基于Coq记录的矩阵形式化方法

马振威[1] 陈 钢[1,2]
(南京航空航天大学计算机学院 南京 210000)[1] (北京京航计算与通信研究所 北京 100074)[2]

摘 要 矩阵在工程系统中有广泛的应用,矩阵运算的正确性对工程系统的可靠性有重要影响。Coq是一种基于带类型λ演算的功能强大的高阶定理证明器。虽然Coq类型系统能够很好地描述可变大小的动态数据类型,但是对于固定大小的类似向量和矩阵的数据类型,其缺乏满意的描述机制。Coq库中也没有向量库或矩阵库,因此在使用Coq来对涉及矩阵的定理或算法进行形式化验证时十分复杂。针对这些问题,文中提出了一种基于Record类型的矩阵实现方法并定义了一组基本的矩阵函数,证明了它们的基本性质。基于文中提供的矩阵类型和相关引理可以比较轻松地完成飞行控制转换矩阵的验证。同其他矩阵实现方法相比,所提方法不仅在实现上相对简洁,在使用上也更加简单、方便。

关键词 Coq,矩阵,形式化验证,定理证明,记录类型
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2019.07.022

Matrix Formalization Based on Coq Record

MA Zhen-wei[1] CHEN Gang[1,2]
(College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210000, China)[1]
(Beijing Jinghang Research Institute of Computing and Communication, Beijing 100074, China)[2]

Abstract Matrix has a wide range of applications in engineering systems, and the correctness of matrix operations has an important impact on the reliability of engineering systems. Coq is a powerful higher-order theorem prover based on the type λ calculus. Although the Coq type system can describe variable-sized dynamic data types well, there is a lack of satisfactory description mechanisms for data types such as fixed-size vectors and matrices. At present, there is no vector library or matrix library in the Coq library, so it is more difficult to use Coq to formally verify the theorem or algorithm involving the matrix. In order to solve these problems, this paper proposed a matrix implementation method based on Record type, defined a set of basic matrix functions and proved their basic properties. The verification of the flight control transformation matrix can be done easily by using the matrix types and related lemmas provided in this paper. Compared with other matrix implementation methods, this method is not only relatively simple to implement, but also simpler and more convenient to use.

Keywords Coq, Matrix, Formal verification, Theorem proving, Record type

1 引言

矩阵运算是工程软件中(如飞行控制系统[1-2])极为常用的一种运算,也是一种很容易发生错误的运算。形式化方法为软件可靠性验证提供了一种有效的工具。大部分常用的形式验证技术(如模型检查)并不保障软件的完全可靠性,甚至在可验证的软件性质方面也有一定的局限,因此限制了矩阵验证的效果。基于高阶逻辑的定理证明技术在理论上有能力提供程序的完全正确性验证,在实践中也能够进行范围更广的程序性质验证。

然而,高阶定理证明器的基础是带类型的λ演算,因此,在这些系统中,实现矩阵的基础性数据结构是λ表达式,而不是命令式语言中的数组。这种数据结构适合于描述具有无限可扩展特性的归纳数据结构,比如列表。但是,对于具有固定长度的向量和矩阵这样的数据结构,其缺乏一种自然和直接的描述方法。目前,针对定长数据结构的形式化技术大体上有3种:1)用函数表示数组,这是HOL系统和一些Coq研究组采用的方法,采用这一技术可以完成很多向量和矩阵性质

到稿日期:2018-06-01 返修日期:2018-09-28 本文受南京航空航天大学研究生创新基地(实验室)开放基金(kfjj20171610),中央高校基本科研业务费专项资金资助。
马振威(1992-),男,硕士生,CCF会员,主要研究方向为定理证明;陈 钢(1958-),男,博士,教授,CCF会员,主要研究方向为定理证明,E-mail:gangchensh@qq.com(通信作者)。

的证明，但是这种描述并不直接对应于程序语言中的数据结构，也难以通过代码抽取的方法将它们转换成程序语言中的代码；2）采用依赖类型进行描述，这一技术可以描述指定长度的向量，并且可以描述输入、输出数据具有特定长度关系的函数，也能够通过代码抽取获得程序代码，但是这一技术对复杂函数的描述相当复杂，对函数的性质证明非常困难；3）采用元组结构描述矩阵，对于小规模的矩阵问题，这一技术描述简单且便于进行定理证明，但并不适合描述任意规模的矩阵。

为了解决上述问题，本文提出了一种将列表和记录相结合的矩阵描述方法，其原理是把向量和矩阵的类型用记录定义；本文还实现了向量以及矩阵的一些运算函数（包括加法、减法、乘法等函数），并对其基本性质进行了证明。由此也说明了基于记录的矩阵定义方法是解决矩阵形式化问题的一个有效方法。

本文第 2 节介绍了 Coq 定理证明器，包括它的工作原理以及 Coq 记录的概念，还讨论了这一技术相比较于其他矩阵实现方式的优点；第 3 节介绍如何使用 Record 实现向量和矩阵；第 4 节介绍了关于向量和矩阵操作函数的定义和基本性质的验证；第 5 节介绍了如何将定义的矩阵向各个数域扩展以方便用户使用，并以一个实例来介绍矩阵的应用；第 6 节总结全文并展望。

2 背景知识

本节介绍本文工作的背景以及相关工作，包括定理证明器 Coq 的介绍和目前 Coq 中与矩阵相关的研究工作，并将这些工作与本文的工作进行了比较，说明了本文工作的特点和优势。

2.1 Coq

Coq[3-4] 是一个基于高阶逻辑的定理证明辅助工具，它以归纳构造演算为理论基础。Coq 可以进行命题演算的证明、时序逻辑的证明、谓词逻辑的证明、高阶逻辑的证明，甚至可以自定义逻辑，然后在这种新定义的逻辑系统中进行逻辑推导。

Coq 采用反向推理的方法构造证明，在设立一个当前证明目标之后，通过使用合适的证明命令，产生一组保证该目标成立的条件，这些条件被称为子目标。如果子目标已知，则证明完成；如果子目标未知，则继续寻找使子目标成立的条件作为新的子目标，直到满足子目标为已知条件或公理。当所有子目标都得到证明，则推理完成。

Coq 的一个重要特点是能够基于 Curry-Howard 同构原理进行程序抽取。也就是说，它可以将 Coq 中定义的数据结构和函数转变成函数程序语言 OCaml 或 Haskell 的程序。这一程序抽取能力使得 Coq 更适合于进行软件的正确性验证，以及可证正确的软件开发。

2.2 相关工作

HOL[5] 定理证明器较早实现了矩阵理论的形式化。首都师范大学施智平团队开展了函数矩阵方面的研究工作[6-7]。HOL 中用函数实现矩阵，这一思路在 Coq 的 ssreflect 项目中得以继承，后面将详细描述。

Coq 标准库中与向量最相似的类型是 List 类型。List 类型的定义如下：

```
Inductive List A:Type:=
  |nil:List A
  |cons:A->List A->List A.
```

从定义可以看出，List 是元素类型相同、长度可变的序列，由于列表的元素也可以是列表，因此用嵌套列表的方式可以实现二维甚至多维嵌套列表。但是，List 的长度是不确定的，无法通过它的类型判断其长度，即无法直接定义一个固定长度的 List。

在 Coq 系统中可以通过依赖类型[8] 定义带长度的列表类型，因此可以定义指定长度的向量类型和指定大小的矩阵类型。Nicolas Magaud[9] 使用了这个方法来定义矩阵。首先基于依赖类型的向量类型可用 Inductive 命令归纳[10]，其定义为：

```
Inductive vect (A:Set):nat->Set:=
  |vnil:vect A 0
  |vcons:forall n:nat,A->vect A n->vect A (S n).
```

该定义是 List 定义的推广，与 List 不同的是，该定义中包含了一个表示表长的参数，在递归定义中这个参数严格“递减”，用来表示每一个子表的长度。使用这种方式的确可以定义一种向量类型，但其存在以下缺点：首先，由于向量的各子向量都包含长度参数，使向量操作函数的定义及其性质的验证十分复杂、困难；其次，Nicolas Magaud 只给出了关于向量的一些简单的性质证明，很多基础性质并没有完成证明。Denes 和 Bertot 指出，这一定义方法在处理二元函数的问题时面临较大的困难[11]。

在 ssreflect 库中包含了一个矩阵库，它把二维矩阵定义为：

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

这里，'I_m 和 'I_n 可以看成是自然数的子集 $0\cdots m$ 和 $0\cdots n$。这个实现方案本质上是把矩阵定义为从自然数矩阵 $[0\cdots m]*[0\cdots n]$ 到实数的函数。在这个定义之下，ssreflect 库中证明了大量的矩阵性质。但是这样种矩阵的实现方式同函数式语言中基于列表的矩阵实现方式是有差距的。此外，ssreflect 的矩阵实现方案过于复杂，不易理解和使用。

针对 ssreflect 矩阵的问题，Denes 和 Bertot 提出了一个称为 effective matrices 的方案。他们把矩阵简单地定义成：

```
Definition seqmatrix:= seq (seq R)
```

其中，R 是环类型，seq 是 ssreflect 中的列表。然后定义了一个从 ssreflect 矩阵类型到 seqmatrix 类型的同态映射，以建立起两者之间的关系。对于每一个矩阵函数，都需要验证从 ssreflect 矩阵到 seqmatrix 的同态关系成立。这一方案避开了直接使用指定长度的向量和矩阵类型，转而通过同态关系来

保障矩阵运算的可靠性。该方案的好处是能够利用 ssreflect 中矩阵性质的证明结果，缺点是矩阵函数并没有直接定义在向量和矩阵类型之上。

本文期望找到一种矩阵定义方案，它能够以接近函数式语言的方式实现向量和矩阵，容易理解，便于进行矩阵函数的构造和矩阵性质的证明，以自然的方式支持程序抽取[12]，从而能够有效地支持矩阵软件的开发和形式验证。为此，本文提出一种基于 Record 类型的定义矩阵的方法，它比前述方案更加简洁、直接，并且基本达到了上述目标。这一方案不依赖复杂的 ssreflect 库，因此用户更容易掌握。

2.3 Record 类型

Record 类型是一个允许定义记录的宏，它的语法如下：

Record *ident params*：*sort*：= *ident*0 {*ident*1 binders1：*term*1；…；*identn* bindersn：*termn*}。

使用 Record 可以定义出这样一种类型，在这个类型中有一些数据，并且这些数据必须满足某些性质。本节以一个 Coq 标准库的实数库中的例子来介绍 Record 的使用方法：

```
Record Metric_Space:Type:=
{Base:Type;
dist:Base-> Base-> R;
dist_pos:forall x y:Base,dist x y >= 0;
dist_sym:forall x y:Base,dist x y=dist y x;
dist_refl:forall x y:Base,dist x y=0<->x=y;
dist_tri:forall x y z:Base,dist x y<= dist x z+dist z y}.
```

这个 Record 定义了度量空间，其中 Base 表示度量空间的集合的类型，dist 是这种类型上两点间的距离函数；dist_pos 表示任何两点之间的距离大于或等于 0；dist_sym 表示任意两点之间，从一点到对方的距离是相等的；dist_refl 表示如果两点的距离为 0，那么这两点为同一个点；dist_tri 表示任意两点的距离小于或等于它们到第三个点的距离的和。

3 基于 Record 的矩阵

本节介绍如何使用 Record 结构来定义任意大小的向量和二维矩阵。向量类型和矩阵类型所接受的数据类型和满足的性质都是不同的，所以它们的定义是不完全相同的；并且矩阵的定义并没有依赖于向量类型，这一点与 Nicolas Magaud 的方法不同，具体将在下文详细介绍。

3.1 向量的定义

使用 Record 结构定义向量：

```
Record Vec (A:Set) (len:nat):Set:=mkVec
  {lis:list A;
len_cond:List.length lis = len }.
```

其中，A 是向量元素的类型；*len* 是向量的长度；*lis* 是一个 list 类型，用来保存数据；*len_cond* 表示 *lis* 的长度等于 *len*。这样就定义了一个向量类型，从它的参数中就可以确定这个向量的元素类型和长度。初始化时只需要将向量的数据用一个 List 表示，再证明它满足 *len_cond* 性质，最后使用构造函数将 *lis* 和引理代入即可。对于 *lis* 的 *len_cond* 性质的验证，只需要固定的几个策略即可轻松完成。例如：定义一个长度为 3、元素类型是自然数、数据依次为 1，2，3 的向量，首先需要定义相关的 List：

```
Definition l3:= [1;0;0].
```

接着证明这个 List 的长度为 3：

```
Lemma l3_length:length l3=3.
Proof. autopro. Qed.
```

最后使用构造函数来定义这个向量：

```
Definition v:= mkVec nat 3 l3 l3_length.
```

检查 Coq 的类型，可以看到 v 的类型为：

```
v:Vec nat 3
```

3.2 矩阵的定义

矩阵的定义的实现相较于向量的定义更为复杂，因为矩阵不仅有长度（行数），还有宽度（列数）。对于数据部分，使用嵌套的列表来保存数据，也就是用 list（list A）型的列表存储数据；另外，不仅要指定列表的长度，还要指定列表中嵌套的每个列表的长度。因此，在使用 Record 来定义矩阵时，除了需要数据外，还需要两个性质。具体的定义如下：

```
Record Matrix(m n:nat):Set:= mkMat
  { mat:list (list A);
    mat_length:length mat = m;
    mat_all_len:all_len_n mat n }.
```

其中，*mat* 是一个嵌套列表，用于存储 Matrix 类型的数据部分；*mat_length* 是 Matrix 类型的一个属性，表示列表 *mat* 的长度为 *m*，*mat_all_len* 是 Matrix 类型的另一个属性，表示列表 *mat* 中所有嵌套的列表的长度都为 *n*。这样就定义出了可用于表示矩阵的 Matrix 类型，其中 *m* 代表矩阵的行数，*n* 代表矩阵的列数。可以通过如下方法定义一个元素类型为自然数的 3 行 3 列的单位矩阵。

首先对于数据部分：

```
Definition m3:=[[1;0;0];[0;1;0];[0;0;1]].
```

接着是需要满足的两个性质：

```
Lemma m3_length:length m3 = 3.
Proof. autopro. Qed.
Lemma m3_all_len:all_len_n nat m3 3.
Proof. autopro. Qed.
```

最后，使用构造函数来初始化这个矩阵：

```
Definition MI3:= mkMat nat 3 3 m3 m3_length m3_all_
len.
```

通过检查可以看到，这个矩阵的类型为：

```
MI3:Matrix nat 3 3
```

矩阵的初始化只需要将数据使用嵌套列表保存起来，而它的两个性质的证明只要使用提供的自定义策略“autopro”即可。

4 向量和矩阵的操作及基本性质

本节介绍向量和矩阵的操作函数，并且验证关于它们的

一些基本性质。由于向量和矩阵的数据部分都是使用列表来保存,因此向量和矩阵的操作实际上是对其中的列表或嵌套列表的操作,再将操作的结果转换成对应的向量或矩阵类型。因为 Coq 使用的是归纳证明的方法,所以这些操作函数的设计十分重要,因为函数设计得不合理将导致无法使用归纳法来证明它的性质,函数设计得过于复杂也将给证明带来困难。

4.1 向量的操作及基本性质

首先介绍向量的操作函数。实现某一个操作的方式有很多,但是一个恰当的函数定义至少需要满足两个条件:1)它应该是正确的,也就是它的输入、输出同所期望的结果一致;2)正如在第 2 节所说,Coq 的证明方法是数学归纳法,所以函数的定义要便于使用数学归纳法来进行性质证明,这也是函数设计中十分重要的一点。本节用向量加法来介绍函数的实现。

首先要明确的是向量加法函数类型应为:

vec_add:forall n :nat,Vec A n -> Vec A n ->Vec A n

函数的两个参数是长度为 n 的向量,最终结果仍是长度为 n 的向量。

对于 Vec 类型的操作实际上是对它所包含的 List 的操作,所以定义向量的类型操作函数之前要先定义对 List 的操作函数。例如,若要定义向量的加法函数,首先要定义 List 的加法函数:

```
Fixpoint list_add (add:A->A->A) (l1:list A) (l2:
list A):list A:=
  match l1,l2 with
  |hd1::tl1,hd2::tl2 => add hd1 hd2::(list_add add tl1
tl2)
  |_,_=> nil
  end.
```

有了这个函数就可以将两个 List 对应的值相加,从这个函数的定义中可以推出函数操作的结果仍是 List 类型,并且结果 List 的长度等于两个参数 List 中长度较小的值。将 vec 类型中的 List 取出作为这个函数的两个参数,由于这两个 vec 的类型相同,取出的 List 的长度也必然相等,函数操作的结果也必然是长度不变的 List。

正如 2.1 节所述,初始化一个向量不仅需要数据,还需证明数据满足“len_cond 性质。下面这个引理证明了两个向量类型数据相加得到的结果仍然满足“len_cond 性质,也就是两个长度为 n 的向量相加得到的向量长度也为 n。

```
Lemma vec_add_cond:forall (n:nat) (v1:Vec A n) (v2:
Vec A n),
    let l1:=lis A n v1 in
    let l2:=lis A n v2 in
    let l:=list_add A f l1 l2 in
  List.length l=n.
```

这个引理证明了两个长度相等的 List 经过 list_add 函数运算后得到的 List 的长度等于原来 List 的长度 n。有了这个引理就可以定义出完整的向量加法操作函数:

```
Definition vec_add (n:nat) (v1:Vec A n) (v2:Vec A n)
  :Vec A n:=
  let l1:=lis A n v1 in
  let l2:=lis A n v2 in
  let l:=list_add A f l1 l2 in
  let l_cond:=vec_add_cond n v1 v2 in
  mkVec A n l l_cond.
```

通过检查可以发现函数 vec_add 的类型为:

vec_add:forall A:Set,(A-> A-> A)-> forall n: nat,Vec A n-> Vec A n-> Vec A n

本文还证明了一些关于向量加法的性质。

向量的加法交换律(v1+v2=v2+v1):

```
Lemma each_eq:forall (n:nat) (v1:Vec A n) (v2:Vec A n),
  (forall a b,f a b=f b a)-> vec_eq n (vec_add n v1 v2)
(vec_add n v2 v1).
```

向量的加法分配律(v1+v2+v3=v1+(v2+v3)):

```
Lemma vec_add_assoc:forall n v1 v2 v3,(forall a b c,f (f
a b) c=f a (f b c))-> vec_eq n (vec_add n (vec_add n v1
v2) v3) (vec_add n v1 (vec_add n v2 v3)).
```

关于这些性质的证明本文不详细介绍。除此之外,本文还定义了向量的叉积函数、常数乘以向量的函数以及一些相关引理。

4.2 矩阵操作及基本性质

矩阵的操作函数比较于向量的操作函数更加困难。首先本文定义了矩阵操作的一个基本操作函数转置函数,这个函数十分重要,它是矩阵乘法的基础。与向量函数一样,矩阵函数的操作实际上也是对它所包含的数据类型(list (list A))的操作。

首先,既然是转置函数,那么先要定义一个获取某一列的函数 getcloumn,对于 list(list A)的操作就是将每一个子列表中的第 n 个元素取出(使用 getitem 函数)组成一个新的 list。起初本文将 getitem 函数定义为 list A-> nat-> option A 类型,即:

```
Fixpoint getitem (l:list A) (n:nat):=
  match n with
  |O => None
  |S O => match l with
        |nil => None
        |a::t => Some a
      end
  |S a => match l with
        nil => None
        h::t => getitem t a
      end
  end.
```

这个函数的功能是从一个列表中取出第 n 个元素,如果

存在则返回 Some A，如果不存在则返回 None。这样定义看起来是更加合理的，因为只有当这个位置存在元素时才能够取到元素，但是在后续的函数中使用这个函数时，都有一个前提，也就是 n 小于列表的长度，可以证明在这个前提条件下是不可能取到 None 值的，所以这样定义这个函数是没有必要的且增加了后续函数定义和相关引理证明的难度。本文对于 getitem 函数的定义为：

```
Fixpoint getitem (l:list A) (n:nat):=
  match n with
  |O => Zero
  |S O => match l with
          |nil => Zero
          |a::l' => a
        end
  |S n' =>match l with
          |nil => Zero
          |a::l' => getitem l' n'
          end
  end.
```

本文对 getitem 做了特殊的处理：如果 n 超出了列表的长度则默认返回 0 元素 Zero。这样做看似不合理，但是在后续的函数定义中，n 不会超过列表的长度，所以不存在取出 0 元素的情况，并且这样做可以很大程度地简化相关引理的证明。

在 getitem 函数的基础上，将 getcloumn 函数定义为：

```
Fixpoint getcolumn l n:=
  match l with
  |nil => nil
  |a::l' => (getitem a n)::getcolumn l'n
  end.
```

接着，对 list(list A)的“转置”函数的定义为：

```
Fixpoint getmatrix' l n:=
match n with
|O => nil
|S t => getcolumn l (S t)::getmatrix' l t
end.
Definition getmatrix l n:= rev (getmatrix' l n).
```

至此就实现了对 list(list A)的转置操作。

有了对嵌套列表的转置函数，就可以将它扩展为矩阵的操作函数：

```
Definition trans m n old:=
  let ll:= mat A m n old in
  let ll_length:= mat_length A m n old in
  let ll_all_len:= mat_all_len A m n old in
   mkMat A n m(getmatrix A Zero ll n) (length_getmatrix ll n) (alllen_getmatrix_t2 n m n ll ll_length ll_all_len).
```

其中，length_getmatrix 是一个引理，用来证明转置后的嵌套列表的行数等于原嵌套列表的列数，也就是满足 mat_length 性质；alllen_getmatrix_t2 是用来证明转置后的嵌套列表的列数（每一个子列表长度）等于原嵌套列表的行数（嵌套列表的长度），也就是满足 mat_all_len 性质，这样就可以使用转置后的嵌套列表和它的两个性质来将它初始成一个新的矩阵，也就是最终的转置矩阵。这个函数的类型为：

```
trans:forall m n:nat,Matrix A m n-> Matrix A n m
```

本文还定义了其他常用的矩阵操作函数。

矩阵乘法函数：

```
Definition matrix_mul m n p left right:=
  let ll:= get_mul_m' m n p left right in
  mkMat A m p ll (get_mul_m_length m n p left right) (alllen_get_mul_m m n p left right).
```

该函数的类型为：

```
matrix_mul:forall m n p:nat,Matrix A m n-> Matrix A n p-> Matrix A m p
```

常数乘以矩阵函数：

```
Definition C_mul_Matrix c m n old:=
  let new := const_mul_matrix c m n old in
    mkMat A m n new (const_mul_matrix_length c m n old) (const_mul_matrix_allen c m n old).
```

该函数的类型为：

```
C_mul_Matrix:A-> forall m n:nat,Matrix A m n-> Matrix A m n
```

这些函数的定义中都使用了辅助函数来实现，本文不一一介绍。

表 1 展示了部分已完成证明的引理，基于这些引理就可以实现基本的矩阵算法的验证。

最后，使用 Coq 提供的程序抽取机制，可以将 Coq 定义转换成 OCaml 代码，将 Coq 模块转换成 OCaml 模块。使用抽取机制的主要动机是生成满足已证性质的程序，以保障程序的可靠性。借助这一机制可以把 Coq 中定义的矩阵以及矩阵函数转换成 OCaml 的数据类型和函数。

表 1　部分引理

Table 1　Part of lemma

function	lemma_name	lemma
Vector		
vec_add	vec_add_assoc	$\boldsymbol{v}1+\boldsymbol{v}2+\boldsymbol{v}3=\boldsymbol{v}1+(\boldsymbol{v}2+\boldsymbol{v}3)$
	vec_add_comm	$\boldsymbol{v}1+\boldsymbol{v}2=\boldsymbol{v}2+\boldsymbol{v}1$
dot_product	dot_product_comm	$\boldsymbol{v}1\cdot\boldsymbol{v}2=\boldsymbol{v}2\cdot\boldsymbol{v}1$
const_mul_vec	cmv_assoc	$k*(l*\boldsymbol{v})=(k*l)*\boldsymbol{v}$
	cmv_comm	$k*(l*\boldsymbol{v})=l*(k*\boldsymbol{v})$
	cmv_distribution	$c*(\boldsymbol{v}1+\boldsymbol{v}2)=(c*\boldsymbol{v}1)+(c*\boldsymbol{v}2)$
Matrix		
matrix_mul	matrix_mul_assoc	$\boldsymbol{m}1*\boldsymbol{m}2*\boldsymbol{m}3=\boldsymbol{m}1*(\boldsymbol{m}2*\boldsymbol{m}3)$
matrix_add	matrix_add_comm	$\boldsymbol{m}1+\boldsymbol{m}2=\boldsymbol{m}2+\boldsymbol{m}1$
	matrix_add_assoc	$\boldsymbol{m}1+\boldsymbol{m}2+\boldsymbol{m}3=(\boldsymbol{m}1+\boldsymbol{m}2)+\boldsymbol{m}3$
const_mul_matrix	cmv_comm	$c1*c2*\boldsymbol{m}=c2*c1*\boldsymbol{m}$
	cmv_assoc	$(c1*c2)*\boldsymbol{m}=c1*(c2*\boldsymbol{m})$
	cmv_distribution	$c*(\boldsymbol{m}1+\boldsymbol{m}2)=(c*\boldsymbol{m}1)+(c*\boldsymbol{m}2)$

5 向量与矩阵的使用

本文定义的向量和矩阵是多态类型[13-14]的，即矩阵元素可以是任意给定类型，所有元素类型相同。多态类型的优点是减少重复性工作并且能更加方便地将向量和矩阵向各个数域扩展，在 Coq 中许多类型都是这样定义的。但是这种做法在使用时是不友好的，因为每使用一个函数都需要补充它的元素类型，如何使得这些函数和引理的使用更加简单是这一节的主要内容。

本节将介绍如何将矩阵往实数域扩展以方便将来使用。

5.1 实数矩阵

由于矩阵的定义和一些基本性质已经完成，所以实数矩阵(不仅是 Coq 库中的实数[15]，也可以是第三方库如 Coquelicot[16]中的实数)的定义很简单，只要将 A 换成 R(实数类型)，再用一个新的名称来代替扩展后的新的类型：

```
Definition mkMatR:= mkMat R.
Definition transR:=trans R 0.
Definition all_len_n_R:= all_len_n R.
Definition matrixR_mul:= mul2 R 0 Rmult Rplus.
Definition Matrix_eq3 m1 m2:=matrix_eq R 3 3 m1 m2.
Definition Matrix_mul3 m1 m2:=matrixR_mul 3 3 3 m1 m2.
Notation "m1 * m m2":= (Matrix_mul3 m1 m2) (at level 50,left associativity).
Notation "m1 =m m2":== (Matrix_eq3 m1 m2) (at level 50,left associativity).
```

使用这些函数来初始化实数矩阵会更加简便，可以将这些定义放在单独的文件里并编译好，这样以后在使用实数矩阵时只需要导入这个库即可。

使用这些函数时只需要指定矩阵的大小即可，如果已经确定了将要使用矩阵的大小并且不想每次使用函数时都指定大小，那么可以更精确地对矩阵进行扩展。比如想要使用 3 行 3 列的实数矩阵，则可以这样定义：

```
Definition mkMatR3:= mkMatR 3 3.
Definition transR3:= transR 3 3.
Definition all_len_n_R3 l:= all_len_n R l 3.
```

这样在使用时就会更加简便。

5.2 矩阵在转换矩阵验证中的使用

在飞行控制系统中，对飞行器做受力分析，根据空气动力学建立运动学和动力学方程进行飞行控制过程的设计时，使用不同的坐标系所带来的复杂程度是不一样的，因此需要在飞行控制系统中建立多个坐标系来进行受力分析。另外，为了方便分析飞行器在各个坐标系中的受力情况，需要使用转换矩阵将一个坐标系中所受的力转换到另一个坐标系中，这就是转换矩阵的作用。本节以一个飞行控制系统中转换矩阵的验证为例子来展示矩阵的使用。

已知的条件是气流坐标系 Sa、稳定坐标系 Ss、机体坐标系 Sb 之间的转换关系和气流坐标系 Sa 到机体坐标系的转换矩阵：

1)气流坐标系 Sa 在水平面旋转侧滑角 β 得到稳定坐标系 Ss。

2)稳定坐标系 Ss 在对称面旋转迎角 α 得到机体坐标系 Sb。

3)气流坐标系 Sa 到机体坐标系 Sb 的转换矩阵。

根据条件，可以推出 Sa 到 Ss 的转换矩阵为 $\boldsymbol{Sa2Ss}$：

$$\begin{bmatrix} \cos\beta & -\sin\beta & 0 \\ \sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Ss 到 Sb 的转换矩阵为 $\boldsymbol{Ss2Sb}$：

$$\begin{bmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{bmatrix}$$

已知 Sa 到 Sb 的转换矩阵为 $\boldsymbol{Sa2Sb}$：

$$\begin{bmatrix} \cos\alpha * \cos\beta & -\cos\alpha * \sin\beta & -\sin\alpha \\ \sin\beta & \cos\beta & 0 \\ \sin\alpha * \cos\beta & -\sin\alpha * \sin\beta & \cos\alpha \end{bmatrix}$$

要证明 $\boldsymbol{Sa2Sb = Ss2Sb * Sa2Ss}$。

使用本文定义的矩阵，在 Coq 中证明过程如下：

```
Parameter alpha:R.
Parameter beta:R.
```

定义 Sa 到 Ss 的转换矩阵为：

```
Definition SaSs':=
  [[cos beta  ;-(sin beta) ; 0];
   [sin beta  ; cos beta  ; 0];
   [0  ;0  ; 1]].
```

定义 Ss 到 Sb 的转化矩阵为：

```
Definition SsSb':=
  [[cos alpha  ;0  ;-(sin alpha) ];
   [0  1  ;0];
   [sin alpha  ;0  ;cos alpha]].
```

定义 Sa 到 Sb 的转换矩阵为：

```
Definition SaSb':=
[[(cos alpha) * (cos beta) ;-(cos alpha) * (sin beta);
-(sin alpha)];
[ sin beta;cos beta;0];
[ (sin alpha) * (cos beta); -(sin alpha) * (sin beta);
cos alpha]].
```

证明：初始化所需的性质：

```
Lemma SaSs'_length:length SaSs' = 3%nat.
Proof. auto. Qed.
Lemma SaSs'_alllen:all_len_n_R_3 SaSs'.
Proof. unfold all_len_n_R_3. simpl. auto. Qed.
Lemma SsSb'_length:length SsSb' = 3%nat.
Proof. auto. Qed.
```

Lemma SsSb′_alllen：all_len_n_R_3 $SsSb'$.

Proof. unfold all_len_n_R_3. simpl. auto. Qed.

Lemma SaSb′_length：length $SaSb'$ = 3%nat.

Proof. auto. Qed.

Lemma SaSb′_alllen：all_len_n_R_3 $SaSb'$.

Proof. unfold all_len_n_R_3. simpl. auto. Qed.

进行初始化：

Definition ***SaSs***：= mkMatR_3_3 $SaSs'$ SaSs′_length SaSs′_alllen.

Definition ***SsSb***：= mkMatR_3_3 $SsSb'$ SsSb′_length SsSb′_alllen.

Definition ***SaSb***：= mkMatR_3_3 $SaSb'$ SaSb′_length SaSb′_alllen.

使用 Ltac 设置自动证明策略，这个自动证明策略能够完成飞行控制中大部分的坐标转换矩阵的验证，若需要使用特殊公式化简，那么需要增加相应的引理，如使用 $\sin^2 x + \cos^2 x = 1$ 来化简三角函数。

Ltac eqprof：=

unfold Matrix_eq3；

unfold Matrix_mul3；

unfold matrixR_mul；

unfold mul2；

unfold get_mul_m′；

unfold get_mul_m；

simpl；

repeat split；ring.

证明：***SsSb*** * ***SaSs*** 的结果与已知的矩阵相同：

Lemma trans_prof：(***SsSb*** * ***m SaSs***)＝***mSaSb***.

Proof. eqprof.

Qed.

结束语　Coq 定理证明器的归纳定义机制可以方便地描述能够无穷延展的数据结构，如自然数、表和树等，然而，在 Coq 中描述具有固定大小的矩阵类型是一个困难问题。本文比较了 Coq 中现有的矩阵实现方法并研究了一种新的基于 Record 类型的矩阵实现方案，定义了一组向量和矩阵的操作函数，并验证了这些函数满足的部分性质，主要包括矩阵加法和乘法的定义以及它们的基本代数性质。同 Coq 中已有的矩阵实现方案比，本文提出的方法更为简洁、直观、易用。未来将继续完善相关引理库并进行逆矩阵及线性方程求解等方面的形式化验证工作。

参 考 文 献

[1] 吴森堂. 飞行控制系统[M]. 北京：北京航空航天大学出版社，2013.

[2] Cook M V. Flight Dynamic Principles[M]. British：Butterworth-Heinemann，2007：20-33.

[3] CHLIPALA A. Certified Programming with Dependent Types：A Pragmatic Introduction to the Coq Proof Assistant[M]. The MIT Press，2011.

[4] TEAM T C D. The Coq Proof Assistant Reference Manual -- Version V8. 7. 1[C]// INRIA. 2016.

[5] NIPKOW T，WENZEL M，PAULSON L C. Isabelle/HOL：a proof assistant for higher-order logic[C]// Springer-Verlag，2002：xiii-xiv.

[6] SHI Z，SONG X，ZHANG J，et al. Formalization of Matrix Theory in HOL4[J]. Advances in Mechanical Engineering，2014，56(5)：1-14.

[7] HARRISON J. A HOL theory of euclidean space[C]// International Conference on Theorem Proving in Higher Order Logics. Springer-Verlag，2005：114-129.

[8] MCBRIDE C. Epigram：Practical Programming with Dependent Types[M]// Advanced Functional Programming. Springer Berlin Heidelberg，2005：130-170.

[9] MAGAUD N. Programming with Dependent Types in Coq：a Study of Square Matrices[C]// Unpublished. 2005.

[10] WILSON S，FLEURIOT J，SMAILL A. Inductive Proof Automation for Coq[R]. England：The University of Edinburgh，2010：3-9.

[11] DENES M，BERTOT Y. Experiments with computable matrices in the Coq system[R]. France：INRIA，2011：2-27.

[12] MULLEN E，PERNSTEINER S，WILCOX J R，et al. Euf：minimizing the Coq extraction TCB[C]// The ACM Sigplan International Conference. ACM，2018：172-185.

[13] SOZEAU M，TABAREAU N. Universe Polymorphism in Coq[M]// Interactive Theorem Proving. Cham：Springer，2014：499-514.

[14] ZILIANI B，SOZEAU M. A unification algorithm for Coq featuring universe polymorphism and overloading[C]// ACM Sigplan International Conference on Functional Programming. ACM，2015：179-191.

[15] KREBBERS R，SPITTERS B. Type classes for efficient exact real arithmetic in Coq[J]. Logical Methods in Computer Science，2013，9(1)：51-59.

[16] BOLDO S，LELAY C，MELQUIOND G. Coquelicot：A User-Friendly Library of Real Analysis for Coq[J]. Mathematics in Computer Science，2015，9(1)：41-62.