软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.006255]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于 Coq 的分块矩阵运算的形式化

麻莹莹[1], 马振威[2], 陈 钢[1]

[1](南京航空航天大学 计算机科学与技术学院,江苏 南京 211106)

[2](上海寻梦信息技术有限公司,上海 200051)

通讯作者: 陈钢, E-mail: gangchensh@nuaa.edu.cn

摘 要: 矩阵是工程领域中常用的一种数据结构,在深度学习领域,矩阵乘法是神经网络训练中的核心技术之一,面对大型矩阵的运算问题,分块矩阵技术可将大矩阵运算转换为小矩阵运算以实现并行运算,并且能够大幅度减少矩阵运算步骤并且提高矩阵运算速度.本文首先对目前学术界的矩阵形式化工作进行了系统总结并且分析了矩阵形式化的主要几种方法;其次介绍并完善了基于 Coq 记录类型的矩阵形式化方法,其中包括提出新的矩阵等价定义、对之前的形式化工作进行了整理和完善,并证明了一组新的引理;在此基础上进一步实现了分块矩阵运算的形式化,讨论了该类型的归纳证明的难点和解决方法;最终实现了矩阵与分块矩阵形式化的不同类型的基础库.

关键词: 矩阵;形式化方法;分块矩阵;深度学习;形式化工程数学;高阶定理证明;Coq

中图法分类号: TP311

中文引用格式: 麻莹莹,马振威,陈钢.基于 Coq 的分块矩阵运算的形式化.软件学报,2021,32(6).

英文引用格式: Ma YY, Ma ZW, Chen G. Formalization of operations of block matrix based on Coq. Ruan Jian Xue Bao/Journal of Software, 2021,32(6).

Formalization of Operations of Block Matrix Based on Coq

MA Ying-Ying[1], MA Zhen-Wei[2], CHEN Gang[1]

[1](School of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China)

[2](Shanghai Xun-Meng Information Technology Co., Ltd., Shanghai 200051, China)

Communicating author: CHEN Gang, E-mail: gangchensh@nuaa.edu.cn

Abstract: Matrix is a commonly used data structure in the field of engineering. In the field of deep learning, matrix multiplication is one of the key technologies in neural network training. Faced with the problem of operations of large matrices, the block matrix technology can be used to convert large matrix operations into small matrix operations to realize parallel computation, which can greatly reduce matrix operation steps and improve the efficiency of matrix operation. Firstly, this paper systematically summarizes the current matrix formalization work in academia and analyzes the main methods of matrix formalization. Secondly, it introduces and improves the matrix formalization method based on Coq record type, which includes putting forward a new definition of matrix equivalence, sorting out and perfecting the previous formalization work and proving a new set of lemmas; then on this basis, the formalization of block matrix operations is further realized, and the difficulties and solutions of this type of inductive proof are discussed. Finally, basic libraries with different types for matrix and block matrix formalization are realized.

Key words: matrix; formal method; block matrix; deep learning; formalized engineering mathematics; higher order theorem proving; Coq

1 引言

矩阵是工程领域中常用的一种数据结构.比如,飞行控制系统中使用矩阵实现坐标转换和动力学方程计算,在深度学习领域,矩阵乘法是神经网络训练中的核心关键计算.但是,传统命令式语言中的矩阵运算程序容易产生各种错误,比如下标越界,这些错误会引起非法存储访问从而导致程序崩溃.深度学习的引入使得矩阵计算进

收稿时间: 2020-09-03; 修改时间: 2020-10-26, 2020-12-19; 采用时间: 2021-01-18; jos 在线出版时间: 2021-02-07

一步复杂化,也增加了矩阵计算错误的可能性.深度学习中的超大规模矩阵计算往往超出传统CPU的能力,为了提高矩阵计算的速度,往往需要采用 GPU、超级计算机或者专用的硬件加速器,这些特殊的硬件计算装置通过引入并行计算来大幅度减少矩阵计算的时间.实现矩阵并行计算的一个核心技巧就是把矩阵分解成分块矩阵,把原本顺序执行的矩阵计算转变成并行执行的分块矩阵计算,然后把这些并行计算任务分配到相应的硬件资源上,通过硬件并行计算大幅度提高矩阵计算速度.然而,分块矩阵的计算进一步提升了矩阵算法的复杂性和可理解性,也使得矩阵计算代码更容易出现错误.

传统的程序测试方法并不能全面的挖掘矩阵程序中的所有错误.形式化方法[1]为提高软件可靠性提供了更有力的技术.从原理上讲,软件的形式化方法就是用数学方法证明软件在任何情况下都满足所期望的性质.形式化方法分成很多种类,一些方法(比如模型检查)通过自动证明工具验证软件满足某些性质,但是这种方法通常不能保证软件满足所有期待的性质.本文采用基于高阶定理证明器的形式化验证方法,这种方法使用通用的定理证明工具,在高阶逻辑系统中全面验证软件性质.基于定理证明[2]的验证方法的工作原理同数学定理证明的过程类似,它需要首先证明一组中间引理,然后在这些中间结果的基础上证明最后的定理,这一方法同数学研究一样需要逐步证明和积累数学定理.因此,为了矩阵计算软件的验证,我们一方面需要在定理证明器中描述矩阵计算的各种函数,另一方面需要证明这些函数所需满足的各种数学性质.在矩阵软件形式化的过程中,我们需要首先证明矩阵的各种数学性质,比如矩阵加法的结合律交换律,乘法结合律,加法和乘法的分配律,加法和乘法同 0 元和单位元作用的有关性质等等.这样一个过程,我们称为矩阵的形式化.

分块矩阵在神经网络的计算中起到特别重要的作用.神经网络的训练是一种非常耗时的计算工作,主要的计算量用于超大规模的矩阵的加法和乘法.为了减少计算时间,人们引入了各种硬件计算进行矩阵运算加速,这些技术包括多核计算、高性能计算、GPU、TPU 及各种专用的硬件加速器、FPGA 等等.这些硬件提速的核心原理就是把矩阵运算并行化,而并行化的基本技术之一就是采用分块矩阵.分块矩阵的实现代码比普通矩阵代码要复杂的多,更难以理解,因此也更需要通过形式化方法进行软件验证.

本文的主要贡献是在对基于Coq记录的矩阵形式化方法[3]的整理和完善的基础上,进一步实现了分块矩阵运算的形式化,接着详细分析了在矩阵形式化的证明过程中所遇到的难点,总结了归纳证明的经验,最后形成了矩阵与分块矩阵的基础库,并利用 Coq 代码抽取机制将矩阵及分块矩阵运算函数转换为可执行的 OCaml 代码.在文献[3]中构建的理论是模块化的,矩阵理论是一个函子,当它作用到一个关于元素类型的模块之后可以产生对应于这类元素的具体的矩阵理论.用这种方法我们可以获得自然数、整数、实数和复数的矩阵理论,对于矩阵函子中证明的加法和乘法性质在实例化的理论中依然有效,避免了大量的重复性证明.但是,上述这个方法并不能应用于分块矩阵,原因如下:一是普通矩阵的等价判断函数不适用于分块矩阵;二是矩阵函子所形成的分块矩阵模块中的乘法与转置函数不能产生正确结果;三是矩阵函子要求输入的用于描述矩阵元素的模块中的加法和乘法都满足交换律,分块矩阵是元素为矩阵的矩阵,而矩阵的乘法并不满足交换律,因此,无法通过函子作用直接获得分块矩阵的主要性质.所以,在这篇文章中,我们对分块矩阵的形式化进行了专门的研究.

在 Coq 中定义的矩阵函数是函数式风格的函数.函数式编程可靠性高但计算效率低,因此函数式矩阵运算本身并不适合直接在硬件中实现.不过,深度学习软件的发展趋势是把算法的高层描述同算法的底层优化分离开来,一个有代表性的做法就是亚马逊公司开发的 TVM 框架,它定义了两种语言,一种是面向用户的高层算法描述语言,另一种是面向实现的用于描述优化策略的策略语言.在这种工作方式之下,高层算法的描述并不一定需要采用命令式语言,只要是任何一种便于编译优化的语言即可.文献[4]即采用了基于函数式语言的高层算法描述方法,并且提出了比 TVM 更为简洁有效的策略优化技术.这一工作也说明,基于函数式语言的深度学习算法高层描述不但具有可行性,而且具有便于代码优化的良好特性.本文不但描述了基于函数式语言的分块矩阵计算基本函数,并且通过形式化方法验证了它们的重要特性,保障了这些函数的可靠实现.因此,这项工作为深度学习算法的并行实现及形式验证提供了基础性的支撑.

本文第 2 节介绍了 Coq 定理证明器并且总结分析了矩阵形式化的相关工作;第 3 节介绍并完善了基于 Coq 记录的矩阵形式化方法;第 4 节提出了基于 Coq 记录的分块矩阵形式化方法;第 5 节对矩阵与分块矩阵形式化

证明中所遇到的困难进行了总结和分析;最后总结全文.

2 背景知识

2.1 Coq定理证明器

Coq 定理证明器[5]是基于高阶带类型λ演算的交互式高阶逻辑定理证明器.它不但表达能力强,而且具备多个具有实用价值的证明机制,比如可扩充的重写机制、基于模块的形式化理论构建机制以及程序抽取机制等等.这些优秀的特点使得这一证明工具在学术界越来越受欢迎,成为当今最为流行的定理证明工具之一.

Coq 基于归纳构造演算,有着强大的数学模型基础和很好的扩展性.Coq 中的归纳类型扩展了传统设计语言中有关类型定义的概念,这与大多函数式程序设计语言中的递归类型定义相似.基于模式匹配和递归,每个归纳类型对应一个计算结构,对于含有递归类型参数（如自然数、表等）的目标证明通常采用归纳证明方法,归纳证明的过程取决于该递归类型的归纳定义.每个归纳声明定义了一组数据值,这些值可以用声明过的构造子来构造.例如布尔值可以用 true 和 false 来构造;自然数可以用 0 或 S 应用到另一个自然数上来构造;而列表可以用 nil 或者将 cons 应用到一个自然数和另一个列表上来构造.对于归纳类型进行归纳证明原理与数学归纳法类似,以表类型为例证明步骤：1）首先,证明当表 list1 为 nil 时 P list1 成立;2）然后,证明当表 list1 为 cons n list1’时 P list 成立,其中 n 是列表头元素,list1’是某个更小的列表,假设 P list1’成立.利用归纳证明所证明的引理能够保证该引理内存在的情况都满足引理内容.

与许多其他编程语言（C,Java,Pascal 等）相似,我们可以在 Coq 中定义块机制,在块（Section）内部,我们可以声明或定义局部变量,并控制他们的作用域.在 Coq 系统中,各个 Section 都有相应作用范围,系统分别使用“Section id”和“End id”表示 Section 的开始和结束.Section 可以嵌套,在其内部我们可以使用关键字“Parameter”和“Variable”来进行全局声明和局部声明,在 Section 结束之后,局部变量会从当前上下文中消失,而在 Section 内部使用到这些变量的定义以及引理都会增加局部变量类型作为参数.

Coq 提供了程序抽取功能,可以将经过认证的程序提取为 OCaml、Haskell 或者 Scheme 之类的语言,在程序抽取时可将 Coq 中指定类型按一定规则映射到指定语言如 OCaml 含有的类型之上.

2.2 相关工作

定理证明技术已成为当今软件工程领域中一种非常重要的形式化技术,该技术已经广泛应用于数学定理证明、协议验证以及硬软件的安全特性验证,是人们解决软件系统正确性、可靠性的重要方法.

在基于定理证明器的矩阵形式化研究中,已存在部分相关工作：在 Isabelle/HOL 和 Coq 中已有向量的形式化[6],[7],[8].Mizar 中描述了向量空间以及线性变换特征值[9],[10].John Harrison 则在 HOL Light 中对欧几里德空间实向量以及矩阵正交变换进行了较完整的形式化[11].在首都师范大学施智平团队中也开展了部分有关矩阵形式化的工作：刘振科等人提出函数矩阵理论在 HOL4 中的形式化[12];康西楠在 HOL4 中已有的矩阵基本性质的基础上给出矩阵变换形式化理论[13];杨秀梅等人在 HOL4 中对向量和函数矩阵进行形式化,并对其连续性、微分以及积分性质进行验证[14].马振威提出基于 Coq 记录的矩阵形式化方法[3].

在矩阵形式化的研究中,首先要解决的一个关键问题是矩阵在定理证明器中的表示方法.一种是把矩阵定义成从下标集合到元素集合的函数映射基于 Isabelle/HOL 系统的研究工作主要采用了这种方法.这种方法的优点在于比较容易证明矩阵的各种性质.在 Coq 的 ssreflect 库中包含的矩阵库也使用该方法的思路定义二维矩阵[15],将矩阵定义为由自然数矩阵生成实数的函数,但该实现比较复杂,并且不易理解以及使用.这类利用函数来实现矩阵形式化的方法有一个缺点,即没有很好的对应计算机软件中的矩阵计算函数,也就是说,这些方法从数学上证明了各种矩阵性质,却没有很好的验证软件中的矩阵函数的性质.

在基于 Lambda 演算的定理证明器中（比如 HOL/ISABELLE、Coq）,可以用表 list 来表示矩阵,并在表的基础上定义矩阵操作函数,并进行矩阵性质的证明.但是,表本身没有长度限制,表类型也没有反映出指定长度或大小的向量和矩阵.在基于依赖类型的定理证明器中,可通过依赖类型定义出可设定长度的表,Nicolas

Magaud 基于依赖类型定义了带有长度的向量类型,其定义如下：

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

该定义与表类型 list 相似,不同在于其增加了一个表示列表长度的参数（类型为自然数）.然而,这一技术在操作和证明上都十分繁琐,不仅无法利用表结构上的各种已知函数,而且当处理更为复杂的二维矩阵问题时将面临较大的困难.

在 Coq 的 Coquelicot 库中也有一种矩阵形式化方法[16],该方法与上述方法都基于依赖类型实现了定长向量,其利用对 pair 函数的递归实现向量,矩阵由向量的嵌套实现,该方法的相比于上述方法构造更加巧妙、易于理解、矩阵函数的实现简单,但后续函数性质的证明较少.

马振威提出一种基于 Coq 记录类型的方法来实现矩阵的形式化 [3]该方法用包含三个分量的记录来表示二维矩阵,其中一个分量是实现矩阵的一个二维表,另外两个分量分别是对矩阵的长度和宽度满足指定长宽的两个证明.这种方式易于理解,能够让我们能够定义出具有指定长宽的矩阵类型,并且我们能够在向量类型和矩阵类型上定义基本的矩阵操作函数（矩阵加法,转置,乘法等）.文献[3]证明一组矩阵操作的基本性质.这种实现矩阵形式化方法的好处在于易于理解并且方便使用,利用该方法构建的函数可直接通过程序抽取得到可执行的 OCaml 代码.但是,在文献[3]中有一个问题没有得到很好的解决.它就是矩阵的相等性问题.对于两个基于记录的矩阵,传统等号并不能准确表达它们的相等性,因为两个记录相等当且仅当这两个记录的所有分量相等.但对于两个列表相同的矩阵记录,它们所包含的关于长度和宽度的证明是不必要求相等的.

目前,我们还未看到有关于分块矩阵运算的形式化研究工作.

本文期望在解决基于 Coq 记录类型的方法存在问题的基础上对更多矩阵函数性质进行补充证明,并且利用该技术对分块矩阵及其运算形式化,最后不仅能够构建多个不同类型的矩阵形式化的基础库共后续研究人员使用,还能为从事该研究方向的人员提供一些经验上的参考.

3 矩阵形式化

本次工作所采用的矩阵形式化方法是对文献[3]中的矩阵形式化方法的完善,其中包括矩阵等价定义、部分矩阵函数修改以及矩阵函数性质的补充证明.本节介绍基于 Coq 记录的矩阵形式化方法,其中矩阵类型定义、矩阵函数定义方法以及矩阵函数性质证明思路.

3.1 矩阵类型定义

Record 类型是一个允许定义记录的宏,使用 Record 可以定义出一种数据结构,该数据结构中具有一些数据,而这些数据满足某种性质,如下即为利用 Record 定义的矩阵类型：

```
Record Mat （A:Set）(m n:nat) : Set := mkMat
    {
    mat : list (list A);
    mat_height : height mat = m;
    mat_width : width mat n
  }.
```

其中,A 表示矩阵内部元素类型,m 表示矩阵行数,n 表示矩阵列数,height 函数用于返回二维表高度,width 函数用于判别输入二维表的宽度是否等于输入参数,mat 用于存储矩阵内部数据即二维表,mat_height 为二维表的高度的等价证明,mat_width 为二维表的宽度的等价证明.因此在创建一个大小为 m×n 的矩阵时,需要提供一个二维表、该二维表的高度性质证明以及该二维表的宽度性质证明.以下是构建一个 3×3 的矩阵 Tm1 的例子：

```
Definition tm1 :=[[1;2;3];[2;3;4];[3;4;5]].
Lemma tm1_cond1: height tm1 = 3.
Lemma tm1_cond2: width tm1 3.
Definition Tm1:= mkMat nat 3 3 tm1 tm1_cond1 tm1_cond2.
```

由于构建一个矩阵所需要写的代码过多,因此我们可以将固定尺寸的矩阵构造封装成函数,后期构造时只需输入矩阵内部元素即可得到所构造的矩阵,如构建 3×3 矩阵函数 mkMat_3_3：

```
Definition mkMat_3_3' (a11 a12 a13 a21 a22 a23 a31 a32 a33: A):=
  [[a11;a12;a13]; [a21;a22;a23]; [a31;a32;a33]].
Definition cond2_mkMat_3_3':forall a11 a12 a13 a21 a22 a23 a31 a32 a33,
width (mkMat_3_3' a11 a12 a13 a21 a22 a23 a31 a32 a33) 3%nat.
Definition mkMat_3_3 (a11 a12 a13 a21 a22 a23 a31 a32 a33 :A):=
  mkMat A 3 3 (mkMat_3_3' a11 a12 a13 a21 a22 a23 a31 a32 a33)
  eq_refl(cond2_mkMat_3_3' a11 a12 a13 a21 a22 a23 a31 a32 a33).
```

其中 eq_refl 是自定义的一个证明目标,其能够自动证明二维表的高度.在此基础上 Tm1 的构建代码大大减少,如下：

```
Definition Tm1:= mkMat_3_3   1 2 3   2 3 4   3 4 5.
```

3.2 矩阵类型等价性质

在文献[3]中对矩阵类型的等价判断是采用 Coq 中的 eq 函数表示,其采用一个公理 Meq 作为二维表（即矩阵内部数据）相等与矩阵相等的转换理论,如下：

```
Axiom Meq : forall (A:Set) (m n:nat) (m1 m2:Mat A m n) , mat A m n m1 = mat A m n m2 <-> m1 = m2.
```

其中 A 为矩阵元素类型,m 和 n 分别为矩阵的高度以及宽度,m1 和 m2 为元素类型为 A、大小为 m×n 的矩阵,mat 是获取矩阵内部二维表的函数,其参数分别为元素类型、矩阵高度、矩阵宽度以及矩阵.由于 eq 函数对于 Record 类型相等要求的特性,即要求 Record 类型中每一条记录相等,而当两个矩阵相等时（即内部数据相等）,不要求其关于该数据的高度以及宽度证明相等,因此该公理存在部分错误,即由二维表相等无法推导出矩阵相等.

为了解决该公理存在的问题,我们提出了一个新的矩阵等价判断函数 M_eq,定义如下：

```
Definition M_eq{A:Set} {m n:nat} (m1 m2:Mat A m n ):= mat A m n m1 = mat A m n m2.
```

其实现原理如下图所示：

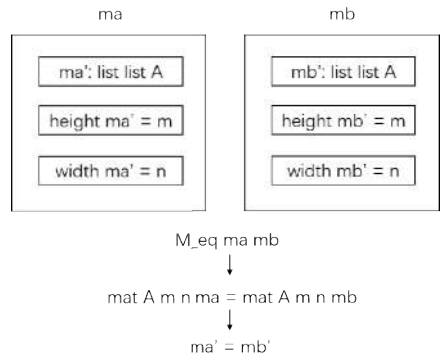

Fig.1 Implementation principle of M_eq function

图 1　M_eq 函数实现原理

该函数用于判断两个矩阵类型中的二维表（即矩阵内部数据）是否相等,该判断方法有效避免了关于 Record 类型使用 eq 函数进行等价判断所存在的问题,也消除其使用错误公理而产生的漏洞.该函数具有自反性、对称性以及传递性,因此我们借助 Coq 中的重写扩张机制构建重写性质,使其成为一个重写可用的二元关系.下面使用“===”标记表示 M_eq 函数.下面是 M_eq 的自反性、对称性和传递性的引理表示：

```
Lemma M_eq_ref : forall {A:Set} {m n:nat} (m0:Mat A m n), m0 === m0.
Lemma M_eq_sym : forall {A:Set} {m n:nat} (m0 m1:Mat A m n),
  m0 === m1 -> m1 === m0.
Lemma M_eq_trans : forall {A:Set} {m n:nat} (m0 m1 m2:Mat A m n),
  m0 === m1 -> m1 === m2 -> m0 === m2.
```

3.3 矩阵函数构造

关于矩阵类型的运算操作函数主要分为矩阵取负运算函数、加减运算函数、矩阵数乘运算函数、矩阵乘法运算函数、以及矩阵转置函数.矩阵处理函数的构造主要分为三个步骤,一是构建对二维表处理操作函数,二是完成该函数输出二维表的高度和宽度等价证明,三是利用前两者构建处理矩阵并且生成矩阵的函数（封装）.采用该方法构造的矩阵处理函数使得矩阵的高度与宽度证明贯穿于函数处理过程中,能够真正的保证输入与输出矩阵高度与宽度固定,因此无法产生矩阵越界错误.以矩阵乘法函数为例,矩阵乘法的原理如下：

若 A 为 m×n 矩阵,B 为 n×p 矩阵,则其两者的乘积 AB 会是一个 m×p 矩阵.

$$(AB)_{ij} = \sum_{r=1}^{n} a_{ir} b_{rj}$$

矩阵乘法函数的构造需依次构造多个递归函数,其分别为 product’,l_mul_dl、dl_mul_dl、mat_mul_mat.其中 product’函数为表的内积函数,其接受两个表作为输入参数并返回其内积结果.l_mul_dl 函数是表与二维表相乘函数,其利用递归将 l 与 m 中每个表分别进行内积,将其结果以表类型输出,在完成该函数的构造后还需对其输出结果（表）的长度进行证明,如下：

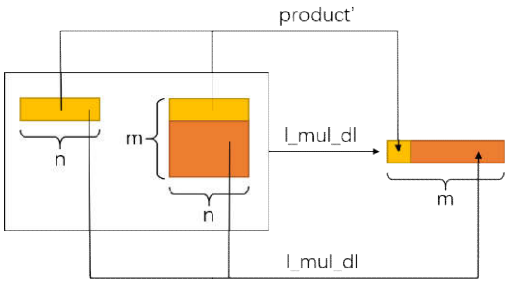

Fig.2　Implementation principle of l_mul_dl function

图 2　l_mul_dl 函数实现原理

dl_mul_dl 为二维表与二维表的相乘函数,其基于 l_mul_dl 函数利用递归实现两个二维表中的表依次内积,并以二维表类型输出,在完成该函数的构造后还需对其输出结果（二维表）的高度与宽度进行证明,如下：

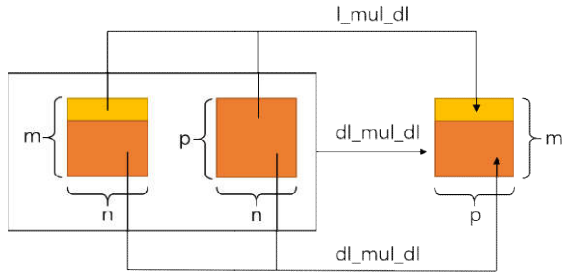

Fig.3 Implementation principle of dl_mul_dl function

图 3 dl_mul_dl 函数实现原理

mat_mul_mat 以两个矩阵类型作为输入参数,由于矩阵乘法原理要求乘号左边的矩阵的每一行乘上与乘号右边的矩阵的每一列作内积,根据 dl_mul_dl 函数的实现原理需要将第二个矩阵进行转置后再进行运算,在完成该函数的构造后还需对其输出结果（二维表）的高度与宽度进行证明,如下:

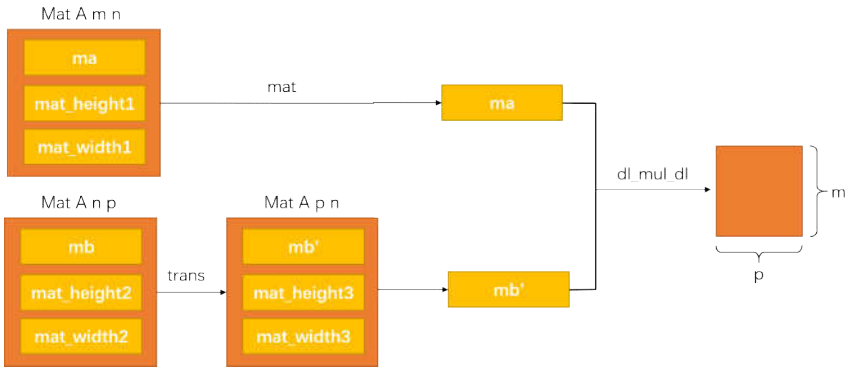

Fig.4 Implementation principle of mat_mul_mat function

图 4 mat_mul_mat 函数实现原理

根据以上函数以及有关函数输出结果高度与宽度证明性质构造矩阵乘法函数 matrix_mul,其定义如下：

```
Definition matrix_mul {m n p:nat}(left :Mat A m n)( rightMat A n p):=
  let ll := mat_mul_mat left right in
  mkMat A m p ll (height_mat_mul_mat left right)(width_mat_mul_mat left right).
```

该函数是将 mat_mul_mat 函数以及该函数输出结果的高度与宽度等价性质三部分封装成一个输入为矩阵类型、输出为矩阵类型函数.

3.4 矩阵函数性质证明

Coq 采用反向推理的方法构造证明,即在证明一条引理时会产生一个证明目标,需要通过使用一组引理以及证明命令来产生保证该目标的成立的子目标,当所有子目标已知或为公理时,则完成证明;反之,则需要继续对子目标进行证明,直到所有的目标都得到证明.对于矩阵函数性质的证明需要根据矩阵函数内部定义来选择归纳目标从而产生符合预期的子目标,此类有关于矩阵的性质证明难点之一在于证明思路的构建,证明思路决定了变量是否需要归纳以及变量归纳证明的顺序,证明思路需要根据函数内部定义而建立,下面以矩阵乘法右分配律为例：

矩阵乘法右分配律形式为$A \times (B + C) = A \times B + A \times C$,如下图所示：

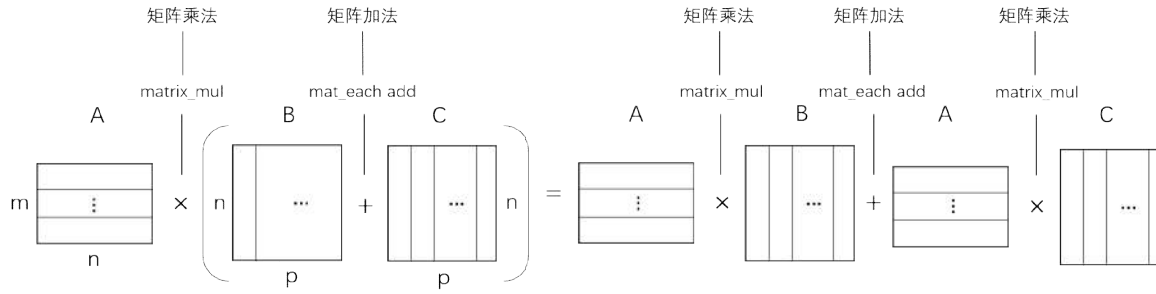

Fig.5 The right distributive law of matrix multiplication function and matrix addition function

图 5 矩阵乘法函数与矩阵加法函数的右分配律

在证明该定理时需要将变量进行归纳,该定理中的变量有 m、n、A、B、C,在 3.3 节中介绍矩阵乘法函数（matrix_mul）定义的实现是由表与表之间的内积函数、表与二维表的乘法函数、二维表与二维表的乘法函数到矩阵以及矩阵乘法函数（返回二维表）的顺序实现的,如下：

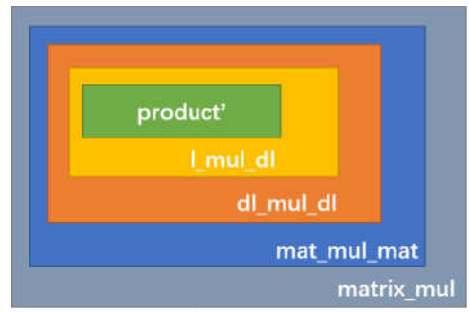

Fig.6 The structure of the definition of matrix multiplication

图 6 矩阵乘法函数定义结构

由于 matrix_mul 为非递归函数,其与矩阵加法函数的右分配律性质展开后形成的证明目标将作为一条新待证引理（mat_mul_mat 函数同理）,完成该待证引理的证明即可完成该性质的证明,下图为 matrix_mul 函数展开后需要证明的子目标即 mat_mul_mat 函数与二维表加法函数的右分配律：

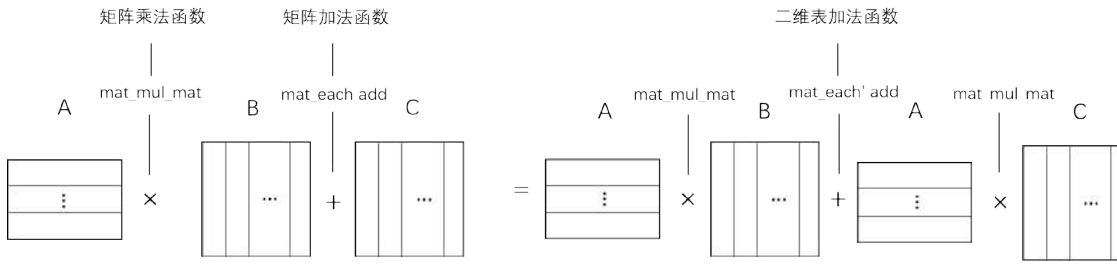

Fig.7 The right distributive law of mat_mul_mat function and two-dimensional list addition function

图 7 mat_mul_mat 函数与二维表加法函数的右分配律

由于 mat_mul_mat 是一个非递归函数,其是在 dl_mul_dl 函数的基础之上定义的,主要功能是分别取出两个输入矩阵中的二维表,并对第二个二维表进行转置,再利用 dl_mul_dl 函数对两个二维表进行乘法函数操作,将 mat_mul_mat 函数展开化简后得到需要证明的两个子目标,分别是二维表转置函数与二维表加法性质

$((A+B)^T=A^T+B^T)$和 dl_mul_dl 函数与二维表加法右分配律性质.

二维表转置函数与二维表加法性质的证明在这部分不详细介绍,dl_mul_dl 函数与二维表加法右分配律如下,此处以及下文代码部分采用 Section 机制 A、m 和 n 为局部变量(A: Set; m n:nat):

```
Lemma dl_mul_dl_distr_r : forall m n (a b c:list(list A)),
  width a n -> width b n -> width c n -> height b = m -> height c = m ->
  dl_mul_dl a ( mat_each' A add b c) = mat_each' A add (dl_mul_dl a b)(dl_mul_dl a c).
```

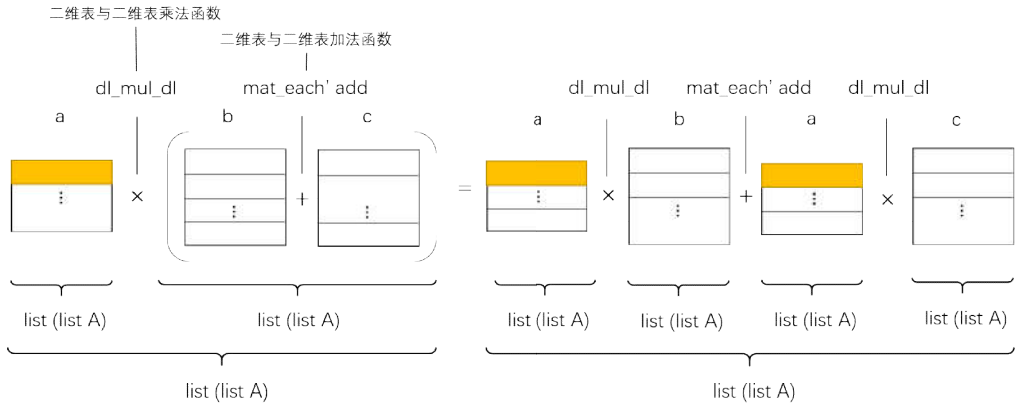

Fig.8 The right distributive law of two-dimensional list multiplication and two-dimensional list addition

图 8 dl_mul_dl 函数与二维表加法的右分配律

由于 dl_mul_dl 函数定义在 l_mul_dl 函数之上,利用数学归纳法对二维表变量 a（图 8）进行归纳展开时,当 a 的形式为 a::a0 时,会取出其二维表中的第一个子表,并拆分成含有 l_mul_dl 的形式,如下所示:

```
l_mul_dl a (mat_each' A add b c) :: dl_mul_dl a0 (mat_each' A add b c) =
list_each A add (l_mul_dl a b) (l_mul_dl a c)
:: mat_each' A add (dl_mul_dl a0 b) (dl_mul_dl a0 c)
```

这提示二维表乘法与二维表加法的右分配律定理的证明需要一个新定理的证明支持,即表与二维表乘法函数 l_mul_dl 与二维表加法的右分配律:

```
Lemma l_mul_dl_distr_r : forall m n (a:list A) (b c:list(list A)) ,
  length a = n -> width b n -> width c n -> height b = m -> height c = m ->
  l_mul_dl a (mat_each' A add b c)
  = list_each A add (l_mul_dl a b) (l_mul_dl a c).
```

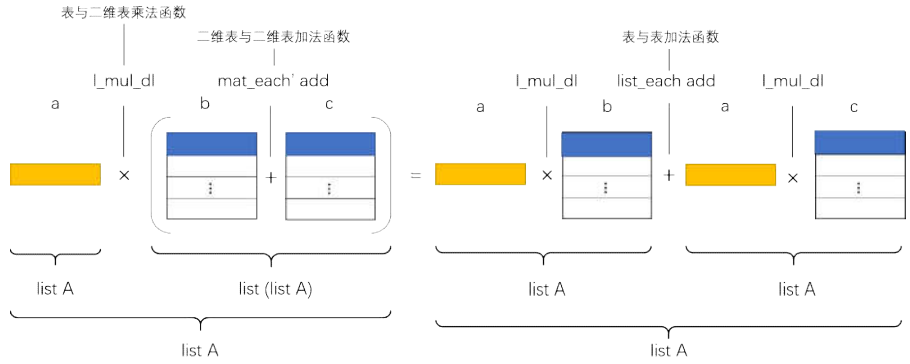

Fig.9 The right distributive law of list and two-dimensional list multiplication and two-dimensional list addition

图 9 表与二维表乘法与二维表加法的右分配律

由于 l_mul_dl 函数定义在 product'函数之上,再次利用数学归纳法对二维表变量 b 和 c(图 9)进行归纳展开时,当 b 的形式为 a3::b 且 c 的形式为 a4::c 时,会取出其二维表中的第一个子表,并拆分成含有 product'的形式,如下所示:

```
product' A Zero add mul a0 (list_each A add a3 a4) =
add (product' A Zero add mul a0 a3) (product' A Zero add mul a0 a4)
```

这提示表与二维表乘法和二维表加法的右分配律定理的证明需要一个新定理的证明支持,即表内积函数 product'与加法 add 的右分配律:

```
Lemma product_distr_r : forall n (a b c:list A) ,
  length a = n -> length b = n -> length c = n ->
  product' A Zero add mul a (list_each A add b c)
  = add (product' A Zero add mul a b) (product' A Zero add mul a c).
```

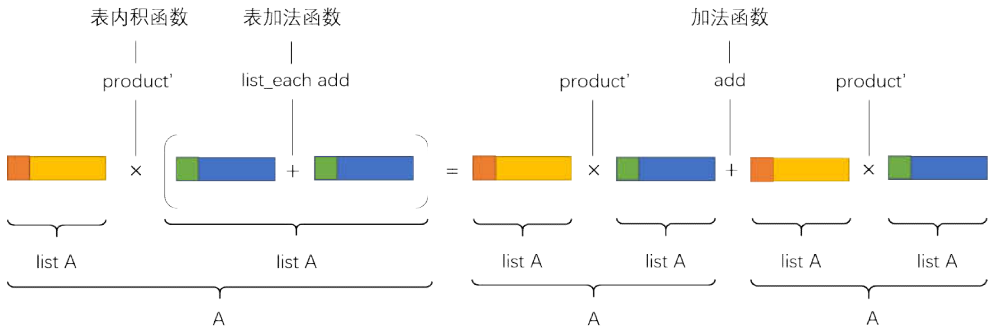

Fig.10 The right distributive law of inner product function and addition

图 10 product'函数与加法的右分配律

由于 product'函数定义在 mul 函数之上,利用数学归纳法对 a、b 和 c(图 10)进行归纳展开时,当 a 的形式为 a0::a,b 的形式为 a1::b 且 c 的形式为 a2::c 时,利用化简拆分成含有 mul 的形式,如下所示:

```
add (mul a (add a1 a2)) (product' A Zero add mul a0 (list_each A add b c)) =
add (add (mul a a1) (product' A Zero add mul a0 b))
  (add (mul a a2) (product' A Zero add mul a0 c))
```

这提示表内积函数与加法的右分配律定理的证明需要一个新定理的证明支持,即乘法 mul 与加法 add 的右分配律：$a*(b+c)=a*b+a*c$,其表示如下：

Fig.11 The right distributive law of multiplication and addition

图 11 乘法与加法的右分配律

大部分的乘法与加法都具有右分配律性质,因此只要在证明中添加该定理作为前提,即可通过反向推导证明矩阵乘法与矩阵加法具有右分配性质.

上述依次需要证明的定理整理如下：

Table 1 Lemmas needed to prove the right distributive law of matrix multiplication

表 1 矩阵乘法右分配律证明所需引理

序号	引理内容	引理名称
1	乘法与加法的右分配律	mul_distr_r
2	表内积函数与加法的右分配律	product_distr_r
3	表与二维表乘法和二维表加法的右分配律	l_mul_dl_distr_r
4	二维表乘法与二维表加法的右分配律	dl_mul_dl_distr_r
5	矩阵乘法与矩阵加法右分配律	matrix_mul_distr_r

在利用数学归纳法进行归纳证明,尤其当证明的性质中含有多个变量时,不建议在证明开始时盲目对所有变量进行统一归纳展开,一是因为在有些变量展开后证明中的前提条件产生假命题,在这种情况下可以直接跳过该目标证明,若在该情况下继续进行对其他变量进行归纳展开会增加许多重复代码与不必要的证明工作;二是对于变量是否进行归纳展开主要依据在于证明思路,如在图 8 中含有三个变量 a、b 和 c,选择 a 作为主要归纳对象的原因在于 dl_mul_dl 函数在运算过程中首先对左侧变量进行模式匹配,当 a 归纳为不同结构时可进行化简并且能够进一步产生与 l_mul_dl 函数相关引理的形式,若先对 b 或 c 进行归纳,不仅不能进一步化简,还会使得证明目标变得更加繁琐.但对于一些更为复杂的引理如乘法结合律以及单位矩阵乘矩阵性质,没有明确的证明思路时,需要构造特殊的函数以及引理来支持其每一个归纳证明目标的证明,这使得这些引理的证明更加复杂并且难以理解,具体的部分我们将在第 5 节详细讨论.

本次工作在已有引理基础上对未完成的引理进行补充以及证明, 其内容如下：

Table 2 Existing matrix lemma

表 2 现有矩阵引理

函数	引理	引理名字	引理内容
matrix_each add	加法交换律	matrix_add_comm	A+B = B+A
	加法结合律	matrix_add_assoc	(A+B)+C=A+(B+C)
	左加零矩阵性质	matrix_add_zero_l	O+A = A
	右加零矩阵性质	matrix_add_zero_r	A+O = A
matrix_each sub	减法交换律	matrix_sub_opp	A-B = -(B-A)
	减法结合律	matrix_sub_assoc	A-B-C = A-(B+C)
matrix_mul	乘法左分配律	matrix_mul_distr_l	(A±B)×C=A×C±B×C
trans	转置性质	tteq	$(A^T)^T = A$

Table 3 Supplemental matrix lemma

表 3 补充矩阵引理

函数	引理	引理名字	引理内容
matrix_each sub	左减零矩阵性质	matrix_sub_zero_l	O-A = -A
	右减零矩阵性质	matrix_sub_zero_r	A-O=A
	自身相减性质	matrix_sub_self	A-A = O
const_mul_l const_mul _r	数乘交换律	const_mul_comm	c1*A = A*c1
	数乘左分配律	const_mul_l_distr_l	(c1+c2)*A = c1*A+c2*A
	数乘右分配律	const_mul_l_distr_r	c*(A±B) = c*A ± c*B
	数左乘 0 性质	const_mul_l_0	0*A = O
	数右乘 0 性质	const_mul_r_0	A*0=O
matrix_mul	矩阵乘法左分配律	matrix_mul_distr_l	(A±B) ×C=A×C ±B×C
	矩阵乘法右分配律	matrix_mul_distr_r	C×(A±B) = C×A ±C×B
	矩阵左乘零矩阵性质	matrix_mul_zero_l	O×A = O
	矩阵右乘零矩阵性质	matrix_mul_zero_r	A×O = O
	矩阵左乘单位矩阵性质	matrix_mul_unit_l	I×A = A
	矩阵右乘单位矩阵性质	matrix_mul_unit_r	A×I = A
	矩阵乘法结合律	matrix_mul_assoc	A×B×C=A×(B×C)
trans	矩阵转置相等性质	meq_teq	A=B <-> $A^T = B^T$
	矩阵转置与加法性质	teq_add	$(A±B)^T = A^T±B^T$
	矩阵转置与数乘性质	cteq	$(k*A)^T = k*A^T$
	矩阵转置与乘法性质	teq_mul	$(A×B)^T= B^T×A^T$

对于一些相似的性质,我们无需一一证明,可以通过已有的矩阵函数性质推导得到,这样可以大幅度减少工作量,如单位矩阵右乘性质：

$$A \times I = A(\text{apply meq\_teq})$$
$$(A \times I)^T = A^T(\text{apply teq\_mul})$$
$$I^T \times A^T = A^T(\text{apply trans\_MI})$$
$$I \times A^T = A^T（\text{apply matrix\_mul\_unit\_l}）$$

本文定义的矩阵以及矩阵函数是多态类型,其能够使得矩阵向各个域扩展,因此在所有性质证明完成后使用 Module 对其进行封装,当输入实数模块参数时获得实数矩阵模块,其中包括所有的矩阵类型、函数定义以及相关性质.本次矩阵形式化工作中提出了整数矩阵、实数矩阵、复数矩阵以及函数矩阵的基础库.

坐标系转换矩阵构成了飞行控制系统的基本组成部分,其可靠性对飞行安全有重要影响.飞行控制系统设计中使用了五种坐标系,它们之间存在 20 种转换矩阵,其中许多转换矩阵相当复杂,利用人工计算难免会产生错误.在本次研究工作中,我们利用矩阵形式化基础库对飞行控制系统中坐标系之间的转换矩阵做了验证,下面我们利用实数矩阵库来验证其中一个简单的例子,即飞行控制系统中气流坐标系与稳定坐标系的旋转矩阵：

验证内容：

$$\begin{bmatrix} cos\alpha & 0 & -sin\alpha \\ 0 & 1 & 0 \\ sin\alpha & 0 & cos\alpha \end{bmatrix} \times \begin{bmatrix} cos\beta & -sin\beta & 0 \\ sin\beta & cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} cos\alpha * cos\beta & -cos\alpha * sin\beta & -sin\beta \\ sin\beta & cos\beta & 0 \\ sin\alpha * cos\beta & -sin\alpha * sin\beta & cos\alpha \end{bmatrix}$$

下图为对该转换矩阵验证的代码部分:

```
Require Import Reals.
Open Scope R_scope.
Require Export Matrix.Mat.RMatrix.
Require Export Matrix.Mat.RMtacs.


(** * Sa -> Sb *)

(* 迎角 α  (angle of attack) *)
Parameter alpha : R.

(* 侧滑角β (sidelip angle) *)
Parameter beta : R.

(* 由  气流坐标轴系(Sa)  转动侧滑角β到  稳定坐标轴系(Ss)*)
Definition coordinate_transform_SaSs:= mkMat_3_3
  (cos beta)     (- sin beta)    0
  (sin beta)     (cos beta)      0
     0            0              1.

(* 由 稳定坐标轴系 Ss 转动迎角 α 到 机体坐标轴系 Sb 的转换矩阵 *)
Definition coordinate_transform_SsSb := mkMat_3_3
  (cos alpha)    0   (- sin alpha)
      0          1         0
  (sin alpha)    0   (cos alpha).



Definition coordinate_transform_SaSb  := mkMat_3_3
  ((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).
(*Require Import Nsatz.*)
(* multiplication of SbSs and SsSa *)
Definition mul_SsSb_SsSa :=
  RMmul coordinate_transform_SsSb coordinate_transform_SaSs.
```

Fig.12　Verification of Rotation Matrix of Airflow Coordinate System and Stable Coordinate System

图 12　气流坐标系与稳定坐标系的旋转矩阵的验证

4　分块矩阵形式化

本节将介绍利用 3 中的矩阵形式化方法所构建的分块矩阵形式化方法,由于在 3 中的矩阵形式化方法中所定义的类型、函数以及证明的函数性质都是带多态参数的,因此我们可以利用该多态性构造同样具有多态性的分块矩阵类型、分块矩阵函数以及证明分块矩阵函数所满足的性质.但我们无法通过函子将矩阵作为元素类型输入到矩阵模块之中,原因在于:矩阵使用的等价判断函数 M_eq 不适用于分块矩阵;矩阵乘法不具有乘法交换律,其无法形成一个满足矩阵模块要求的输入元素模块.因此在本节中我们提出新的分块矩阵的等价性判别函数、分块矩阵运算函数,并且对分块矩阵函数的性质采用了一种新的方式进行证明.

4.1　分块矩阵类型定义

在分块矩阵类型定义中,我们使用矩阵类型定义的双层嵌套来表示分块矩阵类型定义.根据 3.1 中矩阵类型的定义,当矩阵元素类型为 A 且其高度和宽度分别为 m 和 n 时,其类型表示为 Mat A m n,以该矩阵的类型作为矩阵内部元素的类型即可构造出一个内部分块大小为 m×n 分块矩阵.下图为高度和宽度分别为 m2 和 n2、内部元素为 m×n 矩阵的分块矩阵的内部结构:

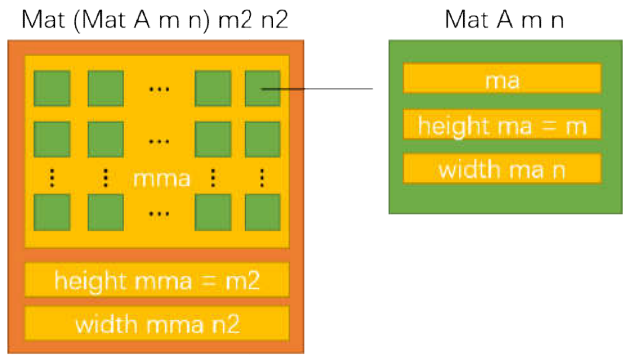

Fig.13 Type definition of block matrix

图 13 分块矩阵类型定义

4.2 分块矩阵类型等价性质

在 3.2 节中我们提出了一个新的函数 M_eq 来判别矩阵相等,但该函数不适用于分块矩阵,当使用 M_eq 来判别分块矩阵相等时会出现下图情况:

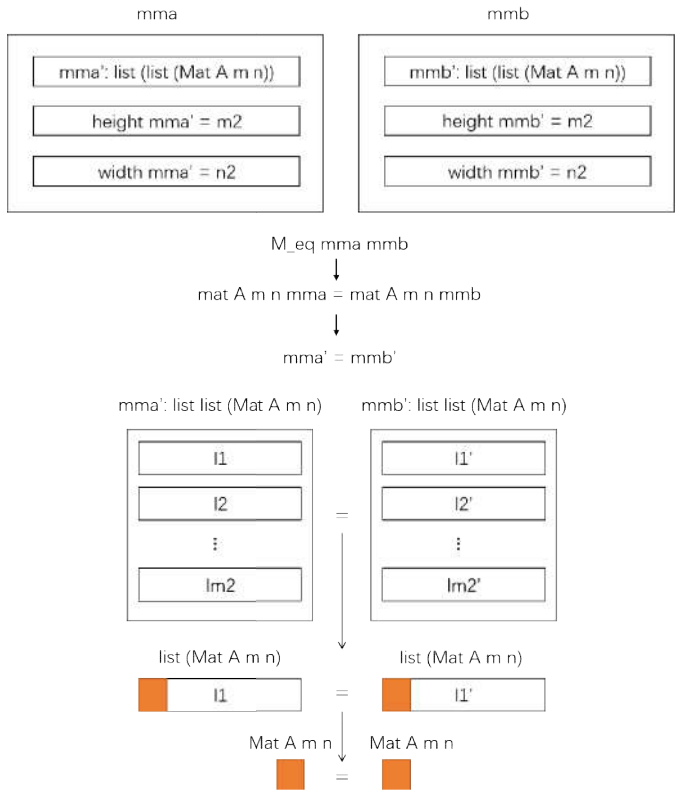

Fig.14 Problems caused by M_eq discriminating the equality of block matrices

图 14 由 M_eq 判别分块矩阵相等引起的问题

M_eq 函数的作用是提取出矩阵类型中的二维表后用 eq 函数判别是否相等,由图 14 可以看出使用 M_eq 判别分块矩阵时,需要用 eq 函数判别两个二维表相等,这要求二维表中所有元素相等(使用 eq 函数作为判别函

数）,但该分块矩阵中的二维表内元素为用 Record 表示的矩阵类型,在 3.2 节中我们说明了矩阵类型不能使用 eq 函数来判别矩阵类型相等的问题,因此我们提出了针对分块矩阵的等价判别函数 MM_eq,该函数是一个递归函数,其依赖于两个递归函数 list_Meq 和 dlist_Meq.为了代码的简洁性,在分块矩阵形式化的代码部分我们也采用了 Section 机制,在下文的代码中 A、m、n、m2 和 n2 都为局部变量其类型为(A:Set)(m n m2 n2:nat).

list_Meq 主要对由矩阵构成的表进行相等判断,该函数对表中的每个相应的矩阵利用 M_eq 函数进行相等判断,最后返回的是所有矩阵相等的与命题,其实质是利用 M_eq 来判别两个表内所有相应矩阵相等.下面是 list_Meq 函数的定义以及图示：

```
Fixpoint list_Meq(l1 l2:list(Mat A m n)):=
  match l1,l2 with
  | hd1::tl1,hd2::tl2 => hd1===hd2 /\list_Meq tl1 tl2
  | nil,nil => True
  | _,_ =>False
  end.
```

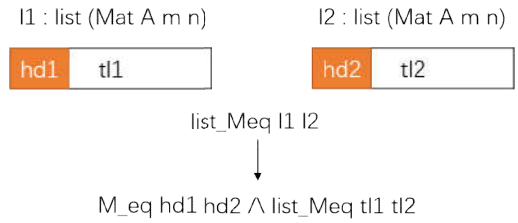

Fig.15 The implementation principle of the list_Meq function

图 15 list_Meq 函数的实现原理

dlist_Meq 函数主要用于对由矩阵构成的二维表进行相等判断,对二维表中每个矩阵的表利用 list_Meq 函数进行相等判断,最后返回的是所有由矩阵构成的表相等的与命题,其实质就是利用 M_eq 来判别两个二维表内所有相应矩阵相等.下面是 dlist_Meq 函数的定义以及图示：

```
Fixpoint dlist_Meq(dl1 dl2:list(list(Mat A m n))) :=
  match dl1,dl2 with
  |hd1::tl1,hd2::tl2 => list_Meq hd1 hd2 /\dlist_Meq tl1 tl2
  | nil,nil => True
  | _,_ =>False
  end.
```

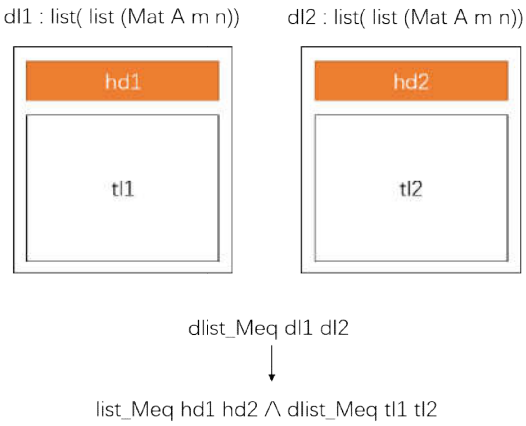

Fig.16 The implementation principle of the dlist_Meq function

图 16 dlist_Meq 函数的实现原理

在这上述两个函数的基础之上我们定义了分块矩阵（即矩阵的矩阵）相等判断函数 MM_eq,具体定义以及图示如下：

```
Definition MM_eq{A:Set} {m n m2 n2:nat} (ma mb:Mat (Mat A m n) m2 n2 ):=
  dlist_Meq (mat (Mat A m n) m2 n2 ma) (mat (Mat A m n) m2 n2 mb).
```

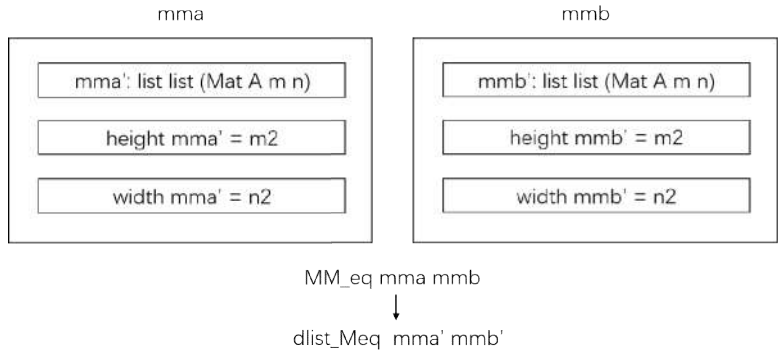

Fig.17 The implementation principle of the MM_eq function

图 17 MM_eq 函数的实现原理

该函数的定义解决了分块矩阵相等判别问题,后续我们对该函数的自反性、对称性以及传递性进行了证明,并利用 Coq 中的重写扩展机制实现了该函数的重写,在此基础上我们可以对分块矩阵函数性质进行验证.

4.3 分块矩阵函数定义

本节主要介绍了分块矩阵函数的定义, 由于 3. 3 节中定义的矩阵运算函数具有多态性, 分块矩阵的函数可以利用其多态性进行构造, 但并不是所有的分块矩阵运算函数都可以通过该方法定义, 例如分块矩阵的乘法函数以及转置函数, 在下面部分我们将讨论为什么无法通过上述方式构造, 以及我们的解决方法.

分块矩阵的加减法实现原理如下：

$$\begin{bmatrix} A_{11} & \cdots & A_{1n} \\ \vdots & \ddots & \vdots \\ A_{m1} & \cdots & A_{mn} \end{bmatrix} \pm \begin{bmatrix} B_{11} & \cdots & B_{1n} \\ \vdots & \ddots & \vdots \\ B_{m1} & \cdots & B_{mn} \end{bmatrix} = \begin{bmatrix} A_{11} \pm B_{11} & \cdots & A_{1n} \pm B_{1n} \\ \vdots & \ddots & \vdots \\ A_{m1} \pm B_{m1} & \cdots & A_{mn} \pm B_{mn} \end{bmatrix}$$

分块矩阵加减法的实现可以利用矩阵函数 matrix_each 的多态性直接实现,matrix_each 函数具有六个参数分别是元素类型、元素运算函数、矩阵高度、矩阵宽度以及两个相同大小的矩阵,该函数的主要功能是将两个矩阵中相应位置的元素一一代入提供的运算函数从而生成一个新的矩阵.当我们构造分块矩阵加法或减法时可以通过输入矩阵类型、矩阵加法即可生成一个分块矩阵加/减法函数.具体分块矩阵加法函数定义如下:

```
Definition MMadd:=@matrix_each (Mat A m n) (@matrix_each A add m n).
```

同理,分块矩阵取负函数可由矩阵函数 matrix_map 的多态性直接实现,具体定义如下:

```
Definition MMopp m n := @matrix_map (Mat A m n) (matrix_map A opp).
```

分块矩阵转置原理是将分块矩阵以矩阵为元素进行转置后再对内部元素一一转置,具体如下:

$$\begin{bmatrix} A_{11} & \cdots & A_{1n} \\ \vdots & \ddots & \vdots \\ A_{m1} & \cdots & A_{mn} \end{bmatrix}^T = \begin{bmatrix} {A_{11}}^T & \cdots & {A_{m1}}^T \\ \vdots & \ddots & \vdots \\ {A_{1n}}^T & \cdots & {A_{m\,n}}^T \end{bmatrix}$$

在矩阵形式化部分我们所定义的矩阵转置函数 trans 具有的五个参数分别是元素类型、缺省值、矩阵高度、矩阵宽度以及输入矩阵,该函数主要作用是将矩阵进行转置.但如果我们直接使用该函数作为分块矩阵的转置函数,会出现由于缺少内部矩阵转置操作而产生错误结果的现象,具体函数定义如下:

```
Definition MMtrans:=trans (Mat A m n) (MO A m n).
```

利用该函数对分块矩阵操作后产生结果如下:

$$\text{MMtrans} \begin{bmatrix} A_{11} & \cdots & A_{1n} \\ \vdots & \ddots & \vdots \\ A_{m1} & \cdots & A_{mn} \end{bmatrix} = \begin{bmatrix} A_{11} & \cdots & A_{1m} \\ \vdots & \ddots & \vdots \\ A_{1n} & \cdots & A_{mn} \end{bmatrix} \neq \begin{bmatrix} {A_{11}}^T & \cdots & {A_{m1}}^T \\ \vdots & \ddots & \vdots \\ {A_{1n}}^T & \cdots & {A_{m\,n}}^T \end{bmatrix}$$

因此该 MMtrans 所产生的转置结果不等于分块矩阵转置的结果,原因在于其内部分块没有转置,面对这个问题我们构造新的转置函数 mmtrans,在实现对分块矩阵整体转置的同时对其内部分块进行转置.我们提出了一个新的递归函数 mhead,该函数与 List 库中 head 函数相似,不同之处在于其对由矩阵类型构成的表取头元素时会同时对该元素进行转置操作,该函数定义如下:

```
Definition mhead (l:list(Mat A m n)):=
  match l with
  | [] => trans A Zero (MO A Zero m n)
  | x::_ =>trans A Zero x
  end.
```

然后我们基于 mhead 函数提出了一个提取由矩阵类型构成的二维表第一列的函数 mheadcolumn,该函数在取出第一列的同时会对该列内的每一个矩阵元素进行一个转置;接着我们提出了一个对由矩阵类型构成的二维表进行转置的函数 mgettrans;最后完成对上述函数输出的二维表的长宽证明并封装成一个分块矩阵转置函数 mmtrans,上述三个函数的定义同普通矩阵转置函数类似.

同样,矩阵形式化中提出的矩阵乘法函数也无法适用于分块矩阵乘法函数,原因在于矩阵乘法函数 matrix_mul 需要提供的参数中有一个为元素相乘函数,对于 A 类型元素构成的矩阵则需要提供一个 A->A->A 类型的乘法函数,然而两个矩阵能够相乘的条件在于前一个矩阵的列数与后一个矩阵的行数相等,这意味着矩阵乘法函数的类型可以为 Mat A m n -> Mat A n p -> Mat A m p,这不符合矩阵乘法函数输入参数的要求,因此我们针对该问题提出一个专门针对分块矩阵的乘法函数.分块矩阵乘法原理如下:

$$\begin{bmatrix} A_{11} & \cdots & A_{1n} \\ \vdots & \ddots & \vdots \\ A_{m1} & \cdots & A_{mn} \end{bmatrix} \times \begin{bmatrix} B_{11} & \cdots & B_{1p} \\ \vdots & \ddots & \vdots \\ B_{n1} & \cdots & B_{np} \end{bmatrix}$$

$$= \begin{bmatrix} A_{11} \times B_{11} + \cdots + A_{1n} \times B_{n1} & \cdots & A_{1n} \times B_{1p} + \cdots + A_{1n} \times B_{np} \\ \vdots & \ddots & \vdots \\ A_{m1} \times B_{11} + \cdots + A_{mn} \times B_{n1} & \cdots & A_{m1} \times B_{1p} + \cdots + A_{mn} \times B_{np} \end{bmatrix}$$

矩阵乘法中相乘的两个矩阵必须满足第一个矩阵的列数与第二个矩阵的行数相等（如 m×n 和 n×p），因此实现分块矩阵的乘法对分块矩阵内部分块大小有要求,若第一个分块矩阵内部分块为 m×n,则要求第二个分块矩阵内部分块需要为 n×p,并且要求两个分块矩阵大小为 m2×n2 和 n2×p2,其构造方法如下：

首先是两个元素为矩阵的表的“内积”函数 product,其实现原理同矩阵乘法函数定义中使用到的 product’函数类似,区别在于输入元素加法与乘法函数由普通加法与乘法变为矩阵的加法与乘法函数并且其缺省值类型为一个矩阵,其定义如下：

```
Fixpoint product(l1:list (Mat A m n))(l2:list (Mat A n p)) :=
  match l1 , l2 with
    | h1::t1,h2::t2 => Madd' (Mmul h1 h2) (product t1 t2)
    | _ ,_ => mZero
  end.
```

其中 Madd’表示为矩阵加法函数,Mmul 表示为矩阵乘法函数,mZero 为零矩阵,其实现原理如下图所示：

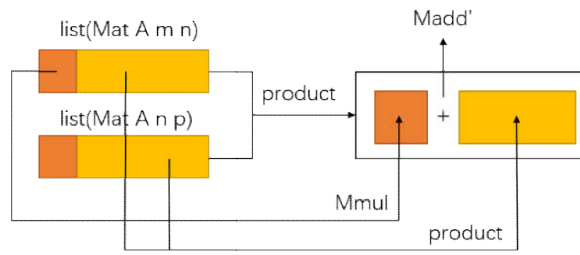

Fig.18　The implementation principle of the product function

图 18　product 函数的实现原理

在该函数的基础之上我们继续实现了元素为矩阵的表与二维表相乘函数 ml_mul_dl,元素为矩阵的二维表与二维表相乘函数 mdl_mul_dl,以及分块矩阵相乘函数（输出类型为元素为矩阵的二维表）mmat_mul_mat 和（输出类型为分块矩阵的）mmatrix_mul,上述函数的实现方法同 3.3 节中矩阵乘法函数的定义类似.但我们在分块矩阵的乘法函数（输出类型为元素为矩阵的二维表）mmat_mul_mat 之中使用的转置函数为普通矩阵转置函数,而非分块矩阵转置函数,具体如下：

```
Definition mmat_mul_mat {m2 n2 p2:nat} (left:Mat (Mat A m n) m2 n2 ) (right:Mat (Mat A n p) n2 p2):=
  let l1 := mat (Mat A m n) m2 n2 left in
  let right' := trans (Mat A n p) (MO A Zero n p) right in
  let l2 := mat (Mat A n p) p2 n2 right' in
    mdl_mul_dl l1 l2.
```

此处使用普通矩阵转置函数的原因在于我们在每个内部分块做矩阵相乘时会进行对第二个矩阵进行转置操作,而当我们使用分块矩阵的转置函数则会使得其内部分块也进行转置,这两个操作在对内部分块的转置操

作重复.对于一个分块矩阵内部分块大小为 m×n 则要求其相乘分块矩阵内部分块为 n×p,若使用分块矩阵转置函数则会使得分块矩阵内部分块大小分别为 m×n 和 p×n,这不符合矩阵元素乘法函数的参数要求.

下面是利用定义好的函数进行计算验证的例子.

验证$\begin{bmatrix}\begin{bmatrix}1&0\\0&1\end{bmatrix}\begin{bmatrix}0&2\\2&0\end{bmatrix}\end{bmatrix}\times\begin{bmatrix}\begin{bmatrix}3&1\\1&3\end{bmatrix}\\\begin{bmatrix}1&3\\3&1\end{bmatrix}\end{bmatrix}=\begin{bmatrix}9&3\\3&9\end{bmatrix}$,其证明过程如下,其中 Mmake_simpl 与 MMmult_simpl 为自定义的分别针对于矩阵构建以及矩阵乘法的化简策略,用于简化证明过程:

```
Definition tm1:= mkMat_2_2  1 0
                            0 1.
Definition tm2:= mkMat_2_2  0 2
                            2 0.
Definition tm3:= mkMat_2_2  3 1
                            1 3.
Definition tm4:= mkMat_2_2  1 3
                            3 1.
Definition tm5:= mkMat_1_2  tm1 tm2.
Definition tm6:= mkMat_2_1  tm3 tm4.
Definition tm7:= mkMat_1_1(mkMat_2_2  9 3
                                      3 9).
Lemma test1: MMmult tm5 tm6 == tm7.
Proof.
  unfold tm5,tm6,tm7. Mmake_simpl. MMmult_simpl.
  split. split. f_equal2. ring. f_equal. ring. f_equal. ring.
  f_equal. ring. auto. auto.
Qed.
```

Fig.19 Verification example of block matrix multiplication function

图 19 分块矩阵乘法函数验证例子

4.4 分块矩阵函数与普通矩阵函数等价性证明

分块矩阵函数与普通矩阵函数的等价性证明是分块矩阵形式化工作中的一个重要内容,其原因有两个:1)保证分块矩阵运算函数的正确性;2)将普通矩阵运算函数的性质利用该等价性移植于分块矩阵运算函数之上,能够大大减少证明过程.分块矩阵能够使得矩阵运算并行化,但分块矩阵的函数相比于普通矩阵函数更加复杂且容易出错,因此为了保证利用分块矩阵技术对矩阵进行运算后得到的结果仍然等于其原结果,我们需要对分块矩阵函数与普通矩阵函数进行等价性证明.

为了证明分块矩阵函数的等价性,我们构造了一个将分块矩阵转换为矩阵的函数 MM_to_M,该函数的作用是将一个分块矩阵类型转换成一个普通矩阵.该函数的定义思路如下:分块矩阵中不同分块的合并涉及到二维表的横向合并以及二维表的竖向合并,二维表的竖向合并可使用 app 函数实现,二维表的横向合并有多种设计方法,对于方法的选择以及原因将在 5.2 节中详细说明.

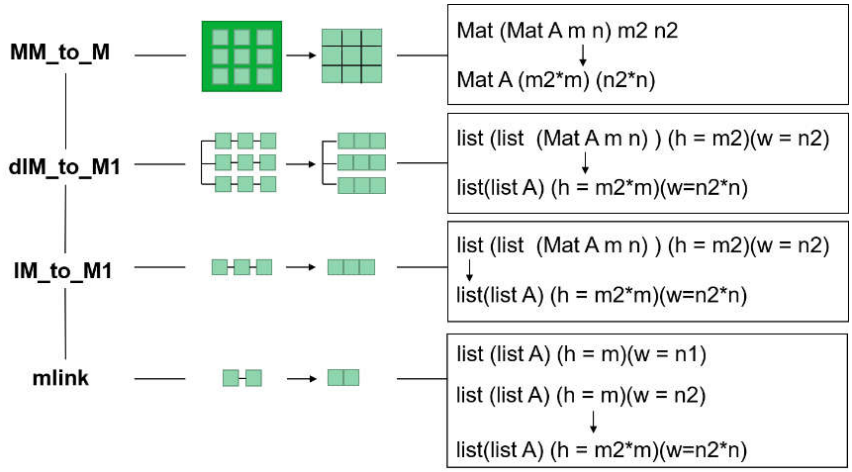

Fig.20 The implementation principle of the MM_to_M function

图 20 MM_to_M 函数的实现原理

以分块矩阵转置函数和普通矩阵转置函数为例, 首先利用定义好的分块矩阵转置函数对分块矩阵进行转置, 然后将其结果通过分块矩阵与矩阵的转换函数转换为矩阵, 证明其等价于先转换为矩阵再进行矩阵转置得到的结果, 其内容如下图所示：

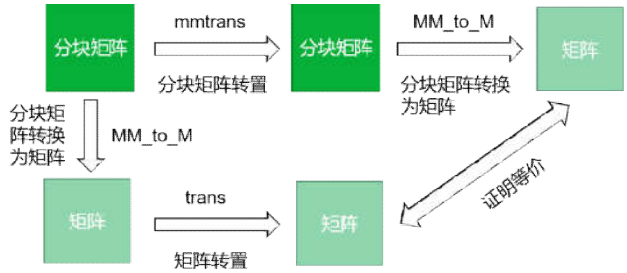

Fig.21 Proof of the equivalence of transposed function of block matrix

图 21 分块矩阵转置函数等价性证明思路

分块矩阵转置函数等价引理表示如下：

```
Lemma MMtrans_eq_Mtrans: forall (me:Mat (Mat A m n) m2 n2)
  MM_to_M(mmtrans A Zero m n me) === trans A Zero (MM_to_M me).
```

该引理的证明依赖于一个重要的基础引理, 即二维表的横向连接函数和二维表转置函数的关系, 若完成该引理的证明即可完成上述分块转置函数等价引理的证明, 其表示如下：

```
forall (A : Set) (Zero : A) (h1 h2 w : nat) (ma mb : list (list A)),
  height ma = h1 -> height mb = h2 -> width ma w -> width mb w ->
  gettrans A Zero (ma ++ mb) w = mlink A (gettrans A Zero ma w) (gettrans A Zero mb w).
```

其中 gettrans 是对二维表的转置, “++”代表 app 函数表示表的连接, mlink 表示二维表的左右连接. 该引理用数学的方法表示为$\begin{bmatrix}A\\B\end{bmatrix}^T = [A^T \quad B^T]$. 在该证明中所遇到的困难之处将在 5.2 节详细讨论.

在分块矩阵函数等价性证明部分, 我们成功证明了以下三个分块矩阵函数等价性引理, 具体如下：

1) 分块矩阵加减运算函数等价引理：

```
Lemma MMf_eq_Mf: forall (me mf:Mat (Mat A m n) m2 n2),
  MM_to_M (matrix_each (Mat A m n) (matrix_each A f) me mf)
  === matrix_each A f (MM_to_M me) (MM_to_M mf).
```

2) 分块矩阵取负运算函数等价引理：

```
Lemma MMm_eq_Mm: forall (me:Mat (Mat A m n) m2 n2),
  MM_to_M (matrix_map (Mat A m n) (matrix_map A f1) me) === matrix_map A f1 (MM_to_M me).
```

3) 分块矩阵转置函数等价引理：

```
Lemma MMtrans_eq_Mtrans: forall (me:Mat (Mat A m n) m2 n2)
  MM_to_M(mmtrans A Zero m n me) === trans A Zero (MM_to_M me).
```

4) 分块矩阵乘法函数等价引理：

```
Lemma MMmul_eq_Mmul: forall (me:Mat (Mat A m n) m2 n2) (mf:Mat (Mat A n p) n2 p2)
  MM_to_M (mmatrix_mul A Zero add mul m n p me mf)
  === matrix_mul A Zero add mul(MM_to_M me)(MM_to_M mf).
```

利用以上引理, 可直接将普通矩阵函数所具有的性质移植到分块矩阵的函数之上, 无需对分块矩阵函数所满足的性质一一证明, 加快了证明效率. 对于证明较为简单的分块矩阵加法交换律, 利用归纳证明其证明步骤如下：

```
Lemma mmatrix_each'_comm: forall (m2 n2 :nat)
  (left right:list (list (@Mat A m n))),
  (forall a b ,Madd a b === Madd b a)->
  height left = m2 -> height right = m2 ->
  width left n2 -> width right n2 ->
  dlist_Meq
  (mat_each' (@Mat A m n) Madd left right)
  (mat_each' (@Mat A m n) Madd right left).
Proof.
  induction m2.
  induction left.
  induction right.
  - simpl. auto.
  - simpl. auto.
  - intros. inversion H0.
  - induction left. induction right.
    + simpl. auto.
    + simpl. auto.
    + induction right.
      { simpl. auto. }
      { intros. simpl. split. apply mlist_each_comm with (n2:=n2) .
        apply H. inversion H2. auto. inversion H3. auto.
        apply IHm2 with(n2:=n2). auto. inversion H0. auto. inversion H1. auto.
        inversion H2. auto. inversion H3. auto. }
Qed.

(** ** mmatrix_comm *)
Lemma mmatrix_comm : forall {m2 n2:nat}(left right:@Mat (@Mat A m n) m2 n2) ,
  (forall a b , Madd a b === Madd b a) ->
  matrix_each (@Mat A m n) Madd left right
  == matrix_each (@Mat A m n) Madd right left.
Proof.
  intros m2 n2.
  induction left.
  - induction right.
    + intros. unfold matrix_each.
      unfold mat_each. simpl.
      unfold MM_eq. simpl.
      apply mmatrix_each'_comm with(m2:=m2) (n2:=n2).
      apply H. apply mat_height. apply mat_height0.
      apply mat_width. apply mat_width0.
Qed.
```

Fig.22 The ordinary inductive proof process of addition distributive law of block matrix

图 22 分块矩阵加法分配律的普通归纳证明过程

对于使用分块矩阵函数与普通矩阵函数的等价性的分块矩阵加法交换律证明如下：

```
Lemma mmatrix_each_comm: forall (ma mb: Mat(Mat A m n) m2 n2),
  (forall a : A, add a Zero = a) ->
  (forall a b : A, add a b = add b a) ->
  matrix_each (Mat A m n) (matrix_each A add) ma mb
  == matrix_each (Mat A m n) (matrix_each A add) mb ma.
Proof.
  intros. apply MM_to_M_M_eq.
  rewrite ?MMf_eq_Mf. apply matrix_comm.
  auto. auto. auto.
Qed.
```

Fig.23 The proof process of addition distributive law of block matrix using the above equivalence

图 23 分块矩阵加法分配律利用等价性的证明过程

从该例子的证明中我们可以明显发现证明的简化, 并且该简化对于难度更大的引理证明仍然适用. 因此我们在证明分块矩阵函数与矩阵函数等价性, 不仅能够保证分块矩阵运算函数的正确性, 还能够在很大程度上简化分块矩阵函数性质证明过程, 减少重复证明工作.

目前已成功证明的分块矩阵函数性质引理如下：

Table 4 Lemma of the properties of the block matrix function

表 4 分块矩阵函数性质引理

引理	引理名字
加法交换律	mmatrix_add_comm
加法结合律	mmatrix_add_assoc
左加零矩阵性质	mmatrix_add_zero_l
右加零矩阵性质	mmatrix_add_zero_r
减法交换律	mmatrix_sub_opp
减法结合律	mmatrix_sub_assoc
左减零矩阵性质	mmatrix_sub_zero_l
右减零矩阵性质	mmatrix_sub_zero_r
自身相减矩阵性质	mmatrix_sub_self
乘法左分配律	mmatrix_mul_distr_l
乘法右分配律	mmatrix_mul_distr_r
乘法结合律	mmatrix_mul_assoc
左乘零分块矩阵性质	mmatrix_mul_zero_l
右乘零分块矩阵性质	mmatrix_mul_zero_r
两次转置性质	mmtrans_same
转置加减性质	mmteq_f
转置乘性质	mmtrans_mul

5 矩阵与分块矩阵形式化证明总结与分析

本节主要对在本次研究工作中的证明部分的经验进行总结,并利用具体例子对几个难点进行了详细的分析.

5.1 证明总结

本次有关矩阵函数性质证明的研究工作是在文献[3]的基础上进行的,对其缺少的引理进行了补充证明,在

数量上增加了十余条矩阵函数性质,并且部分性质证明难度较高.函数性质证明难度的高低取决于其涉及到的函数的定义的复杂性,例如矩阵乘法结合律以及矩阵与单位矩阵相乘性质.由于引理证明难度的增加导致其需要的辅助引理的数量也大大增加,文献[3]中提供的代码量在 2000 行左右,本次矩阵与分块矩阵形式化研究工作在其基础上进行扩充,最后统计有关于矩阵与分块矩阵形式化的代码部分在 10000 行左右.

无论是在矩阵形式化还是在分块矩阵形式化的研究工作中都有大量有关函数性质的证明,我们在进行对函数性质的证明时主要采用归纳证明的方式进行证明.引理的证明与个人证明经验挂钩,在证明中有许多的策略,证明经验能够帮助我们针对不同情况选择不同的证明策略.在对函数性质的证明过程中我们可以发现一些函数的潜在问题,然后根据这些问题对函数进行一定调整与修改并重新对该函数性质进行证明,因此函数的构造与函数性质的证明并不是一蹴而就的,而是一个反复的过程.在函数性质证明中发现的函数定义存在问题则需要在函数中进行相应调整,在调整后需要对调整后的函数的性质进行重新证明.采用这种方法所产生的函数能够在很大程度上修正了函数定义中存在的问题.当函数定义存在一些难以察觉的问题时,即产生结果正确却在证明时与引理产生一定冲突和矛盾,此时我们需要重新修改函数的定义.对于部分引理利用当前已有函数和引理无法完成证明的情况,我们需要构造新的辅助函数并且提出新的辅助引理,辅助函数构造方式的选择也成为证明途中一个阻碍.在引理的归纳证明中,需要对引理中的变量进行归纳,如何选择归纳变量以及确定变量归纳的顺序对于引理的证明来说非常重要,正确的选择能够使得引理更快地得到证明.

下面是我们对本次研究工作中归纳证明时的经验的总结:

- 对必要的变量进行归纳证明,每条引理中存在多个变量,当引理较为复杂时,其中变量的数量也较多,若对每个变量都进行归纳会加大证明的困难程度,因此对于一些不必要变量可以不用归纳,归纳变量的选择由该引理中的函数定义结构决定.在对一个变量进行归纳证明时,重复同样证明操作即表示该变量有可能存在不必要的归纳.
- 归纳变量顺序可变动（根据经验以及具体例子而定）,归纳变量顺序不同所依赖的证明思路也有所不同,在本次工作中主要采用由高度-宽度-二维表的归纳顺序进行归纳证明.对于一些涉及特殊函数的引理需要观察其函数内部定义结构,从而选择有利于进一步化简的变量进行归纳.
- 根据归纳变量所产生的归纳条件进行化简拆分,得到归纳条件中的形式,再进行重写证明.遇到无法证明的引理存在两种情况：1）引理本身存在矛盾,这可能由引理本身描述错误或函数定义存在问题导致引理中产生冲突或无法证明的情况;2）当前函数以及引理无法满足其证明化简的要求.对于第二种情况,则需要考虑构造新的函数以及引出新引理进行化简证明.在下一小节中将详细分析这两种情况.
- 引理的等价变换和泛化在引理的证明中起到一个重要的作用,当我们证明一个引理时可能会产生多个子目标,若当前子目标则需要新的化简后才可证明则需要将该化简泛化为一个引理.同样我们对于一个引理的证明中也可以依据该引理涉及到的函数性质对函数做一些等价转换,以达到在一定程度上简化证明步骤的目的.

5.2 难点分析

对于上一小节中我们所提及的引理证明存在的两种情况将在下面以例子详细解释：

1）由函数定义引起引理无法证明的情况

引理本身存在矛盾这种情况可能由两种原因导致：一种是引理本身描述错误,这可以通过检查发现错误的原因并且对其进行修改;第二种是引理内部涉及的函数定义不合理而导致引理产生矛盾或者是无法证明的情况,这需要对函数定义进行修改,下面以单位矩阵左乘性质为例分析第二种原因：

文献[3]中采用的单位矩阵生成函数 MI 由以下几个函数构成：

list_i 函数的作用是产生一个长度为 n 的表,其中第 i 个元素为 1 其余元素为 0,如 list_i 4 3 = [0;0;1;0].其定义如下：

```
Fixpoint list_i n i :=
```

```
  match n,i with
  | O,_ =>List.nil
  | S n',O => List.cons Zero (list_i n' O)
  | S n',S O => List.cons One (list_i n' O)
  | S n', S i' => List.cons Zero (list_i n' i')
  end.
```

dlist_i 函数的作用是产生一个类型为二维表的单位矩阵,其中参数 n 表示产生单位矩阵的阶数,dlist_i 3 = [[1;0;0];[0;1;0];[0;0;1]].其定义如下：

```
Fixpoint dlist_i' n i :=
  match i with
  | O => List.nil
  | S i' => List.cons (list_i n i) (dlist_i' n i')
  end.
Definition dlist_i n := List.rev (dlist_i' n n).
```

单位矩阵生成函数 MI 定义如下：

```
Definition MI n := let m := dlist_i n in mkMat A n n m (dlist_i_length n) (all_len_n_dlist_i n).
```

该定义从结果上显示是正确的,即对于任意参数 n 都可产生 n 阶单位矩阵,但其定义中所使用的 rev 函数却在后期证明中成为证明的一大阻碍,rev 函数的作用是将一个表内元素进行逆向排放,下面以单位矩阵左乘性质引理为例具体说明.

对于需要证明的单位矩阵左乘性质引理如下：

```
Lemma matrix_mul_unit_l : forall  {m n:nat} (ma:Mat A m n),                    （1）
  matrix_mul (MI m) ma === ma.
```

其中 matrix_mul 为矩阵乘法函数,MI 为单位矩阵生成函数（MI m 代表 m 阶单位矩阵）,ma 代表 m×n 的矩阵,该引理所表示的性质为单位矩阵与矩阵相乘结果等于矩阵本身.

在证明 matrix_mul_unit_l 引理时首先对已封装的处理矩阵的函数(matrix_mul、MI)以及矩阵相等判断函数(M_eq)展开,并进行一些变换操作化简成有关于二维表类型的引理,引理整理后如下:

```
Lemma dlist_mul_unit_l': forall m n (ma:list(list A)), height ma = m -> width ma n ->
（2）
dl_mul_dl A Zero add mul (rev (dlist_i' A Zero One m m))  (gettrans A Zero ma n) = ma.
```

其中 gettrans 是对二维表的转置函数其带有四个参数分别为二维表内元素类型 A、A 类型中零的表示、二维表以及二维表的宽度.引理（2）等号左侧的结构看起来非常复杂,为了简化引理我们证明了 dl_mul_dl 函数与 rev 函数的关系,具体如下：

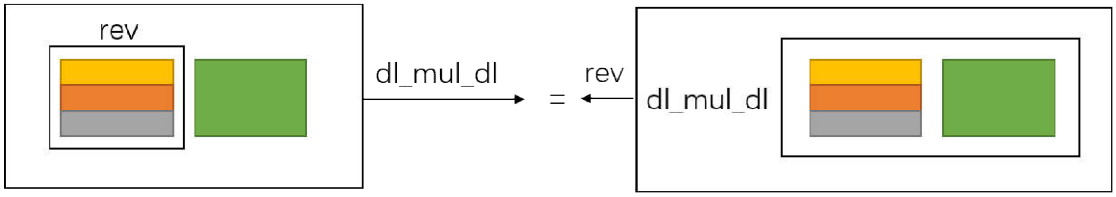

Fig.24 The relationship between dl_mul_dl function and rev function

图 24　dl_mul_dl 函数与 rev 函数的关系

接着使用该引理对（2）进行化简并为新的证明目标,如下：

```
Lemma dlist_mul_unit_l': forall m n (ma:list(list A)), height ma = m -> width ma n ->  （3）
rev(dl_mul_dl A Zero add mul (dlist_i' A Zero One m m) (gettrans A Zero ma n)) = ma.
```

由于 rev 函数对一个参数使用两次后的结果等于其本身,我们将等式两边都使用 rev 函数以消除左侧的 rev 函数,并对等式左侧 ma 的转置项(gettrans A Zero ma n)替换为普通二维表,该化简的目的是为了减少引理等号左侧的复杂性并将其转移至引理等号右侧,最终需要进行归纳证明的引理如下：

```
Lemma dlist_mul_unit_l': forall m n (ma:list(list A)), height ma = m -> width ma n ->  （4）
dl_mul_dl A Zero add mul (dlist_i' A Zero One n n) ma = rev(gettrans A Zero ma n).
```

在该引理的归纳证明中,我们需要对 n 和 ma 两个变量进行归纳,在 n=S n 和 ma=a::ma 时,化简后得到如下形式子目标：

```
l_mul_dl A Zero add mul (list_i A Zero One (S n)(S n)) (a::ma)  （5）
:: dl_mul_dl A Zero add mul (dlist_i' A Zero One (S n) n) (a :: ma) =
rev (gettrans A Zero (tl a :: tailcolumn A ma) n) ++ (hd Zero a :: headcolumn A Zero ma) :: nil
```

该子目标看起来较为复杂,但其整体形式为 a::A=B++(b::nil),如下：

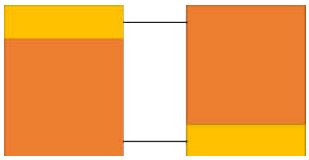

Fig.25　Proof problem caused by rev function

图 25　rev 函数引起的证明问题

该问题产生的原因在于 rev 函数,rev 函数将 a::ma 形式的参数转变为 ma++(a::nil)形式,其中++为 app 函数用于连接两个表,而我们通常对该类问题进行证明时,采用一定方法将子目标化简为 a::A=b::B 形式,在利用 f_equal 策略使该目标转化为两部分各自相等的子目标,若对其内部继续进行归纳化简,证明会变得更加复杂繁琐.因此面对该问题时,我们分析了问题出现的原因并且发现重新思考与 rev 函数相关的函数定义,在该例子中即为单位矩阵生成函数的定义.

因此我们不采用文献[3]中所定义的单位矩阵生成函数,虽然该函数产生结果满足我们的要求,但其内部构造不利于有关单位矩阵的性质证明,因此我们对其单位矩阵生成函数进行了修改.对于 list_i 函数我们不决定修改,原因在于该函数满足我们的要求,并且没有在证明过程中产生阻碍,我们对单位矩阵的二维表的生成函数做了以下修改：

```
Fixpoint dlist_i' m n i :=
  match m with
  | O => List.nil
  | S m' => List.cons (list_i n (S i)) (dlist_i' m' n (S i))
  end.
```

dlist_i'函数的作用是产生一个行数为 m,每行长度为 n 的二维表,dlist_i'n n 0 即为 n 阶单位矩阵（二维表类型）,在完成函数的构造后对该函数输出结果（二维表）的高度以及宽度进行等价性证明,下面是对该函数

进行的测试部分：

```
Definition tm1:= dlist_i' nat 0 1 3 3 0. （构建 3 阶单位矩阵）
Compute tm1.
```

```
= [[1; 0; 0]; [0; 1; 0]; [0; 0; 1]]
: list (list nat)
```

Fig.26 The test result of dlist_i' fucntion

图 26 dlist_i'函数测试结果

经过测试,我们可以看到该函数所产生的二维表满足单位矩阵的特性.利用上述函数以及性质将 dlist_i'封装成单位矩阵生成函数 MI,如下：

```
Definition MI n := let ma := dlist_i' n n 0 in mkMat A n n ma (height_dlist_i' n n 0) (width_dlist_i' n n 0).
```

在上述无 rev 函数定义的单位矩阵生成函数定义下,对引理（1）进行展开化简,得到如下引理：

```
Lemma dlist_mul_unit_l': forall m n (ma:list(list A)), height ma = m -> width ma n -> （6）
  dl_mul_dl A Zero add mul (dlist_i' A Zero One m m 0) gettrans A Zero ma n = ma.
```

为了使证明更加简单,我们利用转置函数的特性对该引理进行化简,如下：

```
Lemma dlist_mul_unit_l': forall m n (ma:list(list A)), height ma = m -> width ma n-> （7） dl_mul_dl A Zero add mul (dlist_i' A Zero One n n 0) ma = gettrans A Zero ma n.
```

下面对该引理进行归纳证明,对变量 n 进行归纳证明时,得到一个子目标如下：

```
dl_mul_dl A Zero add mul (dlist_i' A Zero One n (S n) 1) ma =                （8）
dl_mul_dl A Zero add mul (dlist_i' A Zero One n n 0)(tailcolumn A ma)
```

观察该子目标发现,需要将 dlist_i' A Zero One n (S n) 进行化简,但利用已有函数无法进行化简,在无法化简的情况下,因此需要构造新的函数 link1 来辅助该化简证明,该函数的作用是将一个表左接于一个二维表,其实现原理如下：

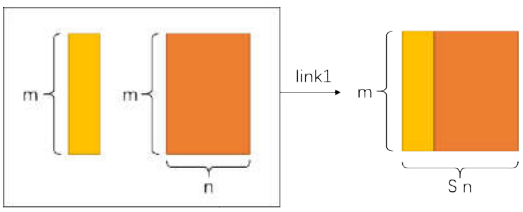

Fig.27 The implementation principle of link1 function

图 27 link1 函数实现原理

对于子目标（8）中的 dlist_i' A Zero One n (S n) 1 部分,我们需要利用 link1 函数进行化简,其化简引理如下：

```
Lemma dlist_i'_link1: forall m n i,                                          （9）
dlist_i' A Zero One m (S n) (S i) = link1 (list_o A Zero m) ( dlist_i' A Zero One m n i).
```

利用该性质将子目标（8）进行化简,得到：

```
dl_mul_dl A Zero add mul (link1 (list_o A Zero n) (dlist_i' A Zero One n n 0)) ma =
（10）
dl_mul_dl A Zero add mul (dlist_i' A Zero One n n 0) (tailcolumn A ma)
```

观察该子目标,需要证明 dl_mul_dl 函数与 link1(list_o A Zero n)函数之间的关系,其原理如下：

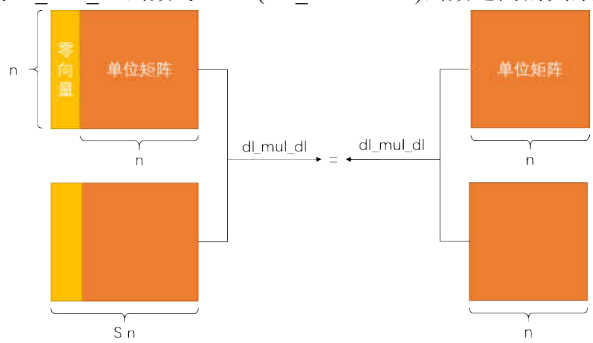

Fig.28 The relationship between dl_mul_dl function and link1 function

图 28 dl_mul_dl 函数与 link1 函数的关系

为了使该引理更易证明,将其泛化成以下形式：

```
Lemma dlist_mul_unit_l: forall m n i ma, width ma (S n) ->                （11）
  dl_mul_dl A Zero add mul (link1 (list_o A Zero m) (dlist_i' A Zero One m n i))ma
    = dl_mul_dl A Zero add mul (dlist_i' A Zero One m n i) (tailcolumn A ma).
```

完成该引理的证明后,加上一些辅助引理的证明即可完成单位矩阵左乘性质的证明,整体的证明框架如下：

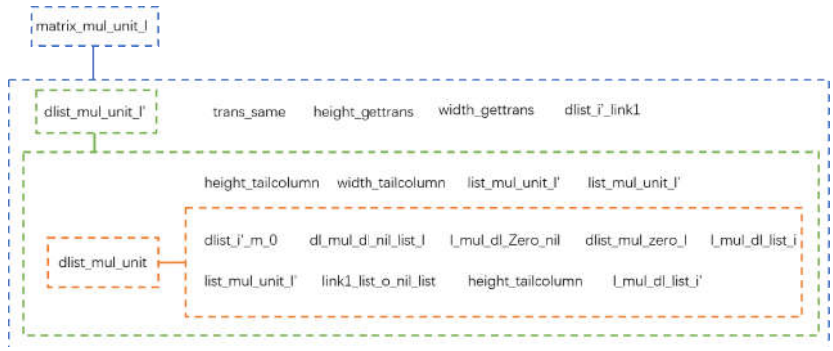

Fig.29 A proof framework for the property of left multiplication of identity matrix

图 29 单位矩阵左乘性质证明框架

由上图可知,该单位矩阵左乘性质的证明依赖于 17 个引理的证明,并且每个引理都需要有策略地对多个变量按一定顺序进行归纳证明,部分引理依赖于其他引理,因此一个矩阵函数性质的证明过程较为复杂以及繁琐,对如 3.4 节中分析的乘法右分配律性质,可以在证明过程中产生较为清晰的证明思路,但对于一些证明思路并不明显的性质如本节例子的单位矩阵左乘性质,证明过程中产生的一些引理主要是根据其归纳证明过程中子目标需要的化简步骤以及子目标的泛化而产生的,这些引理专门为这一个性质而服务,以支持该性质的证明.

2）辅助函数有多种构造方式的情况

在分块矩阵函数与矩阵函数等价性证明中,我们需要一个辅助函数——将分块矩阵转化为矩阵的函数

MM_to_M,然而该函数的定义依赖于一个二维表的横向合并函数 mlink,该函数可以由多种实现方法,构造方式的选择还需通过后续的证明决定,以下是两种不同的实现方法:

```
Fixpoint mlink ma mb n:=
  match n with
  | O => mb
  | S n' => link A (headcolumn A Zero ma)
                (mlink (tailcolumn A ma) mb n')
```

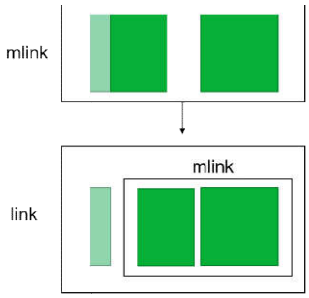

Fig.30 The implementation principle of the first mlink definition

图 30 第一种 mlink 定义实现原理

```
Fixpoint mlink(ma mb:list (list A)):=
  match ma with
  | nil => mb
  | x::x' => match mb with
                | nil => ma
                | y::y' => (x++y):: (mlink x' y')
                end
  end.
```

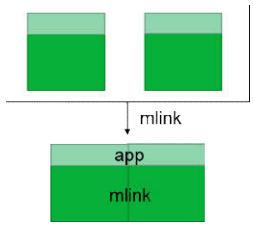

Fig.31 The implementation principle of the second mlink definition

图 31 第二种 mlink 定义实现原理

这两种定义所产生的操作结果经过测试都满足我们的要求,即横向合并两个二维表,根据这两种定义可以构造出两种分块矩阵到矩阵的转换函数,虽然两种定义的函数产生的结果相同,但由于定义的不同,在证明过程中前两种定义方法产生矛盾以及无法证明的问题.

第一种 mlink 的定义使用文献[3]中的 link 函数作为辅助函数,但在归纳变量化简后中会出现过于复杂而导致无法继续证明的情况,具体如下图所示:

```
1 subgoal
A : Set
Zero : A
h1, h2, w : nat
IHw : forall ma mb : list (list A),
      height ma = S h1 ->
      height mb = h2 ->
      width ma w ->
      width mb w ->
      gettrans A Zero (ma ++ mb) w = mlink  A Zero (gettrans A Zero ma w) (gettrans A
Zero mb w) (S h1)
a : list A
ma : list (list A)
IHma : forall mb : list (list A),
       height ma = S h1 ->
       height mb = h2 ->
       width ma (S w) ->
       width mb (S w) ->
       gettrans A Zero (ma ++ mb) (S w) =
       mlink  A Zero (gettrans A Zero ma (S w)) (gettrans A Zero mb (S w)) (S h1)
mb : list (list A)
H : height (a :: ma) = S h1
H0 : height mb = h2
H1 : width (a :: ma) (S w)
H2 : width mb (S w)
______________________________________(1/1)
(hd Zero a :: headcolumn A Zero (ma ++ mb)) :: gettrans A Zero (tl a :: tailcolumn A (ma +
+ mb)) w =
match
  mlink  A Zero (headcolumn A Zero ma :: tailcolumn A (gettrans A Zero (tl a ::
tailcolumn A ma) w))
    (headcolumn A Zero mb :: gettrans A Zero (tailcolumn A mb) w) h1
with
| nil => nil
| y :: y' =>
    (hd Zero a :: y) :: link A (headcolumn A Zero (gettrans A Zero (tl a :: tailcolumn A
ma) w)) y'
end
```

Fig.32 Problems caused by mlink constructed using link function

图 32 使用 link 函数构造的 mlink 所产生的问题

其原因在于,mlink 利用第一个二维表的宽度作为递归参数,利用 headcolumn 以及 tailcolumn 函数将二维表按列分割,跟二维表的归纳方式不同（按行分割）,因此不利于后续归纳证明.在该情况下,即使对所有涉及到的变量进行多次归纳仍难以化简,并且给证明制造了更多的困难,因此虽然该函数的定义所产生的结果满足我们的需求,但由于无法证明后续引理,我们选择了第二种定义,第二种定义主要使用辅助函数 app 来辅助 mlink 的构造,该定义较第一种定义更为简单,无需提供输入参数中第一个二维表的宽度,并且后续的一系列有关于该函数的引理都可以顺利证明.

在后续的证明中仍需要构造新的辅助函数以及大量辅助引理以支撑证明,在分块矩阵函数等价性证明中还需要构造两个辅助函数,分别是 headcolumn_n 以及 tailcolumn_n,这两个函数的作用分别是取二维表前 n 列、取二维表除前 n 列以外剩余的列数,这两个函数主要起过渡作用,用于建立部分函数之间的关系.

在矩阵和分块矩阵形式化研究工作中,通常函数定义无法一步到位,一个函数的实现有多种方式,一开始定义的函数看起来正确但是具有一些潜在问题难以发现,在后期性质证明时产生冲突或无法证明,则需要思考是否需要修改函数定义;性质证明无法一步到位,在证明一个引理时往往需要多个引理以支撑,这些性质可以分为辅助引理与关键引理,当目前无引理可帮助化简证明时,需要构造过渡函数以创建新的引理进行进一步化简证明,如 link1 函数、headcolumn_n 函数以及 tailcolumn_n 函数,此时辅助引理的构造方式也需要在后续证明中选择;定义与证明同时进行时,遇到证明失败时难以确定是函数定义问题还是证明策略问题,需要在不断尝试中进一步理解问题所在.

6 总结

利用 Coq 定理证明器中的 Record 类型所实现的矩阵形式化方法与其他矩阵实现方法相比,其实现更加简洁且更容易理解,使用起来也更加方便,并且可以使用代码抽取机制将经过验证的函数抽取为可执行的 OCaml 代码.我们在完善了矩阵形式化方法的基础上,为矩阵函数的引理进行了补充证明,并且建立了基于不同类型的矩阵基础库,具体包括实数矩阵库、整数矩阵库、复数矩阵库以及函数矩阵库.后续在矩阵形式化方法的基础上实现分块矩阵形式化方法,其能够实现将高阶矩阵运算转换为多个低阶矩阵运算,在后续有关于分块矩阵形

式化证明工作中,我们不仅证明了分块矩阵运算的正确性,并将矩阵形式化中已证明的函数移植于分块矩阵形式化库中,最后我们建立了基于不同类型的分块矩阵基础库.目前我们已经实现用函数式语言描述矩阵,在未来的工作中我们考虑定义一组重写优化规则并进行代码优化,将该矩阵运算以及分块矩阵运算函数转换为高效率并行 C 代码,并验证该重写优化的等价性,从而构建一个深度学习代码优化系统,并完成深度学习矩阵运算的形式化验证.本次研究工作的代码地址：https://gitee.com/yingyingma/Matrix.

致谢

感谢石正璞对本次工作提出的问题和建议以及在编译方面的支持工作,感谢沈楠、邓昊对本文存在的语病和问题的检查和纠正,感谢张正、崔敏以及讨论班的同学对本次研究工作的热心讨论和研究.

References:

[1] Almeida JB, Frade MJ, Pinto JS, Sousa SMD. An overview of formal methods tools and techniques[C]. in Rigorous Software Development, London: Springer, 2011: 15-44.

[2] Formalizing the safety of Java, the Java virtual machine, and Java card. ACM Computing Surveys (CSUR),2001,33(4).

[3] Ma ZW, Chen G. Matrix Formalization Based on Coq Record. Computer Science, 2019. DOI: 10.11896/j.issn.1002-137X.2019.07.022

[4] Bastian Hagedorn, Johannes Lenfers, Thomas Koehler, Qin XY, Sergei Gorlatch, and Michel Steuwer: Achieving High-Performance the Functional Way - A Functional Pearl on Expressing High-Performance Optimizations as Rewrite Strategies, ACM SIGPLAN International Conference on Functional Programming (ICFP 2020). DOI: 10.1145/3408974

[5] Bertot Y, Casteran P. Interactive Theorem Proving and Program Development[M]// Interactive theorem proving and program development. Coq'Art: The Calculus of inductive constructions. Springer, 2004.

[6] S Obua. Proving bounds for real linear programs in Isabelle /HOL[C]. Theorem Proving in Higher Order Logics, 2005:227-244

[7] IOANA PAŞCA. Formal proofs for theoretical properties of Newton's method. Mathematical Structures in Computer Science,2011,21(4). DOI: 10.1017/S0960129511000077

[8] Ioana Paşca. Formally verified conditions for regularity of interval matrices[P]. Intelligent computer mathematics,2010.

[9] Jesse Alama. The Rank-Nullity Theorem. Formalized Mathematics,2007,15(3).

[10] Karol P k. Eigenvalues of a Linear Transformation. Formalized Mathematics, 2008, 16(4):289-295 . DOI: 10.2478/v10037-008-0035-x

[11] Solovay RM, Arthan RD, Harrison J. Some new results on decidability for elementary algebra and geometry. Annals of Pure and Applied Logic,2012,163(12). DOI: 10.1016/j.apal.2012.04.003

[12] Shi ZP, Liu ZK, Guan Y, Ye SW, Zhang J, Wei HX, Luo GM. Formalization of Function Matrix Theory in HOL. Journal of Applied Mathematics,2014,2014. DOI: 10.1155/2014/201214

[13] Kang XN, Shi ZP, Ye SW, Guan Y. Formalization of Matrix Transformation Theory in HOL4. Computer Simulation, 2014. DOI: 10.1080/19443994.2014.944221

[14] Yang XM, Guan Y, Shi ZP, Wu AX, Zhang QY, Zhang J. Higher-order Logic Formalization of Function Matrix and its Calculus. Computer Science, 2016. DOI: 10.11896/j.issn.1002-137X.2016.11.005

[15] https://math-comp.github.io/mcb/

[16] Boldo S, Lelay C, Melquiond G. Coquelicot: A User-Friendly Library of Real Analysis for Coq. Mathematics in Computer ence, 2015, 9(1):41-62. DOI: 10.1007/s11786-014-0181-1

附中文参考文献:

[3] 马振威,陈钢.基于 Coq 记录的矩阵形式化方法.计算机科学,2019, 46(7):139-145.

[12]刘振科,施智平,关永,等.函数矩阵理论在 HOL4 中的形式化.小型微型计算机系统,2013, 34(3):654-658.

[13]康西楠,施智平,叶世伟,关永.矩阵变换理论在 HOL4 中的形式化.计算机仿真,2014,31(3):289-294.

[14] 杨秀梅,关永,施智平,等.函数矩阵及其微积分的高阶逻辑形式化.计算机科学,2016,43(11):24-29.