On the Simplification of Tensor Expressions

by

Nabil Obeid

Graduate Program in Computer Science

3

Submitted in partial fulfillment
of the requirements for the degree of
Master of Science

Faculty of Graduate Studies
The University of Western Ontario
London, Ontario
August, 2001

Abstract

Tensors are mathematical objects that generalize vectors and matrices. They describe geometrical quantities and they are used in various applied settings including mathematical physics. The indicial notation of tensors permits us to write an expression in a compact manner and to use simplifying mathematical operations.

In a large number of problems in differential geometry and general relativity, the time consuming and straightforward algebraic manipulation is obviously very important. Thus, tensor computation came into existence and became necessary and desirable at the same time.

Over the past 25 years, few algorithms have appeared for simplifying tensor expressions. Among the most important tensor computation systems, we can mention SHEEP, Macsyma ITensor Package, MathTensor and GRTensorII.

Meanwhile, graph theory, which had been lying almost dormant for hundreds of years since the time of Euler, started to explode by the turn of the 20^{th} century. It has now grown into a major discipline in mathematics, which has branched off today in various directions such as coloring problems, Ramsey theory, factorization theory and optimization, with problems permeating into many scientific areas such as physics, chemistry, engineering, psychology, and of course computer science.

Investigating some of the tensor computation packages will show that they have some deficiencies. Thus, rather than building a new system and adding more features to it, it was an objective in this thesis to express an efficient algorithms by removing most, if not all, restrictions compared to other packages, using graph theory. A summary of the implementation and the advantages of this system is also included.

Acknowledgments

I am thankful to my supervisor, Professor Stephen Watt, who taught me everything I needed to know about tensor expressions, for accepting to supervise me, for his kindness and generous contributions of time and for his careful commentary of my thesis. Without him, this work would never been completed.

Also, I would like to thank every person in the SCL lab for their helpful advice and useful comments on this thesis. Furthermore, I am sincerely grateful to my parents who kept supporting me regardless of the consequences and to my family, especially my wife, for their endless support and love. I shall never forget that.

Finally, I would like to acknowledge the School of Graduate Studies, the Department of Computer Science and Professor Watt for their financial support.

Contents

List of Tables

List of Figures

Abbreviations

NM(T)	name of a tenosr **T**
i	index i
SI(i)	symbol of index i
_i?	verify the type of i
IL(T)	list of indices of **T**
FL(T)	list of symbols of free indices of **T**
DL(T)	list of symbols of dummy indices of **T**
Pos(i, **IL(T)**)	position of i in **IL(T)**
TM	tensor monomial
TE	tensor expression
G(T)	graph of a tensor monomial **T**
V(G(T))	set of vertices of **G(T)**
EF(i)	edge represents a free index i
ED(i)	edge represents a dummy index i
E(G(T))	set of edges of **G(T)**

Chapter 1

Introduction

1.1 Overview

During the second half of the 19^{th} century and the first two decades of the 20^{th} century, tensors were introduced, were systematized and were brought to definitive form. This introduction made tensors the ideal tool in several areas such as mathematics, physics and mechanics.

Tensors are mathematical objects that generalize vectors and matrices, describe geometrical quantities and can be used in various applied settings, including mathematical physics. The indicial notation of tensors permits us not only to write an equation in a compact manner, but also to use simplifying mathematical operations.

Quantities such as the mass of a satellite or the temperature at certain points in a body have a definite magnitude. They can be represented adequately by single numbers or scalars. These are tensors of order or rank zero. Throughout this thesis, we will use the term rank in preference to the term order.

Properties such as the position or velocity of a satellite or the flow of heat in a body have both magnitude and direction. They can be represented by directed line segments

or by vectors. These are objects with one index or tensors of rank one. For example, V_b or U^a are vectors or tensors of rank one [Dan97].

Other quantities such as the stress inside a fluid may be characterized by matrices. These are objects with two indices or tensors of rank two. For example, $\mathbf{V}_{ij}, \mathbf{V}_i{}^j, \mathbf{V}^i{}_j$ and $\mathbf{V}^{ij}$ are tensors of rank two.

Tensors of higher rank must have a corresponding number of indices. For example, $\mathbf{R}^a{}_{bij}$ is a Riemann tensor of rank 4 since it has 4 non-repeated indices. In mathematics, tensorial objects must satisfy many special properties which we shall not detail here. We shall describe only the formal properties which must be respected by simplification rather than transformation.

Thus, tensors are defined as quantities which are represented as letters with uppercase and/or lowercase indices attached to the letters. In general, the number of uppercase and/or lowercase indices determine the rank of the tensor. For example, a tensor with n indices is an n rank tensor.

Tensors may be used to express the relation between physical quantities and they are ideal to formulate physical laws such as partial differential equations. These expressions are commonly known as tensor expressions. For example, a tensor expression such as $\mathbf{R}^a{}_{bij} + \mathbf{V}_{ij}U^aV_b$ can be created by multiplying $\mathbf{V}_{ij}$ by U^a and then by V_b and finally adding the result to $\mathbf{R}^a{}_{bij}$.

Tensor expressions can have intricate properties. For example, the *Riemann tensor* has special symmetry properties with respect to the permutation of its indices. Thus, the problem of simplification of tensor expressions naturally appears in this context. The objective of this thesis is to express effective algorithms to simplify tensor expressions using graph theory.

1.2 Basic Tensor Arithmetic

While tensors are used to great effectiveness in various areas of pure and applied mathematics as multi-linear objects and differential geometry and for the purpose of this thesis, we shall treat them as purely formal objects obeying arithmetic rules.

The purpose of this section is to establish several basic rules of operations with tensors. These operations are algebraic in character. We present these rules here. The development of this section is summarized from [Sok64].

Definition 1.1 *The **name** of a tensor is a letter from the alphabet used to express the tensor. For example, the name of the Riemann tensor* $\boldsymbol{R}_{abc}{}^{d}$ *is* $\boldsymbol{R}$.

Definition 1.2 *A **non-repeated** index is an index shown only once in a subscript or in a superscript position in a tensor. Meanwhile, a **repeated** index is an index shown twice in a tensor in a subscript and in a superscript position.*

Property 1.1 *The **sum** or **difference** of two tensors exists if and only if each tensor has the same non-repeated subscript indices and the same non-repeated superscript indices. Two tensors with this property are said to be of the same type.*

Property 1.2 *The **multiplication** of two tensors is possible if and only if the intersection of the subscript indices of both tensors is empty and the intersection of the superscript indices of both tensors is empty too. In this thesis, the multiplication of two tensors is commutative; i.e.,* $V^{c}\boldsymbol{U}_{ab} = \boldsymbol{U}_{ab}V^{c}$.

Remark 1.1 *Tensors do not obey all rules of ordinary arithmetic. For example, we can only add or multiply two tensors if the conditions of summation, as in Property 1.1, and multiplication, also as in Property 1.1, exist.*

Property 1.3 *The **intersection** of the non-repeated indices and the repeated indices should always be empty.*

Property 1.4 *A set of indexed quantities is said to be* ***symmetric*** *in any number of its indices, subscript or superscript, if their values remain unchanged by any permutation of that group of indices. For example, let* $\boldsymbol{A}_{kij}$ *be a symmetric tensor. Then,* $\boldsymbol{A}_{kij} = \boldsymbol{A}_{ijk} = \boldsymbol{A}_{kji}$.

Property 1.5 *A tensor is said to be* ***antisymmetric*** *in a specific group of their indices, subscript or superscript, if they remain unchanged by a finite even number of permutation of these indices and if they simply change their sign by a finite odd number of permutation of the index group. In general, the permutation of indices in a tensor is not always commutative; i.e.* $\boldsymbol{U}_{ab} \neq \boldsymbol{U}_{ba}$. *Note that, a tensor can combine this property and Property 1.4.*

Remark 1.2 *A linear combination of tensors exists if and only if Property 1.1, Property 1.2 and Property 1.3 are satisfied.*

Example 1.1 *Let* $\boldsymbol{U}_{abc}\,\boldsymbol{V}^{ec}\,\boldsymbol{W}^{b}$, $\boldsymbol{U}_{acb}\,\boldsymbol{V}^{eb}\,\boldsymbol{W}^{c}$ *and* $\boldsymbol{A}^{ad}$ *be three different tensor monomials. Thus,*

$$(\boldsymbol{U}_{abc}\,\boldsymbol{V}^{ec}\,\boldsymbol{W}^{b} + \boldsymbol{U}_{acb}\,\boldsymbol{V}^{eb}\,\boldsymbol{W}^{c}) \cdot \boldsymbol{A}^{ad} \tag{1.1}$$

is a tensor expression since:

1. *each expression in the sum in (1.1) has the same non-repeated indices,* $\{a, e\}$, *which satisfies Property 1.1,*
2. *the operands in the product have disjoint non-repeated index set and so satisfy Property 1.2, and*
3. *the non-repeated index set* $\cap$ *repeated index set* $= \emptyset$, *as in Property 1.3.*

1.3 Graph Theoretic Approach

Certain problems in physics can be formulated as problems in graph theory. Tensors are the perfect example to show that. For example, let $\mathbf{U}_i{}^j{}_k$ be a tensor and let G be an empty graph. In each graph, $\mathbf{V}_0()$ will be a reserved vertex to represent the set of non-repeated indices. Thus, $\mathbf{U}_i{}^j{}_k$ can be formulated as the following:

1. the name of the tensor, **U**, will be represented as a vertex, say $\mathbf{V}_1()$, in G,
2. the indices, {i, j, k}, will be represented as the edges of G, such that
   (a) the type of each index, subscript or superscript, will be represented as the direction of the edge, and
   (b) the position of the index in the tensor will be shown on the edge.

The method of labeling the vertices and the edges is fully described in Chapter 5.

The main computational problem is to devise an algorithm to reorganize each tensor expression into a simplified form. This simplification may involve the combination of a very large number of similar terms in a sum or factors in a product. Each monomial in every expression need to be represented as a separate graph. Adding or comparing these graphs allows us to add or to multiply tensor monomials and thus simplify the tensor expression.

For example, let $Expr = \mathbf{R}^{ab}{}_{cd}\mathbf{U}_{ai}\mathbf{V}^{cjk}{}_k$ be a tensor. The free indices of $Expr$ are b, d, i, j which are added to $\mathbf{V}_0()$. Meanwhile, $\mathbf{V}_1()$, $\mathbf{V}_2()$ and $\mathbf{V}_3()$ represent the name of $\mathbf{R}^{ab}{}_{cd}$, $\mathbf{U}_{ai}$ and $\mathbf{V}^{cjk}{}_k$ respectively. Thus, the graph will contain 4 vertices: $\mathbf{V}_0(b, d, i, j)$, $\mathbf{V}_1(R)$, $\mathbf{V}_2(U)$ and $\mathbf{V}_3(V)$.

In $Expr$, there exist 4 non-repeated and 3 repeated indices. The non-repeated ones are represented as edges related to $\mathbf{V}_0(b, d, i, j)$. Since each repeated index can be found

in one or 2 tensors, it will be represented as an edge going from one vertex to another but not $\mathbf{V}_0(b, d, i, j)$. The type of the indices will specify the direction of the edges. The full representation of *Expr* can be found in Figure 1.1.

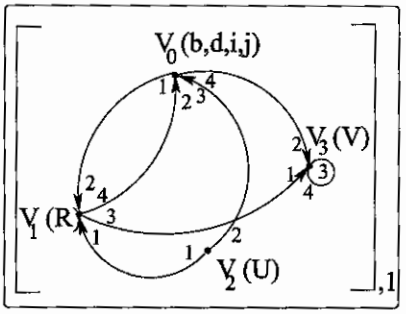

Figure 1.1: Representation of $\mathbf{R}^{ab}{}_{cd}\mathbf{U}_{ai}\mathbf{V}^{cjk}{}_{k}$

Note that, the representation of tensors in this package uses a rank-independent notation. It is a useful notation since:

1. this representation provides a natural and straightforward conceptual interface,
2. the treatment of the problem of dummy indices is natural, and
3. it is easy to add and to compare graphs.

1.4 Tensor Packages and Related Work

The use of computers to manipulate mathematical equations and expressions in symbolic forms is becoming more acceptable than manipulating the numerical quantities represented by those symbols. The advantages of using such a symbolic system can be resumed as the following: symbolic integration or differentiation, substitution of one expression into another, simplification of an expression, change of subject, etc.

Some of the best known symbolic mathematical software packages are **AXIOM**, **MACSYMA**, **Maple**, **Mathematica** and **REDUCE**. Others of historical interest include **MATHLAB** and **SHEEP** [Int95].

Over the past 25 years, a few different algorithms and packages have appeared to simplify tensor expressions. Tensor analysis requires both *Indicial* and *Component* Tensor Manipulations.

1. *Component tensor manipulation* means that geometrical tensor objects are usually represented as arrays or matrices. Tensor operations such as contraction or covariant differentiation are carried out by actually summing over the repeated indices. That is, one explicitly performs operations on the appropriate tensor components stored in an array or matrix.

2. *Indicial tensor manipulation* is implemented by representing tensors as functions of their covariant, contravariant and derivative indices. Tensor operations such as contraction or covariant differentiation are performed by manipulating the indices themselves rather than the components to which they correspond.

Among the early systems for relativistic applications, we can mention **LAM**, **ALAM** and **CLAM**. **ALAM** and **CLAM** are systems for symbolic mathematics especially for General Relativity. They were first implemented in ATLAS assembly language and later Lisp. These systems are only capable of component tensor calculations and they have been used for various applications [Ray70] [RC71].

In late 1970 and up to early 1980, **SHEEP**, which is an algebraic package for symbolic mathematics especially tensor analysis and general relativity developed by Inge Frick, surpassed the previous systems. It was implemented in assembly language then in LIPS and it is specialized in manipulating components of tensors [ea93]. It was written to treat a data type *indicial formula* representing formulas containing tensors or spinors with symbolic or letter indices [Hör79].

The **MACSYMA** Itensorial package implements symbolic tensor manipulation of two distinct types:

1. Explicit tensor manipulation, **ETENSOR**, and
2. Indicial tensor manipulation, **ITENSOR**.

The **MACSYMA ITENSOR** package was the first computer system capable of performing indicial tensor calculations while **ETENSOR** is specialized in manipulating components of tensors. Macsyma, as with many other software systems, has been used to investigate the validity of gravity theory [The83].

STENSOR of L. Hörnfeldt [Hör85] can also, as with all other systems, cope with symbolic indices, covariant differentiation and complicated symmetries. **STENSOR** has been used for applications in classical relativity, quantum gravity and super gravity. **STENSOR** is known as component calculations and indicial tensor manipulations.

MathTensor, which was implemented in Mathematica, developed by L. Paker and S. Christensen [CP90], is one of the more recent tensor computation package which has both indicial manipulation and component calculations. Among the new features, **MathTensor** provides the definition of rules and their applications, the differential forms and the pattern matching techniques used in the simplification of algebraic expressions.

In 1992, S.A. Fulling et al. [FKWC92] described an algorithm to enumerate the independent monomials built from the Riemann tensor and its covariant. They presented explicit tables for monomials of rank up to 12 in derivatives.

In the same year and using the same algorithm, Meller and Wybourne [WM92] enumerated up to the rank 14.

GRTensor II, developed by K. Lake and P. Musgrave [grt94], is a computer algebra package for performing calculations in the general area of differential geometry. The

purpose of this package is the calculation of tensor components in curved space times specified in terms of a metric or set of basis vectors. This package contains a library of standard definitions of a large number of commonly used curvature tensors, as well as the Newman-Penrose formalism. **GRTensor II** is only dedicated to component tensor calculations.

In 1994, M. Kavian [kMG97] developed **TCP**, Tensor Computation Package, and implemented in Maple. **TCP** was devised to have some new features compared to other packages. Among these features, a database of rules and components containing definition, identities and precomputed values of various invariants associated with different metric tensors.

In 1996, V.A. Ilyin and A.P. Kryukov [IK96] presented a package written using REDUCE to simplify tensor expressions called **ATENSOR**. The proposed algorithm is based on the consideration of tensor expressions as vectors in some linear space which is formed by all the elements of the group algebra of the corresponding tensor expression.

After two years, R. Portugal [Por98] presented an algorithm based on Gröbner bases. This algorithm simplifies tensor expressions by representing them in a canonical form taking into account the symmetries with respect to index permutations and the renaming of dummy indices.

In the following year 1999, Portugal [Por99] presented another algorithm for simplifying tensor expressions. First, he defines the canonical form of a single tensor and shows that the problem of finding the canonical form of a generic tensor expression reduces to finding the canonical form of a single tensor using the automorphism group of the symmetries. Then, the algorithm for simplifying the cyclic symmetry of the Riemann tensor is presented. By using this algorithm, Portugal shows how to simplify Riemann tensor polynomials.

1.5 Representation of Tensors in Computer Algebra Systems

In addition to the *Indicial* and the *Component* tensor manipulations, it is important in tensor analysis to check certain conditions, such as the symmetric properties of a tensor under interchange of the indices and also some other properties of certain tensors.

One useful approach for a tensor analysis package is the ability to simplify tensor expressions with a pattern matching technique giving some flexibility to define some rules then using them whenever appropriate.

Coordinate transformations have also become one of the new features of some tensor packages such as *MathTensor*. Coordinate transformation transforms the components of a tensor from one coordinate system to another [PC94].

Solving the Indicial tensor equations by converting those equations to components and then solving for the associated Finite Difference Equations is another strategy which is used by the *Macsyma ITensor Package* [The83].

In any tensor package, we can always define a tensor by calling a function and specifying its attributes. In *MathTensor*, this can be done with the command *DefineTensor* as follows:

- `DefineTensor[cyc, "c", {{3,1,2}, 1}]`

In *Macsyma ITensor Package*, the symmetric properties of tensors are defined by a function called **decsym**. For example,

- `decsym(B, 5, 3, [SYM(1,2), ANTI(3,4)], [CYC(ALL)])`

declares B to be symmetric in its first and second indices and antisymmetric in its third

and fourth covariant indices and cyclic in all of its contravariant indices.

One of the major differences between *MathTensor* and *Macsyma ITensor Package* is the representation of the indices. In *MathTensor*, the covariant and contravariant indices are represented in only one bracket such that $la, lb, ..., lo$ represent the covariant free indices and $ua, ub, ..., uo$ represent the contravariant free indices [The83].

For example, **MAXWELL[la, lb]** represents the Maxwell field tensor $\mathbf{F}_{ab}$ with covariant indices a and b while **MAXWELL[la, ub]** represents the Maxwell tensor $\mathbf{F}_{a}^{\ b}$ with contravariant index a and covariant index b.

1.6 Comparison of different packages

The advantage of using the *symbolic coordinate indices* can be summarized by representing any tensor as a n dimensional vector where n is the total rank of the tensor. For example, all the functions to manipulate vectors in *Mathematica* can be used in order to simplify expressions including tensorial ones.

Meanwhile, one of the disadvantages for using such a notation to define a tensor in a system as *Macsyma ITensor Package* is the representation of the lower and the upper indices. These indices should be separated in two brackets. Thus, this definition will not allow tensors to have mixed lower and upper indices.

For instance, The *Riemann tensor* $\mathbf{R}^{a}_{\ b}{}^{c}_{\ d}$ can not be represented in *Macsyma ETensor Package.* Therefore, this package is unable to perform index manipulations which is the key element to the algebraic simplification of expressions containing tensorial objects [Kav94].

In contrast, we are able to represent such a tensor with a mixture of lower and upper indices in *MathTensor* but we are not allowed to use some letters to represent the dummy

indices. For instance, the letters $lp, ..., lz$ and $up, ..., uz$ are only reserved to be used to represent the dummy indices.

GRTensor is dedicated only to Component tensor calculations and not for algebraic operations on tensorial expressions.

In *TCP*(*Tensor Computation Package*), the process of renaming the dummy indices is not well controlled all the time. Representing on the situation, *TCP* changes the name of the free and dummy indices to be different ones. One example of such performance would be the **Binachi** identity.

For example, let $TM1 = \mathbf{R}_{efpq}\mathbf{F}^{pq}$ be a tensor monomial. It is defined as the following
TM1 := Tensor(Riem(abcd)*Maxwell(AB));

In this tensor, there are only two free indices c and d, which are replaced by **e** and **f**. Meanwhile, the dummy indices a and b are replaced by p and q. The problem occurs when simplifying such a tensor containing all the alphabets as indices.

Chapter 2

Tensor Algebra

2.1 Tensors Defined

Tensors are a beautiful and simple language useful to describe natural phenomena. They are defined as quantities having physical significance and satisfy certain transformation law. Tensor fields are the abstract symbols of this language. Each tensor field represents a single physical quantity that is associated with certain points in three-dimensional space and instants of time.

Tensors, which can be defined using mathematical objects as mentioned in Section 1.1, are classified as:

1. **Zero-rank Tensors** or **Scalars**. Tensors of rank zero are **scalars**. They are quantities that are independent of the orientation of axes. A tensor of rank zero or a scalar determines the scalar field. For example, the work, the pressure, and the density. Note particularly that we can always specify each of these quantities merely by giving a single number denoting its magnitude or its value.

2. **First-rank Tensors** or **Vectors**. Tensors of rank one are the ordinary **vectors** defined as quantities having magnitude and direction. A tensor of rank one deter-

mines a vector field. For example, the position, the velocity, the gravity assist, and the mechanics of a particle.

3. **Higher-rank Tensors**. The quantity which we shall call a **tensor** is, in reality, a tensor of rank two or higher. Note that scalars and vectors are in the same family as tensors. For example, the stress inside a solid or fluid, the trajectories of point masses in a gravitational field, the motion of finite rigid bodies, the transfer of heat by conduction, and the deformation of solids.

A tensor is more than an array of numbers since the entries transform in related ways under coordinate transformations. Tensors are in particular useful to be used for equations which keep the same form under coordinate trajectories. In fact, a special form of differentiation, called the *covariant derivative*, can be used to write the partial differential equation in a tensorial form. Normal partial derivatives are not tensorial quantities.

In applications, several different specific tensors appear. For example, the Riemann tensor, the Ricci tensor, and metric tensors. In the differential geometry, field theory, the general theory of relativity and fluid mechanics, the laws of motion and field equations are expressed in a tensorial form to reflect the coordinate invariance of the equations. An additional advantage of tensor formalism is the compactness of tensorial expressions.

2.2 Definition of Spinors

Consider two vectors $\mathbf{X}_1 = (x_1, y_1, z_1)$ and $\mathbf{X}_2 = (x_2, y_2, z_2)$ of the Euclidean space E_3 with the same origin which are orthogonal and have equal norms. These vectors define a plane, and if we consider these vectors as being ordered, this order defines a direction of rotation.

In order to introduce algebraically a representation of the order in which the vectors $\mathbf{X}_1$ and $\mathbf{X}_2$ have been considered, it is convenient to multiply the components of the second vector by the imaginary number i. We can thus form the three complex numbers:

$$\begin{cases} x = x_1 + ix_2, \\ y = y_1 + iy_2, \\ z = z_1 + iz_2. \end{cases} \tag{2.1}$$

Then, these three numbers are able to form the components of Z which is equal to:

$$\mathbf{Z} = (x, y, z). \tag{2.2}$$

Thus,

$$\mathbf{Z} \cdot \mathbf{Z} = \|\mathbf{Z}\|^2 = x^2 + y^2 + z^2 = 0 \tag{2.3}$$

as $\mathbf{X}_1$ and $\mathbf{X}_2$ are orthogonal with equal norms. The relation (**2.3**) between the three complex numbers x, y and z allows us to express them by means of just two complex numbers.

This relation can be put into the form:

$$\begin{aligned} z^2 &= -(x^2 + y^2) \\ &= -(x + iy)(x - iy). \end{aligned} \tag{2.4}$$

Then, if we set

$$\begin{cases} x + iy = -2\phi^2, \\ x - iy = 2\psi^2, \end{cases} \tag{2.5}$$

ψ and ϕ will allow us to calculate x, y and z. In fact, we have:

$$\begin{cases} x = \psi^2 - \phi^2, \\ y = i(\psi^2 + \phi^2), \\ z = \pm 2\psi\phi. \end{cases} \tag{2.6}$$

For the value of z, one of the signs, $+$ or $-$, can be chosen arbitrarily where we choose the negative sign. Thus, the two complex numbers ψ and ϕ form a representation for the two vectors $\mathbf{X}_1$ and $\mathbf{X}_2$ as well as the rank chosen for these two vectors.

We define a **spinor** to be the pair (ψ, ϕ) which is related to the vectors $\mathbf{X}_1$ and $\mathbf{X}_2$ by the relation (2.6) [Hla99]. In general, spinors are subject to the same simplification issues as tensors except when otherwise mentioned.

2.3 First View of Tensors

In this section, we will discuss the physical representation characterized by *scalars*, *vectors* and *tensors of rank two or higher* which will be denoted by bold capital letters such as $\mathbf{R}$, $\mathbf{U}$ and $\mathbf{V}$. This representation is used for spinors as well as tensors, except when otherwise mentioned.

2.3.1 Aims of Tensor Calculus

Tensor Calculus is a branch of geometry that allows us to formulate geometrical and physical theorems, usually as differential equations, in a way independent from the influence of the underlying arbitrarily chosen coordinate system.

It formulates equations valid for a family, or group, of coordinates that are obtained from each other by well-defined transformations, of various degrees of generality. For this reason tensor calculus is an ideal tool in several areas of mathematics such as differential geometry [Pap99].

It is desirable and often convenient to use tensor calculus as a mathematical background in which such laws can be formulated. In particular, Einstein found it an excellent tool for the presentation of his General Relativity theory.

As a result, tensor calculus came into great prominence and is now invaluable in its applications to most branches of theoretical physics. It is also indispensable in the differential geometry of hyperspace.

2.3.2 Tensor Indices and their Order

In mathematical or physical approaches, the name of symbols or *kernels*, tensorial or not: uppercase and/or lowercase, Latin and/or Greek is characterized by one or more indices or suffixes. These indices are indicated with notational variations such as: upper vs. lower, Greek vs. Roman, superscript(up or contravariant - not to be taken for powers!) vs. subscript(down or covariant), accented vs. unaccented, diacritical marks, etc.

For the rest of the thesis, the notations of covariant and contravariant will be used to define the position of each index.

For example, let

$$\mathbf{R}^{i}{}_{jkl},\ \mathbf{F}_{a\beta},\ \mathbf{C}_{i'jkl},\ \mathbf{R}_{jk_0} \tag{2.7}$$

be tensors such that $\mathbf{R}$, $\mathbf{F}$ and $\mathbf{C}$ are the kernels and $a, \beta, i, i', j, k, k_0$ and l are the indices at which i is the only contravariant index.

To avoid ambiguity with raising of a lower index and lowering of an upper index, it is sometimes necessary for tensors to specify the order of indices when both subscripts and superscripts occur. For example, the tensor $\mathbf{V}^{l}{}_{nm}$ in (2.8) is not necessarily the same as the tensor $\mathbf{V}^{l}{}_{mn}$.

Note that, we should not place the upper and the lower index in the same vertical line. For example,

$$\mathbf{T}^{ij}{}_{kl}^{mn}\,\mathbf{U}_i{}^k\,\mathbf{V}^l{}_{nm}. \tag{2.8}$$

Note that the index rank is always important for spinors which is not the case for tensors. For example, the tensor $T_k{}^k$ is equal to the tensor $T^k{}_k$ which is not the case for spinors. Thus, the position of each index in a tensor is important.

Finally, if two or more covariant or two or more contravariant indices are equal in a tensor, then the tensor itself will be equal to zero [Spa65].

2.3.3 Index Conventions

We shall now adopt an important convention with regard to indices. Since we are dealing with sum of products, and since it is difficult to keep writing summation signs within summation indices, we shall follow Einstein's example and adopt the convention by which we omit the summation index wherever it occurs simultaneously at two types one covariantly and one contravariantly. Thus, in these sums:

$$\sum_{k=1}^{n}\alpha_k, \quad \sum_{k=1}^{n}\beta_k^2, \quad \sum_{k=1}^{n}\alpha^i\beta_k \tag{2.9}$$

we cannot omit the summation index. Thus, the terms in (2.9) are not tensors. Only certain sums are admitted in a monomial to be a tensor. However, the sums

$$\sum_{k=1}^{n}\alpha^k\beta_k, \quad \sum_{i=1}^{n}\mathbf{A}^{iK}\beta_i, \quad \sum_{i,k=1}^{n}\mathbf{A}^{ik}\mathbf{B}_{ki}$$

can be abbreviated (omitting the summation index) as:

$$\alpha^k\beta_k, \quad \mathbf{A}^{ik}\beta_i, \quad \mathbf{A}^{ik}\mathbf{B}_{ki}. \tag{2.10}$$

This shorthand notation for a sum is called the ***summation convention*** (introduced by Einstein).

Since the repeated index is to be summed, it follows that the particular letter used for the repeated index is quite immaterial, and we may substitute it by any letter we please without altering the value of the expressions we are dealing with.

For this reason, the repeated index is often referred to as a **dummy index**. For example, in (2.10) we can replace the dummy index i with the index j without changing the summation. An index which is not repeated in any single term is known as a **free index** [McC57].

2.3.4 Rank of Tensors

The **rank** of a tensor is the sum of the covariant and the contravariant free indices.

1. A **scalar** contains only dummy indices. For example, $h_i{}^{li}{}_l$ and $\mathbf{U}^{ij}\mathbf{V}_{ji}$.
2. A **vector** have only one free index. For example, A_i, A^i and $\mathbf{U}_i{}^k{}_l\mathbf{V}_k{}^l$.
3. A **tensor of rank two** is classified into one of three classes:
   (a) **Contravariant Tensor** contains exactly 2 contravariant free indices. For example, $\mathbf{T}^{kl}$.
   (b) **Covariant Tensor** is a tensor which contains 2 covariant free indices. For example, $\mathbf{T}_{kl}$.
   (c) **Mixed Tensor** is a partly covariant and partly contravariant tensor. For example, $\mathbf{T}_i{}^j$ or $\mathbf{T}^j{}_i$. In particular, a mixed tensor transform as a contravariant vector with respect to the covariant index and like a covariant vector with respect to the contravariant index.

Thus, the components of a mixed tensor need exactly two free indices such that one index is a covariant index while the other is contravariant one. The position of these indices will indicate the class into which the tensor belongs.

If these classes are accepted, there is nothing to prevent an extension to higher rank tensors, with mixture of covariant and contravariant indices. Therefore, tensors of higher rank can be defined in the same way as tensors of rank two.

2.3.5 Tensors vs. Matrices

Tensors and Matrices are different and they are frequently confused for each other.

1. The Cartesian components of the tensor $\mathbf{T}_{ij}$ can always be used to create a square matrix T:

$$T = \begin{pmatrix} T_{11} & T_{12} & T_{13} \\ T_{21} & T_{22} & T_{23} \\ T_{31} & T_{32} & T_{33} \end{pmatrix}$$

   We call T the matrix of the tensor $\mathbf{T}$. Then, the components can be manipulated by standard techniques of matrix algebra. Thus, a tensor of rank two can be represented by an $n \times n$ square matrix. Meanwhile, a general $n \times m$ non-square matrix cannot be represented by a tensor.

2. A tensor with rank higher than two can not be in general represented by a one single matrix but its components can be arranged to form matrices of higher dimensions. For instance, the components of $\mathbf{B}_{ijk}$ form a cubic array in three dimensions.

2.4 Algebraic Operations with Tensors

We must now summarize the algebraic rules for manipulating tensors. In this section, tensors are taken to be of the same rank, which could be two or higher, except when otherwise mentioned. Most formulas are also valid when these tensors are vectors.

The fundamental operations of tensor algebra are addition, contraction, multiplication, permutation of indices, and symmetries. Addition and multiplication are governed by most of the rules as in the arithmetic of real numbers.

2.4.1 Addition

The sum of two tensors having the same free covariant indices and the same free contravariant indices is a tensor of the same name and the same rank as the original ones. Tensors are added by adding their corresponding coordinates. For example, consider the two tensors:

$$\begin{cases} \mathbf{A} \equiv \mathbf{U}_{ij}{}^{k}, & \textit{and} \\ \mathbf{B} \equiv 2\mathbf{U}_{ij}{}^{k} \end{cases} \tag{2.11}$$

Their sum is

$$\mathbf{C} \equiv \mathbf{U}_{ij}{}^{k} + 2\mathbf{U}_{ij}{}^{k} \equiv 3\mathbf{U}_{ij}{}^{k}. \tag{2.12}$$

Thus,

$\mathbf{T} = \mathbf{U} + \mathbf{V} = \mathbf{V} + \mathbf{U}$;
$(\mathbf{U} + \mathbf{V}) + \mathbf{T} = \mathbf{U} + (\mathbf{V} + \mathbf{T}) = \mathbf{U} + \mathbf{V} + \mathbf{T}$;
there exists a null tensor $\mathbf{0}$ s.t. $\mathbf{0} + \mathbf{T} = \mathbf{T}$;
for each $\mathbf{T}$, there exists -$\mathbf{T}$ s.t. $\mathbf{T} + (\text{-}\mathbf{T}) = 0$.

Therefore, tensors with the same free covariant indices and the same free contravariant indices form an *Abelian group* under addition. It is always true that tensors with a different number of covariant and/or different number of contravariant indices cannot be added together.

For instance, it is clear that we cannot expect to give any tensorial meaning to the expression $\mathbf{U}^{ij} + \mathbf{V}_i$. Moreover, the operation can be immediately extended to find the sum of any number of tensors provided that they all have the same free covariant indices and the same free contravariant indices. Subtraction of similar tensors is immediate.

2.4.2 Contraction

Let $\mathbf{T}$ be a mixed tensor of rank r. If we set a subscript index to be equal to a superscript one, then we create a pair of dummy indices. According to the summation conversion as in Section 2.3.3, summing this tensor with respect to the dummy index produces a new tensor, mixed or not, of rank $r - 2$. This process of forming a new tensor by summing over a new dummy indices is called **contraction.**

The only restriction on naming the contracted indices is that the name should not conflict with the names of free or dummy indices. In general, the treatment of the dummy indices is one of the most difficult problems in any computer algebra package.

For example, let $\mathbf{U}_{ij}{}^{k}$ be a mixed tensor of rank 3. If we set k equal to j, then the result appears as $\mathbf{U}_{ij}{}^{j}$. Since only one index is left as a free one, the result is the same as a covariant vector. We say that this vector is obtained from a third-rank tensor by contracting j and k.

This operation can evidently be repeated several times with respect to any pair of indices, one of which is a subscript index and the other is a superscript index. As just shown, contraction lowers the rank of a tensor by two.

2.4.3 Tensor Product

There exist different types of multiplication in tensor algebra. The most knows ones are: the outer and the inner product. Tensors in tensor product can have different name or rank.

Scalar Product

The multiplication of a tensor $\mathbf{T}$ with a scalar λ produces a tensor $\mathbf{S}$ with the same characteristics as $\mathbf{T}$ such that $\mathbf{S} = \lambda\mathbf{T}$. We denote the scalar product of $\mathbf{T}$ with λ by $\lambda \cdot \mathbf{T}$. This indicates that each coordinate of $\mathbf{T}$ is obtained by multiplying the corresponding coordinate of $\mathbf{T}$ by the same scalar factor λ. Thus, tensors can be multiplied by numbers just like vectors can [GoŁ74].

The multiplication of similar tensors with a negative scalar is immediate. Multiplication by a zero always yields a tensor with all of its coordinates are zero. Such tensors are referred to as *null* or *zero tensors*. Thus,

$$\begin{aligned} &0 \cdot \mathbf{T} = \mathbf{T} \cdot 0 = \mathbf{0}, \\ &(-\lambda) \cdot \mathbf{T} = \mathbf{T} \cdot (-\lambda) = -\lambda\mathbf{T}, \\ &c \cdot (d\mathbf{T}) = (cd) \cdot \mathbf{T} = cd \cdot \mathbf{T}. \end{aligned}$$

Outer Product

Just as for vectors, different tensors can be multiplied together. In fact, we can always construct a new tensor from any two given ones, say $\mathbf{S}$ and $\mathbf{T}$ of rank s and r respectively, by taking their outer product. We denote this by $\mathbf{T} \cdot \mathbf{S}$ of rank t such that $t = s + r$.

Thus, the components of the outer product of two tensors is the product of the components of the separate tensors. It is obvious that the result of the multiplication will always be a tensor of the sum of given ranks. The only restriction for applying the outer product

is that the intersection of the covariant indices of both tensors and the intersection of the contravariant indices of both tensors should always be empty, as mentioned in Property 1.2.

For example, let $\mathbf{S}^{ab}$ and $\mathbf{T}^{k}_{\ l}$ be two tensors of rank 2. Then, the outer product of both tensors is a tensor $\mathbf{T}^{k}_{\ l}\ \mathbf{S}^{ab}$ of rank $2+2=4$. Meanwhile, the outer product of tensors is always associative and distributive with respect to addition. Thus,

$$\boxed{\begin{aligned}&(\mathbf{S}+\mathbf{T})\cdot\mathbf{U}=\mathbf{S}\cdot\mathbf{U}+\mathbf{T}\cdot\mathbf{U};\\&(\mathbf{S}\cdot\mathbf{T})\cdot\mathbf{U}=\mathbf{S}\cdot(\mathbf{T}\cdot\mathbf{U})=\mathbf{S}\cdot\mathbf{T}\cdot\mathbf{U}.\end{aligned}}$$

Note that, tensors do always not obey the rules of ordinary arithmetic. For example, let $\mathbf{S}$ and $\mathbf{T}$ be two tensors of rank s and t respectively. Then, the equation $\mathbf{S}\cdot\mathbf{T}=0$ does not necessarily imply that $\mathbf{S}$ or $\mathbf{T}$ are null tensors.

Inner Product

The inner product of two given tensors $\mathbf{S}$ and $\mathbf{T}$ of rank s and t respectively can be constructed from the outer product of both followed by the contraction of any two indices such that each index is coming from different tensor. The rank of the inner product is always less than the sum of both ranks by two; i.e., $< s+r-2$.

Note that it is not intended to use more than one contraction in the inner product. For example, contracting the outer product $\mathbf{T}_{kl}\mathbf{S}^{a}_{\ b}$ of the tensors $\mathbf{T}_{kl}$ and $\mathbf{S}^{a}_{\ b}$ yields to either: $\mathbf{T}_{kl}\mathbf{S}^{k}_{\ b}$ or $\mathbf{T}_{kl}\mathbf{S}^{l}_{\ b}$.

In particular, the inner product of two vectors U_i and V^j is a tensor $\mathbf{U}_k V^k$ of rank zero and it is called *scalar product.*

Integral Powers

Similar to vectors, a tensor can be multiplied by itself n times, n is an even number, to give a scalar. By convention, $\mathbf{T}^2$ is a self contraction of all free indices. For example, let $T^{ia}{}_{a}$ be a tensor of rank one. Thus, $(T^{ia}{}_{a})^2 = T^{i}{}_{a}{}^{a} T_{ib}{}^{b}$.

Thus, integral powers of a tensor $\mathbf{T}$ are defined inductively by:

$$\boxed{\begin{aligned} &T^0 = 1, \\ &T^1 = T, \\ &\mathbf{T}^n = \mathbf{T}^{n-1} \cdot \mathbf{T}, \\ &\mathbf{T}^m \cdot \mathbf{T}^n = \mathbf{T}^{m+n}, \\ &(\mathbf{T}^m)^n = \mathbf{T}^{mn}. \end{aligned}} \tag{2.13}$$

where m and n are nonnegative even integers.

It should be noted that the integral powers do not obey all the time the rules of ordinary powers. For example, if $\mathbf{T}$ is a given tensor, then the relations in (2.13) do not apply for an odd integer ≥ 3.

2.4.4 Raising and Lowering Indices

The metric tensor is represented by the special symbol g. The main use of this tensor in this thesis is usually used to raise and/or lower indices of a tensor.

To raise and/or lower an index of a tensor $\mathbf{T}$, we form an inner product of $\mathbf{T}$ with the covariant or the contravariant metric tensor. For example, let v^j be a contravariant vector. The contraction of v^j with the metric tensor g_{ij} forms $g_{ij}v^j$. Thus, the new vector transforms covariantly and it is denoted by v_i.

Similarly, $v^i = g^{ij}v_j$. It should be noted that the method of raising and/or lowering

indices is not confined to vectors, but may be applied to any index of a tensor of any rank.

Thus, these operations change the type of a tensor $\mathbf{T}$ but preserve its rank. They do not alter the tensor in any fundamental way; i.e., it is the same tensor but in a different representation. Thus, $\mathbf{R}_{kl}S_r$ and $\mathbf{R}^k{}_l S_r$ are equivalent.

Also these operations are *invertible*: raising and then lowering the same index, and vice versa, leads to the original tensor.

2.4.5 Permutation of Indices

In general, we can permute a set of indices in a given tensor. This permutation leads to another tensor. For example, let $\mathbf{U}_{ijk...m}$ and $\mathbf{U}_{jki...m}$ be two tensors. Thus, $\mathbf{U}_{jki...m}$ differs from the tensor $\mathbf{U}_{ijk...m}$ by permuting some of its indices. The importance of this operation appears by defining the property of symmetries for tensors [Aki72].

2.4.6 Symmetry Properties

If two covariant, contravariant or mixed indices of a tensor of rank r can be interchanged without changing the sign of the tensor, then the tensor is said to be **symmetric** with respect to both indices. For instance, if $\mathbf{H}_{ij}{}^{kl} = \mathbf{H}_{ji}{}^{kl}$, the tensor $\mathbf{H}_{ij}{}^{kl}$ is symmetric with respect to i and j.

If symmetry in a tensor holds for the permutation of any two same type or mixed indices, then the tensor is a *completely symmetric* or simply a *symmetric* tensor. For instance, the metric tensor is symmetric since $g_{ab} = g_{ba}$. Note that the symmetry is invariant under transformation of coordinates.

Similarly, if the interchange of same type or mixed pair indices of a tensor changes the sign of the tensor, then it will be called *completely antisymmetric* or simply *antisymmetric*

with respect to both indices.

Meanwhile, the tensor is antisymmetric if the sign of the tensor changes by an odd number of permutation of any two indices of the tensor. For example, if $\mathbf{T}_{ij} = -\mathbf{T}_{ji}$, then the tensor $\mathbf{T}_{ij}$ is antisymmetric with respect to i and j. The symmetry and the antisymmetry of tensor $\mathbf{T}_{ij}$ can be summarized by the formulas:

$$\begin{array}{|ll|}\hline \mathbf{T} \text{ is symmetric} & \Longleftrightarrow \mathbf{T}_{ij} = \mathbf{T}_{ji}, \\ \mathbf{T} \text{ is antisymmetric} & \Longleftrightarrow \mathbf{T}_{ij} = -\mathbf{T}_{ji}. \\ \hline \end{array}$$

Many tensors have special symmetries under the permutations of their indices. They can have a combination of both types of symmetries. For instance, $\mathbf{R}_{abcd}$ represents the Riemann tensor with four indices, namely **a, b, c** and **d**. This tensor is antisymmetric under the change of the first pair of indices, $\{a, b\}$, or the last pair of indices, $\{c, d\}$. At the same time, it is symmetric with respect to both pairs, $\{a, b\}$ and $\{c, d\}$.

Other symmetries relate multiple terms. For example, the sum of terms obtained by cyclic interchange of the last three indices of the Riemann tensor is zero. For example, let $\mathbf{R}^{d}{}_{cba}$ be a Riemann tensor. Thus,

$$\begin{cases} \mathbf{R}^{d}{}_{cba} = -\mathbf{R}^{c}{}_{dba} = -\mathbf{R}^{c}{}_{dab} = \mathbf{R}^{a}{}_{bcd}, \\ \mathbf{R}^{d}{}_{bca} + \mathbf{R}^{d}{}_{cab} + \mathbf{R}^{d}{}_{abc} = 0. \end{cases} \tag{2.14}$$

2.5 Introduction to Specific Tensors

As introduced in the previous section, the **Riemann tensor** has exactly 4 different indices which they can permute only as explained previously. From the presence of this tensor, we can derive many different ones such as the **Ricci tensor**, the **Einstein tensor**, and the **Weyl tensor**.

The **Ricci tensor** of first kind is simply the contraction of the first and the last indices of a the Riemann tensor introduced in (2.14) such that:

$$\mathbf{R}_{cb} \equiv \mathbf{R}^{a}{}_{cba}. \tag{2.15}$$

The last index can be raised to yield the Ricci tensor of the second kind:

$$\mathbf{R}_{c}{}^{b} \equiv g^{br}\mathbf{R}_{cr}. \tag{2.16}$$

As a consequence of the Riemann tensor, the Ricci tensor is symmetric. If this tensor is contracted by letting $c = b$, we get the **Ricci curvature scalar** such that:

$$\mathbf{R}^{a}{}_{a} \equiv \mathbf{R}. \tag{2.17}$$

Meanwhile, the combination of

$$\mathbf{R}^{ab} - \frac{1}{2}\, g^{ab}R \;\equiv\; \mathbf{G}^{ab}. \tag{2.18}$$

is known as the **Einstein tensor**. It is called after Einstein since its importance for gravity was first understood by Einstein.

Now, by subtracting from $\mathbf{R}_{abcd}$ the appropriate terms formed from its contractions, we can construct a tensor that has no non-zero contractions which known as the **projective tensor** or **Weyl tensor**. This tensor has all the symmetry properties of the Riemann tensor and it has the following form for space of dimension $n > 2$:

$$\begin{aligned} \mathbf{C}^{ijhl} \;\equiv\; & \mathbf{R}_{ijhl} \\ & - \frac{1}{(n-2)}(g_{ih}\mathbf{R}_{jl} - g_{il}\mathbf{R}_{jh} - g_{jh}\mathbf{R}_{il} + g_{jl}\mathbf{R}_{ih}) \end{aligned} \tag{2.19}$$

$$+\frac{1}{(n-1)(n-2)}R(g_{ih}g_{jl}-g_{il}g_{jh})$$

It can be shown that the Weyl tensor is zero in 3 dimensions.

Finally, The **Maxwell tensor**, denoted by $\mathbf{F}^{ij}$, is an antisymmetric tensor and it was introduced to explain Maxwell's theory in a compact form. Maxwell started from the idea that a spatially distributed electromagnetic field possesses certain properties of elasticity, and when constructing its model he used the analogy with the theory of elastic continuum which was then already developed in all details.

Chapter 3

Graph Theory

3.1 Introduction

Throughout the many branches of mathematics, one frequently encounters the fundamental concepts of **sets** and **relations**. The theory of graphs offers no exception to that. In fact, a graph may be defined as a finite nonempty set with some kind of relations.

Graph theory, which had arisen out of puzzles solved for the sake of curiosity, has now grown into a major discipline in mathematics with problems permeating into almost all subjects such as physics, chemistry, engineering, psychology, and computer science. This subject, which has been lying almost dormant for hundreds of years since the time of Euler, suddenly started exploding by the turn of the 20^{th} century, and it has branched off today in various directions such as coloring problems, Ramsey theory, factorization theory, computer science and optimization.

Some puzzles and various problems of a practical nature have been instrumental in the development of various topics in graph theory. The famous Königsberg Bridge problem has been the inspiration for the development of graph theory. In fact, graph theory can be counted as a mathematical subject for over three centuries.

In this chapter, we will introduce a few terms of graph theory, present some examples, state some graph isomorphisms and canonical relabeling, and describe some useful graph operations.

3.2 Basic Graph Theory

We can think of a *graph* as a set of points in a plane and as a set of line segments, possibly curved, each of which either joins two points or joins a point to itself [GY94].

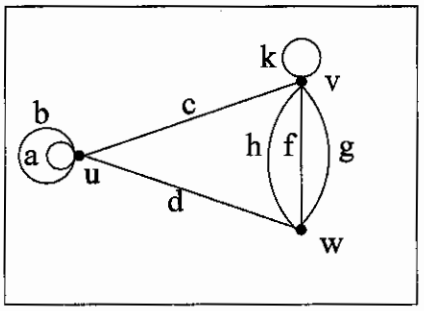

Figure 3.1: Representation of a multi-graph

Graphs are highly versatile models for analyzing a wide range of practical problems in which points and connections between them have some physical or conceptual interpretation. In this thesis, each graph consists of a list of vertices, list of edges and their components. The construction of graphs in here is similar to the one used in graph theory.

The following development is summarized from [GY94].

Definition 3.1 *A* ***Graph*** $G = (V, E)$ *is a mathematical structure consisting of two sets:* V *and* E*. The elements of* V *are called* ***vertices***, ***nodes*** *or* ***kernels*** *and the elements of* E *are called* ***edges****. Each edge has a set of one or two vertices associated to it. A* ***loop-edge*** *is an edge going from a vertex to itself.*

Example 3.1 *The graph in 3.1 has vertex-set* $V = \{u, v, w\}$ *and edge-set* $E = \{a, b, c, d, f, g, h, k\}$*. The edges* $\{a, b, k\}$ *are edges that join a single vertex to itself, and the set of*

$\{c, d, f, g, h\}$ *are edges connected to more than one vertex.*

Remark 3.1 *Let $G = (V, E)$ be a graph. We denote $|V|$ and $|E|$ to be the number of vertices and the number of edges respectively in the graph G.*

Remark 3.2 *An edge between two vertices may be considered as a connection in either direction. Assigning a direction makes one of these forward and the other backward. In a line drawing, the choice of forward direction is indicated by placing an arrow on the edge.*

Definition 3.2 *A graph is said to be labeled, if its n vertices are distinguished from one another by labels such as $\{v_1, v_2, ..., v_n\}$ [RB99]. For example, the graph in Figure* 3.1 *is labeled.*

Definition 3.3 *Let $G = (V, E)$ be a graph such that $|V| > 1$. We say that V is in a lexicographical order if the vertices are ordered based on their labels. For example, $\{v_1, v_2, ..., v_n\}$ are ordered in a lexicographical order.*

Remark 3.3 *Let $G = (V, E)$ be a graph such that $V = \{v_0, v_1, ..., v_n\}$. We can always* ***relabel*** *a graph G with an integer i by adding the subscript integer of each vertex to the integer i. For example, let $G = (V, E)$ be a graph such that $V = \{v_0, v_1, v_2\}$. By relabeling G with the integer $i = 3$, the new graph $G_1 = (V_1, E_1)$ will be modified such that the list of vertices $V = \{v_3, v_4, v_5\}$.*

Definition 3.4 *A* ***directed edge*** *is an edge, one of whose endpoints is designated as the* ***tail*** *with the other endpoint designated as the* ***head****.*

Definition 3.5 *Let e_1 and e_2 be 2 directed edges. We say that $e_1 = e_2$ if the head of e_1 is equal to the head of e_2 and if the tail of e_1 is equal to the tail of e_2.*

Definition 3.6 *Let e_1 and e_2 be 2 different directed edges. We say that $e_1 < e_2$ if the label of the head of e_1 is smaller of the label of the head of e_2. If both labels are equal, then $e_1 < e_2$ if the label of the tail of e_1 is smaller of the tail of e_2.*

Definition 3.7 *Let V be a non-empty set, and let $E \subseteq V \times V$. A **directed** graph or **digraph** $G = (V, E)$ is a graph such that all edges are directed. An **undirected** graph $G_1 = (V_1, E_1)$ is a graph with all edges undirected. For example, the abstract representation of computer programs can be represented only by directed graphs.*

Definition 3.8 *A directed or undirected graph $G = (V, E)$ is called a **multi-graph** or **multi-edges graph** if for some $u, v \in V$, there are two or more directed or undirected edges from u to v or from v to u.*

Definition 3.9 *Let $G = (V, E)$ be a multi-graph such that $|E| > 1$. By ordering the elements of E as mentioned in Definition 3.6, we can say that E is in a lexicographical order.*

Definition 3.10 *Let $G = (V, E)$ be a multi-graph. We say that G is in order if V and E are in lexicographical order as mentioned in Definition 3.3 and in Definition 3.9.*

Definition 3.11 *Let G be a directed, undirected or multi-graph. For any vertex v of G, the **degree** of that vertex is the total number of edges in G, denoted by $deg_G(v)$, that begins or ends with v. The degree of a loop is considered to count as 2.*

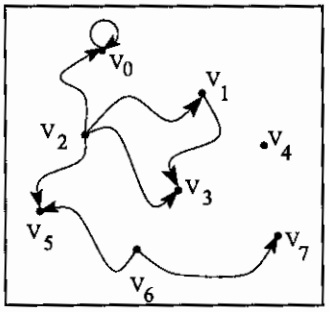

Figure 3.2: A graph and its degree.

Example 3.2 *In **Figure 3.2**, $deg_G(v_1) = deg_G(v_3) = deg_G(v_5) = deg_G(v_6) = 2$, $deg_G(v_2) = 4$, $deg_G(v_4) = 0$, and $deg_G(v_7) = 1$. For v_0, the $deg_G(v_0) = 3$ since we count the loop twice.*

3.3 Graph Isomorphism and Canonical Relabeling

Deciding when two line drawings represent the same graph can be quite difficult for graphs containing more than a few vertices and edges. A related task is deciding when two graphs with different specifications are *structurally equivalent*, that is, whether they have the same pattern of connections.

Designing a practical algorithm to make these decisions is a famous unsolved problem, called the **graph-isomorphism problem.** Since the shape or length of an edge and its position in space are not part of a graph's specification, each graph has infinitely many special representations.

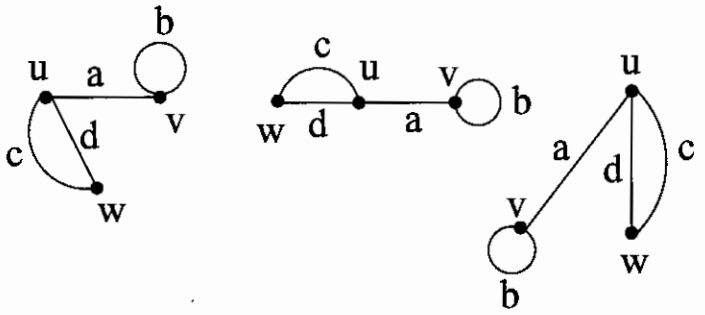

Figure 3.3: Different drawings of the same undirected graph.

The vertices and edges in the three drawings above have matched labels. Since each graph has exactly three matched vertices and same four edges, it is easy to recognize that these three drawings represent the same graph.

Definition 3.12 *Let* $G_1 = (V_1, E_1)$ *and* $G_2 = (V_2, E_2)$ *be two graphs. A function* $f : V_1 \to V_2$ *is called a* ***graph isomorphism*** *if:*

1. f *is one-to-one and onto and*

2. *for all* $u, v \in V_1, (u, v) \in E_1$ *if and only if* $(f(u), f(v)) \in E_2$.

When such a function exists, G_1 and G_2 are called ***isomorphic*** *graphs. [Gra94]*

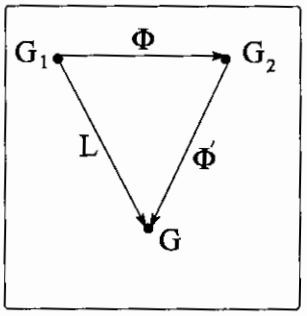

Figure 3.4: Canonical labeling of a graph

Definition 3.13 *Let $G = (V, E)$ be a graph and let $\boldsymbol{K}$ be a class of graphs having the same vertex set V. We assume that if G_1 and G_2 are isomorphic graphs with $V_1 = V_2 = V$, then $G_1 \in \boldsymbol{K}$ implies $G_2 \in \boldsymbol{K}$. Let $|V| = n$. By a* ***canonical labeling*** *of the class $\boldsymbol{K}$, we mean a function L whose domain is $\boldsymbol{K}$ such that:*

1. *$L(G)$ is a labeling of G, i.e. a bijection $V \to \{1, ..., n\}$ for any $G \in \boldsymbol{K}$,*
2. *If G_1 and G_2 belong to $\boldsymbol{K}$, then they are isomorphic if and only if the map $L(G_2)^{-1} \circ L(G_1) : V \to V$ is an isomorphism $G_1 \to G_2$.*

Clearly, a canonical labeling can be used to decide whether the graphs G_1 and G_2 are isomorphic, provided at least one of them belongs to $\boldsymbol{K}$. [Bab80]

3.4 Graph Operations

Computer scientists often regard a graph as a modifiable data structure. Accordingly, the configuration that results when a vertex or an edge is added to or deleted from a graph G is considered to be a new value of G. These basic operations are part of the *graph*, just as the operations of addition and scalar multiplication are part of the definition of a vector space.

The following development is summarized from [GY94].

Definition 3.14 *Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are equal if, and only if, $V_1 = V_2$ and E_1 is identical to E_2.*

Remark 3.4 *Let $G = (V, E)$ be a multi-graph and let $v \in V$. We define **Ord**(v) to be an ordered list of edges, as defined in 3.6, such that the header is always v.*

Remark 3.5 *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two ordered multi-graphs. We say that $E_1 < E_2$ if $|E_1| < |E_2|$. If $|E_1| = |E_2|$, then $E_1 < E_2$ if the first element of E_1 is smaller than the first element of E_2 and so on as explained in Definition 3.6.*

Remark 3.6 *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs. We say that $G_1 < G_2$ if $|V_1| < |V_2|$. If $|V_1| = |V_2|$, then $G_1 < G_2$ if $E_1 < E_2$.*

Definition 3.15 *If v is a vertex of a graph G, then the **vertex-deletion** $G - v$ is a graph induced by the vertex-set $V_G - \{v\}$. That is,*

1. *$V_{G-v} = V_G - \{v\}$, and*
2. *$E_{G-v} = \{e \in E_G : v \notin \textbf{head}(e) \text{ and } v \notin \textbf{tail}(e)\}$.*

More generally, if $U \subseteq V_G$, then the result of iteratively deleting all the vertices in U is denoted by $G - V$.

Definition 3.16 ***Adding a vertex** v to a graph G, where v is a new vertex not already in V_G, means creating a new graph, denoted $G \cup \{v\}$, with vertex-set $V_G \cup \{v\}$ and edge-set E_G.*

Definition 3.17 *If e is an edge of a graph G, then the **edge-deletion** $G - e$ is a graph induced be the edge-set $E_G - \{e\}$. That is,*

1. *$V_{G-e} = V_G$, and*

2. $E_{G-e} = E_G - \{e\}$.

More generally, if $D \subseteq E_G$, then the result of iteratively deleting all the edges in D is denoted by $G - D$.

Definition 3.18 ***Adding an edge*** *between two vertices u and w of a graph G means creating a new graph, denoted $G \cup \{e\}$, with vertex-set V_G and edge-set $E_G \cup \{e\}$, where e is a new edge with endpoints u and w.*

Remark 3.7 *Each operation of adding or deleting a vertex or adding or deleting an edge is called a* ***basic operation****. In general, we can combine and/or repeat one or more basic operation. Whether or not a person is designing a software, it is mathematically interesting to analyze or synthesize a graph construction as a sequence of basic operations. In general, all non-basic operations can be constructed by combining and/or iterating the basic operations.*

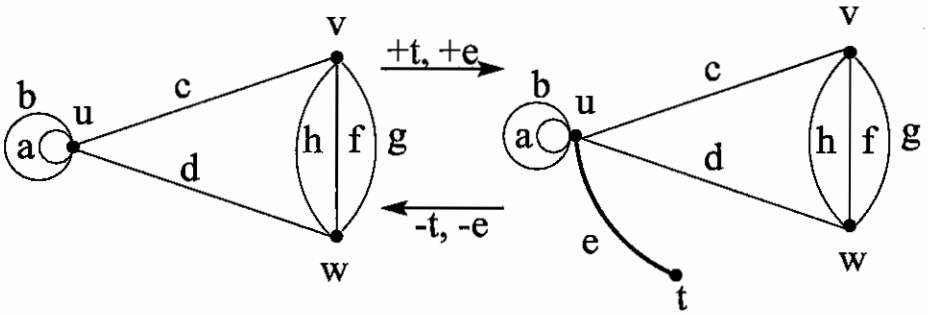

Figure 3.5: Adding and deleting a vertex t and an edge e.

In mathematics, we can always generate many new graphs from a given set of graphs. In this section, we consider some of the methods to generate new graphs from a given pair of graphs.

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two directed graphs or directed multi-graphs. Note that these two graphs do not need to be disjoint.

Table 3.1: Binary Operations of Graphs

Operation	Symbol	Number of vertices	Number of edges
Union	$G_1 \cup G_2$	$V_1 + V_2 - V_1 \cap V_2$	$E_1 + E_2 - E_1 \cap E_2$
Sum	$G_1 + G_2$	$V_1 + V_2$	$E_1 + E_2$
Intersection	$G_1 \cap G_2$	$V_1 \cap V_2$	$E_1 \cap E_2$
Join	$G_1 \vee G_2$	$V_1 + V_2$	$E_1 + E_2 + E_1 E_2$

Definition 3.19 ***Union of two graphs:*** *The graph $G = (V, E)$, where $V = V_1 \cup V_2$ and $E = E_1 \cup E_2$ is called the union of G_1 and G_2 and is denoted by $G_1 \cup G_2$.*

Remark 3.8 *When G_1 and G_2 are vertex disjoint, $G_1 \cup G_2$ is denoted by $G_1 + G_2$ and is called the sum of the graphs G_1 and G_2. It is always true that the union and the sum of graphs are commutative; i.e., $G_1 \cup G_2 = G_2 \cup G_1$ and $G_1 + G_2 = G_2 + G_1$.*

Definition 3.20 ***Intersection of two graphs:*** *If $V_1 \cap V_2 \neq \emptyset$, then the graph $G = (V, E)$, where $V = V_1 \cap V_2$ and $E = E_1 \cap E_2$, is the intersection of G_1 and G_2 and it will denoted by $G_1 \cap G_2$.*

Definition 3.21 ***Join of two graphs:*** *Let G_1 and G_2 be vertex-disjoint graphs. Then the join graph, $G_1 \vee G_2$, of G_1 and G_2 is a new graph in which each vertex of G_1 is adjacent to every vertex of G_2.*

For each of the above operations, we can calculate the number of vertices and the number of edges in the resulting graph, as shown in Table 3.1.

Remark 3.9 *There exist more operations to be considered such as:*

1. *The Cartesian product: $G_1 \times G_2$,*

2. *Composition product:* $G_1[G_2]$,

3. *Normal product:* $G_1 \circ G_2$, *and*

4. *Tensor product or Kronecker product:* $G_1 \otimes G_2$.

These operations are less important for us than the ones in Table 3.1.

Remark 3.10 *By* ***contracting*** *an edge e, we refer to the operation of removing e and identifying its end vertices. A graph G is contractible to a graph H if H can be obtained from G by a sequence of contractions.*

Remark 3.11 *We can always create a new directed graph or directed multi-graph, say* $G_3 = (V_3, E_3)$ *from* G_1, *by changing the direction of one or more of its directed edges.*

Finally, we need to introduce the powers for a graph G.

Definition 3.22 *The* k^{th} ***power*** G^k *of* G *has* $V(G^k) = V(G)$, *where* u *and* v *are adjacent in* G^k *whenever* $deg_G(u, v) \leq k$ *[RB99].*

3.5 Matrix Representations

A graph G can be completely determined either by its adjacencies or by its incidences. There are a variety of standard data structure representations for graphs. This information can be conveniently stated in matrix form.

Indeed, with a given graph adequately labeled, there are several associated matrices, including the adjacency matrix, incidence matrix, distance matrix, and cocycle matrix.

It is often possible to make use of these matrices in order to identify certain properties of a graph. Furthermore, representing graphs by matrices remains important as a

conceptual and theoretical tool. It helps us to bring the power of linear algebra to graph theory.

For example, suppose we want to test a conjecture of a graph with the aid of a computer. A standard technique is to represent, store, and manipulate the graph in computer memory using a matrix. A common matrix used in this way is the **adjacency matrix**.

The adjacency matrix is called a *binary matrix* if all the entries of that matrix are either 0 or 1.

Definition 3.23 *The **adjacency matrix** of a directed multi-graph G, denoted A_G, is the matrix whose rows and columns are both indexed by identical orderings of V_G, such that:*

$$A_G[u,v] = \begin{cases} \text{the number of edges from } u \text{ to } v & , \quad \text{if } u \neq v, \\ \text{the number of self-loops at } v & , \quad \text{if } u = v \end{cases}$$

While the adjacency matrix for an undirected graph is symmetric, the adjacency matrix for a directed graph is antisymmetric.

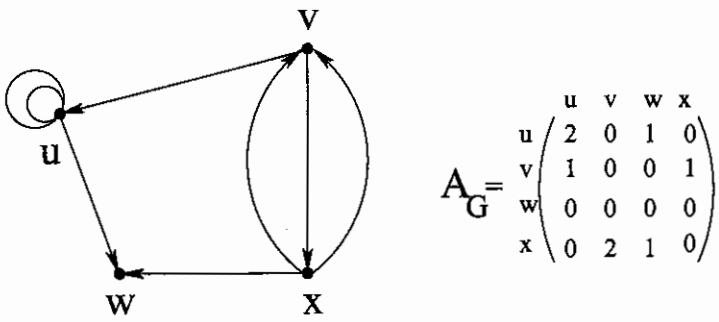

Figure 3.6: A graph and its adjacency matrix.

Example 3.3 *Let $G = (V, E)$ be a multi-graph. Then, $\boldsymbol{A}_G$ is the adjacency matrix for the directed multi-graph G, as shown in* ***Figure 3.6****, by using the vertex ordering u, v, w, and x.*

Usually the vertex order is implicit from the context, in which case the adjacency matrix A_G can be written as a matrix without an explicit row or column labeling.

Proposition 3.1 *Let G be a direct multi-graph with $V_G = \{v_1, v_2, ..., v_n\}$. Then the sum of the elements of row i of the adjacency matrix A_G equal to the outdegree of vertex v_i, and the sum of the elements of column j equal to the indegree of vertex v_j.*

Remark 3.12 *Proposition 3.1 implies that two graphs are isomorphic if it is possible to order their respective vertex-sets in a way that their adjacency matrices are identical.*

Since the adjacency matrix of a graph is square, we can investigate its determinant. Clearly, the determinant of A is independent of the labeling of the nodes of G. Hence we may say that the *determinant* of a graph G is the determinant of any adjacency matrix of G [FB89].

Chapter 4

Formulation of Tensor Algebra as an Algebra of Graphs

4.1 Introduction

In general, tensors are mathematical objects with a finite number of indices. Tensors can have different properties such as symmetries and invariance with respect to a specific renaming of dummy indices. This indicial notation method permits us:

1. to use simplifying mathematical operations, and
2. to write equations in a compact manner.

As mentioned in Section 1.3, each tensor consists of four parts. They are summarized as follows by its:

1. **name** such as **R**, **T**, **C**, ...,
2. **indices** which may be categorized free or dummy,
3. **position** of each index first, second, third, ..., and

4. **type** of each index such as subscript or superscript.

On the other hand, graph theory is a mathematical branch which has many applications in different areas. Graphs successfully serve as mathematical models for many concrete real-world problems.

Certain problems in physics, chemistry, statistics, engineering, psychology, communications science, operational research, and computer technology can be formulated as problems in graph theory. Also, many branches of mathematics, such as group theory, number theory, matrix theory, combinatorics, probability, and topology have lots of interactions with graph theory.

The purpose of this chapter is to formulate each basic operation in tensor algebra to a similar one in graph theory.

4.2 Relabeling Dummy Indices

In every tensor, the dummy indices are the contracted ones. It could happen sometimes that some terms in an expression might cancel if the indices are labeled properly. In most algorithms, the method of relabeling dummy indices can sometimes be quite complicated to execute.

Since the indices in this algorithm are represented as edges in a graph and since the name of the dummy indices is not important anymore, all labeling restrictions have been eliminated. We simplified this problem by representing in the output each dummy index with the character "%" followed by an integer.

This method make the algorithm more efficient, gives the user more flexibility and eliminates most if not all restrictions for relabeling the dummy indices.

For example, let $Expr = \mathbf{R}_{abcd}\mathbf{R}^{abcd} + \mathbf{R}_{mnlk}\mathbf{R}^{lkmn}$ be a tensor expression. The

dummy indices a, b, c, d and k, l, m, n will be relabeled $\%1, \%2, \%3$ and $\%4$ respectively and $Expr = \mathbf{R}_{\%1\%2\%3\%4}\mathbf{R}^{\%1\%2\%3\%4} + \mathbf{R}_{\%1\%2\%3\%4}\mathbf{R}^{\%3\%4\%1\%2}$. Applying the symmetries will create identical terms and the answer is: $2\mathbf{R}_{\%1\%2\%3\%4}\mathbf{R}^{\%1\%2\%3\%4}$.

4.3 Tensor Monomials as Multi-graphs

A tensor can be formulated and be associated to a graph by representing:

1. its **name** as a **vertex** in the graph,
2. the **indices** of the tensor, free or dummy, as **edges** in the graph,
3. the **type** of the indices, subscript or superscript, as **direction** of edges, and
4. the **position** of each indices will be shown on its corresponding edge.

In general, by multiplying two or more tensors, we can create a tensor monomial. The representation of a monomial is similar to the representation of each tensor separately as a graph followed by adding these graphs as mentioned later in Chapter 6.

4.4 Tensor Algebra and Algebra of graphs

After describing the relation between tensors and graphs, we now need to formulate the relation between tensor algebra and algebra of graphs. Throughout the thesis, we reserve the vertex $\mathbf{V}_0()$ to represent the free indices in a given tensor. Meanwhile, the numbers on the edges represent the position of the indices in their corresponding tensors.

Since it is easier to add or to compare two graphs in graph theory rather than adding or multiplying two tensors in tensor algebra, graph theory will be used to treat and to simplify all tensor expressions.

4.4.1 Addition

The problem of adding two tensors can be formulated as a question: whether two tensor expressions are equal or not, taking into account the graph theory properties? Then, the simplest and shortest canonical form problem for an expression arises as a central one.

The method consists of representing each tensor as a graph and compare them as described in Section 3.14. The same method will be used to identify multiples of common monomials. For instance, let $U_a{}^a{}_b$, V_b, $U^c{}_{cb}$ and $U_b{}^i{}_i$ be 4 symmetric tensors. Adding them will create an expression that looks like: $V_b + 3U_a{}^a{}_b$.

As another example, let us consider the two Riemann tensor expressions:

$$\mathbf{R}_{abmn} \cdot \mathbf{R}^{mncd} \qquad and \qquad \mathbf{R}_{abij} \cdot \mathbf{R}^{ijcd}. \tag{4.1}$$

It is not difficult to see that both monomials are equal. Indeed, the second term can be transformed to the first one by renaming their dummy indices.

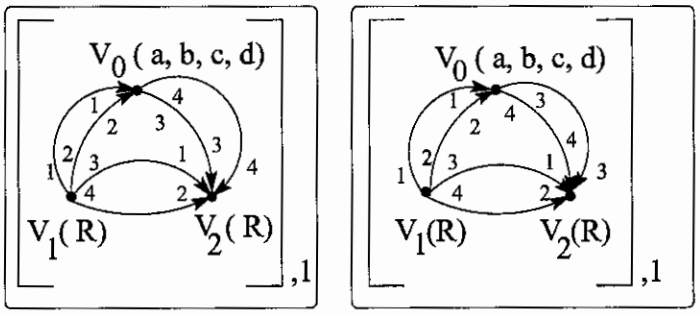

Figure 4.1: Representation of $\mathbf{R}_{abmn} \cdot \mathbf{R}^{mncd}$ and $\mathbf{R}_{abij} \cdot \mathbf{R}^{ijcd}$ respectively.

By representing each monomial as a graph and as shown in Figure 4.1, it is obvious that both graphs are identical, as mentioned in Definition 3.14, since there is no difference between the edges representing the dummy indices.

4.4.2 Contraction

For a tensor of rank ≥ 2, we set a subscript index to be equal to a superscript index, then we create a pair of dummy indices one of which is covariant with the other contravariant. This process of forming a new tensor of rank smaller than the original one by summing over the dummy index is called **contraction**. This method can be formulated as the contraction of edges shown in Remark 3.10.

Meanwhile, the contraction can be created in the presence of either a tensor or a tensor monomial and it can be done by modifying the edges of the graph.

In the presence of a tensor, the contraction can be formulated as a loop-edge going from $\mathbf{V}_1$ to itself and then eliminating both edges which represent both free indices. The full description of this method can be found in Table 4.1 where $m = n$.

For instance, the index c in $3\mathbf{A}^{abi}{}_{cib}$ is contracted with a which create a new tensor looks like: $3\mathbf{A}^{abi}{}_{aib}$. The corresponding graph of that tensor is represented as the first graph in Figure 4.4.

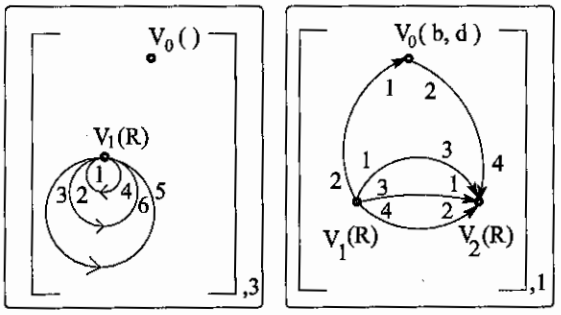

Figure 4.2: Representation of $3\mathbf{A}^{abi}{}_{aib}$ and $\mathbf{R}_{abmn} \cdot \mathbf{R}^{mnad}$ respectively

Meanwhile, the only difference appears in the case of a tensor monomial is that the edge of the new dummy index will be going from one vertex to another depending on the position of the contracted indices. Table 4.1 summarize this situation where $m \neq n$. For

Table 4.1: Contracting i_1 and i_2 in a tensor $\mathbf{T}$.

1. $\mathbf{G(T)} \leftarrow \mathbf{G(T)} - \mathbf{EF}(i_1) - \mathbf{EF}(i_2)$
2. $\mathbf{G(T)} \leftarrow \mathbf{ED}(i_1)$
3. $\mathbf{FL(T)} \leftarrow \mathbf{FL(T)} - i_1 - i_2$
4. return $\mathbf{T}$

example, let $\mathbf{R}_{abij} \cdot \mathbf{R}^{ijcd}$ be a tensor monomial from the previous section. By contracting c and a, the new tensor is represented as the second graph in Figure 4.4.

4.4.3 Tensor Product

For the next three different product types, the tensors do not have to have the same name or the same rank.

Scalar Product

If a tensor is obtained by the multiplication of a scalar λ by a tensor $\mathbf{S}$ of rank s, then the new tensor, which will be denoted by $\lambda\mathbf{S}$, will have a rank equal to the rank of $\mathbf{S}$.

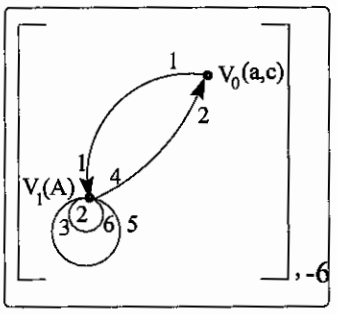

Figure 4.3: Representation of $2 * (-3\mathbf{A}^{abi}{}_{cib})$

The method consists of representing the tensor as a graph and then multiplying the

given scalar by the integer of the tensor shown on the bottom right of the graph.

For example, let $3\mathbf{A}^{abi}{}_{cib}$ be a tensor as shown in Figure 5.1. Then, by multiplying that tensor by -2, the new tensor $-6\mathbf{A}^{abi}{}_{cib}$ will be shown as in Figure 4.3.

Outer Product

If a tensor monomial is obtained by the multiplication of any two given tensors, say $\mathbf{S}$ and $\mathbf{T}$ of rank s and r respectively, then the new tensor, which is denoted by $\mathbf{T} \cdot \mathbf{S}$, will have a rank t such that $t = s + r$.

Thus, the components of a product of two tensors is the product of the components of the separate tensors. It is obvious that the product will always be a tensor of certain rank. The only restriction applies with this type of product is that the set of covariant indices of both tensors and the set of contravariant indices of both tensors should always be empty.

The product of two tensors can be formulated as the addition of their graphs mentioned in Remark 3.8. In fact, the multiplication of two tensors creates a tensor monomial which it can be represented as a new graph. The full method of outer product is described later in Table 7.2.

Inner Product

As we mentioned in Section 2.4.3, the inner product is the combination of the outer product and the contraction of indices. Meanwhile, the inner product of two given tensors of rank s and r is a tensor of rank less than the sum of both ranks. The representation of this type of product can be represented as shown in Section 4.4.3 followed by Section 4.4.2.

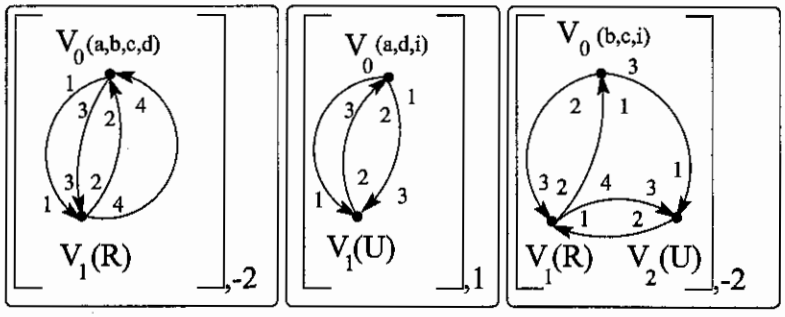

Figure 4.4: Representation of $-2\mathbf{R}^{a\ c}_{b\ d}$, $\mathbf{U}^{i\ d}_{a}$ and $-2\mathbf{R}^{a\ c}_{b\ d}\mathbf{U}^{i\ d}_{a}$ respectively.

For example, let $T_1 = \mathbf{R}^{a\ c}_{b\ d}$ and $T_2 = \mathbf{U}^{i\ d}_{a}$ be two tensors of ranks 4 and 3 respectively. Then, multiplying T_1 by T_2 will create a new tensor T_3 which is equal to $\mathbf{R}^{a\ c}_{b\ d}\mathbf{U}^{i\ d}_{a}$ of rank 3 as shown in Figure 4.4.

4.4.4 Raising and lowering indices

We can always create a new tensor from a given one by raising and/or lowering one or more index as in Section 2.4.4. By raising and/or lowering one of the dummy indices, we need to lower/raise the other dummy index.

Notice that raising and/or lowering the indices does not change the rank of the given tensor. Similarly, we can surely create a graph G_1 from a graph G by changing the direction of one or more edges as in Section 3.11.

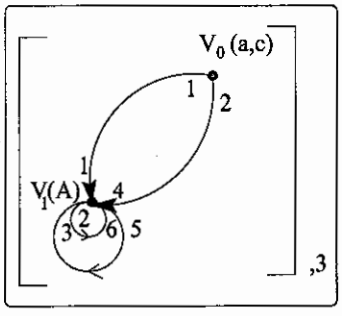

Figure 4.5: Representation of $3\mathbf{A}^{a\ ic}_{b\ \ i}$

For example, by lowering and raising the indices b and c in the tensor $3\mathbf{A}^{abi}{}_{cib}$, we create a new tensor which looks like $3\mathbf{A}^{a}{}_{b}{}^{ic}{}_{i}{}^{b}$. The difference appears by comparing Figure 5.1 to Figure 4.5.

4.4.5 Permutation of indices

In general, we can permute any given set of indices in a tensor. This permutation will either lead us to a new tensor or to the same one. By counting the number of permutations taken to order a given set of indices into a lexicographical order, the symbol of permutation, denoted by $\in$, can be either:

$$\in = \begin{cases} -1 & \textit{if number of permutation is odd,} \\ 0 & \textit{if two or more indices are equal, and} \\ +1 & \textit{if number of permutation is even.} \end{cases} \tag{4.2}$$

The permutation is simply changing the position of the indices in a tensor. Thus, if each index is represented as an edge in a graph, this operation can be simulated by changing the numbers on the representing edge. Since these numbers are not fixed, they can be changed depending on the new position of each index.

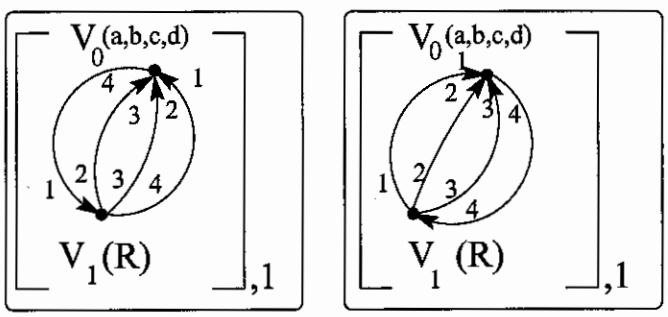

Figure 4.6: Representation of $\mathbf{R}^{d}{}_{cba}$ and $\mathbf{R}_{abc}{}^{d}$ respectively.

Table 4.2: Permuting the indices in a tensor **T**.

1.	$\in \leftarrow 0$
2.	$\mathbf{L}(\mathrm{I}) \leftarrow \mathbf{S}(\mathbf{L}(\mathrm{I}))$
3.	**if** $\mathbf{L}(\mathrm{I}).\mathrm{i} = \mathbf{L}(\mathrm{I}).\mathrm{j}$
	goto 9
4.	else
5.	**if** $\#\mathbf{S}(\mathbf{L}(\mathrm{I})) = 2n$ **then**
6.	$\in \leftarrow 1$
7.	**else**
8.	$\in \leftarrow -1$
9.	return $\in$

Thus, the method of permuting the indices consists of creating a list of integers of the indices, based on their lexicographical order, and order that list. Note that, this list will be extracted later from the graph.

For example, let $\{k, j, i\}$ be a set of indices from the tensor $\mathbf{A}^{kji}$. Based on their lexicographical order, they shall be numbered as $I = (3, 2, 1)$. Thus, ordering I will make $\in_{ijk} = -1$ since it takes 3 rounds to make it look like: $(1, 2, 3)$.

The importance of this operation appears when we define the symmetries for tensors. For example, let $\mathbf{R}^{d}{}_{cba}$ be a Riemann tensor. By permuting the four indices and by following exactly the steps from Table 4.2, $\mathbf{R}^{d}{}_{cba}$ will look like $\mathbf{R}_{abc}{}^{d}$ as shown in **Figure 4.6**.

4.4.6 Symmetries

Symmetries with respect to index permutations means that a tensor $\mathbf{T}$ obeys one or more equations of the kind:

$$\mathbf{T}^{i_1 \ \ldots \ i_n} = \in T^{\pi(i_1 \ \ldots \ i_n)}, \tag{4.3}$$

where $\pi(i_1 \ ... \ i_n)$ is some permutation of $i_1 \ ... \ i_n$ and

$$\in = \begin{cases} -1 & if \ \mathbf{T} \ is \ antisymmetric \ and \ \#\pi(i_1 \ ... \ i_n) \ is \ odd, \\ 0 & if \ \mathbf{T} \ is \ null, \\ +1 & elsewhere. \end{cases} \tag{4.4}$$

Since symmetries are defined with respect to the permutations of indices, it can be formulated in a similar way as in Section 4.4.5. That means, changing the integers on the edges of the graph and the sign of the tensor if necessary.

Meanwhile, there exist 4 different types of symmetries for any given set of indices in a tensor. They can be either: non-symmetric, symmetric, antisymmetric or mixed of any of the previous ones. If any of the first three cases occurs, the problem can be solved by copying the desired list of integers and permute them into a lexicographical order. Depending on the number of permutation of indices, $\in$ can be either -1, 0 or 1.

If the indices of a tensor have more than one type of symmetries, then we separate the indices into index type groups n. Thus, each index group will be added to a tensor name and then multiply all of the new tensors using the inner product. This implies that the problem can be categorized into smaller parts and solved based on one of the previous cases. The full description of each type and this method can be found later in Chapter 8.

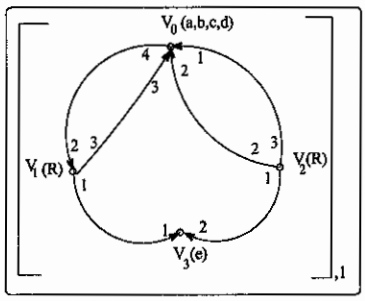

Figure 4.7: Representation of $e^{ij}\mathbf{R}_i{}^d{}_c\mathbf{R}_{jba}$.

For example, let $\mathbf{R}^d{}_{cba}$ be a Riemann tensor such that $\mathbf{R}^d{}_{cba}$ is an antisymmetric tensor with respect to the first or the last pair indices and symmetric with respect to both sets. That means, $\mathbf{R}^d{}_{cba} = -\mathbf{R}_c{}^d{}_{ba}$, $\mathbf{R}^d{}_{cba} = -\mathbf{R}^d{}_{cab}$ and $\mathbf{R}^d{}_{cba} = \mathbf{R}_{ba}{}^d{}_c$.

Since this tensor is a mixed one, the indices are separated into 3 groups: $I_1 = \{d, c\}$, $I_2 = \{b, a\}$ and $I_3 = \{I_1, I_2\}$. Thus, the Riemann tensor $\mathbf{R}^d{}_{cba}$ will be equivalent to $e^{ij}\mathbf{R}_i{}^d{}_c\mathbf{R}_{jba}$ such that e^{ij} is a symmetric tensor and the rest are antisymmetric tensors as shown in Figure 4.7.

Permuting the indices of each vertex separately will modify $e^{ij}\mathbf{R}_i{}^d{}_c\mathbf{R}_{jba}$ to look like: $e^{ij}\mathbf{R}_{abj}\mathbf{R}_c{}^d{}_i$. Converting back will give us: $\mathbf{R}_{abc}{}^d$.

Chapter 5

Canonical Labeling of Graphs

5.1 Introduction

Geometrically, we define a graph to be a set of non-empty points in space, called vertices denoted by **V** interconnected by a set of lines of two-element subsets of **V** called edges denoted by **E**. The structure of a graph, such as drawings, incidence tables, and concrete descriptions of the abstract structure, is what characterizes a graph itself and makes it independent of its representation. Meanwhile, the structure of both vertices and edges can always be modified by adding or eliminating some of their components [Gib85].

The purpose of this chapter is to formulate the components of a tensor, the name, the indices, their types and their positions, as parts of the components of the corresponding graph by presenting:

1. an algorithm to label the vertices,
2. another algorithm to label the edges, and
3. applying the symmetries.

Table 5.1: Algorithm of Vertex Labeling.

1. $\mathbf{G(T)} \leftarrow \mathbf{V}_0()$
2. $\mathbf{V}_0() \leftarrow \mathbf{FL(T)}$
3. $\mathbf{V}_1() \leftarrow \mathbf{NM(T)}$
4. $\mathbf{G(T)} \leftarrow \mathbf{V}_1(\mathbf{NM(T)})$
5. return $\mathbf{G}$

5.2 Algorithm of Vertex Labeling

The first basic element of a graph is the vertices. Each vertex in the graph is represented as a point such that distinct vertices are represented by distinct points. Note that the set of vertices need always to be labeled. Since the name of a tensor and the name of the free indices differentiate a tensor from another, they will both be reserved to be included in the representation of different vertices.

Upon the creation of each graph, we first include a special vertex $\mathbf{V}_0()$. This vertex, which is included in every graph, represent the set of free indices. Next, we create the second vertex $\mathbf{V}_1()$ in the graph by including the name of the tensor in it. The implementation of each vertex consists of a single integer followed by list of strings. This algorithm is described in Table 5.1 and the implementation is fully described later in Chapter 9.

For example, let $\mathbf{T}_1 = 3\mathbf{A}_{abi}{}^{cib}$ be a tensor. The graph of $\mathbf{T}_1$ will first contain the vertex $\mathbf{V}_0()$. Next, we include the list of free indices of $\mathbf{T}_1$ into $\mathbf{V}_0()$. Then, we increase the number of vertices in the graph by adding the vertex $\mathbf{V}_1()$ and glue to it the name of the tensor $\mathbf{A}$. Figure 5.1 shows both vertices $\mathbf{V}_0(a, c)$ and $\mathbf{V}_1(\mathbf{A})$ of the corresponding graph of the tensor $3\mathbf{A}_{abi}{}^{cib}$.

5.3 Algorithm of Edge Labeling

The diagrammatic representation of a graph helps us to visualize many concepts relating to graphs and to the systems of which they are models. Each edge is represented by a simple arc joining two, not necessarily distinct, vertices. In a diagrammatic representation of a graph, it is possible that two edges intersect at a point that is not necessarily a vertex of the graph. This section takes into consideration the description of the algorithm to create and to label the edges in a graph depending on the indices, their types and their positions.

Each index is represented as an edge in the graph. The position is needed to separate an index from another. If an index i is free, we state the position of i in the list of free indices beside $\mathbf{V}_0$ and the position of i in the tensor beside its vertex. The edge will be directed depending on the type of the index. The priority is given to the subscript index over the superscript one. This method helps us to separate similar tensors such as A_{ab} and A_{ba} or A_a and A^a.

Each edge is represented as: $e = [\mathbf{V}, i, \mathbf{V}', j]$ such that $\mathbf{V}$ and $\mathbf{V}'$ represent the head and the tail of e respectively while i and j represent the position of the index in $\mathbf{V}$ and $\mathbf{V}'$ respectively. The full description of this algorithm is mentioned in Table 5.2 and the implementation of this method is described later in Section 9.2.2.

For example, let $\mathbf{T}_2 = 3\mathbf{A}_{abi}{}^{cib}$ be a tensor. $\mathbf{T}_2$ has 6 indices such that 2 are free and 4 or 2 pairs are dummy indices. Thus, $\mathbf{FL}(\mathbf{T}_2) = \{a, c\}$ and $\mathbf{DL}(\mathbf{T}_2) = \{d, i\}$. As shown in Section 5.2, the graph will contain 2 different vertices, $\mathbf{V}_0(a, c)$ and $\mathbf{V}_1(A)$.

The first index a, which is a subscript index, is part of $\mathbf{FL}(\mathbf{T}_2)$. Thus, this index will be represented as an edge going from $\mathbf{V}_1(A)$ to $\mathbf{V}_0(a, c)$. The integers on that edge are: the position of a in $\mathbf{T}_2 = 1$ and the position of a in $\mathbf{FL}(\mathbf{T}_2) = 1$. Therefore, the representation of the edge is: $[\mathbf{V}_1(\mathbf{A}), 1, \mathbf{V}_0(a, c), 1]$.

Table 5.2: Algorithm of Edge Labeling.

```
1.   j ← 0
2.   for i in IL(T) do
       begin
3.     j ← j + 1
4.     if SI(i) ∈ FL(T) then
         begin
5.       if _i? then
6.         addedge(V1, j, V0, Pos(i, FL(T)))
7.       else
8.         addedge(V0, Pos(i, FL(T)), V1, j)
         end
9.     else
        begin
10.      ifnot _i? then
           begin
11.        if _SI(i) ∈ IL(T) then
12.          addedge(V1, j, V1, Pos(_SI(i), IL(T)))
13.        else go to 16
14.      else
15.        if ^SI(i) ∈ IL(T) then
16.          addedge(V1, Pos(_SI(i), IL(T)), V1, j)
17.        else go to 16
           end
         end
18.      IL(T) ← IL(T) - IL(T).1
       end
19.  return G
```

In contrast, the second index does not belong to $\mathbf{FL}(\mathbf{T}_2)$. Indeed, the superscript index b is the first element of $\mathbf{DL}(\mathbf{T}_2)$ and since it is a dummy index, it will be represented as a loop-edge going from $\mathbf{V}_1(\mathbf{A})$ to itself. The position of both b's in $\mathbf{T}_2$ are 2 and 6. Thus, the representation of the second edge is: $[\mathbf{V}_1(\mathbf{A}), 2, \mathbf{V}_1(\mathbf{A}), 6]$. Meanwhile, the rest of the indices in $3\mathbf{A}_{abi}{}^{cib}$ c and i are represented as edges in the graph shown in Figure 5.1.

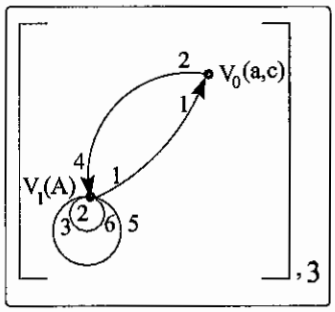

Figure 5.1: Representation of $3\mathbf{A}_{abi}{}^{cib}$

One of the advantages of this structure is the simplicity of comparing any two similar tensors or any two similar tensor monomials, as in (5.1), by checking only the edges of the graphs. For example:

$$\begin{cases} T^a{}_{ab} \text{ and } T^a{}_{ba}, \\ T^a{}_{ab}{}^b \text{ and } T^a{}_{ba}{}^b, \\ V^a V_a \text{ and } V_a V^a, \\ F^a{}_b F_a{}^b \text{ and } F^a{}_a F_b{}^b, \\ F^a{}_{bi} F_a{}^b \text{ and } F^a{}_{ai} F_b{}^b. \end{cases} \tag{5.1}$$

The algorithm for edge labeling, which is fully described in Table 5.2, is implemented to run in $O(n)$-time where n is the number of indices in a tensor. The complexity is dominated by the *for* statement that appears in line 2.

5.4 Symmetries

A tensor is said to be *symmetric* in any number of its indices, subscript and/or superscript, if it remains unchanged by any permutation of indices. Thus, if $\mathbf{A}^r{}_{kl} = \mathbf{A}^r{}_{lk}$, then $\mathbf{A}^r{}_{kl}$ is symmetric in its subscript indices and $\mathbf{A}^r{}_{kl}$ is symmetric in its superscript indices if $\mathbf{A}^r{}_{kl} = \mathbf{A}_k{}^r{}_l$ [JF97].

Symmetry, as explained in Section 2.4.6, is similar to the permutation of indices. That means, changing the position of the indices in a tensor. Thus, symmetry represents changing integers on the edges of a graph.

The importance of symmetries appears by simplifying a tensor expression after applying tensor algebra. It happens that tensors look sometimes different when in reality they are completely similar. It is a similar case as graph isomorphism in Section 3.3.

After labeling the edges in a graph, we need to identify the type of symmetry of each tensor. A set of indices is either:

1. **non-symmetric**,
2. totally symmetric or simply **symmetric**,
3. totally antisymmetric or simply **antisymmetric**, or
4. **mixed** of any of the above.

In the first case, the indices in a non-symmetric tensor can't be permuted at all. That means, the tensor should remains unchanged and the graph remains unchanged.

For other cases, we first order the edges based on the integers beside $\mathbf{V}_0$. Now, if all the indices are symmetric, we create a desired list of integers, denoted by **LSI**, from all

the edges beside V_1. Thus, sorting **LSI**, relabeling the edges in the graph based on **LSI**, and reordering the indices in the tensor will complete the job.

Similarly to the symmetric case, we proceed in the antisymmetric case. The only difference occurs by counting the number of permutation to order **LSI**. If it is an odd number, we change the coefficient of the tensor to be a negative one. Otherwise, we proceed as similar to the symmetric case.

For example, let $\mathbf{A}_{jik}$ and $\mathbf{A}_{kij}$ be a symmetric and antisymmetric tensors respectively. From Figure 5.2, the list of integers for the first graph is $LSI_1 = (2, 1, 3)$ while the list of integers for the second graph is $LSI_2 = (3, 1, 2)$. Note that, both lists represent the same indices but in different order. Thus, by sorting LSI_1 and LSI_2 and modifying the edges in both graphs, we create a new graph which represent the tensor $\mathbf{A}_{ijk}$ as shown in Figure 5.2.

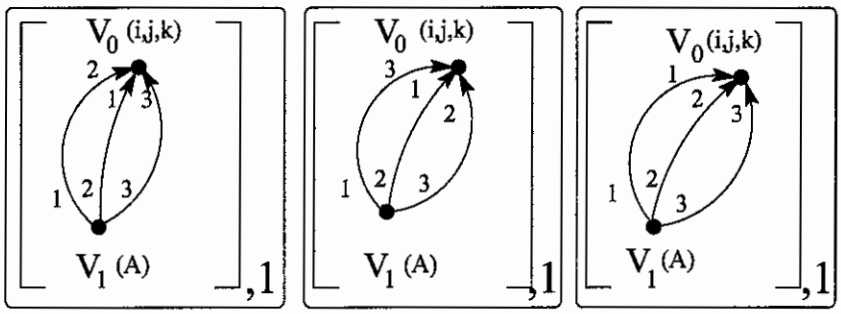

Figure 5.2: Representation of $\mathbf{A}_{jik}$, $\mathbf{A}_{kij}$ and $\mathbf{A}_{ijk}$

Finally, in the case of mixed type of indices, we split them into different groups such that each group contains only one specific type. Then, we create new vertices in the graph equal to the number of index groups. For each vertex, we create set of edges which represent one group of indices. This does not elegantly handle the most general case of arbitrary symmetries, but it is sufficient for tensors which appears in most applications.

Since each set of indices represent one and only one type of symmetries, we can proceed similar to any of the previous method, non-symmetric, symmetric or antisymmetric, to reorganize them. If non-symmetry appears, we keep **LSI** unchanged without any modification.

For example, let $\mathbf{R}_{cbd}{}^{a}$ be a **Riemann tensor** such that $I = \{c, b, d, a\}$. This tensor is antisymmetric with respect to $I_1 = \{c, b\}$ and $I_2 = \{d, a\}$ and symmetric with respect to $I_3 = \{I_1, I_2\}$. Thus, by splitting both groups I_1 and I_2:

$$\mathbf{R}_{cbd}{}^{a} = e^{ij}\mathbf{R}_{cbi}\mathbf{R}_{d}{}^{a}{}_{j} \tag{5.2}$$

such that $\mathbf{R}_{cbi}$ and $\mathbf{R}_{d}{}^{a}{}_{j}$ are antisymmetric tensors and e^{ij} is a symmetric tensor.

Next, the lists of integers are: $\mathbf{LI}(I) = (3, 2, 4, 1)$, $\mathbf{LI}(I_1) = (3, 2)$, and $\mathbf{LI}(I_2) = (4, 1)$. By sorting $\mathbf{LI}(I_1)$ and $\mathbf{LI}(I_2)$, both lists will look like $(2, 3)$ and $(1, 4)$ respectively. Thus, $\mathbf{R}_{cbi} = -\mathbf{R}_{bci}$ and $\mathbf{R}_{d}{}^{a}{}_{j} = -\mathbf{R}^{a}{}_{dj}$. Since e^{ij} is a symmetric tensor and since the first integer in $\mathbf{LI}(I_1) = 1$ is smaller to the first integer in $\mathbf{LI}(I_2) = 2$, we can switch the places of the tensors in equation (5.2). Thus, $\mathbf{R}_{cbd}{}^{a} = e^{ij}\mathbf{R}^{a}{}_{dj}\mathbf{R}_{bci} = \mathbf{R}^{a}{}_{dbc}$.

The full description of each type, non-symmetry, symmetry, antisymmetry and mixed symmetry, is described later in Chapter 8.

Chapter 6

Algebraic Simplification of Tensors

6.1 Introduction

Since each graph consists of vertices and edges, two graphs are equal if and only if their vertices and their edges are equal. This equality is insufficient to compare two tensor monomials since we may be able to relabel their graphs to make them equal. We therefore need to be able to determine whether two graphs are isomorphic or not. Thus, a good strategy is needed to compare and to simplify two different graphs. That means finding a single unified general form for every graph.

The purpose of this chapter is to present a method to be used for relabeling the vertices and the edges. Then, an equality test on the relabeled graph gives a test for isomorphism. Thus, we need to:

1. relabel the identical vertices in a graph using the method of **canonical relabeling**,
2. reshuffle the free indices of a tensor into a lexicographical order using **tensor canonicalization**, and
3. relabel all the edges, **EF**s and **ED**s, in a graph.

6.2 Canonical Relabeling of Vertices

Two different graph specifications are often alternative descriptions for structurally equivalent graphs. Developing a universally applicable method for deciding the structural equivalence is called the Graph Isomorphism Problem. Some strategies and tests work well in many instances, but a practical method has not yet been developed to handle all cases [GY94]. In contrast, the canonical relabeling can always be used to decide whether two graphs G_1 and G_2 are isomorphic or not.

Beyond pictures and index tables, possible descriptions of graphs include various kinds of different matrices. It is not surprising that some forms of matrix representations were introduced for graphs.

The adjacency matrix, as defined in Definition 3.23, is the classical matrix representation for graphs, which allows us to establish certain properties of the graph using matrix-theoretic methods. It does serve us as a good start towards developing more efficient data structures.

In canonical relabeling, the adjacency matrix can be used to find the best permutation for identical vertices in a graph as mentioned in Definition 6.4. For each graph, we can always define an adjacency matrix of size $n \times n$ where n is the number of vertices in a graph. The cost of creating such a matrix is $O(n^2)$. Note that the adjacency matrix contains the number of edges going from a vertex to another.

Definition 6.1 *Two vertices are identical if and only if they have the same degree, the same number of **EF**s and the same name of the tensor.*

Remark 6.1 *By joining the identical vertices in separate lists, we can always create a list of identical vertices denoted by **LIV**.*

Remark 6.2 *Let $\boldsymbol{A}_G$ be the adjacency matrix of a graph G. Since every permutation gives a new relabeled graph based on the original one, we can create an adjacency matrix for each relabeled graph.*

Remark 6.3 *Let $\boldsymbol{A}_G$ be an adjacency matrix of a graph G of size $n \times n$ and let **LSI** be a matrix of size $1 \times n^2$ where n is the number of vertices. $\boldsymbol{A}_G$ can always be transferred to a list of single integers **LSI** by copying the rows of $\boldsymbol{A}_G$ into **LSI**.*

Definition 6.2 *Let **LSI** be a list of single integers such that $\boldsymbol{LSI} = [r_1, r_2, \ ..., r_{n^2}]$ where n is the number of vertices in a graph. Thus, we define the base of **LSI** c such that $c = (max\ r_i) + 1$, $i = 1 \ldots n^2$.*

Definition 6.3 *Let **LSI** be a list of single integers and let c be its base. We define **Measure** as a single integer such that*

$$\boldsymbol{Measure} = \sum_{i=1}^{n^2} \mid r_i \mid \times c^i \tag{6.1}$$

*The cost of finding **Measure** is $O(n^2)$.*

Definition 6.4 *The best permutation of all vertices in a graph G, or the canonical representative graph, can be chosen by taking into consideration the biggest integer calculated from each **Measure** of all **LSI**. The complexity of determining the canonical representative graph in here is $O(n^2k!)$ where k is the size of the largest vertex class.*

Example 6.1 *Let $T = \boldsymbol{A}_{ij}{}^{k} \boldsymbol{A}_{abk} \boldsymbol{V}^{nm}$ be a tensor and let G be its corresponding graph as shown in Figure 6.1. By checking the degree of each tensor, the number of free indices in each tensor and the name of the tensors, the vertices can be separated into 2 lists: L_1 and L_2 such as $L_1 = \{\boldsymbol{V}_1(A), \boldsymbol{V}_2(A)\}$, where both elements have degree $= 3$, and $L_2 = \{\boldsymbol{V}_3(V)\}$ which has degree $= 2$.*

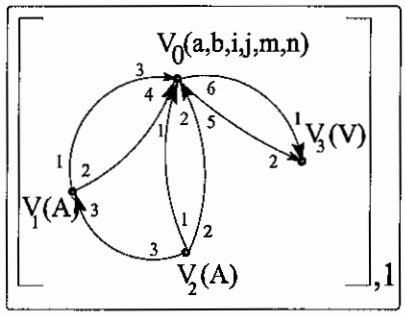

Figure 6.1: The representation of $\mathbf{A}_{ij}{}^{k}\mathbf{A}_{abk}\mathbf{V}^{nm}$

Since L_2 has only one element and since the permutation of one element is the element itself, we keep $\boldsymbol{V}_3(\boldsymbol{V})$ unchanged. Therefore, we eliminate L_2. Meanwhile, the number of permutations of L_1 is $2! = 2$. That means, G can be either itself or a new one G_1 in which $\boldsymbol{V}_1(\boldsymbol{A})$ is interchanged with $\boldsymbol{V}_2(\boldsymbol{A})$ and vice-versa. In both graphs G and G_1, the adjacency matrices are represented as in Figure 6.2 respectively.

	V_0	V_1	V_2	V_3
V_0	0	0	0	2
V_1	2	0	0	0
V_2	2	1	0	0
V_3	0	0	0	0

	V_0	V_1	V_2	V_3
V_0	0	0	0	2
V_1	2	0	1	0
V_2	2	0	0	0
V_3	0	0	0	0

Figure 6.2: The adjacency matrices for G and G_1 respectively

For the graph G, the corresponding list $\boldsymbol{LSI}(G) = [0,0,0,2,2,0,0,0,2,1,0,0,0,0,0,0]$ where the base $c = 2+1 = 3$. By using formula 6.1, $\boldsymbol{Measure(G)} = \sum_{i=1}^{16} r_i * 3^{(16-i)} =$ 5103. Similarly, $\boldsymbol{LSI}(G_1) = [0,0,0,2,2,0,1,0,2,0,0,0,0,0,0,0]$ where $c = 3$. Meanwhile, $\boldsymbol{Measure(G_1)} = \sum_{i=1}^{16} r_i * 3^{(16-i)} = 21870$. Since the $\boldsymbol{Measure(G_1)}$ is bigger than the one of G, we consider G_1 to be the model graph.

6.3 Relabeling the Edges

After relabeling canonically the vertices, we need a good strategy to relabel the edges in a consistent way to make the isomorphic graphs comparable. The order of the indices, free or dummy, will not be the same as the original order. Thus, relabeling the edges in the graph and then reordering the indices in its corresponding tensor will complete the job.

Depending on the type of symmetries in a tensor, the indices will reshuffle into a different position such that the free indices will have the priority over the dummy ones. This kind of changement can be modified by relabeling the integers on the edges.

Definition 6.5 *We define* ***LIF*** *to be the list of single integers from all* ***EF****'s in the graph beside* $\boldsymbol{V}_i$, $i = 1, \ldots, |\boldsymbol{V(G)}| - 1$, *such that the priority is given to the integer which represent the smallest in the lexicographical order.*

Definition 6.6 *We define* ***LID*** *to be the list of single integers from all* ***ED****'s in the graph beside* $\boldsymbol{V}_i$, $i = 1, \ldots, (|\boldsymbol{V(G)}| - 1)$. *The priority is given to the edges going to* $\boldsymbol{V}_j$ *followed by the edges coming from* $\boldsymbol{V}_j$, $j = 1, \ldots, |\boldsymbol{V(G)}| - 1$, *respectively.*

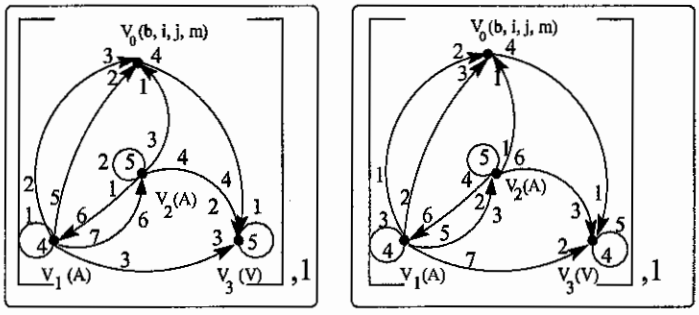

Figure 6.3: The graphs of $\mathbf{A}_{ljd\ i\ t}^{\ \ l\ k}\mathbf{A}_{kabn}^{\ \ \ \ at}\mathbf{V}_{c}^{\ ndmc}$ and $\mathbf{A}_{ijl\ t\ d}^{\ \ l\ k}\mathbf{A}_{bka\ \ n}^{\ \ at}\mathbf{V}^{mdn\ \ c}_{\ \ c}$.

Definition 6.7 *We define* ***LIG*** *to be the list of single integers from* ***LIF*** *followed by* ***LID****. That means,* ***LIG*** *= [****LIF****,* ***LID****].*

Remark 6.4 *Repeating the same procedure for all vertices of a given graph, all the edges will be relabeled in a consistant way and this will complete the job.*

Example 6.2 *Let* $\boldsymbol{A}_{ljd\ i\ t}^{\ l\ k}\boldsymbol{A}_{kabn}^{\ \ \ \ at}\boldsymbol{V}_{c}^{\ ndmc}$ *be a tensor monomial such that every tensor is a symmetric one and let G be its graph as shown in Figure 6.3.*

First, we begin relabeling the edges for $\boldsymbol{V}_1$*. The list of single integers for* ***EF****s is* ***LIF*** *= [5, 2] since 5 represent the index i and 2 represent the index j and since i is smaller than j in the lexicographical order. Meanwhile, the list of single integers for* ***ED****s is* ***LID*** *= [1, 4, 7, 6, 3] since 1 and 4 belong to the edge going from* $\boldsymbol{V}_1$ *to itself, 7 belongs to the edge going from* $\boldsymbol{V}_1$ *to* $\boldsymbol{V}_2$*, 6 belongs to the edge going from* $\boldsymbol{V}_2$ *to* $\boldsymbol{V}_1$*, and finally 3 belongs to the edge going from* $\boldsymbol{V}_1$ *to* $\boldsymbol{V}_3$*.*

Thus, ***LIG*** *= [5, 2, 1, 4, 7, 6, 3] since* ***LIG*** *is the combination of* ***LIF*** *followed by* ***LID****. Ordering* ***LIG*** *will give us [1, 2, 3, 4, 5, 6, 7]. By continuing like that for all the vertices and by modifying the edges in the graph* $\boldsymbol{G}$*, the result is a new graph as shown in Figure 6.3 which represent* $\boldsymbol{A}_{ijl\ t\ d}^{\ \ l\ \ k}\boldsymbol{A}_{bka}^{\ \ \ \ at}\boldsymbol{V}_{\ \ c}^{mdn\ \ c}$.

6.4 Tensor Canonicalization

The canonical function is used to reshuffle the indices of a tensor into lexicographical order. The lexicographical order of the indices plays an important role in the simplification process. The simplest case for ordering the set of indices of a tensorial object into a lexicographical order may be obtained by using the symmetric or the antisymmetric properties of the tensor.

Since the indices in each tensor are either free or dummy, we will assign higher priority to the free indices over the dummy ones and to the subscripts over the superscripts depending on the type of the symmetry of the tensor. Therefore, the free indices will be

shown at the beginning followed by the dummy ones. This can be obtained as a result of relabeling the edges in a graph as mentioned in the previous section.

The following examples clarify various aspects of the canonicalization function in this algorithm.

Example 6.3 *Let* $\boldsymbol{R}_{cabd}$ *and* $\boldsymbol{R}_{b\ ca}^{\ d}$ *be two Riemann tensors. By reshuffling the indices of both tensors, they will respectively look like:*

$$-\boldsymbol{R}_{acbd}, \qquad -\boldsymbol{R}_{acb}^{\ \ d}$$

The canonical function in the above examples goes through the symmetric and antisymmetric index groups and sort them.

Example 6.4 *Let* $\boldsymbol{T} = \boldsymbol{A}^{im\ \ jl}_{\ jik}$ *be a symmetric tensor. By reshuffling the indices of* $\boldsymbol{T}$*, it should look like:* $\boldsymbol{A}_{k\ ij}^{\ lm\ \ ij}$ *since the priority is for free indices over the dummy ones and for the subscription over the superscript.*

Example 6.5 *Let* $\boldsymbol{R}_{b\ ca}^{\ d}$ *be a Riemann tensor and let* $\boldsymbol{F}^{ca}$ *be a Maxwell tensor. From the previous example,* $\boldsymbol{R}_{b\ ca}^{\ d} = -\boldsymbol{R}_{acb}^{\ \ d}$ *and since Maxwell tensor is an antisymmetric tensor, then* $\boldsymbol{F}^{ca} = -\boldsymbol{F}^{ac}$*. The outer product of both tensors is* $\boldsymbol{R}_{acb}^{\ \ d}\boldsymbol{F}^{ac}$ *such that* a *and* c *are two contracted indices. This product will produce:*

$$\boldsymbol{R}_{acb}^{\ \ d}\boldsymbol{F}^{ac} = \boldsymbol{R}_{b\ ac}^{\ d}\boldsymbol{F}^{ac} = \boldsymbol{R}_{b\ \%1\%2}^{\ d}\boldsymbol{F}^{\%1\%2}$$

as a final result.

Example 6.6 *Finally, let* $\boldsymbol{R}^{a}_{\ bed;c}$ *be a Riemann tensor. By ordering to map the indices* a, b, d, e *and the derivative* c*, where* c *is a non-symmetric index, into the lexicographical order, the final result looks like:* $-\boldsymbol{R}^{a}_{\ bde;c}$.

Chapter 7

Basic Algebraic Operations on Tensor Expressions

7.1 Introduction

Higher rank tensors can be constructed from lower rank ones by forming the outer product. A tensor expression can be obtained from the summation of tensor monomials. Thus, two tensor expressions are equivalent if their terms are equivalent. Note that, the sum of any two tensor monomials is commutative.

Most of the operations in tensor algebra, such as contraction, raising and/or lowering indices, and symmetries, modify only the edges in a graph. Meanwhile, the multiplication modifies both the set of vertices and the set of edges in the graph. Since the addition of two tensors is the same as comparing two graphs, it effects neither the vertices or the edges of both of them.

One aspect of the structure in graph theory is the system of smaller graphs inside a graph, called subgraphs. Subgraphs arise explicitly or implicitly in almost any discussion of graphs. For instance, in the process of building a graph G from scratch, vertex by

vertex and edge by edge, the entire sequence of graphs formed along the way is a nested chain of subgraphs in G. This idea was the key to test and to proof the perfectness of the sum of two graphs which is formulated as the product of two tensors.

The basic operations of adding a vertex or an edge into a graph, together with the operations for deleting an edge or a vertex from a graph, provide a mechanism for transforming a graph, step by step, into another one. Thus, the idea of representing a tensor monomial as a graph consists to represent each tensor individually. This process will allows us to apply whatever we need later from tensor algebra such as addition, subtraction or multiplication. We prioritize the tensor operations of multiplication over addition or subtraction.

For example, let $Expr = 2(\mathbf{T}_{abc}\mathbf{V}^{ec}V^{b} - \mathbf{T}_{acb}\mathbf{V}^{eb}V^{c}) \cdot \mathbf{R}^{ad}$ be a tensor expression such that each tensor is a symmetric tensor. The operators in this expression are multiplication and substraction. The simplification is performed by applying the rules mentioned in Section 6.3 and Section 4.2. This means that to simplify $Expr$, we need first to multiply $\mathbf{R}^{ad}$ to each monomial in $Expr$, to apply the symmetries and finally to rename the dummy indices. This can be translate as:

$$\begin{aligned} Expr &= 2\mathbf{T}_{abc}\mathbf{V}^{ec}V^{b}\mathbf{R}^{ad} - 2\mathbf{T}_{acb}\mathbf{V}^{eb}V^{c}\mathbf{R}^{ad} \quad (multiplication) \\ &= 2\mathbf{T}_{cba}\mathbf{V}^{ec}V^{b}\mathbf{R}^{da} - 2\mathbf{T}_{bca}\mathbf{V}^{eb}V^{c}\mathbf{R}^{da} \quad (symmetries) \\ &= 2\mathbf{T}_{\%1\%2\%3}\mathbf{V}^{e\%1}V^{\%2}\mathbf{R}^{d\%3} - 2\mathbf{T}_{\%1\%2\%3}\mathbf{V}^{e\%1}V^{\%2}\mathbf{R}^{d\%3} \quad (renaming) \\ &= 0. \end{aligned} \tag{7.1}$$

Figure 7.1 shows explicitly the representation of $Expr$ as a tree and the steps needed to proceed and to find the solution. The purpose of this chapter is to develop several algebraic rules to manipulate tensors. The fundamental operations are addition and multiplication.

7.2 Graphical Pattern Matching Technique

As mentioned in Section 4.4.1, we can always add two tensors or tensor monomials by comparing their graphs. If the graphs are equal, we add the coefficients of both tensors and pass it to one of the graphs. Otherwise, we sort them, as mentioned in Remark 3.6, in a list of graphs and return it as an expression.

The question rises when we try to add two or more, equal or not, tensor expressions. One of the several techniques to proceed with this kind of addition is the simplification through graphical pattern matching technique.

The technique of graphical pattern matching consists of comparing two lists of sorted graphs such that each list is formulated of graphs representing the monomials of the expression. The output is one sorted list of graphs as described in Table 7.1.

This technique has been implemented to run in $O(m+n)$-time where m and n are the number of monomials in both expressions, and assuming the monomials are in canonical form. Thus, this algorithm tends to be more efficient since the implementation, in the worst case, can be run in $O(2n)$-time where n is the $max(m,n)$.

Meanwhile, the method used in most other algorithms consists of searching for similar monomials, then compare them and finally relabeling the dummy indices if comparison occurs.

For example, let A and B be two tensor expressions such that:

$$\begin{cases} \mathbf{A} = \mathbf{R}^{a\ c}_{\ b\ d} + \mathbf{U}^{c\ a}_{\ d} V_b + \mathbf{U}^{ac}{}_{bd}, \\ \mathbf{B} = -\mathbf{U}^{c\ a}_{\ d} V_b + 3\mathbf{R}^{a\ c}_{\ b\ d}. \end{cases} \tag{7.2}$$

Based on the graph of each monomial in expression $\mathbf{A}$, the graphs should be sorted

Table 7.1: Graphical Pattern Matching Technique for $\mathbf{L}_1$ and $\mathbf{L}_2$.

```
1.   L3 ← ∅
2.   while L1 ≠ ∅ or L2 ≠ ∅ do
       begin
3.     if L1.1 < L2.1 then
         begin
4.       L3 ← L1.1 + L3
5.       L1 ← L1 − L1.1
         end
6.     else
         begin
7.       if L1.1 > L2.1 then
           begin
8.         L3 ← L2.1 + L3
9.         L2 ← L2 − L2.1
           end
10.      else
           begin
11.        L3 ← L1.1 + L2.1 + L3
12.        L1 ← L1 − L1.1
13.        L2 ← L2 − L2.1
           end
         end
       end
14.  if L1 = ∅ then
15.    L3 ← L2
16.  else L3 ← L1
17.  return T
```

as: $\mathbf{LG}_1 = [\mathbf{R}^{a\ c}_{b\ d},\ \mathbf{U}^{ac}{}_{bd},\ \mathbf{U}^{c\ a}_{d}V_b]$ since:

1. the graph of the first two monomials contains only one vertex, and since
2. $\mathbf{E}(\mathbf{G}(\mathbf{R}^{a\ c}_{b\ d})) < \mathbf{E}(\mathbf{G}(\mathbf{U}^{ac}{}_{bd}))$ as fully described in Remark 3.5.

As similar to the first expression, the monomials in the expression $\mathbf{B}$ are sorted as: $\mathbf{LG}_2 = [3\mathbf{R}^{a\ c}_{b\ d}, -\mathbf{U}^{c\ a}_{d}V_b]$. By comparing $\mathbf{LG}_1$ to $\mathbf{LG}_2$ and by following the steps from Table 7.1, the final result is represented as: $\mathbf{LG} = [4\mathbf{R}^{a\ c}_{b\ d}, \mathbf{U}^{ac}{}_{bd}]$ which can be translated back as: $\mathbf{C} = \mathbf{A} + \mathbf{B} = 4\mathbf{R}^{a\ c}_{b\ d} + \mathbf{U}^{ac}{}_{bd}$.

7.3 Addition

As mentioned in Section 2.4.1 and Section 4.4.1, the addition or the subtraction of n tensors of rank k is again a tensor of the same rank. Note that, the summation of 2 tensors or 2 tensor expressions will proceed by comparing their graphs.

Proposition 7.1 *Let* $\boldsymbol{TE}_1 = a_1\boldsymbol{TM}_1 + ... + a_m\boldsymbol{TM}_m$ *and* $\boldsymbol{TE}_2 = b_1\boldsymbol{TM}_1 + ... + b_n\boldsymbol{TM}_n$ *be two tensor expressions. By sorting the graphs of both expressions into canonical forms, we can add* $\boldsymbol{TE}_1$ *to* $\boldsymbol{TE}_2$ *in* $O((m+n)c)$ *where* m *and* n *are the number of monomials in* $\boldsymbol{TE}_1$ *and in* $\boldsymbol{TE}_2$ *respectively and* c *is the cost of graph canonicalization.*

Proposition 7.2 *Let* $\boldsymbol{TE}_1$ *and* $\boldsymbol{TE}_2$ *be two tensor expressions as in Proposition 7.1. Using the method of graph isomorphism, the cost of adding* $\boldsymbol{TE}_1$ *to* $\boldsymbol{TE}_2$ *is* $O(mnI)$ *where* I *is the cost of a graph isomorphism and* m *and* n *are the number of monomials in* $\boldsymbol{TE}_1$ *and in* $\boldsymbol{TE}_2$ *respectively.*

Comparing Proposition 7.1 to Proposition 7.2, we can find out that it is more efficient to use the technique of graphical pattern matching of sorted graphs, mentioned in Table 7.1, compared to the graph isomorphism technique, assuming that $O(c) = O(I)$.

While isomorphism tests or canonicalization could dominate this analysis, we note that the number of vertices in the graph is usually limited, but the number of terms n and m may be very large. Therefore, we assume that I and C are in practice of constant cost, and we might to optimize the complexity in terms of n and m.

Example 7.1 *Let* $\boldsymbol{Expr}_1$ *and* $\boldsymbol{Expr}_2$ *be two tensor expressions such that all tensors are symmetric. Thus,*

$$\begin{cases} \boldsymbol{Expr}_1 = \boldsymbol{T}_{abc}\,\boldsymbol{V}^{ec}V^{b} + \boldsymbol{R}_{aj}{}^{je}, \\ \boldsymbol{Expr}_2 = A^{e}B_{a} - \boldsymbol{T}_{acb}\,\boldsymbol{V}^{eb}V^{c} - 2\boldsymbol{R}^{ei}{}_{ia}. \end{cases} \tag{7.3}$$

By checking the conditions of summation and by representing each term as a graph, we first apply the symmetries and then we rename the dummy indices in both expressions. Thus, (7.3) will become:

$$\begin{cases} \boldsymbol{Expr}_1 = \boldsymbol{T}_{a\%1\%2}\,\boldsymbol{V}^{e\%1}V^{\%2} + \boldsymbol{R}_{a}{}^{e}{}_{\%1}{}^{\%1}, \\ \boldsymbol{Expr}_2 = A^{e}B_{a} - \boldsymbol{T}_{a\%1\%2}\,\boldsymbol{V}^{e\%1}V^{\%2} - 2\boldsymbol{R}_{a}{}^{e}{}_{\%1}{}^{\%1}. \end{cases} \tag{7.4}$$

Using the technique of graphical pattern matching, we sort $\boldsymbol{Expr}_1$ *and* $\boldsymbol{Expr}_2$ *as 2 lists of graphs such that as:*

$$\begin{cases} \boldsymbol{L}(\boldsymbol{Expr}_1) = [\boldsymbol{R}_{a}{}^{e}{}_{\%1}{}^{\%1},\ \boldsymbol{T}_{a\%1\%2}\,\boldsymbol{V}^{e\%1}V^{\%2}], \\ \boldsymbol{L}(\boldsymbol{Expr}_2) = [-2\boldsymbol{R}_{a}{}^{e}{}_{\%1}{}^{\%1},\ A^{e}B_{a},\ -\boldsymbol{T}_{a\%1\%2}\,\boldsymbol{V}^{e\%1}V^{\%2}]. \end{cases} \tag{7.5}$$

Thus, Adding $\boldsymbol{Expr}_1$ *to* $\boldsymbol{Expr}_2$ *is the same as comparing the graphs of both lists one-by-one. This comparison produces:*

$$\boldsymbol{L}(\boldsymbol{Expr}_3) = [-\boldsymbol{R}_{a}{}^{e}{}_{\%1}{}^{\%1},\ A^{e}B_{a}], \tag{7.6}$$

which can be translated back to:

$$\boldsymbol{Expr}_3 = -\boldsymbol{R}_{a}{}^{e}{}_{\%1}{}^{\%1} + A^{e}B_{a}. \tag{7.7}$$

7.4 Product

As mentioned in Section 2.4.3, the multiplication of two tensors does not depend on their rank and they should not always have to be equal. In fact, we can always construct a new tensor from any two given tensors by taking their outer product while its rank will be the sum of their ranks. Thus, the procedure for multiplying two or more tensor expressions will be addressed as mentioned in Table 7.2.

Proposition 7.3 *Let* $\boldsymbol{TE}_1 = a_1\boldsymbol{TM}_1 + ... + a_m\boldsymbol{TM}_m$ *and* $\boldsymbol{TE}_2 = b_1\boldsymbol{TM}_1 + ... + b_n\boldsymbol{TM}_n$ *be two tensor expressions. Sorting the graphs into canonical forms, the required time to multiply* $\boldsymbol{TE}_1$ *to* $\boldsymbol{TE}_2$ *is* $O(mnC)$ *where* C *is the cost of canonicalizing tensor monomials and* m *and* n *are the number of monomials in* $\boldsymbol{TE}_1$ *and in* $\boldsymbol{TE}_2$ *respectively.*

Proposition 7.4 *Let* $\boldsymbol{TE}_1$ *and* $\boldsymbol{TE}_2$ *be two tensor expressions as in Proposition 7.3. Using graph isomorphism, the cost of multiplying* $\boldsymbol{TE}_1$ *to* $\boldsymbol{TE}_2$ *is* $O((mn)^2 I)$ *in the classical method where* I *is the cost of graph isomorphism and* m *and* n *are the number of monomials in* $\boldsymbol{TE}_1$ *and in* $\boldsymbol{TE}_2$ *respectively.*

Similar to the addition and by comparing both propositions, we conclude that it is more efficient to use the method in the first proposition compared to Proposition 7.4.

Remark 7.1 *Let* $\boldsymbol{TE} = a_1\boldsymbol{TM}_1 + ... + a_m\boldsymbol{TM}_m$ *be a tensor expression and let* λ *be a scalar. Thus,* $\lambda\boldsymbol{TE} = a_1\lambda\boldsymbol{TM}_1 + ... + a_m\lambda\boldsymbol{TM}_m$.

Remark 7.2 *Let* $\boldsymbol{TE} = a_1\boldsymbol{TM}_1 + ... + a_m\boldsymbol{TM}_m$ *be a tensor expression. Thus,* $(\boldsymbol{TE})^k$ *exists if and only if* k *is an even number.*

Example 7.2 *Let* $\boldsymbol{TE}_1 = \boldsymbol{T}^{ab} + \boldsymbol{U}^{ba}$ *be a tensor expression. Thus,*

1. $(\boldsymbol{TE}_1)^0 = 1$,

2. $(\boldsymbol{TE}_1)^1 = \boldsymbol{T}^{ab} + \boldsymbol{U}^{ba}$,

3. $(\boldsymbol{TE}_1)^2 = (\boldsymbol{T}^{ab} + \boldsymbol{U}^{ba})(\boldsymbol{T}^{ab} + \boldsymbol{U}^{ba}) = \boldsymbol{T}^{\%1\%2}\boldsymbol{T}_{\%1\%2} + 2\boldsymbol{T}^{\%1\%2}\boldsymbol{U}_{\%2\%1} + \boldsymbol{U}^{\%1\%2}\boldsymbol{U}_{\%1\%2}$,

4. $(\boldsymbol{TE}_1)^3$ = *error*,

5. $(\boldsymbol{TE}_1)^4 = ((\boldsymbol{TE}_1)^2)^2 = (\boldsymbol{T}^{\%1\%2}\boldsymbol{T}_{\%1\%2} + 2\boldsymbol{T}^{\%1\%2}\boldsymbol{U}_{\%2\%1} + \boldsymbol{U}^{\%1\%2}\boldsymbol{U}_{\%1\%2})^2$
   and so on.

Example 7.3 *Let* $\boldsymbol{Expr}_1$ *and* $\boldsymbol{Expr}_2$ *be two tensor expressions such that all tensors are symmetric. Thus,*

$$\begin{cases} \boldsymbol{Expr}_1 = \boldsymbol{T}_{abc}\boldsymbol{V}^{ec}V^b + \boldsymbol{R}_{aj}{}^{je} + A^eB_a, \\ \boldsymbol{Expr}_2 = A^aB_{el}. \end{cases} \tag{7.8}$$

By verifying the condition and by mapping each monomial to a graph, we first apply the symmetries and then we rename the dummy indices in both expressions to be like:

$$\begin{cases} \boldsymbol{Expr}_1 = \boldsymbol{T}_{a\%1\%2}\boldsymbol{V}^{e\%1}V^{\%2} + \boldsymbol{R}_{a\%1}{}^{e\%1} + A^eB_a, \\ \boldsymbol{Expr}_2 = A^aB_{el}. \end{cases} \tag{7.9}$$

Multiplying $\boldsymbol{Expr}_1$ *to* $\boldsymbol{Expr}_2$*, which is the same as adding every graph in* $\boldsymbol{Expr}_1$ *to every graph in* $\boldsymbol{Expr}_2$ *individually, the result should be:*

$$\boldsymbol{Expr}_3 = \boldsymbol{T}_{a\%1\%2}\boldsymbol{V}^{e\%1}V^{\%2}A^aB_{el} + \boldsymbol{R}_a{}^e{}_{\%1}{}^{\%1}A^aB_{el} + A^eB_aA^aB_{el}. \tag{7.10}$$

Again, by applying the symmetries and renaming the dummmy indices, 7.10 will be:

$$\boldsymbol{Expr}_3 = \boldsymbol{T}_{\%1\%2\%3}\boldsymbol{V}^{\%1\%4}V^{\%2}A^{\%3}B_{l\%4} + \boldsymbol{R}_{\%1\%2}{}^{\%1\%3}A^{\%2}B_{l\%3} + A^{\%1}B_{\%2}A^{\%2}B_{l\%1}. \tag{7.11}$$

Table 7.2: Multiplication of $\mathbf{TE}_1$ by $\mathbf{TE}_2$.

1.	$i \leftarrow 0$
2.	$j \leftarrow 0$
3.	$\mathbf{L}_1 \leftarrow \emptyset$
4.	$\mathbf{L}_2 \leftarrow \emptyset$
5.	$\mathbf{L}_3 \leftarrow \emptyset$
6.	**for** #TM in $\mathbf{TE}_1$ **do**
	begin
7.	$i \leftarrow i + 1$
8.	$\mathbf{L}_1.i \leftarrow \mathbf{G}(\mathbf{TE}_1.i)$
	end
9.	**for** #TM in $\mathbf{TE}_2$ **do**
	begin
10.	$j \leftarrow j + 1$
11.	$\mathbf{L}_2.j \leftarrow \mathbf{G}(\mathbf{TE}_2.j)$
	end
12.	$\mathbf{L}_1 \leftarrow \mathbf{S}(\mathbf{L}_1)$
13.	$\mathbf{L}_2 \leftarrow \mathbf{S}(\mathbf{L}_2)$
14.	$k \leftarrow i \times j$
14.	**for** k **do**
15.	$\mathbf{L}_3.k \leftarrow \mathbf{L}_1.i + \mathbf{L}_2.j$
16.	$\mathbf{TE}_3 \leftarrow \mathbf{L}_3$
17.	return $\mathbf{TE}_3$

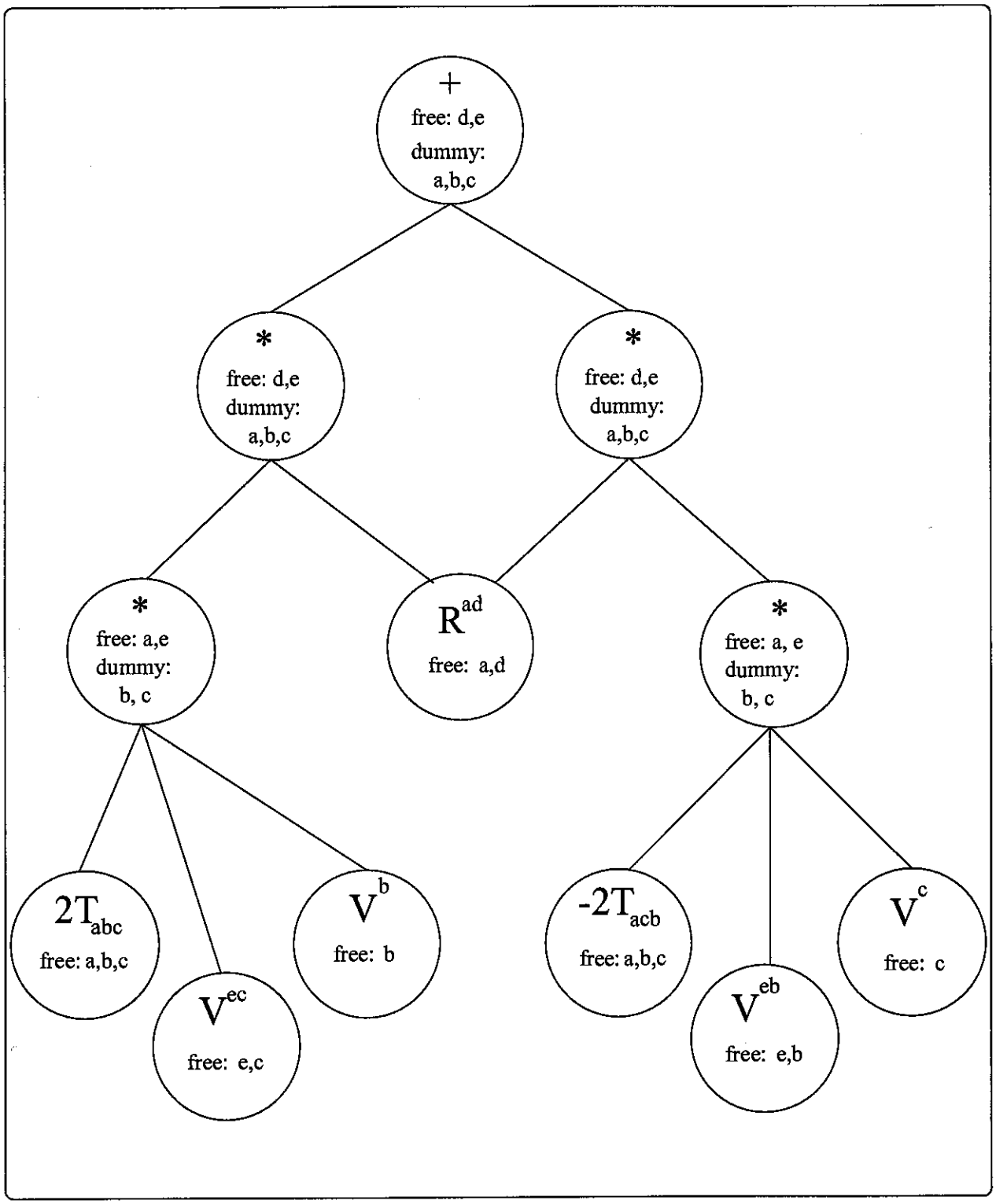

Figure 7.1: The representation of $2(\mathbf{T}_{abc}\mathbf{V}^{ec}V^{b} - \mathbf{T}_{acb}\mathbf{V}^{eb}V^{c}) \cdot \mathbf{R}^{ad}$ as a tree.

Chapter 8

Symmetries

8.1 Introduction

Some attempts have been made to describe algorithms for simplifying tensor expressions. In general, the treatment of symmetries in tensor expressions and the presence of dummy indices make simplification a complicated task.

The product of any two symmetric tenors is not usually a symmetric tensor monomial. For example, if $\mathbf{A}_{ij}$ and $\mathbf{B}_{kl}$ are symmetric tensors of rank 2, then $\mathbf{A}_{ij}\mathbf{B}_{kl}$ is not generally a symmetric tensor of rank 4. Indeed, $\mathbf{A}_{ij}\mathbf{B}_{kl} \neq \mathbf{A}_{ik}\mathbf{B}_{jl}$. Similarly, if $\mathbf{T}$ is a mixed tensor with n number of index groups, then the indices are not interchangeable within these groups. For example, $\mathbf{R}_{abcd} \neq \mathbf{R}_{acbd}$.

The importance of symmetries appears in the process of comparing two tensors. Thus, two tensors can have same indices while they can positioned differently. For example, let $\mathbf{A}_{jik}$ and $\mathbf{A}_{kji}$ be two symmetric tensors. By applying the symmetries, the difference of both tensors is zero. Indeed, by interchanging both sets of indices into lexicographical order, then $\mathbf{A}_{jik} = \mathbf{A}_{kji} = \mathbf{A}_{ijk}$. Initially, the user needs to identify the type of symmetry used in every tensor. Thus, a set of indices have one of these types:

1. non-symmetry,

2. symmetry,

3. antisymmetry, or

4. mixed of any of the above or mixed symmetries.

The purpose of this chapter is to introduce all types of symmetries and the necessary steps to treat each type. The implementation of symmetries is described in chapter 9.

8.2 Non-Symmetry

The user can always present a tensor $\mathbf{T}$ at which it can be neither a symmetric nor an antisymmetric tensor. This situation occurs by dropping the symmetry requirements, admitting that if $\mathbf{T}^{ij}$ and $\mathbf{T}^{i}{}_{j}$ are given tensors, then:

$$\begin{cases} \mathbf{T}^{ij} \neq \mathbf{T}^{ji}, & \mathbf{T}^{ij} \neq -\mathbf{T}^{ji}, \\ \mathbf{T}^{i}{}_{j} \neq \mathbf{T}_{j}{}^{i}, & \mathbf{T}^{i}{}_{j} \neq -\mathbf{T}^{i}{}_{j}. \end{cases} \tag{8.1}$$

Meanwhile, if the indices in a tensor are not permutable, the tensor remains unchanged. That means, we do not modify neither the edges nor the integers on the edges in the graph. Thus, these tensors are not useless. Indeed, we can add or multiply a non-symmetric tensor to a symmetric and/or an antisymmetric tensor.

For instance, the derivative in a given tensor is non-symmetric. Indeed, if the derivative is a free index, it counts as a non-symmetric index regardless OF type of symmetry of the tensor. Thus, if $\mathbf{R}_{abcd;e}$ is a Riemann tensor, then $\mathbf{R}_{abcd;e} \neq \mathbf{R}_{aecd;b} \neq \mathbf{R}_{abce;d}$.

8.3 Symmetry

Meanwhile, if altering any covariant and/or contravariant index in a tensor $\mathbf{T}$ does not alter the sign of the tensor, we define $\mathbf{T}$ to be a symmetric tensor with respect to its indices. Thus, symmetry change the position of the indices validating the process of comparing two tensors. This can happen by taking into consideration that free indices have priority over the dummy ones and subscript dummy indices are preferred to be positioned first compare to their superscript indices.

Reshuffling the indices in a symmetric tensor will change their positions without changing the sign of the tensor. That means, the integers on their corresponding edges beside the vertex need to be relabeled. This job can be accomplish by following the exact steps occur in Section 6.3 for relabeling the **EF**s and **ED**s in a graph. The full method of permuting the indices is addressed in Table 8.1.

Example 8.1 *Let* $\boldsymbol{A}^{a\ k}_{\ i\ ja}$ *and* $\boldsymbol{A}_{jia}{}^{ka}$ *be two symmetric tensors. By reshuffling the indices in both tensors, by following the steps in Table 8.1, they appear to be equal such that* $\boldsymbol{A}^{a\ k}_{\ i\ ja} = \boldsymbol{A}^{a\ k}_{\ i\ ja} = \boldsymbol{A}_{ij}{}^{k\ a}_{\ a}$.

Example 8.2 *Let*

$$\begin{cases} \boldsymbol{T}_1 &= \boldsymbol{A}^{a}_{\ ib}, \\ \boldsymbol{T}_2 &= \boldsymbol{B}^{b}, \\ \boldsymbol{T}_3 &= \boldsymbol{B}_{c}, \\ \boldsymbol{T}_4 &= \boldsymbol{A}^{c\ a}_{\ i}, \end{cases} \tag{8.2}$$

be 4 *different symmetric tensors. Thus, by applying the symmetry to each one, these tensors will be:* $\boldsymbol{A}^{a}_{\ bi}$, $\boldsymbol{B}^{b}$, $\boldsymbol{B}_{c}$ *and* $\boldsymbol{A}^{ac}_{\ \ i}$ *respectively. Meanwhile, the difference between* $\boldsymbol{T}_1\boldsymbol{T}_2$ *and* $\boldsymbol{T}_4\boldsymbol{T}_3$ *is zero since* $\boldsymbol{T}_1\boldsymbol{T}_2 = \boldsymbol{A}^{a}_{\ ib}\boldsymbol{B}^{b}$ *and* $\boldsymbol{T}_4\boldsymbol{T}_3 = \boldsymbol{A}^{c\ a}_{\ i}\boldsymbol{B}_{c} = \boldsymbol{A}^{a\ c}_{\ i}\boldsymbol{B}_{c}$

Note that, symmetry should be re-applied again whenever contraction occurs. In

Table 8.1: Applying non-symmetry, antisymmetry or symmetry to $\mathbf{T}$.

1.	$\mathbf{G} \leftarrow \mathbf{G}(\mathbf{T})$
2.	**if** $\mathbf{T}$ is non-symmetric **then**
3.	return $\mathbf{T}$
4.	$\mathbf{CT} \leftarrow coefficient(\mathbf{T})$
5.	$\mathbf{LI}(\mathbf{V}_i) \leftarrow$ List of integers beside $\mathbf{V}_i, i = 1...(\#\mathbf{V}(\mathbf{G}) - 1)$
6.	$\mathbf{LI}(\mathbf{V}_i) \leftarrow \mathbf{S}(\mathbf{LI}(\mathbf{V}_i))$
7.	$n \leftarrow \#\mathbf{S}(\mathbf{LI}(\mathbf{V}_i))$
8.	**if** $\mathbf{T}$ is antisymmetric & $i = 2n + 1$ **then**
9.	$\mathbf{CT} \leftarrow -\mathbf{CT}$
10.	$\mathbf{T} \leftarrow \mathbf{CT}(\mathbf{T})$
11.	$\mathbf{IL}(\mathbf{T}) \leftarrow \mathbf{S}(\mathbf{IL}(\mathbf{T}))$
12.	return $\mathbf{T}$

example 8.2, the inner product of $\mathbf{T}_1$ and $\mathbf{T}_2$ create a dummy index which forces us to re-apply the symmetries again.

8.4 Antisymmetry

The definition of an antisymmetric tensor $\mathbf{T}$ follows exactly from the definition of the symmetric one except that transposing any adjacent pair of indices in a tensor $\mathbf{T}$ modifies the sign of $\mathbf{T}$.

To reshuffle the indices in an antisymmetric tensor into a lexicographical order, we proceed in a similar way as mentioned in Section 8.3 by taking into consideration the number of transpositions k to accomplish the job. If k is an odd number, then the symbol

of permutation $\in = -1$. Otherwise, the sign of the tensor remains unchanged. The full description of this method is addressed in Table 8.1.

Example 8.3 *Let* $\boldsymbol{T} = \boldsymbol{A}_{aji}{}^{a}{}_{k}$ *be an antisymmetric tensor. Thus, the list of integers* $\boldsymbol{LI}$ *from the edges beside* $\boldsymbol{V}_1$ *of its graph is equal to* $(3, 2, 5, 1, 4)$ *and it takes* 5 *steps to order* $\boldsymbol{LI}$*. Thus, by proceeding in reshuffling the indices,* $\boldsymbol{T} = -\boldsymbol{A}_{ijka}{}^{a}$ *where* $\in = -1$.

8.5 Mixed Symmetries

In the previous sections, we introduced a method to reshuffle the indices having one type of symmetries. Since the user can always produce a tensor with more than one type of symmetries, we need to provide a method to be applied in this case too. The obvious example for a similar case is the Riemann tensor.

Let $\mathbf{T}$ be a mixed symmetric tensor. By dividing this problem into 4 separate parts, we can summarized as:

1. At the beginning, the indices will be split into n separate groups such that each group contains one and only one type of symmetries. This kind of splitting will be given by the user.

2. In the graph, we eliminate the vertex which represent the tensor $\mathbf{T}$ and create instead of that n new vertices where n is the number of separate index groups.

3. Each vertex will contain and will represent one index group. Thus, the edges will be relabeled as similar to Section 8.2, Section 8.3 and Section 8.4.

4. Finally, we join back all the edges of the new vertices within the original vertex and then reshuffle the indices in $\mathbf{T}$ based on its graph.

The full method of permuting the indices in a tensor with mixed type of symmetries is addressed in Table 8.2.

Table 8.2: Applying mixed symmetry to **T**.

1.	$\mathbf{G} \leftarrow \mathbf{G}(\mathbf{T})$
2.	$i \leftarrow 0$
3.	$\mathbf{LGI} \leftarrow$ list of index groups
4.	$\mathbf{CT} \leftarrow coefficient(\mathbf{T})$
5.	**for** $n \in \#\mathbf{LGI}$ **do**
	begin
6.	$\mathbf{G}_n \leftarrow \mathbf{V}(\mathbf{NM}(\mathbf{T}))$
7.	$\mathbf{G}_n \leftarrow \mathbf{E}(\mathbf{LGI}.n)$
8.	$\mathbf{E}(\mathbf{G}_n) \leftarrow relabel(\mathbf{E}(\mathbf{G}_n))$[1]
9.	$i \leftarrow \#\mathbf{S}(\mathbf{E}(\mathbf{G}_n))$
10.	**if** **T** is antisymmetric **then**
	begin
11.	**if** $i = 2n + 1$ **then**
12.	$\mathbf{CT} \leftarrow -\mathbf{CT}$
	end
	end
13.	$\mathbf{E}(\mathbf{G}) \leftarrow relabel(\mathbf{E}(\mathbf{G}))$[2]
14.	$\mathbf{IL}(\mathbf{T}) \leftarrow \mathbf{S}(\mathbf{IL}(\mathbf{T}))$
15.	return **T**

1.*As mentioned in Section* 6.3.

2.*Based on all* $\mathbf{G}_n$.

Example 8.4 *Let* $\boldsymbol{R}_{cbd}{}^{a}$ *be a* ***Riemann*** *tensor. This tensor is antisymmetric with respect to* $I_1 = \{c, b\}$ *and to* $I_2 = \{d, a\}$ *and it is symmetric with respect to* $I = I_1, I_2$*. Thus, by proceeding as similar to the example in Section 5.4,* $\boldsymbol{R}_{cbd}{}^{a} = \boldsymbol{R}^{a}{}_{dbc}$.

Example 8.5 *Let* $\boldsymbol{T} = \boldsymbol{A}_{lj}{}^{ki}{}_{;n}$ *be a tensor such that that* $\boldsymbol{T}$ *is antisymmetric with respect to* $I_1 = \{l, j\}$ *and to* $I_2 = \{k, i\}$*, symmetric with respect to* $I_3 = \{I_1, I_2\}$ *and non-symmetric with respect to* $I_4 = \{n\}$*. Therefore,* T *has* 4 *different index groups* I_1, I_2, I_3, I_4*. By following the method addressed in Table 8.2,* $\boldsymbol{T}$ *will be:*

$$\boldsymbol{A}_{lj}{}^{ki}{}_{;n} = e^{ab}\boldsymbol{A}_{ljа}\boldsymbol{A}^{ki}{}_{b}\boldsymbol{A}_{;n} = \boldsymbol{A}^{ik}{}_{jl;n} \tag{8.3}$$

Table 8.3: The algorithm for simplifying a monomial **TM**.

Step1:		apply tensor algebra
Step2:		$\mathbf{G} \leftarrow \mathbf{TM}$[1]
Step3:		apply symmetries[2]
Step4:		canonical labeling of $\mathbf{V}(\mathbf{G})$[3] based on:
	i.	# **EF**s of each $\mathbf{V} \in \mathbf{V}(\mathbf{G})$,
	ii.	degree of each $\mathbf{V} \in \mathbf{V}(\mathbf{G})$, and
	iii.	name of each $\mathbf{T} \in \mathbf{TM}$
Step5:		canonical relabeling of edges:
	i.	renumber **EF**s according to symmetries[4]
	ii.	renumber **ED**s according to destinations[4,5]
Step6:		$\mathbf{TM} \leftarrow \mathbf{G}$

1. *By referring to Chapter* 5.

2. *Following the steps in Table* 8.1 *or Table* 8.2.

3. *As mentioned in Section* 6.2.

4. *Based on Section* 6.3.

5. *Lower dummy index has priority over upper dummy one.*

Chapter 9

Aldor Implementation

9.1 Introduction to Aldor

The first question which comes into mind is: what is *Aldor* and why *Aldor*? We can say that the original motivation for *Aldor* came from the field of computer algebra: to provide an improved extended language for *computer algebra system AXIOM*. The following development in this section is fully summarized from [SMW94].

In general, Aldor is a programming language that attempts to achieve power through uniformity. Rather than building a language by adding features, they tried instead to build a language by removing restrictions.

While the design of Aldor emphasizes generality and composibility, it also emphasizes efficiency. Usually these objectives seem to pull in different directions. An achievement of Aldor's implementation is its ability to attain both simultaneously.

The desire to model the extremely rich relationships among mathematical structures has driven the design of Aldor into a somewhat different direction than that of other contemporary programming languages. Aldor places more emphasis on uniform handling of functions and types, and less emphasis on a particular object model. This provides the

flexibility necessary to solve the original problem, and has already proved of significant use outside of this initial context.

Aldor is unusual among compiled programming languages, in that types and functions are *first class*: that is, both types and functions may be constructed dynamically and manipulated in the same way as any other value. This provides a natural foundation for both object-oriented and functional programming styles, and leads to programs in which independently developed components may be combined in quite powerful ways.

9.2 Implementation of graphs

Since tensors need to be represented as graphs which consist of two basic elements vertices and edges, it is important to show and to present the method of implementation of each of them. Thus, the representation of each graph will be shown formally and not as points and segments. For example, the tensor $-6\mathbf{A}^{abi}{}_{cib}$ represented as:

$$\mathbf{G}[[\mathbf{V}_0(a,c),1,\mathbf{V}_1(\mathbf{A}),1],[\mathbf{V}_1(\mathbf{A}),4,\mathbf{V}_0(a,c),2],[\mathbf{V}_1(\mathbf{A}),2,\mathbf{V}_1(\mathbf{A}),6],[\mathbf{V}_1(\mathbf{A}),3,\mathbf{V}_1(\mathbf{A}),5]]$$

which is similar to Figure 4.3 where $\mathbf{V}_0(a,c)$ and $\mathbf{V}_1(\mathbf{A})$ are vertices and each factor of $\mathbf{G}$ is an edge.

9.2.1 Vertices

In every graph, each vertex is represented as a single integer followed by a list of strings.

1. The use of the single integer is important in relabeling the vertices. That means in separating and in identifying a vertex from another. Meanwhile, each vertex is mapped to its subscript integer. For instance, the initial vertex $\mathbf{V}_0$ is equivalent to the integer zero, the vertex $\mathbf{V}_1$ is equivalent to the integer 1, and in general the vertex $\mathbf{V}_n$ is equivalent to the integer n.

2. Since the name of a given tensor and its list of free indices differentiate a tensor from another, the list of strings is implemented in two separate cases:

   (a) the list of free indices in a tensor is added to the initial vertex $\mathbf{V}_0$ to make it look like $\mathbf{V}_0()$, and

   (b) the next vertex, $\mathbf{V}_0$, is modified by adding the name of the given tensor.

The structure of the vertices in this method helps us to separate and to label each vertex in a graph as mentioned in Definition 3.2. Thus, it helps us to proof the uniqueness of each vertex in a graph.

9.2.2 Edges

Since the edges play an important role to differentiate two graphs having the same number of vertices, we will choose the most readable method to represent the edges and show their uniqueness. Thus, each edge represented as:

$$e = [vertex, integer, vertex, integer]$$

where they can be created by checking each index in $\mathbf{IL(T)}$. We denote i and j to be the position of a given index in a tensor $\mathbf{T}$ and in $\mathbf{FL(T)}$ respectively. Remember that the indices in any $\mathbf{T}$ are either free or dummy.

1. If the index is free, we check its type(subscript vs. superscript).

   (a) If it is a subscript index, the representing edge will be directed from $\mathbf{V}_1$ to $\mathbf{V}_0$. Thus, $e_1 = [\mathbf{V}_1, i, \mathbf{V}_0, j]$.

   (b) Similarly, if the index is a superscript index, the $\mathbf{EF}$ of the index is represented as an edge directed from $\mathbf{V}_0$ to $\mathbf{V}_1$. Thus, $e_2 = [\mathbf{V}_0, j, \mathbf{V}_1, i]$.

2. Meanwhile, if the index is a dummy one, it will be represented as a loop-edge going from the vertex to itself. We always choose the edge to be directed from the subscript dummy index to the superscript one. Thus, $e_3 = [\mathbf{V}_1, i, \mathbf{V}_1, i']$ where i' is the position of the subscript dummy index.

Note that, in the case of a monomial, the **ED** will be directed from one vertex, say $\mathbf{V}$, to another one, $\mathbf{V}'$, depending on the position of the subscript dummy index. Thus, $e_4 = [\mathbf{V}, i, \mathbf{V}', i']$.

9.3 Implementation of Tensors

In designing this package, emphasis has been placed on the interface, allowing simple user input of calculations and tensor definitions, as well as presenting readable output. Calculations are specified in an intuitive manner using a minimal number of commands.

Since each tensor is formulated and mapped to a graph, the user needs to provide the necessary elements of a tensor. Thus, each tensor should be represented as **TM** followed by the 4 different components of a tensor separated by a comma. The components should be stated between brackets().

At the beginning, we need to specify the coefficient of a tensor. The user could present any real number, of course not infinity. it could be a negative or a positive number. Note that, this number needs to be given if it is one. If the coefficient is zero, then the tensor is a zero tensor.

9.3.1 Name of Tensor

The name of the tensor can be represented as an uppercase or a lowercase, Latin or Greek letter. It should be placed after the coefficient and between " ". For instance, "**R**" is the

name of the Riemann tensor $\mathbf{T} = 2\mathbf{R}[\_a, \hat{}b, \_c, \hat{}d]$. Note that, we don't reserved any word for any special tensor.

9.3.2 Indices and Types

After the coefficient and the tensor name, the user will present the indices. They could be presented as an upper or lower case, Greek or Roman, subscript or superscripts, ... except for % which is a reserved character to treat the dummy indices.

We defined the indices associated with the tensor by specifying the type of each index, such as cov for covariant or cont for contravariant, followed by the name of the index stated between quotes " ". These indices should be collected in brackets [ ] and in a given order where the position of each index in the tensor is similar to the one in the list.

Example 9.1 *Let* $\boldsymbol{T} = 2\boldsymbol{R}[\_a, \hat{}b, \_c, \hat{}d]$ *be a Riemann tensor. Thus, the list of indices is represented as: [cov"a", cont"b", cov"c", cont"d"].*

9.3.3 Symmetries

In the last bracket, the user should define the symmetric type of the tensor. The symmetric type is represented as the name of the type followed by a set of integers between brackets where they represent the position of these indices. The name of the types is defined as the follow:

- *nsym*: which represent a non-symmetric set of indices,
- *sym*: which represent a symmetric set of indices, and
- *asym*: which represent an antisymmetric set of indices.

Meanwhile, a mixed symmetric tensor can be defined as a combination of any type mentioned above. Since the tensor can have more than one type of symmetries, the

mixed symmetry should be represented in brackets. For instance, the type of the tensor $\mathbf{T} = 2\mathbf{R}^{a}{}_{b}{}^{c}{}_{d}$ mentioned in example 9.1 should be represented as: $[asym[1,2], asym[3,4], sym[[1,2],[3,4]]]$.

Example 9.2 *The tensor* $\boldsymbol{T} = 2\boldsymbol{R}^{a}{}_{b}{}^{c}{}_{d}$ *is a Riemann tensor which have four free indices, namely a and c as contravariant indices and b and d as covariant indices. It will be represented as:*

$\boldsymbol{TM}(2, "\boldsymbol{R}", [cov"a", cont"b", cov"c", cont"d"], [asym[1,2], asym[3,4], sym[[1,2],[3,4]]])$.

9.4 Output

The output in *Aldor* will be given as the coefficient of the tensor and the name of the tensor followed by the set of the tensor's indices. A subscript free index will be represented as _ followed by the name of the index while a superscript free index will be represented as ^ followed by the name of the index. Similarly, a dummy index will be represented as the type followed by $\%i$ where i is an integer. The indices will shown in a bracket [ ] and in order as it should be. Thus, the tensor $2\mathbf{R}^{a}{}_{b}{}^{c}{}_{d}$ will be shown as:

$$2\mathbf{R}[\_a, \hat{}\, b, \_c, \hat{}\, d]. \tag{9.1}$$

9.4.1 Dummy Indices

In *MathTensor* and *Macsyma Itensor package*, there exists a restriction of using certain letters for the indices in tensors. For instance, in *MathTensor*, we can only use the letters $a, b, ..., o$ as index names and $p, q, ..., z$ as dummy index names.

It is inconvenient for the user, in particular, to define a rule which contains some dummy indices and has to type additional commands to indicate the type of the index used in that rule.

In this approach, it has been an objective to treat the dummy indices from the user's viewpoint. The dummy indices are generated internally and then they are converted to some appropriate index symbols in the output. Leaving all the choices to the user and without any restriction, the output for every dummy index will be shown as the symbol "%" followed by a single integer.

For example, the expression $\mathbf{U}^{a}{}_{b}{}^{c}\,\mathbf{V}^{d}{}_{d}{}^{b}$ will be shown as $\mathbf{U}^{ac}{}_{\%1}\,\mathbf{V}_{\%2}{}^{\%1\%2}$ where b and d are replaced by %1 and %2 respectively. Thus, the user can use any dummy index name, Greek or Latin letters, symbols or numbers, without worry if it has been previously defined.

9.5 Code to Show

It is useful to present a part of the implementation in Aldor. This is the tensor monomial category of how to add or to multiply 2 of them.

```
define TensorMonomialCategory(K: Ring, aST: string, V: VertexCategory,
E: EdgeCategory(V)): Category == BasicType with {

coerce       : Integer       -> %;
coerce       : Integer       -> %;

coeff        : %             -> K;
symbollist   : %             -> List TK(aST);

freeindices  : %             -> List String;
dummyindices : %             -> List String;

CanonicalGrf : %             -> Multigraph(V, E);
```

```
normalize     : %                -> %;

applyTM       : (K, TK(aST))     -> %;
TM            : (K, string, List TensorIndex, List TensorSymmetries) -> %;

-             : %                -> %;
+             : (%, %)           -> %;

*             : (K, %)           -> %;
*             : (%, %)           -> %;

contrat       : (%, List TensorIndex) -> %;
<             : (%, %)           -> Boolean;
}
```

9.6 Experimental Implementation

Finally, we will present some examples to show how this package works. We will define three tensors $\mathbf{T}_1, \mathbf{T}_2$ and $\mathbf{T}_3$ such that: $\mathbf{T}_1 = \mathbf{R}_{ba}{}^{dc}$, $\mathbf{T}_2 = \mathbf{U}^{a}{}_{ic}$, and $\mathbf{T}_3 = \mathbf{V}_{c}{}^{aj}$. Then, we will multiply $\mathbf{T}_1$ by $\mathbf{T}_2$ and $\mathbf{T}_1$ by $\mathbf{T}_3$ and finally add the result.

> T1 := **TM**(1, "**R**", [cov"b", cov"a", cont"d", cont"c"], [asym[1, 2], asym[3, 4], sym[[1, 2], [3, 4]]);

$$T1 = \mathbf{R}[\_a, \_b, \hat{}\, c, \hat{}\, d]$$

> T2 := **TM**(1, "**U**", [cont"a", cov"i", cov"c"], [sym[1, 2, 3]);

$$T2 = \mathbf{U}[\hat{}a, \_c, \_i]$$

> T3 := **TM**(1, "**V**", [cov"c", cont"a", cont"j"], [sym[1, 2, 3]);

$$T3 = \mathbf{V}[\hat{}a, \_c, \hat{}\, j]$$

> $T4 := T1 \cdot T2;$

$$T4 = \mathbf{R}[\_b, \_\%1, \hat{}\, d, \hat{}\, \%2]\mathbf{U}[\_i, \_\%2, \hat{}\, \%1]$$

> $T5 := T1 \cdot T3;$

$$T5 = \mathbf{R}[\_b, \_\%1, \hat{}\, d, \hat{}\, \%2]\mathbf{V}[\hat{}j, \_\%2, \hat{}\, \%1]$$

> $T6 := T4 + T5;$

error, can't add these monomials.

Chapter 10

Conclusion

This thesis reflects an approach for the implementation of tensor expressions on a computer algebra system. We are going to summarize the advantages of this approach compared to other packages.

- **Canonical labeling** reduces the number of monomial isomorphism tests compared to other implementation.
- **First algorithm based on graph theory** and not based on any other algebraic solution.
- **Uniqueness of representation.** The introduction of this representation in any computer algebra system is unique. It resembles to the way scientists usually write tensors. For example, the symmetric tensor $\frac{1}{3}\mathbf{U}_i{}^c{}_b{}^i$ is represented as:

$$TE(\frac{1}{3}, "U", [cov"i", cont"c", cov"b", cont"i"], [sym[1,2,3,4]]). \tag{10.1}$$

- **Aldor made programming easier**. As it has been explained in the previous chapter, both types and functions may be constructed dynamically and manipulated in the same way as any other value.

- **Easier to create a new tensor** or to use special tensors introduced in Chapter 2. For instance, it is easy to create a new symmetric tensor $\mathbf{U}_{a\ c\ e}^{\ b\ d\ f}$ or to introduce the **Riemann** tensor $\mathbf{R}_{a\ c}^{\ b\ d}$ as shown in Section 9.3.

- **Dynamic assignment of monomials contain tensorial quantities.** in this approach, a monomial containing tensorial quantities is treated as the same way as a tensor. For example, $TM1 := \frac{1}{3}\mathbf{U}_{i\ b}^{\ c\ i}\mathbf{R}_{a\ c}^{\ b\ d}$ is treated as a tensor of rank 2. In addition to that, $TM1$ associates, using the symmetric properties of Riemann tensor, with the property that it is antisymmetric in its indices.

- **Space efficient**. We do not need to introduce the concept of database, as the same as any other packages, which contain many substitutions rules and identities for tensors and classify the various rules inside that database. This classification needs a large database to store the identities and substitutions formulae for various quantities. As an example, the tensor

$$\mathbf{R}^{r}{}_{kls}\mathbf{R}^{s}{}_{mnt}\mathbf{R}^{t}{}_{pqr}$$

will be treated in an easier way than looking in a large database and comparing to each elements in the database.

- **Implementation has a simple interface**. For example, the output of the tensor (10.1) will be shown as:

$$\frac{1}{3}\mathbf{U}[\_b,\hat{}\,c,\_\%1,\hat{}\,\%1].$$

- **Not restricted to symmetries.** The definition of tensors in this approach, as opposed to some package, is not restricted to the definition of the symmetric and

antisymmetric properties of a tensor. One is able to define all the attributes associated with a tensor. These attributes can be defined and stored for later access. For example, the user can create a tensor which can be symmetric, antisymmetric or whatever he choose.

- **Same algorithm treats Spinors** as similar to the method of treating tensors.
- **Same algorithm** treats partial derivative and covariant derivative.
- **No restriction of using any index name**. There is no restriction on using any names for indices in a tensor. For instance, we can use a0, uwo, CS and ud at the same time as indices in the Riemann tensor $\mathbf{R}_{a0}{}^{uwo}{}_{CS}{}^{ud}$.
- **No restriction for dummy indices.** In this approach, the restriction of using a set of names to represent the dummy indices has been removed. For example both a and z can be represented as dummy indices in $\mathbf{U}_{a}{}^{za}{}_{zi}$.
- Having this new representation, we are able to apply several operations at the same time. For example,

$$2\mathbf{A}^{a}{}_{b;c} + 3\mathbf{B}^{a}{}_{b;c} \tag{10.2}$$

represents the sum of two tensors having c as covariant derivative.

Bibliography

[Aki72] M. A. Akivis. *An Introduction to Linear Algebra and Tensors.* Dover Publications, Inc, 1th edition, 1972.

[Bab80] László Babai. On the Complexity of Canonical Labeling of Strongly Regular Graphs. *Society for Industrial and Applied Mathematics*, 9(1):212–216, February 1980.

[CP90] S. Christensen and L. Parker. MathTensor: A system for performing tensor analysis by computer. *Mathematica Journal*, 1(1):51–61, Summer 1990.

[Dan97] D. A. Danielson. *Vectors and Tensors for Engineering and Physics.* ADDISON, WESLEY, 2nd edition, 1997.

[ea93] J.E.F. Skea et al. *SHEEP, a Computer Algebra System for General Relativity.* Proc First Brazilian School on Comp Alg, Oxford U Press, 1993.

[FB89] Frank Harary Fred Buckly. *Distance in Graphs.* Addison-Wesley Publishing Company, New York, 1th edition, 1989.

[FKWC92] S.A. Fulling, R.C King, B.G. Wybourne, and C.J. Cummins. Normal forms for tensor polynomials: 1. the Riemann tensor. *Class. Quantum Grav.*, pages 1151–1197, 1992.

[Gib85] Alan Gibbons. *Algorithmic graph theory.* Cambridge University press, London, 1th edition, 1985.

[GoŁ74] StanisŁaw GoŁab. *Tensor Calculus.* Elsevier Scientific Publishing Company, 1974.

[Gra94] Ralph P. Gramaldi. *Discerte and Combinatorial Mathematics An Applied Introduction.* Addison and Wesley, New York, 3th edition, 1994.

[grt94] 1994. `http://grtensor.phy.queensu.ca/`.

[GY94] Jonathan Gross and Jay Yellen. *Graph Theory and its Applications.* CRC Press, New York, 1th edition, 1994.

[Hla99] Jean Hladik. *Spinors in Physics.* Springer, New York, 1th edition, 1999.

[Hör79] L. Hörnfeldt. A system for automatic generation of tensor algorithms and indicial tensor calculus. In Edward W. Ng, editor, *Proceedings of the International Symposium on Symbolic and Algebraic Manipulation (EUROSAM '79)*, volume 72 of *LNCS*, pages 279–290, Marseilles, France, June 1979. Springer.

[Hör85] L. Hörnfeldt. STENSOR. In Bob F. Caviness, editor, *Proceedings of the European Conference on Computer Algebra (EUROCAL '85): volume 2: research contributions*, volume 204 of *LNCS*, pages 165–165, Linz, Austria, April 1985. Springer.

[IK96] V.A. Ilyin and A.P. Kryukov. ATENSOR - REDUCE program for tensor simplification. *Computer Physics Communications*, 96:36–52, 1996.

[Int95] 1995. `http://nightflight.com/cgi-bin/foldoc.cgi?symbolic+mathematics`.

[JF97] Christoph Schweigert Jürgen Fuchs. *Symmetries, Lie Algebras and Representations.* Cambridge University press, London, 1th edition, 1997.

[Kav94] Masoud Kavian. *Tensor Computation on Computer Algebra Systems.* UofW, 1994.

[kMG97] M. kavian, R. G. McLenaghan, and K. O. Geddes. Application of genetic algorithms to the algebraic simplification of tensor polynomials. In Wolfgang W. Küchlin, editor, *ISSAC '97. Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation, July 21–23, 1997, Maui, Hawaii*, pages 93–100, New York, NY 10036, USA, 1997. ACM Press.

[McC57] A. J. McConnell. *Applications Of Tensor Analysis.* Dover Publications, Inc, New York, 1th edition, 1957.

[Pap99] John Papastavridis. *Tensor Calculus and Analytical Dynamics.* CRC Press, 1th edition, 1999.

[PC94] Leonard Parker and Steven M. Christensen. *Math Tensor: A System for Doing Tensor Analysis by Computer.* Addison-Wesley, Reading, MA, USA, 1994.

[Por98] R. Portugal. An algorithm to simplify tensor expressions. *Computer Physics Communications*, 115:215–230, 1998.

[Por99] R. Portugal. Algorithmic simplification of tensor expressions. *Journal of Physics*, 32(44):7779–7789, 1999.

[Ray70] D'Inverno Ray. *ALAM Programmer's Manual*, 1970.

[RB99] K. Ranganathan R. Balakrishnan. *A textbook of Graph Theory.* Springer, New York, 1th edition, 1999.

[RC71] D'Inverno Ray and Russell Clark. *CLAM Programmer's Manual.* King's College, London, 1971.

[SMW94] Samuel S. Dooley Stephen M. Watt, Peter A. Broadbery. *Aldor, Compiler User Guide.* The Numerical Algorithms Group Limited, New York, 2nd printing edition, September 1994.

[Sok64] I. S. Sokolnikoff. *Tensor Analysis, Theory and Applications to Geometry and Mechanics of Continua.* John Wiley, New York, 2nd edition, 1964.

[Spa65] Barry Spain. *Tensor Calculus.* Oliver and Boyd LTD, Great Britain, 3th edition, 1965.

[The83] M. I. T. The Mathlab Group, Lab. for Computer Science. *MACSYMA Reference Manual, Version 9, Volume I.* Symbolics, Inc., Burlington, MA, 2nd printing edition, December 1983.

[WM92] B.G. Wybourne and J. Meller. Enumeration of the order-14 invariants from the Riemann tensor. *Journal of Physics*, A 25:5999–6003, 1992.

Vita

Name	Nabil Obeid.
Place of Birth	Tripoli, Lebanon.
Year of Birth	1967.
Post-secondary Education and Degrees	Université de Moncton, Moncton, N.B., 1992–1995 B.A. Concordia University, Montreal, Quebec, 1995–1998 M.Sc. The University of Western Ontario, London, Ontario, 1998–2001 M.Sc.
Honors and Awards	President's Scholarship for Graduate Student, The University of Western Ontario, 1998–1999. Nominated for Graduate Student Teaching Award, The University of Western Ontario, 2000.
Related work experience	Teaching Assistant, The University of Western Ontario, 1998–1999. Research Assistant, The University of Western Ontario, 1998–2000.

Publications

1. Nabil Obeid, Absolutely Continuous Invariant Measures for Meromorphic Functions. *Master Thesis*, Math, 1998.

2. Nabil Obeid, How to use Derive, *Derive User Software Manual*, Université de Moncton, 1994.