with the following commutativity theorem:

   Theorem 3.3 (`PULL_DEST_CART`)
   $\vdash \forall \mathtt{a\ v\ v_1\ v_2}.$
   $\mathtt{dest\_cart\ v_1} +_{\mathtt{cfun}} \mathtt{dest\_cart\ v_2} = \mathtt{dest\_cart}\ (\mathtt{v_1} +_{\mathbb{C}^{\mathtt{N}}} \mathtt{v_2})$
   $\wedge\ \mathtt{a} \%_{\mathtt{cfun}} \mathtt{dest\_cart\ v} = \mathtt{dest\_cart}\ (\mathtt{a} \%_{\mathbb{C}^{\mathtt{N}}} \mathtt{v})$
   $\wedge\ \mathtt{cfun\_zero} = \mathtt{dest\_cart}\ (\mathtt{cv\_zero})$

   This theorem is easily proved by using the definitions of the involved operations and the fact that `dest_cart` is a bijection.

   Example 2 *Applying this step after Example 1 yields:*
   $\vdash \forall \mathtt{a\ b\ v}.\ \mathtt{dest\_cart}\ ((\mathtt{a} + \mathtt{b})\ \%\ \mathtt{v}) = \mathtt{dest\_cart}\ (\mathtt{a}\ \%\ \mathtt{v} + \mathtt{b}\ \%\ \mathtt{v})$

4. Apply injectivity of `dest_cart` by using the following theorem:

   Theorem 3.4 (`DEST_CART_INJ`)
   $\vdash \forall \mathtt{f\ g}.\ \mathtt{dest\_cart\ f} = \mathtt{dest\_cart\ g} \Leftrightarrow \mathtt{f} = \mathtt{g}$

   which follows trivially from the fact that `dest_cart` is a bijection.

   Example 3 *Applying injectivity of* `dest_cart` *after Example 2 yields:*
   $\vdash \forall \mathtt{a\ b\ v}.\ (\mathtt{a} + \mathtt{b})\ \%\ \mathtt{v} = \mathtt{a}\ \%\ \mathtt{v} + \mathtt{b}\ \%\ \mathtt{v}$
   which corresponds indeed to the $\mathbb{C}^{\mathtt{N}}$ *counterpart of the original theorem.*

More than 50 theorems over $\mathbb{C}^{\mathtt{N}}$ were derived from `cfun` using solely these four steps. As can be seen this procedure is extremely simple and its implementation does not take more than 10 lines. The proofs of all three theorems `FORALL_CFUN`, `PULL_DEST_CART` and `DEST_CART_INJ` take no more than 15 lines, including their statements. Finally, since all the theorems dealing with `cfun` arithmetic have been stored in a single database, deriving the corresponding theorems for $\mathbb{C}^{\mathtt{N}}$ is simply achieved by iterating over this database and applying `cfun_to_cvector`.

This approach is new, as far as we know. The closest related work that we are aware of are [2, 8] which use type isomorphisms in order to improve proof reuse in Coq. These works make a thorough analysis of type isomorphisms and their use to change data representation in presence of dependent types. Also in [6, 7], by reinterpreting the concept of quotient operation of set theory, types are divided by an equivalence relation to create new types; quotient types. Then constants and those theorems about the original types, which are valid about new types,

are automatically lifted to corresponding constants and theorems of the quotient types. All these approaches go much further than we do by being more general and systematic, however our method is a lot simpler and applies not only to type isomorphisms but also simply to sets which are in bijection. This simplicity of our approach allowed us to make the corresponding development very quickly and concisely.

3.2 Inner Spaces as Type Classes

We now derive from properties of `cfun` inner spaces their counterpart properties for $\mathbb{C}^N$ with the dot product (Definition 2.3), exactly as it is the case in theorem provers making use of *type classes* [17, 13]. Note however that, in addition to the instantiation of the axiom for inner spaces, we also deal with the bijection between complex-valued functions with finite dimension and the set of complex vectors. This work thus goes beyond what is done with standard type classes.

This time, we define an Ocaml function `inner_space_to_cdot` which takes as input a theorem *th* of the form `is_inner_space s` $\Rightarrow$ `P s` and turns it into `P' s` where `P'` is the `cdot` counterpart of `P`. For instance the following theorem:

Theorem 3.5 (`INSPACE_ADD_RZERO`)
$\vdash \forall$`s inprod` (`f` : `cfun`).
`is_inner_space` (`s`, `inprod`) $\wedge$ `f` $\in$ `s` $\Rightarrow$ `inprod f cfun_zero` = `0`

should be automatically turned into:

Theorem 3.6 (`CDOT_RZERO`)
$\vdash$ $\forall$`v` : $\mathbb{C}^N$. `v` $\cdot$ `cfun_zero` = `0`

This is implemented as follows:

1. Remove the `is_inner_space` antecedent, by applying Modus Ponens between *th* and the following theorem, which states that the dot product, up to translation to the `cfun` type, is an inner product over the universe of (`N`)`finite_image` $\to \mathbb{C}$:

   Theorem 3.7 (`CDOT_IS_INNER_SPACE`)
   $\vdash$ `is_inner_space` ((: (`N`)`finite_image` $\to \mathbb{C}$), λ`f g`. (`mk_cart f`) $\cdot$ (`mk_cart g`))

   Example 4 *For instance Theorem 3.5 is turned into:*
   $\vdash \forall$`f`. `f` $\in$ (: (`N`)`finite_image` $\to \mathbb{C}$) $\Rightarrow$ `mk_cart f` $\cdot$ `mk_cart cfun_zero` = `0`.

2. Remove trivial membership predicates of the form `f` $\in$ (: (`N`)`finite_image` $\to \mathbb{C}$), by rewriting with `IN_UNIV`: $\vdash \forall$`x` : `A`. `x` $\in$ (: `A`).

Example 5 *The same example becomes:*
$\vdash \forall$`f`. `mk_cart f` $\cdot$ `mk_cart cfun_zero` $=$ `0`.

3. Apply steps 2 and 3 of `cfun_to_cvector`, i.e., rewrite by `FORALL_CFUN` and `PULL_MK_ CART`, yielding, for the example:
$\vdash \forall$`v`. `mk_cart` (`dest_cart v`) $\cdot$ `mk_cart` (`dest_cart cv_zero`) $=$ `0`.

4. Remove the occurrences of `dest_cart` and `mk_cart` by using step 4 of `cfun_to_cvector` and the fact that `dest_cart` and `mk_cart` are inverse. The example finally becomes:
$\vdash \forall$`v`. `v` $\cdot$ `cv_zero` $=$ `0`.

The implementation of this procedure does not take more than 20 lines. Only the theorem `CDOT_IS_ INNER_SPACE` is non trivial, but a similar result would be proved in any formalization of the dot product anyway so its cost is not proper to our method. The above procedure can easily be extended to integrate transformations of theorems dealing with orthogonality, which yielded, in the end, around 30 theorems proved entirely automatically. Note that these theorems are not toy examples: even involved results like the Cauchy Schwartz inequality are "translated" using this method. As far as we know, this approach is also new in HOL Light: the only related work is [11], which provides an implementation of type classes in HOL Light. However it is very unsatisfying since it actually runs again the tactic scripts for every instantiation of the type class.

3.3 Deriving Complex Matrices Arithmetic from Linear Operators

We finally present the way to derive matrix arithmetic from operators' and linear operators' arithmetic. It is well known that, to any linear operator *op* of a finite dimension vector space, one can map a matrix m such that, for every vector v, $op\ v = m.v$, where . denotes matrix-vector multiplication. We first present the transformation `cmatrix_of_cop` allowing to obtain this matrix, and its converse `cop_of_cmatrix`, as well as their important properties.

The matrix canonically associated with a linear operator *op* is classically obtained by taking for every column the outcome of *op* applied to a basis vector. More precisely:

$$M = \begin{pmatrix} (op\ e_1)_1 & (op\ e_2)_1 & \dots \\ (op\ e_1)_2 & (op\ e_2)_2 & \dots \\ \dots & \dots & \dots \end{pmatrix}$$

where $(e_1, e_2, \dots)$ is the canonical base of the space.

In addition, since operators are defined on `cfun`, we need to apply the now usual translations from/to $\mathbb{C}^N$. Formally, this yields the following definition:

Definition 3.8 (`cmatrix_of_cop`)

$\vdash$ $\forall$op. cmatrix_of_cop op
= lambda i j. mk_cart (op (dest_cart (cbasis j)))$i

where `cbasis` comes from Definition 2.10.

Reciprocally, the operator associated with a matrix is classically defined simply by taking matrix-vector multiplication:

Definition 3.9 (`cop_of_cmatrix`)

$\vdash$ $\forall m : \mathbb{C}^{N^M}$. cop_of_cmatrix m = λf : cfun. dest_cart (m ** mk_cart f)

where $**$ is the matrix-vector multiplication (Definition 2.12).

We then prove the following theorem:

Theorem 3.10

$\vdash$ $\forall m : \mathbb{C}^{N^M}$. cmatrix_of_cop (cop_of_cmatrix m) = m

$\vdash$ $\forall$op. is_linear_cop op $\Rightarrow$ cop_of_cmatrix (cmatrix_of_cop op) = op

This theorem is not trivial (especially the second conjunct) and requires in particular some induction on the space dimension.

These theorems allow to develop again a procedure similar to the ones for Sections 3.1 and 3.2. Once again we define an Ocaml function `lincop_to_cmatrix` which takes a property on operators (linear or not) as input and returns its matrix counterpart. For instance, take the theorem stating the left-distributivity between operators when the first operator is linear:

Theorem 3.11 (`LINCOP_ADD_MUL_LDISTRIB`)

$\vdash \forall op_1\ op_2\ op_3$. is_linear_cop op_3
$\Rightarrow op_3\ **\ (op_1 + op_2) = op_3\ **\ op_1 + op_3\ **\ op_2$

Then:

```
# lincop_to_cmatrix LINCOP_ADD_MUL_LDISTRIB;;
val it :  thm = ∀m1 m2 m3. m3 ** (m1 + m2) = m3 ** m1 + m3 ** m2
```

We present once again the steps that allow to implement such a function:

1. Replace every universally quantified operator by a matrix, using the following theorem:

   Theorem 3.12 (`FORALL_COP`)

   $\forall$P. ($\forall$op. P op) $\Rightarrow$ ($\forall$m. P (cop_of_cmatrix m))

   $\forall$P. ($\forall$op. is_linear_cop op $\Rightarrow$ P op) $\Leftrightarrow$ ($\forall$m. P (cop_of_cmatrix m))

which allows us to replace both linear and non-linear operators by matrices (note that a property which holds for non-linear operators also holds for linear ones since these are more general).

2. Pull `cop_of_cmatrix` occurrences to outermost positions, by rewriting with the following theorem:

   Theorem 3.13 (`PULL_COP_OF_CMATRIX`)
   $\forall \mathtt{a}\ \mathtt{m_1}\ \mathtt{m_2}.$
   $\mathtt{cop\_of\_cmatrix}\ \mathtt{m_1}\ **\ \mathtt{cop\_of\_cmatrix}\ \mathtt{m_2} = \mathtt{cop\_of\_cmatrix}\ (\mathtt{m_1} * *\mathtt{m_2})$
   $\wedge\ \mathtt{cop\_of\_cmatrix}\ \mathtt{m_1}\ +\ \mathtt{cop\_of\_cmatrix}\ \mathtt{m_2} = \mathtt{cop\_of\_cmatrix}\ (\mathtt{m_1} + \mathtt{m_2})$
   $\wedge\ \mathtt{a}\ \%\ \mathtt{cop\_of\_cmatrix}\ \mathtt{m} = \mathtt{cop\_of\_cmatrix}\ (\mathtt{a}\ \%\ \mathtt{m})$
   $\wedge\ \mathtt{cop\_zero} = \mathtt{cop\_of\_cmatrix}\ \mathtt{cmatrix\_zero}$

3. Eliminate `cop_of_cmatrix` by injectivity.

Once again, this very simple procedure takes no more than 10 lines to implement, with no more than 20 lines to prove all the required theorems. Still, it is powerful enough to derive automatically more than 50 theorems from operators *and* linear operators properties. The only difference is that the theorem `FORALL_COP` has an implicational form for linear operators. This shows again how flexible our method is since matrices are in bijection with *linear* operators only[1].

This technique can be used to derive other properties and can be applied to other situations, for instance to deal with the alternative representation of complex vectors as two real vectors (one for the real part and one for the imaginary part). In the next section, we now apply all these results to the formalization of electromagnetic planar waves.

4 Conclusion

We presented a toolbox for complex vector spaces in HOL Light: complex function spaces were defined along with useful operations on them; complex vectors, in finite dimension, were formalized, adding some particular cases in dimension 3. An essential aspect of this formalization is that it is *engineering*-oriented: we favoured the notions and theorems that are useful in applications, and we provided tactics to automate the most frequent tasks. We then made use of this theory to formalize

[1]Note that since dependent types are much more powerful than the simple type theory used in HOL, such sets bijections could still often be represented as type isomorphisms in a type theory like the Calculus of Inductive Constructions.

a fragment of electromagnetics, and we verified some classical results for plane waves: the law of the plane of incidence and of reflection.

In addition, an insightful aspect of our formalization is that many of the properties of complex vector spaces have been proved entirely automatically by analyzing their relation with complex function spaces. This reminds of many proof engineering techniques, e.g., type isomorphism or type classes, but our approach presents the advantage of being very easy to implement by making an elegant use of the rewriting functions already present in HOL Light.

As a future work, we plan to develop extensively electromagnetic applications, particularly in the verification of optical systems. We also investigate ray and quantum optics which make both use of the presented formalization. Finally, we consider applying our proof engineering technique to other cases like the dual data representation of complex vectors as vectors of complexes or as a pair of real vectors.

References

[1] S. Kh. Afshar, O. Hasan, and V. Aravantinos. `http://hvg.ece.concordia.ca/projects/optics/cxvec/`, 2014.

[2] G. Barthe and O. Pons. Type Isomorphisms and Proof Reuse in Dependent Type Theory. In *Foundations of Software Science and Computation Structures*, pages 57–71. Springer, 2001.

[3] J. Harrison. A HOL Theory of Euclidean space. In *Theorem Proving in Higher Order Logics*, volume 3603 of *LNCS*, pages 114–129. Springer, 2005.

[4] J. Harrison. The HOL Light Theory of Euclidean Space. *Journal of Automated Reasoning*, 50(2):173–190, 2013.

[5] H. Herencia-Zapana, R. Jobredeaux, S. Owre, P. Garoche, E. Feron, G. Perez, and P. Ascariz. PVS Linear Algebra Libraries for Verification of Control Software Algorithms in C/ACSL. In *NASA Formal Methods*, pages 147–161, 2012.

[6] P. V. Homeier. A Design Structure for Higher Order Quotients. In *Theorem Proving in Higher Order Logics*, pages 130–146. Springer, 2005.

[7] B. Huffman and O. Kunčar. Lifting and Transfer: A Modular Design for Quotients in Isabelle/HOL. In *Isabelle Users Workshop at ITP*, 2012.

[8] N. Magaud. Changing Data Representation within the Coq System. In *TPHOLs*, pages 87–102, 2003.

[9] M.Y. Mahmoud, V. Aravantinos, and S. Tahar. Formalization of Infinite Dimension Linear Spaces with Application to Quantum Theory. In *NASA Formal Methods*, volume 7871 of *Lecture Notes in Computer Science*, pages 413–417. Springer, 2013.

[10] N. Mastronardi, G. H. Golub, S. Chandrasekaran, M. Moonen, P. Van Dooren, and S. Van Huffel. Numerical Linear Algebra in Signal Processing Applications. *EURASIP Journal on Advances in Signal Processing*, 2007, 2007.

[11] S. McLaughlin. An Interpretation of Isabelle/HOL in HOL Light. In *Automated Reasoning*, volume 4130 of *Lecture Notes in Computer Science*, pages 192–204. Springer, 2006.

[12] J. L. Meriam, L. G. Kraige, and W. J. Palm. *Engineering Mechanics Vol. 1: Statics*. Wiley, 1987.

[13] M. Sozeau and N. Oury. First-Class Type Classes. In *Theorem Proving in Higher Order Logics*, pages 278–293, 2008.

[14] J. Stein. `http://coq.inria.fr/pylons/contribs/view/LinAlg/trunk`, 2014.

[15] J. C. Tallack. *Introduction to Vector Analysis*. Cambridge University Press, 1970.

[16] P. Van Dooren. *Numerical Linear Algebra in Signals, Systems and Control*, volume 80. Springer Science+ Business Media, 2011.

[17] M. Wenzel. Type Classes and Overloading in Higher-Order Logic. In *Theorem Proving in Higher Order Logics*, pages 307–322, 1997.