AN EXTERIOR CALCULUS PACKAGE FOR REDUCE
AND
DIMENSIONAL REDUCTION OF GENERALISED THEORIES OF GRAVITY

A DOCTOR OF PHILOSOPHY THESIS

in

Physics

Middle East Technical University

By

Göktürk Üçoluk

December 1989

Aproval of the Graduate School of Natural and Applied Sciences.

Prof. Dr. O. Alpay Ankara

Director

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Prof. Dr. Şinasi Ellialtıoğlu

Chairman of the Department

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Prof. Dr. Tekin Dereli

Supervisor

Examining Committee in Charge:

Prof. Dr. Perihan Tolun (Chairman)
Prof. Dr. Tekin Dereli
Prof. Dr. Hüseyin Akçay
Assoc. Prof. Dr. Yiğit Gündüç
Assoc. Prof. Dr. Metin Önder

ABSTRACT

AN EXTERIOR CALCULUS PACKAGE FOR REDUCE
AND
DIMENSIONAL REDUCTION OF GENERALISED
THEORIES OF GRAVITY

Üçoluk, Göktürk
Ph.D., Department of Physics
Supervisor: Prof. Dr. Tekin Dereli

December 1989, 115 pages

A new symbolic algebraic manipulation software for exterior calculus and the technique of dimensional reduction is developed. The pack is written in RLISP, a dialect of LISP, as an extension to REDUCE. The package is introduced both from user and technical aspects. This system is used in solving some original problems in generalised theories of gravity. The first of these problems is finding the reduced form of the modified double self dual field equations for a stationary, axially symmetric metric with a non-vanishing torsion ansatz. The second is the Kaluza-Klein reduction of a generalised theory of gravity in $D = 5$ dimensions. The third is The Kaluza-Klein reduction of the dimensonally continued Euler form actions in $D = 7$ assuming the product topology $M_4 \times S^3$. Yang-Mills self interaction terms are found. The new RF^3 type coupling terms are explicitly worked out.

Keywords: exterior calculus, dimensional reduction, generalised theories of gravity

ÖZ

REDUCE IÇİN BİR DIŞ CEBİR PAKETİ
VE
GENELLEŞTİRİLMİŞ KÜTLE ÇEKİM KURAMLARININ
BOYUTSAL İNDİRGENİMİ

Üçoluk, Göktürk
Doktora, Fizik Bölümü
Tez Yöneticisi: Prof. Dr. Tekin Dereli

Aralık 1989, 115 sayfa

Bu çalışmada dış cebir hesabı ve boyutsal indirgeme yöntemi için bir sembolik cebirsel işlemleme yazılımı geliştirilmiştir. Paket, LISP'in bir dialekti olan RLISP'te REDUCE dilinin bir geliştirilmesi olarak yazılmıştır. Yazılım hem kullanıcı bakımından hem de teknik boyutta sunulmaktadır. Söz konusu sistem genelleştirilmiş çekim kuramlarında bazı özgün problemlerin çözümünde kullanılmıştır. Bu problemlerin ilki değiştirilmiş çift öz-eşlenik alan denklemlerinin, durağan, eksenel bakışımlı bir metrik için ve sıfır olmayan bir burulma eki ile indirgendiği biçimin elde edilmesidir. İkincisi, $D = 5$ boyutta, genelleştirilmiş bir kütle çekim kuramının Kaluza-Klein boyutsal indirgenimidir. Üçüncüsü, sonuç topolojinin $M_4 \times S^3$ olduğunun varsayıldığı, boyutsal olarak $D = 7$'ye sürdürülmüş Euler eyleminin boyutsal indirgeniminin gerçekleştirilmesidir. Sonuçta Yang-Mills alanlarının kendi kendileri ile etkileşimlerini betimleyen terimler gözlenmiştir. Bilinen RF^2 türü etkileşmelere ek olarak yeni RF^3 türü etkileşim terimleri elde edilmiş, bunların ne oldukları açık olarak hesaplanmıştır.

Bilim dalı sayısal kodu: 404.06.01

Anahtar Kelimeler: dış cebir hesabı, boyutsal indirgeme, genelleştirilmiş çekim kuramları

ACKNOWLEDGMENTS

It is great pleasure to express my gratitude to Prof. Dr. Tekin Dereli for his continuous support and guidance throughout this work and his very valuable friendship.

TABLE OF CONTENTS

ABSTRACT i

ÖZ ii

ACKNOWLEDGMENTS iii

TABLE OF CONTENTS iv

I INTRODUCTION 1

I.1 The Aim of this Work 1

I.2 Symbolic Algebraic Manipulation in Theoretical Physics 2

I.2.1 High Energy Physics 3

I.2.2 Celestial Mechanics 4

I.2.3 General Relativity 5

I.2.4 Other Fields in Physics 6

I.3 The Present state: What Can Be and What Cannot Be Done in SAM 7

I.4 Exterior Calculus in Theoretical Physics 9

I.5 The Structure of this Thesis 11

CHAPTER

II X^TR 13

II.1 An Introduction to X^TR 13

II.1.1 What is X^TR? 13

II.1.2 What is this section about and what it is not ? 13

II.1.3 Getting started 13

II.1.4 Writing programs 14

II.1.5 What will be the output ? 14

II.1.6 Can one get the output as a hard copy? 15

II.1.7 How long will it take to solve a problem in X^TR, can one predict it? 15

II.2 Programming in X^TR 16

II.2.1 Declarations 16

II.2.2 Removing declarations . . . 17
II.2.3 Viewing previously made form declarations . . . 17
II.2.4 Built in exterior calculus operators . . . 18
II.2.4.1 Exterior (Wedge) product . . . 18
II.2.4.2 Exterior differentiation . . . 19
II.2.4.3 Interior product . . . 20
II.2.4.4 Hodge duality . . . 20
II.2.5 How to introduce a metric . . . 21
II.2.6 How to introduce the signature . . . 21
II.2.7 A Handle to Introduce User-Defined-Exterior Operators Easily . . . 22
II.2.8 Facilities for the Dimensional Reduction Technique . . . 23
II.3 Useful Hints for Using REDUCE and $X^{T}R$ . . . 26
II.3.1 To speed up your programs . . . 26
II.3.2 To Reduce Program's Memory Requirement . . . 27
II.3.3 A Peculiarity of REDUCE . . . 27
II.4 Inside $X^{T}R$ . . . 29
II.4.1 The Forms . . . 29
II.4.2 The Linearization . . . 30
II.4.3 Some Important Utility Functions . . . 31
II.5 An Overview of LISP . . . 34
II.5.1 The data structure . . . 34
II.5.2 Programming in LISP . . . 36
II.5.3 Some System Functions . . . 38
II.5.4 How to introduce new functions . . . 42
II.6 An Inside Look to REDUCE . . . 45
II.6.1 The Programing Language Used and the Source Organization . . . 45
II.6.2 The Environments REDUCE is Providing . . . 46
II.6.3 Input . . . 47
II.6.4 The Evaluation . . . 49
II.6.5 Output . . . 53
III PHYSICAL APPLICATIONS . . . 55
III.1 A Study of Stationary, Axially Symmetric Space-time Geometries Satisfying Modified Double Duality Equations Using the Exterior Calculus Package $X^{T}R$ for REDUCE [1] . . . 55
III.1.1 Stationary, axisymmetric double dual curvatures . . . 57

III.1.2 The Solution 60
III.1.2.1 The First type of S_i 60
III.1.2.2 The Second type of S_i 60
III.1.3 The Torsion Ansatz 64
III.2 Kaluza-Klein Reduction of Generalised Theories of Gravity and Non-minimal Gauge Couplings [2] 66
III.2.1 Kaluza-Klein reduction 66
III.2.2 Non-minimal gauge couplings 68
III.3 Direct Curvature-Yang-Mills Field Couplings Induced by the Kaluza-Klein Reduction of Euler Form Actions in Seven Dimensions [3] . . . 72
III.3.1 Kaluza-Klein reduction on $M_4 \times S^3$ 73
III.3.2 RF^3 - type couplings 76
III.3.3 Concluding Comments 78
IV CONCLUSION 79
LIST OF REFERENCES 80
A NOTATION AND CONVENTIONS 83
B The RLISP Syntax 85
C SOURCE OF X^TR 90
VITA 115

CHAPTER I

INTRODUCTION

I.1 The Aim of this Work

All factions of science provide their scientists with some sort of intuitive methodology, a way of "first approach to problems". The tradition of the school of theoretical physics is such that it tries to avoid the use of technological tools. It believes that the tools provided is the thought of mathematics and logic. But nowadays the computations required to solve some problems at the frontiers of theoretical physics are becoming more and more unmanageable for human power, even at group level. In the last century with the advances in technology, physics obtained tools of enormous power. Although, at the first sight, those contrivances seem to aid mainly experimental physics by providing electronical devices like high precision experimental environments and computers to handle numerical data, in the last twenty-five years theoretical physics got its share too. A new field of computer science, namely the field of **computer algebra** evolved. The interdisciplinary character of this kind of work made it necessary that the developers had to be experts in at least three disciplines:

- Applied algebra
- Computational science
- Physics

As a theoretical physicist it is a pride to mention that among the pioneers of this new area the leading number of people are theoretical physicists. Because of their need for large scale symbolic computation, they have concentrated on developing computer algebra programs

and inventing new and faster algorithms in this field. This Ph.D. work originated from a similar stimulation and constitutes of two ideas. **Firstly, to develop a new symbolic computational tool for a formalism which is becoming more and more popular among theoretical physicists, namely, exterior calculus, and then secondly, to use this tool to solve some original problems in the field of relativistic theories of gravity.**

I.2 Symbolic Algebraic Manipulation in Theoretical Physics

Before giving an overview of what is done in symbolic algebraic manipulation (*from now on will be abbreviated as* **SAM**) regarding physics it would be reasonable to answer the question "What is the field of SAM?" or with another naming "What is Computer Algebra?". We may state the definition as [9]

> It is the part of computer science which designs, analyzes, implements and applies algebraic algorithms.

SAM is in close encounter with the mathematical disciplines like algebra, analysis (calculus) and numerical analysis and in an imperative relation to logic. However one has to point out that there is a small difference in SAM compared to these disciplines. In algebra the general rules underlying an algorithm are the primary objects of study whereas in SAM the tool itself, the algorithm in all its aspects, including its efficiency and implementation, is the subject of concern.

It would not be a mistake to say that the origination of SAM programs in 1970's were due to some needs emerging in theoretical physics problems. In those days QED, for example, was trying to compute the sixth order corrections to the anomalous magnetic moment of the electron. The analytic form for the fourth order contribution was computed by hand in 1957 [1]. Because of the sheer magnitude of the calculation, the sixth order result, which involves looking at 72 diagrams, has been computed by a group effort involving many different independent researchers. To be sure the result is correct, each diagram was being computed by two different groups, usually with different SAM computational tools and a tool developed by Campbell and Hearn [2] that later gave rise to the famous SAM language REDUCE.

We would like to give a short survey of what is done up to now as SAM applications in various fields of physics.

I.2.1 High Energy Physics

In this field we can mention the SAM languages REDUCE, SCHOONSCHIP, MACSYMA, ASHMEDAI. Formerly the trend was to perform QED calculations. In the recent years the possibility of computation in QCD is also considered. Both in QED and QCD the main approach is through a perturbative series expension. In QED as well known the form of a process is:

$$P = A_1\alpha + A_2\alpha^2 + A_3\alpha^3 + \cdots$$

where α is the so-called fine structure constant. To calculate each of the A_i's one has to evaluate several Feynman diagrams, each of which graphically represents a different situation of the same physical process. There exist SAM programs to generate these topologically different graphs. [3] It is also a task demanded from SAM to realize the divergences (both infra-red and ultra-violet) in the diagrams and perform the well defined renormalization process. The result leads to the calculation of an integral of the form

$$\int \frac{\mathcal{NUM}}{a_1 a_2 \cdots a_n}\, d^4k_1 \cdots d^4k_n$$

where the a_i's are defined by

$$a_i = \left\{\sum_r (\zeta_r p_r + \eta_r k_r)\right\}^2 + m^2$$

and $\zeta_r,\ \eta_r \in \{0, +1, -1\}$ and p_r and k_r are the momenta of the external and internal particles involved, respectively, and m is the mass. The denominator is handled using the identity:

$$\frac{1}{a_1 a_2 \cdots a_n} = (n-1)! \int_0^1 d\alpha_1 \cdots \int_0^1 d\alpha_n \frac{\delta(\sum_{i=1}^n \alpha_i - 1)}{(\sum_{i=1}^n \alpha_i a_i)^n}$$

$\mathcal{NUM}$ is a product term of Dirac matrices γ_μ and monomials like $(\sum_{\nu=1}^4 q^\nu \gamma_\nu - 1)$, where q^ν are the components of momentum. While the calculation of the numerator is carried out the intermediate results blow up very quickly. It is very common to deal with thousands of terms. With the progress in SAM systems new algorithms have been developed to overcome these sorts of problems. The Kahane-algorithm is a good example for this. [4] It is also worthwhile to mention that only after the SAM systems started to appear, an algorithm like this is invented, although it would be an enormous aid for the hand calculation, too.

Once the algebra for the numerator is done the integration over the internal momenta is performed, finally to obtain the contribution of that particular diagram it is only necessary

to apply a projective operator or to square the result. All these steps can now be performed by SAM systems. Many impressive results, in particular the electron anomaly at higher orders, were only obtainable with SAM systems.

I.2.2 Celestial Mechanics

Another very popular area for the application of SAM systems is the field of celestial mechanics. Here the most tedious task to be faced is in the construction of an analytic perturbation theory; that is a formula for calculating the position, velocity and rotation of a body at any instant in time. The body might be the moon or a planet, or more often these days an artificial satellite or space station. The construction of such a theory requires the solution of differential equations that describe the motion of the body. Since a solution rarely exists in closed form, one expresses it as a power series in terms of the small parameters of the theory, such as the orbital eccentricities, and then solves by the technique of repeated approximation. The most frequently considered series in celestial mechanics are Poisson series which have the form

$$S(\mathbf{x}, \mathbf{y}) = \sum_j P_j(\mathbf{x}) \cos(\mathbf{j} \cdot \mathbf{y}) + Q_j(\mathbf{x}) \sin(\mathbf{j} \cdot \mathbf{y})$$

where $\mathbf{x}$ is a vector of polynomial variables, $\mathbf{y}$ is a vector of trigonometrical variables, and $\mathbf{j}$ is a vector of integers. $P_j(\mathbf{x})$ and $Q_j(\mathbf{x})$ are polynomials, whose coefficients may be real numbers. By the use of standard trigonometric identities, such series are closed under the operations of addition, substraction, multiplication and division. Finally, an important theoretical consideration is that Poisson series can be written in a unique canonical form, in other words, a form to which all equivalent expressions can be uniquely reduced.

The most famous of the calculations of this kind is surely Delaunay's lunar theory, in which the secularized Hamiltonian for the moon's motion is considered. The Hamiltonian describing the physical system requires several pages; up to 600 transformations are to be applied on each term, and combines all the effects like the non-sphericity of the earth, the tilted ecliptic and the influence of the sun. Delaunay spent 20 years to complete his calculation by hand. Deprit et al. [5] used only 20 hours on a small computer equipped with a SAM system for the recomputation. Also it is a tribute to the accuracy of Delaunay that his results published in 1867 were correct up to order 9, with only one exception in an addition of order 7.

I.2.3 General Relativity

This is a field of physics which most heavily makes use of SAM. In general relativity the endeavor is to find a mathematical representation that relates the geometric structure of space-time to its material content (mass & energy). Since the field equations posed by Einstein is the one that fits best with the experimental evidence the main concern is intensified on it.

$$G_{\mu\nu} = -\frac{8\pi G}{c^2} T_{\mu\nu}$$

Where $G_{\mu\nu}$ is the so called Einstein tensor which bears the information of the geometric structure of space and $T_{\mu\nu}$ is the energy momentum tensor. The common approaches can be categorized as:

- Define a metric and try to find the distribution of matter and energy.
- Assume a distribution of matter (and energy) then try to find what geometric structure of the space-time would be possible.
- A midway: Impose certain geometric constraints on the space-time but do not totally fix it, then try to discover a class of solutions of the field equation. (eg. solution classes under certain symmetry assumptions)

It should be mentioned that there exist other field equations alternative to Einstein's which are derived from different Lagrangians by variational methods. Commonly, these are Lagrangians which have additional terms besides the Einstein term. SAM aids also in the computations of these theories. In particular, this Ph.D. thesis is an example for this kind of computations. In the mathematical machinery that leads to the field equations there is hardly an integration involved. This property makes SAM very suitable as a computational tool. Differentiation, union of rational functions over a common denominator, recognition of some patterns and a possible substitution for them by a table-look-up, cancellation of common factors in rationals, substitution and simplification, is the main demand. To have a critic "what can be done and what cannot" the reader is referred to the section on this subject.

One of the most difficult problems in general relativity is how to decide whether two different metrics describe the same gravitational field or not. Now there exists an algorithmic approach to the classification of the metrics.

A number of SAM systems are available for calculations in this field: ALTRAN, FORMAC, SYMBAL, REDUCE, MACSYMA, CAMAL, ALAM, LAM, CLAM, ORTOCARTAN and SHEEP are among them.

I.2.4 Other Fields in Physics

Nowadays *electron optics* is essential and instrumental for, among others, particle accelerators, electron microscopy and the microfabrication techniques for large scale integrated circuits. SAM-tools for research in this area have been REDUCE and CAMAL. [7] The basic physical problem encountered is the calculation of the general trajectory equation of an electron travelling in a cavity in presence of electric and magnetic fields. Mathematically the solution of this problem amounts to solving linear homogeneous second order differential equations. When aberrations are taken into account the algebra becomes cumbersome.

Molecular physics gives a nice and illustrative example for the use of SAM systems. For a long time the calculation of the structure factor for ionic solutions in which one kind is large enough to dominate many of the solution properties was only possible using numerical techniques. It took 15 years to obtain the first analytic solution [7]. The problem is to determine the quantity $c(x)$ in the equation

$$h(x) = c(x) + n\sigma^3 \int h(|\mathbf{x} - \mathbf{y}|)c(y)d\mathbf{y}$$

where $h(x)$ is the total correlation factor, n the particle number density and σ the diameter of the dominating ion. $c(x)$ is the product of the inverse temperature in energy units and the potential between ions. The structure factor is obtained by Fourier transformation of the above equation. REDUCE was used to obtain this analytic result.

In *plasma physics* SAM have aided the study of containment, the stability and the heating of plasma. A standard calculation in this field starts from the equations of motion which either are already or are to be transformed into large systems of partial or ordinary differential equations. Approximate solutions are obtained by both REDUCE and MACSYMA.

Similar problems exist in *fluid mechanics*. A recent application, for example, deals with the solution of the Navier-Stokes equations for a rotating fluid above an infinite disk. [8]

I.3 The Present state: What Can Be and What Cannot Be Done in SAM

Being in contact with other theoretical physicists one can immediately figure out that the demands (or better to say the expectations) of a theoretical physicist from SAM is somewhat different what he (or she) gets. Theoretical physics is an area in which a good degree of discernment, high rate of intelligence is required. The theoretical physicist knows what his goal is. He can (usually) figure out the rough form of an uncalculated result. With this picture in mind he carries out the hand calculation and at each step arrived performs a consistency check; furthermore at each step he evaluates the present state of the calculation with a sight to get rid of the unwanted terms. In a sense he is guiding the calculation through a dark forest. He knows or decides at each step to apply which identity, mathematical trick or whatsoever. Unfortunately, the present state of SAM is very far from this end. In the above stated sense it has no intelligence at all. It has no concept of "better". Its mind is simple, relays on a binary logic. A substitution is either always made (if it is told to do so beforehand) or never. The ability of decision on reasoning is absent: It will never do something like

> ... At this point I shall keep this $\sin^2\theta$ term because I know that at the next step I will perform a differentiation of another term which will give a $\cos^2\theta$ contribution, then I will be able to add them and substitute a 1 instead ...
> ... But the second $\sin^2\theta$ term, yes that must be immediately substituted by $\cos^2\theta - \cos 2\theta$ since ...

But it will be able to take the 1001th derivative of $(1-x^2)^{1/2}$ since the steps to do this is quite simple. The length of the calculation, the size of the intermediate results are of a minor importance for it, provided the algorithm is well defined. What can be done in SAM is listed below

- expansion and ordering of rational functions
- symbolic differentiation of rational functions
- procedural facilities for defining functions
- substitution and pattern matching (*in an orthodox and deterministic manner. Always one directional substitution*)
- automatic simplification of expressions (*e.g., cancellation of the Greatest Common Divisor*)

- calculations with symbolic matrices
- Dirac algebra for the interest of QED, QCD
- various transformations like Fourier, Laplace etc.
- calculation with complex quantities
- arbitrary precision arithmetic
- facilities for solving some differential and integral equations
- symbolic integration
- factorisation of multivariate polynomials
- exterior form algebra

Two of these items are worth talking about. One is the symbolic integration. Indefinite integration is now a mathematically solved problem. That means there exist a well defined algorithm (as like it exists for differentiation) that decides whether an indefinite integral can be taken in closed form or not and if the answer to this is yes then what the result of the integration is. Unfortunately, the algorithm is very complicated and up to now only SAM applications, which do not implement the whole of the algorithm have existed. The theory which is proven 20 years ago is based on an old conjecture originally mapped out by Liouville and Hermite:
If a function $u(x)/v(x)$ where u and v are polynomials over an algebraic extention field is integrated then the result will have the form

$$p(x)/q(x) + c_1 \log(v_1) + c_2 \log(v_2) + \ldots$$

Here p and q are again polynomials of the same field and v_i are factors of the denominator of the integrand.

The second is the factorisation. This is may be the most impressive application field of SAM. Using advanced algorithms adopting p-adic methods, nowadays it is possible to factorise multivariate polynomials consisting of a thousand of terms in less then a minute of time. Interested readers are referred to consult reviews on the subject like [10].

As it is seen from the list above, some fields which would be of great interest to theoretical physicists are absent. There exists preliminary work on some of them but no satisfactory SAM have applications existed yet. These are:

- non-abelian computation (*eg. calculation with Lie algebra valued entities*)
- symbolic indicial tensor manipulations
- graphical study of symbolic parametric functions
- expert system applications for SAM (*By this we mean a system that would take decisions among various calculation methods, justifies and explains it*)

I.4 Exterior Calculus in Theoretical Physics

It is possible to construct an antisymmetric product operation of antisymmetric tensors. These tensors form an anticommutative, Grassmann algebra under this operation which is named as the **wedge product**. Consider an antisymmetric tensor of type $(0, p)$ on a compact orientable manifold M. Such a tensor is called a p-form and written as ω with the definition

$$\omega = T_{i_1 \ldots i_p} dx^{i_1} \wedge \ldots \wedge dx^{i_p}$$

The notation $\wedge$ denotes the wedge product and is defined by:

$$dx^{i_1} \wedge \ldots \wedge dx^{i_p} = 0$$

If any pair of $dx^{i_1} \ldots dx^{i_p}$ are equal the product vanishes; $dx^{i_1} \wedge \ldots \wedge dx^{i_p}$ changes sign if any pair of $dx^{i_1} \ldots dx^{i_p}$ are interchanged. Furthermore it is linear over the zero-forms (i.e., functions), if $\mathbf{P}$, $\mathbf{Q}$ and $\mathbf{R}$ are any forms and α is any scalar function then:

$$[(\alpha \mathbf{P}) + \mathbf{Q}] \wedge \mathbf{R} = \alpha(\mathbf{P} \wedge \mathbf{R}) + \mathbf{Q} \wedge \mathbf{R}$$

The p-form is said to be the element of the degree p in the exterior algebra over the manifold M which is defined as

$$\bigwedge(M) = \sum_{p=0}^{n} \bigwedge^p(M)$$

where $\bigwedge^p$ denotes the space of all rank-p covariant totally antisymmetric tensors. And n is the dimension of the manifold M. An important property of the forms on M is that they vanish if their degree is greater than the dimension of the manifold.

So we have on a manifold M with dimension n:

0-forms : functions ... (*like* $f(x)$)

1-forms : covariant vectors ... (*like* $a_i dx^i$)

2-forms : antisymmetric covariant tensors of rank 2... (*like* $T_{ij} dx^i \wedge dx^j$)

$\vdots$

n-forms : antisymm. covariant tensors of rank n... (*like* $T_{i_1 \ldots i_n} dx^{i_1} \wedge \ldots \wedge dx^{i_n}$)

On the elements of this algebra, that means the differential forms, various operations are defined. These will not be covered here, there exist a large number of text books [11] which can be consulted. What we will mention about is the usefulness of exterior calculus in theoretical physics. What is the power of exterior calculus compared to tensor calculus? Flanders, in his book on differential forms which has already become a standard text book of the subject, depicts some contrasting points as:

- Tensor analysis is a technique for performing calculations with indexed entities. It does not provide a body of substantial results within a physical subject which is available for later applications. Exterior calculus provides such a body.

- In classical tensor analysis one cannot determine the applicability because one is never told what the space is. Everything seems to work in coordinate patches, but it is also known that this is insufficient.

- Tensor fields do not behave nicely under mappings. For example, given a contravariant vector field on an **X**-space and a mapping ϕ from **X**-space to **Y**-space, there is no naturally induced field on the **Y**-space. With exterior forms we have it.

- In tensor calculations the complexity of the indices cause to lose sight of the differences between various types of physical quantities.

- It is often quite difficult using tensor methods to discover the deeper invariants in geometric and physical situations, even in local ones. Using exterior forms, they seem to come naturally according to these principles:

  - All local geometric relations arise one way or another from the equality of mixed partials, i.e., Poincaré Lemma.
  - Local invariants themselves usually appear as the result of applying exterior differentiation to everything in sight.
  - Global relations arise from integration by parts, i.e., Stokes' theorem.

– Existence problems which are not actual partial differential equations (boundary value or Cauchy problems) generally are of the type of Frobenius-Cartan-Kähler system of exterior differential forms and hence can be reduced to ordinary equations.

All these are the reasons for the fact that tensor methods are (*and will be*) replaced by exterior calculus. Especially in modern differential geometry and the prolongation theory of non-linear differential equations, increasing use is made of this elegant formalism.

I.5 The Structure of this Thesis

This thesis, as was said already, consists of two parts.

- The developed X^TR package which implements exterior calculus and dimensional reduction in REDUCE, the SAM language. In the chapter on X^TR a detailed explanation is given how to use it. Also a discussion about the technical details and the interior of the package is included. To have an understanding of these the reader must be equipped with the knowledge about the underlying language LISP and REDUCE's internal structure and functional behavior. But written material and documentation for REDUCE firmware does not exist. Therefore a covering section on LISP and REDUCE's interior is included at the end of the chapter on X^TR. If the reader does not have any intention to extend X^TR or knows already how LISP and REDUCE functions these sections may be omitted.

- Original physical applications in which X^TR has been made use of. It is a fact to be underlined that although X^TR is used in those places where the calculations became perplexing and hard in performing by hand, a lot of human expert effort is spent on top of this. Especially in combining terms, proving under some indicial rearrangements that some terms cancel out, introducing or throwing away surface terms and applying some exterior calculus identities (*like the Bianchi identities*) neither REDUCE nor any SAM language would be in any help. These are jobs still to be carried out by human theoreticians.

  The applications to be found in this thesis are:

  – A study of the modified double self dual curvature equations for a stationary, axially symmetric metric.

- The Kaluza-Klein reduction of Stephenson-Kilmister-Yang theory of gravity in $D = 5$ dimensions. The form of the interactions among the gravitational, electromagnetic and massless scalar fields in four dimensional space-time is exhibited.
- The Kaluza-Klein reduction of Euler form actions on $M_4 \times S^3$. Direct curvature-Yang-Mills coupling terms in the reduced action of the generic types RF^2 and RF^3 are explicitly given.

In the appendices the physical notation used and the source code of $\mathrm{X^TR}$ are given.

CHAPTER II

X^TR

II.1 An Introduction to X^TR

II.1.1 What is X^TR?

As it was said already, exterior algebra is gaining importance as a computational tool in various branches of theoretical physics. Especially in modern differential geometry and the prolongation theory of non-linear differential equations increasing use is made of this elegant formalism. X^TR is a package written by the author which enables computations in this formalism in the algebraic simplification language REDUCE 3. It provides also an additional part which incorporates the Kaluza-Klein dimensional reduction technique.

II.1.2 What is this section about and what it is not ?

This section provides a manual like explanation of how X^TR is activated and used. You will also find some useful hints which will help you in speeding up your calculations. It makes no attempt to discuss neither programming fundamentals nor how to program in REDUCE. It does not attempt to enter the paradise of the Theory of Differential Geometry. For the REDUCE part of programming extensive reference is made to the definitive text the REDUCE 3 Users manual, by A.C. Hearn. For the mathematical aspects the user may consult any text book on Exterior algebra.

II.1.3 Getting started

To have X^TR in your REDUCE session there are three alternatives (listed in the order of preference):

- Run a REDUCE version which includes X^{T}R at a compilative level. (i.e., while REDUCE is compiled the X^{T}R should be included in the compiled source)

- Run REDUCE starting form a checkpoint image which includes the definitions of X^{T}R.

- 1. Run REDUCE (By the conventional ways)
  2. Include the X^{T}R source code by an `IN XTR$` statement. (You don't have to change the environment from ALGEBRAIC to SYMBOLIC)

II.1.4 Writing programs

To make X^{T}R do your wish, you have got three alternatives:

1. If your problem consists of a small set of equations then you may enter these, directly on line, to the interpreter. In this case after the instructions are carried out and the session is over the input lines will be lost and you will not be able to reuse them in another calculation.

2. Creating a file which consists of the statements in the order which you would feed them to the interpreter if you would have chosen the 1 stalternative above. No additional statements are needed. To learn how a text file is created/written on your specific computer consult the operating system manual. To include this file into the REDUCE which is in hold of X^{T}R already, type

   ```
   IN filename;
   ```

   where `filename` is the name of your file that you have written.

3. A combination of the alternatives above. (i.e., some of the statements may be included from a text file and the others may be entered on-line to the interpreter).

II.1.5 What will be the output ?

The output depends heavily on the currently active output specifications of REDUCE. As one may also find out in the REDUCE manual the default settings are for 2-dimensional mathematical printing. We shall point out that this style of printing does not fulfill the actual requirements of 2-dimensionality. Although the powers are printed as expected there is no notion of sub/super indices. To have an index lowering/raising facility an adaptive

version of Van Hultzen's pack is included. You may or may not factor out common terms, form a common denominator, put the denominator (if it is different from one) as a minus powered factor of the numerator, obtain the addition/subtraction separated terms as a column list, etc. For details see your REDUCE manual.

II.1.6 Can one get the output as a hard copy?

Yes, indeed. It is sufficient to direct the output to the printer by a statement:

```
OUT PRINTER;
```

followed by your specific calculation statements. But in fact it is a much better habit to direct the output to a file by:

```
OUT filename;
```

Then when REDUCE is quitted and one is back to the systems environment he can always print this created file using the conventional methods. One can commute among different files and/or the terminal by further `OUT` statements, and a file remains open (that means appendable at its end) until it is closed by a `SHUT` statement. If REDUCE is properly quitted all opened files will automatically be shut. If you shut an open file in REDUCE and then attempt to re-open it by another `OUT` statement the previous content of that file will be lost (it will NOT append at the end). To re-direct the output to the terminal you may use:

```
OUT T;
```

As a final remark: There is no way to get a multi-output at the same time (i.e., you cannot view the output which is going to a file at the same instance on the terminal).

II.1.7 How long will it take to solve a problem in $\mathbf{X^T R}$, can one predict it?

There is no simple answer to the question. Some factors of computation are:

- The number of terms involved in the computations.
- The intermediate size of the expressions expanded or substituted or combined under the basic four arithmetical operations. (i.e., The computation may happily simplify to a few terms, but that does not mean that the intermediate expressions are of the final size!).
- The total available active memory size, the speed of your computer etc.

II.2 Programming in X^TR

II.2.1 Declarations

Exterior algebra is an algebra on the entities called 'forms'. In X^TR all forms have to have a single degree which is a non-negative integer. That also means it is forbidden to have mixed degree forms. The user has to define his forms and their degrees by a `FORMS` declaration statement prior to their usage. Like:

```
FORMS P=3, Q=1, X=0, Y=0;
```

If you want to use an indexed form, you have to declare it in the same manner. If, for example, you have an intention of using the atom `GAMMA` as indexed representing forms of degree 4, you will introduce it to the system by:

```
OPERATOR GAMMA;
FORMS GAMMA=4;
```

Furthermore, the system has to know the space dimension, because in exterior algebra any form with a higher degree then the space dimension has to vanish. This is declared by:

```
SPACEDIM spacedimension;
```

Both the degrees and the spacedimension must be numerical, otherwise X^TR will cause an error. This restrictive choice has its reasons. First, it will speed up the calculations, secondly it will resolve the ambiguity in the comparisons of symbolic valued spacedimension and form degrees.

There exists another way which is a combination of the `FORMS` and `SPACEDIM` statements. When X^TR is used in differential geometry involved calculations usually a coordinate chart is introduced and the number of the coordinates (which are themselves zero-degreed forms) determines the space dimension. This is done by making use of a `COORDINATES` statement. Here is an example:

```
COORDINATES T, R, THETA, PHI;
```

This is equivalent to a sequence of statements:

```
FORMS T=0, R=0, THETA=0, PHI=0;
SPACEDIM 4;
```

A `COORDINATE` statement has some side effects, which we will mention soon.

II.2.2 Removing declarations

Atoms which were declared FORMS will be manipulated according to the exterior algebra rules. To kill this property and making ordinary atoms out of these you may use the REMFORM statement:

```
REMFORM X, P;
```

With the first declaration of the preceding section effective, still Q and Y retain their FORMS property, but X, Y don't. Now they are ordinary atoms again. A similar statement exist for removing COORDINATES, the REMCOORDINATES statements kills the form property of the coordinate atoms. At this stage there appears no difference between REMCOORDINATES and REMFORM, but this is not true. REMCOORDINATES have some side effects too. Furthermore, removing some coordinates (i.e., not all of them) will not cause an automatic re-adjusting of the spacedimension nor two successive COORDINATES declarations will lead to a spacedimension of the total number of coordinates. In this case all atoms subject to these COORDINATES declarations will be defined as zero-forms but the spacedimension will be set according to the last COORDINATES statement.

There is no REMSPACEDIM statement, since a new SPACEDIM declaration will automatically cancel the former one. One may obtain and/or use the spacedimension. This value is stored in the atom XSPCDIM!*.

II.2.3 Viewing previously made form declarations

It is possible to get a listing of FORMS declared atoms and their associated degrees. There are two alternatives:

1. FORMDEGREES ALL;

2. FORMDEGREES a-comma-separated-list-of-atoms;

In the first case you get a complete listing of the presently active FORMS declarations. In the second one you get the listing only for the atoms in the atom list that follows. An example for the second one may be:

```
FORMDEGREES P, Q, THETA;
```

If an atom in the list is not declared to be a form you get a warning. Since a COORDINATE declaration will declare all of the argument atoms to be zero-forms these atoms will also be present in the listing but those will be marked as COORDINATE. There are another group

of declarations which are more specific. You may find them under the heading 'How to introduce a metric'.

II.2.4 Built in exterior calculus operators

The exterior (wedge) product, exterior derivative, interior product and the Hodge duality operation on forms are implemented. The infix/prefix operators associated with those operations are:

Operation	*Conventional symbol*	*REDUCE infix/prefix operator*
Exterior product	$\wedge$	&
Exterior diff.	d	D
Interior product	i	.*.
Hodge duality	$\star$	#

All of these operators are multilinear/linear over scalars (symbolic expressions which are themselves non-forms or functions of them).

II.2.4.1 Exterior (Wedge) product

$X^{T}R$ knows the following:

- The commutation rule,

$$p \wedge q = (-)^{deg(p) \cdot deg(q)} \cdot q \wedge p$$

- A wedge product which contains a form in two different places will cause the product to be zero if this is an odd-degree-form.

- Any wedge product that sums up to a higher degree than the spacedimension in degrees is zero. This feature can be disabled by turning the flag `KILLSUPER` off. This flag is `T` by default.

- The wedge product of two zero forms is the ordinary multiplication of them.

For a human computing expert it is straight forward that a term like $p \wedge q + q \wedge p$ shall simplify to

- zero, if neither p nor q is of even-degree.
- $2p \wedge q$, if otherwise.

For X$^{\mathrm{T}}$R that is true as well, but to achieve the goal, X$^{\mathrm{T}}$R will canonically order any wedge product according to the order of introduction (i.e., the order of first appearance in your program). If you use one single letter identifiers then the order will be the alphabetical order.

II.2.4.2 Exterior differentiation

Exterior differentiation is one of the most frequently used operators in this algebra. It has some properties which are known to the system:

- It acts as an antiderivative type of operator over the wedge product.
- Is nilpotent.
- Acts as the ordinary differentiation if applied to zero-forms.

At this stage there is an ambiguity. If we have a function of one or more zero-forms what will the Exterior differential be? Let's have an example:

```
FORMS R=0; OPERATOR F;
```

Now the question is, what shall `D F(R)` be? There exist two possible choices of preference:

1. It may remain as it is, the answer is `dF(R)`.
2. It is further evaluated to `DF(F(R),R) * dR`.

X$^{\mathrm{T}}$R provides you with a flag, named `DEREXP`, which has control over these preferences. If you set `DEREXP` by a REDUCE statement `ON DEREXP;` then the choice is the second one above, otherwise it is the first. It is always possible to look at a calculation first with `DEREXP` off (which is the default case) and then set `DEREXP` on and reevaluate the result. So you will see the expanded forms. But it is obvious that this scheme will not work the other way around. After you have expanded it once you cannot recombine the results to yield the compact, unexpanded form.

On the terminal and on your printer the exterior derivative will be displayed in lower case unless the hardware of your system disables it or you had set the REDUCE flag `RAISE` on.

II.2.4.3 Interior product

The interior product operator has again some properties known by the system.

- It acts as an antiderivative type of operator over the wedge product.
- Is nilpotent.

The further properties of the interior product differs whether the REDUCE flag `UNKINPRD` is set on or not. `UNKINPRD` stands for `UN`Known-`IN`terior-`PR`o`D`uct. If `UNKINPRD` is turned off, all interior products which do not have an explicit dependence on their left operand, will vanish. Suppose we have a program segment which asks for

```
ON UNKINPRD;
X .*. P;
```

will produce zero as a result, where

```
OFF UNKINPRD;
X .*. P;
```

will yield an evaluation to itself. In both cases

```
X .*.  D X
```

will give a result of `1`. If you are going to work in an orthonormal base you can use directly the index (which shall evaluate to a number) as the first argument of the interior product. That means a piece of code like

```
FOR I:=1:4 SUM (I .*. E(I))
```

is valid and will yield a result of 4 (As it is explained later the operator `E` stands for the orthonormal base 1-forms).

II.2.4.4 Hodge duality

The double Hodge duality identity is built in, but to have this to hand a `SIGNATURE` statement should be issued prior to the usage. If you are working in an orthonormal base then the system automatically replaces the hodges by their proper equivalents. It is of course possible to define your own replacement rules by making use of REDUCE's `LET` statements. There is an additional feature provided by $X^{T}R$. If the flag `DRIFTHODGE` is turned `ON` any expression of the structure

$$(\ldots \wedge \# \bigcirc \wedge \ldots \wedge \#\triangle \wedge \ldots)$$

is converted to

$$(\ldots \wedge \bigcirc \wedge \ldots \wedge \triangle \wedge \ldots)$$

provided that the form degrees of $\bigcirc$ and $\triangle$ sum up to the value stored in `XSPCDIM!*` that means the dimension of the manifold. And

$$(\ldots \wedge \# \bigcirc \wedge \ldots \wedge \triangle \wedge \ldots)$$

is converted to

$$(\ldots \wedge \bigcirc \wedge \ldots \wedge \#\triangle \wedge \ldots)$$

provided that the form degrees of $\bigcirc$ and $\triangle$ are equal.

II.2.5 How to introduce a metric

There is no direct way of introducing the metric. But the user is provided with a facility where he/she is able to introduce his/her orthonormal set of base 1-forms via the `VIERBEIN`. As it is well known, instead of working with a natural base 1-forms it often becomes more convenient to work with an orthonormal set of base 1-forms. They are introduced by setting the non-zero elements of a matrix named `VIERBEIN`. In this case the subsequent step is to allow the system to generate the vierbein and make the necessary `LET` assertions. This is performed by a call to the procedure `GENERATE()`. If the user wants to introduce the inverse vierbein himself then before the `GENERATE` call, he shall introduce the inverse by setting the non-zero elements of the matrix `INVIERBEIN` and then call `GENERATE()` with argument `T`. After the call to `GENERATE()` the user is able to switch between the orthonormal base one form representation and the natural base representation at any instance by turning the flag `INBASE`, `ON` or `OFF`. In the first case all results are obtained as wedge products of `E(1), E(2), ..., E(n)` standing for e^1, e^2, ..., e^n and in the latter case as wedge products of natural base 1-forms. If `INBASE` is `ON` then all hodge maps of wedge products of orthonormal base 1-forms will be replaced by their appropriate equivalents.

II.2.6 How to introduce the signature

The signature is introduced by a `SIGNATURE` statement. The syntax is somewhat similar to the syntax used in human texts. For example,

```
SIGNATURE -1,+1,+1,+1;
```

or

```
SIGNATURE -1,-1,+1,+1,+1;
```

you can find or use the value of the signature under the global variable `SIGNATURE!*`. For the first example above `SIGNATURE!*` will automatically be set to `2`, and for the second to `1`.

The signature is very rarely used in the calculations. The only point of appearance is in the double hodge duality. Therefore, if this kind of computation is not required it is unnecessary to introduce the signature.

II.2.7 A Handle to Introduce User-Defined-Exterior Operators Easily

Very soon, starting programming with X^TR one realizes a need to have an easy way to introduce operators which are linear with respect to the forms. This could be achieved, having some knowledge about the interior structure of X^TR, by making use of the hidden dependency of all the forms on the atom `FORM`. Then one can think to make use of the REDUCE's linearity facility, the `LINEAR` declaration. But this will not be suitable for two reasons.

1. REDUCE's internal linearity would not be able to differentiate between a zero-form which has a dependency on `FORM` (*and shall be taken out, indeed*) and a non-zero-form which has (obviously) a dependency on `FORM`. Both of them would be seen 'depending' on `FORM` and would sit inside the operation. Only non-dependent scalars would be taken out.

2. Unnecessarily and artificially one will have to introduce a second argument, namely the atom `FORM`.

`XTROP` aids for this purpose.

```
XTROP operator-name;
```

declares `operator-name` to be a linear operator over the non-zero form structures.

Since it is a great possibility that an operator defined and used in this manner will itself be a form, too, X^TR provides a way to associate a function as the form-degree evaluation function. This function will automatically receive the form-degree of the argument of the operator calculated. This function shall be defined as a `SYMBOLIC` mode procedure. The way to introduce this function is

```
FDEGREEFN form-degree-evaluation-function-name, operator-name;
```

Here is an example, assume we are going to define a Hodge function of ours. Of course some `LET` rules will be needed, but these are the lines related to the newly introduced context.

```
XTROP MYHODGE;
FDEGREEFN MYFDEGREEFN, MYHODGE;
SYMBOLIC PROCEDURE MYFDEGREEFN U; XSPCDIM!* - U;
```

II.2.8 Facilities for the Dimensional Reduction Technique

Very sketchy, the technique of dimensional reduction can be described as:

1. There exists an n-dimensional manifold M_n with a topology $M_{n_1} \times M_{n_2}$.
2. The physical quantities (*like connection 1-forms, curvature 2-forms, the curvature scalar, etc.*) are calculated for M_n.
3. Similar physical quantities are calculated independently for M_{n_1} and M_{n_2}.
4. Using those results the physical quantities for M_n are tried to be expressed in terms of the physical quantities of M_{n_1} and M_{n_2} only.

The underlying topology of the manifold reflects in the coordinate chart $\mathbf{x}^M$ that it is partitioned as (x^μ, y^m) where the indices run as μ=0,1,...,n_1-1 and m=n_1,...,n. This idea of index separation is the base of the facilities X$^{\mathrm{T}}$R provides.

You know already that it is possible to calculate physical quantities using X$^{\mathrm{T}}$R. So the first three steps mentioned, can be performed using X$^{\mathrm{T}}$R. The fourth step is somewhat complicated. Deriving the first basic physical quantities (*connection and curvature forms)* requires some elaborated human insight. Some terms have to be recognised and intelligently recombined to yield another pre-calculated physical quantity. Sometimes it is also necessary to apply some transformations, drop out surface terms etc. Hence it is not always possible to automate it by means of SAM. But after this is done and other physical quantities are going to be dimensionally reduced then one can make intensive use of X$^{\mathrm{T}}$R. It is possible to define a dimensional reduction space by a declaration

```
REDSPACEDIMS n1, n2;
```

Forms can be defined to be **living** in one of these spaces or both of them. This is done by the declarations

```
LIVESON1 a-list-of-form-variables;
LIVESON2 a-list-of-form-variables;
```

for example,

```
LIVESON2 Q,Y
```

declares the formerly `FORMS` declared `Q` and `Y` to be living `only` in the second dimensional reduction space and having no part in the first. To know these is of vital importance to decide which of the wedge product terms are going to survive. This is because in case of dimensional reduction a wedge product has only the right to exist if its form degrees calculated for each of the two dimensional reduction spaces are lesser or equal to the dimensions of those spaces, respectively. To declare a form to be living in both spaces there exists the declaration

LIFESTYLE a-list-of-form(d_1, d_2);

for instance

```
LIFESTYLE Q(1,3),Y(2,2);
```

will declare the formerly declared 4-form `Q` to act as 1-form in the first dimensional reduction space and as 3-form in the second. Similarly, `Y` will act as 2-form in both of them. It is the user's responsibility to have all the declarations in a total conformity.

It is also possible to declare an indexed form to have a lifestyle according to its indexes. The nice thing about $\mathrm{X^TR}$ is that these indexes are not numbers at all. This is done by the declaration

```
INDEXEDLSTYLE a-list-of-form-variables;
```

which is self explanatory.

But how to define indexes to mark one space or the other? `PUTINDEXKIND` serves for this purpose

```
PUTINDEXKIND 1-or-2, a-list-of-index-variables;
```

The first slot: being `1` or `2` represents the first of the dimensional reduction spaces or the second, respectively.

Indexes can be declared to be `splitting` into two (*or even more parts, but the high degree term killing will not work properly*).

```
SPLIT index-var, first-space-index-var, second-space-index-var;
```

For example, after a

```
SPLIT X,A,B;
```

declaration in case of dimensional reduction (*how this is done will be explained soon*) any indexed term (*operator term, in sense of REDUCE*) will be expanded as the **sum** of the same term written once with `X` substituted by `A` and once by `B`. So, for example, a term like

```
F(X) ∧ G(X)
```

will become

```
(F(A)+F(B)) ∧ (G(A)+G(B))
```

while dimensonal reduction is performed.

And how to fire the dimensional reduction? First turn the flag `DIMRED` on. This will tell X^TR that you want the wedge products calculated checked for the individual form degrees, that means the form degrees according to each of the reduction spaces. If these degrees exceed the corresponding dimension of the dimensional reduction spaces then the term vanishes. If `DIMRED` is not turned on no such check is made. The only check made in that case is whether the total form degree exceeds the dimension of the partitioned manifold. To proceed, call the function `DIMRED()` with two arguments, namely

1. The form expression to be dimensionally reduced.
2. The sum of the two dimensions of the dimensional reduction spaces (*unless for very specific purposes this will be the same with* `XSPCDIM!*`).

Most of the further simplifications will be carried out by user-defined pattern matching rules introduced by the `FOR ALL` statements of REDUCE. X^TR provides some test functions which can be incorporated in the `SUCH THAT` boolean expressions involved. These are

`SPLITP:`

Evaluates to `T` if its argument is an index which was declared `SPLIT`.

`INDEXKIND:`

Returns the associated value of its argument that was introduced by a `PUTINDEXKIND` statement. If such a declaration does not exist then `0` (*zero*) is returned.

`FREEOFINDEX:`

Returns `T` if the first argument does not contain an expression which has an index of the kind which is given in the second argument.

II.3 Useful Hints for Using REDUCE and $X^{T}R$

II.3.1 To speed up your programs

The loveliest program is that program in which one has done the least number of tricks: you simply convert the human style of input to $X^{T}R$, submit it to REDUCE and get the result. But, unfortunately, sometimes (which is not vary rare) when one deals with huge expressions time gets tough and hours of computation time is still insufficient. What to do then is:

- Avoid using unnecessary form dependencies. If you are not going to differentiate (i.e., take the exterior derivative) get rid of the form dependency. For example, if `X` is a zero-form, and you are not going to take the exterior derivative any way it is superficial to have a term like `sin(X)` in the calculations. It is better to use a non-form atom instead (e.g., `sinX` or `S` ....) This will ease the work that the specific $X^{T}R$ routines have to carry out. Because these terms will not be considered from the exterior algebra point of view and will be taken out in the linearization process. All these can still find an application if you have to 'exterior differentiate', you can substitute these terms after the differentiation.

- Don't perform dummy-recalculations!. If you are calculating some mathematical structure which is symmetric or antisymmetric you shall make use of this knowledge. That is also true for derivatives. If you have a commonly used derivative term, calculate it only once, store the result and then re-use it.

- Avoid unnecessary computations. If you can predict that a result is zero (or something as simple as that) be merciful and don't torture the computer. Introduce it.

- Investigate the intermediate results (if there are any!). If you recognize a pattern that frequently occurs, use a `LET` statement to substitute it (usually these are some rationals raised to a rational number power or a standard exponential term)

- Sometimes allowing REDUCE to combine everything over a common denominator is wrong. It may cause the expressions to become swollen. In those cases set the flags (consult the REDUCE manual) to inhibit the common denominator.

- If you don't actually need the intermediate results printed, do not print them. REDUCE spends quite a good deal of time on rearranging terms, finding common

factors, forming the two-dimensional output, etc.

II.3.2 To Reduce Program's Memory Requirement

The worst kind of problems are those where you get a `STORE JAM` error. That means your system cannot provide REDUCE with the necessary amount of memory. The intermediate size of the expressions evaluated is too large. There are some tricks to do in those cases:

- REDUCE is designed in such a manner that in an algebraic evaluation all the sub terms of an operation are first evaluated, substituted and then the final operation is carried out. For example, consider a case in which you have hundred terms summed all together and each of the terms constitutes of multiplicative factors. The calculation proceed as follows: first of all, each multiplication is carried out, if there are any simplifications inside those terms, they are performed, and then, at the very end, all of the hundred simplified terms are put together and searched for cancelling common terms. So, you may easily have situations in which there are terms each of a huge size but cancelling with the previous term and when all of them are put in the memory together you are out of space!. The solution is straightforward. Break those operations into smaller ones. Calculate and add them one by one. This may be time consuming but will work.

- Get rid of the unnecessary results assigned to variables. Remember, everything that can be remembered occupies a place in the memory.

II.3.3 A Peculiarity of REDUCE

It will be quite often the case that the user wishes to introduce `FOR ALL` pattern-matching rules over exterior form operations. REDUCE has a bad habit of calling the `SIMP` procedure (*the algebraic evaluator*) on the left-hand-side of the rule **at the moment of the definition**. This causes problem if the left hand side contains an antisymmetric-like operation (*for example any operator declared* `ANTISYMMETRIC` *or a wedge product*). Because, depending on the result of the LISP-ordering of the free (*pattern matching*) variables the simplificator may decide to reorder the arguments of the operation. This may end in a sign change. Now this rule is recorded with a minus in front of it and loses its chance to match properly for all cases. The way to inhibit this would be introducing a global flag which would prevent this evaluation. But unfortunately this is not present. The related problem for WEDGE is solved by a flag `LETPREVAL`. If this flag is turned `OFF` WEDGE

will return its arguments as it is, and will not attempt **any** simplification. But as it is seen, this does not prevent REDUCE from making an effort to `SIMP`lify. An antisymmetric declared operator will still cause problems. An ad-hoc solution to this is to define the antisymmetricity after the declaration of the pattern-matching rules.

II.4 Inside X^TR

In this section our aim is to sketch the structure of X^TR. While doing this our approach will not be explaining how each line of code or function functions. We will try to answer the questions "**Why** it is done that way" in those places where several alternatives existed. Another aim of this section is to play a guiding role for a work which would use this thesis to extend X^TR further.

II.4.1 The Forms

The base structure in an exterior calculus package is the representation and incorporation of the entities that we call forms. The algebra and how some special operations are defined on this algebra is stated elsewhere in this thesis. As implemented, it is a non-commutative algebra where each form has an associated natural number, its degree. Depending on the dimension (which is also a natural number) only forms of equal or lesser degree than the dimension are allowed to live, higher will vanish. Each operation on these entities (forms) have a well defined effect from the "degree" point of view. Although all functions and variables are called zero-forms on the mathematical ground for technical reasons in X^TR another scheme is adopted. This caused in no aspects a difference from the user point of view. Those variables representing forms and the degrees of them have to be declared at the algebraic users level. These atoms get their degrees stored under the indicator `FDEG` in their property list. In X^TR the form degrees and the dimension of the manifold are restricted explicitly to be non-negative integers. Firstly, it is very seldom that a calculation needs a form degree to be symbolic; Secondly, to restrict them to be numerical enhances the speed performance. If non-numerical form degrees would be allowed then in order to proceed with the calculations some additional information had to be supplied (*like 'is the symbolic degree odd or even', 'does any degree ordering exists among various forms?'*) and also the system had to cope with problems like: 'the wedge of two odd degreed forms is even, therefore,....

Each of the form variables are declared to be dependent on a ghost! atom, namely `FORM`. This trick aids solving the severe problem of linearity. Also they enter a list stored as the value of the global atom `FORMS!*`. So the **forms** are normal algebraic variables just like any scalar variable in the algebraic mode. An alternative way for the implementation could be a new type definition in REDUCE, just like it is done for matrices. Although this seems more modular, it would cause a coding overhead which is actually unnecessary,

because

- there is no need for a special internal representation (*like it is the case for polynomials, or sparse matrices*),
- there exists no special algorithm for the algebra involved in the exterior form calculus (*like Kahene's algorithm for HEP calculations or the Gauss method for matrices*)

II.4.2 The Linearization

Since there is no need for a new data type, an extension of the linearization facilities already provided by REDUCE would suffice. As it is said in the previous section all operators are linear over the non-forms and most of them are linear over zero-forms too (*as it was stated above we consider a technical distinction between non-forms and zero-forms*). In XTR there exist three service functions:

`formlnr():`

Is provided by REDUCE itself for **unary** operator linearization. Has to receive a single argument, which is a list with three members, namely, an atom which stands for the unary operator, then the algebraic expression in prefix form, then an atom for which the dependency check will be carried out. (In our case for example, this is the atom `FORM`.

`linearize():`

Is defined in the source of XTR. Handles linearity for `nary` cases. Has to receive two arguments, first of which is an nary list that starts with the atom representing the operation and continues with the prefix algebraic expressions in sequence, and the second is the atom of dependency (*like it is in the preceding item*).

`getzfout():`

Is defined in the source of XTR. In an nary operation which is already linearized by `linearize()` it is well possible that there exist some zero-forms. Except in some specific cases (*like exterior differentiation*), usually it is desired to get these zero-forms out of the operation, too. This function serves for this purpose. It takes two arguments. An nary algebraic operator expression, and a flag. Consider the case where some elements of the nary operation are expressions with form degrees equal to zero. Then there exists a possibility of leaving a residue of a `1` behind or leave no

trace at all (i.e., remove it from the list). The flag has a control over these choices. A non-NIL flag will leave a 1 behind.

The first two functions described above are general purpose functions where the third is specific for **form** handling. Trying to adopt a general approach for calls to these functions one realizes that the calls to `formlnr()` (*in the case of unary linarization*) or `linearize()` (*in the case of nary linarization*) is done always as the first task. To serve this purpose two general functions, namely, `unarylin()` and `narylin()` are introduced. All form operations are defined so that their simplification function (the function which has its name stored under the indicator `SIMPFN` in the property list of the operation) calls immediately to one of them. Here is an example:

```
SYMBOLIC PROCEDURE WEDGESIMPFN U;
NARYLIN('WEDGE . U, FUNCTION WEDGESIMP);
```

as it is clear, the prefix operator `WEDGE` has the property `WEDGESIMPFN` under the indicator `SIMPFN` which is recognized by the algebraic simplificator `SIMP`. The former versions of REDUCE stripped off the operator (*in this case* `WEDGE`) while calling the simplification function. So it was necessary to append this "chopped off" bit, the name of the operator. In the last version of REDUCE this is fixed: If the operator which has the property `SIMPFN` is also flagged `FULL` then the simplification function receives the whole expression (*without anything being chopped off*). That means if `WEDGE` would be flagged `FULL` then the second line in the example above should read as:

```
NARYLIN(U, FUNCTION WEDGESIMP);
```

Since this package is upgraded from REDUCE 2 to REDUCE 3, in order to keep the downward compatibility at most, we refrained from using this feature throughout the code. Finally, `unarylin()` and `narylin()` carry on with the linearization until linearization yields the same result with the input. Then the function which is given by name as the second argument is called to carry out the simplification (*in the example above* `WEDGESIMP`).

II.4.3 Some Important Utility Functions

`FDEGREE():`

This function calculates the form degree of any algebraic expression. It is possible to define prescriptions for user written operations (*introduced by* `XTROP` *statements*) by means of putting the name of the degree evaluation function under the indicator `FDEGREEFN`. The result is always a non-negative integer.

`SCATTEREDFDEGREE():`

Used for dimensional reduction calculations. It takes a (possibly wedged) form term on which already linearization and zero-form removing are applied. The result is a two element list, where the first element represents the form-degree of the argument which lives in the 1 $^{\text{st}}$dimensional-reduction-space and the second element represents the corresponding similar quantity for the 2 $^{\text{nd}}$. If in any slot there exists a `NIL` this means the "creature" lives **only** in the other space. And if both slots are `NIL` then no meaningful discrimination could be made. This could be, for example, the case in which an exterior differentiation operator is encountered that has an operand with a mixed life style!.

`NEWDEPENDS():`

This is a generalization of REDUCE's internal function `DEPENDS()`. It performs an additional check. It can be the case that an operator has to mask its arguments dependency (*to understand the need for this think of the concept of "invariant" in physics*) or it can also be the case that an operator itself has an hidden dependency. In order to provide a handle for this it is possible to put under the indicators `INDEPENDENT` or `DEPENDENT` an atom which would mean the operator is 'independent of' or 'dependent to' that atom. The function `FREE()` will recognize it.

`FREE():`

Quite the (negation) `NOT()` of `NEWDEPENDS()`.

`ANTIDERDIST():`

This function with four arguments, implements anti-derivational distribution of any function over a list of arguments. Mathematically we can say that if f is an antiderivation then it distributes over a product operation $\odot$ as:

$$f(p_1 \odot p_2) = f(p_1) \odot p_2 + (-)^{g(p_1)} p_1 \odot f(p_2)$$

If this is generalized to a product with n terms, we have

$$f(\bigodot_{i=1}^{n} p_i) = \sum_{i=1}^{n} (-)^{\sum_{j=1}^{i-1} g(p_j)} \left[\bigodot_{j=1}^{i-1} p_j\right] \odot f(p_i) \odot \left[\bigodot_{j=i+1}^{n} p_j\right]$$

where

- $\left[\bigodot_{j=q}^{r} p_j\right]$ with $q > r$ is defined to be 1.

- A sum with a greater lower bound than the upper vanishes.
- $1 \odot p_i \equiv p_i$.

Here we define each of the four arguments of `ANTIDERDIST()` in order:

- A function which will generate the term which represents the case 'the operator is acting on an argument'.
- The list which is subject to the distribution.
- A prefix operator which will join the list members (*one of them each time will have the distributed operation on it*)
- The name of the function which returns a non-negative value. Each time a term in the list is jumped over and it is the turn for the next term to receive the operation the sign is changed by a factor of (-1) raised to the power of this value which is obtained by applying this function to the 'jumped over' term. (*the function g in the above stated equation*)

Here follows an example: Consider the case where the exterior derivative `D` is distributed over the wedge product `WEDGE`. In an expression like `(D (WEDGE X Y Z))` we call `ANTIDERDIST()` as

```
ANTIDERDIST( FUNCTION(LAMBDA (X) (LIST 'D X),
             '( X Y Z),
             'WEDGE
             'FDEGREE  );
```

II.5 An Overview of LISP

II.5.1 The data structure

LISP is a computer language which serves to deal with binary tree structures. A binary tree structure is a conceptual entity which constitutes of node(s). A node can be:

- a *terminal* node
- a *non-terminal (or branch)* node

A non-terminal node is a 2-tuple[1] of links to nodes. With this definition, a binary tree is nothing else than a class of diagraphs. In this definition we have **not** made the assumption that the graph is acyclic, LISP allows that a tree structure may link to itself. What a terminal node is, varies according to the dialect of the LISP used. To be on the safe side we can say that these can be numbers or identifiers.

Because of its ability to cope with binary trees, LISP proves itself to be the best environment for all the computational problems which can efficiently be represented as binary tree structures. To mention a few of them one can state:

- Theorem proving in logic and logic inference
- Game applications *(like chess, bridge)*
- Compiler design
- Natural language processing
- Expert system and knowledge acquisition based system design
- Neural Network simulations
- Symbolic algebraic manipulation (**SAM**)

LISP has a very very simple syntax. It is the syntax of the single object which is called **Symbolic Expression** and is abreviated as **sexpr**. Before we state the syntax definition of all the sexpr, we would like to introduce the syntax for the object that represents the terminal node. These kind of objects are named **atom**.

[1] with 2-tuple we mean an ordered set of cardinality 2

<atom> : : = <literal atom> | <numeral> | −<numeral>

<literal atom> : : = <atom letter>

: : = <literal atom><atom letter>

: : = <literal atom><digit>

<numeral> : : = <digit> | <numeral><digit>

<atom letter> : : = A | B | C . . . | Z

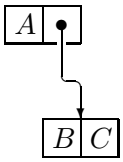
<digit> : : = 0 | 1 | 2 . . . | 9

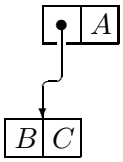
Now we can state the syntax definition of the sexpr:

<sexpr> : : = <atom> | (<sexpr>.<sexpr>)

Adopting the **box-notation** as the graphical representation for the LISP binary trees, we may give some simple examples:

$(A\ .\ B)$ 

$(A\ .\ (B\ .\ C))$ 

$((B\ .\ C)\ .\ A)$ 

It is a common demand in computational problems, like those stated above, to have a handle to represent sets of dynamic length. Sometimes also an ordering among the set is required. LISP provides a convention for representation of an ordered set as a binary tree and calls this a **list**. A formal definition of this mapping from the sets to the binary trees (therefore to the dotted pairs) can be formulated as follows. Assume $\mathcal{R}$ is the function that does the mapping, that means it takes as an argument an ordered set and returns the dotted expression that LISP conventionally associates to it. We define $\mathcal{R}$ recursively:

$$\mathcal{R}(\emptyset) \quad = \quad NIL$$

$$\mathcal{R}(\{x_1, x_2, \ldots, x_n\}) \quad = \quad (x_1 \,.\, \mathcal{R}(\{x_2, x_3, \ldots, x_n\}))$$

Here NIL is a special atom of LISP which serves for other purposes too. This will be explained later.

Since the list structure occurs so often in LISP programming, a shorthand syntax for i/o has been provided. This enables a much better readability. The conversion of this syntax to the dotted pair representation is performed at i/o level as follows:

$$() \quad \rightarrow \quad NIL \tag{II.1}$$

$$(x_1\ x_2\ \cdots\ x_n) \quad \rightarrow \quad (x_1 \,.\, (x_2 \,.\, (\cdots \,.\, (x_n \,.\, NIL)\cdots))) \tag{II.2}$$

We will also make intensive use of this shorthand syntax.

II.5.2 Programming in LISP

LISP programing is based on a simple evaluation scheme, namely, the evaluation of single s-expressions (sexpr) consecutively. LISP takes an sexpr, evaluates it and as the result of this evaluation always returns a value which is also an sexpr. The first concept a novice LISP programmer has to understand is that every evaluation returns a single sexpr (of course if no error has occurred while the evaluation). The software mechanism that performs the evaluation of an sexpr is called the **evaluator**. Although the general data structure is more flexible the evaluator expects an atom or a list. If the submitted is an atom then the value assigned to it will be returned. This will become clearer very soon. If it is a list then a relatively more complex method is applied. The first member of the list is evaluated and is expected to evaluate to an internally or user defined function or explicitly to a lambda expression, otherwise an error occurres. Both the internally defined and the user defined functions fall into two categories regarding the parameter passing. Either they are pre-evaluted or not (i.e., in computational science terms this corresponds to call-by-value and call-by-name argument passing). The remaining elements of the list are treated as the arguments and depending on the category the function falls in, they are evaluated or not. Then the actual evaluation of the function with these arguments is performed. The result is, as said, again an sexpr. The evaluator is also as a system defined function available. Therefore a function which takes its arguments at the first sight not pre-evaluated is always able to perform an evaluation of any sexpr (including the arguments) anytime it decides to, a good example of this is the system function COND which will be explained later. Before we end this section we would like to mention the two

atoms NIL and T which are defined by the LISP system. NIL serves as the terminator for lists and as the boolean **false** value. T is the boolean T value that the system returns (when a test function is applied). LISP is very optimistic in evaluation. In inputs and evaluations all non-NIL values are treated as boolean true (equivalent to T). Both NIL and T have values assigned to themselves. That means when you evaluate NIL it will return NIL as value, similarly T will return T as value.

II.5.3 Some System Functions

In this section we will introduce some of the basic system functions and comment on them.

Function: QUOTE

Number of arguments: 1

Type of arguments: any sexpr

Parameter passing: non-pre-evaluated

Explanation: This function serves for a very simple but useful purpose. It returns its argument non-evaluated. So, it is possible to inhibit an argument from being evaluated by QUOTing it. As a logical abbreviation LISP provides the single quote sign. The use of it is straightforward. $'\triangle$ is an abbreviation for $(QUOTE\ \triangle)$

Example :

input	*output*
$(QUOTE\ \ A)$	A
$(QUOTE\ \ (A\ .\ B))$	$(A\ .\ B)$
$'(A\ .\ B)$	$(A\ .\ B)$

Function: CONS

Number of arguments: 2

Type of arguments: any sexpr

Parameter passing: pre-evaluated

Explanation: This function is the basic constructor to create a dotted pair of two sexpr's.

Example:

input	*output*
$(CONS\ 'A\ 'B)$	$(A\ .\ B)$
$(CONS\ 'A\ '(C\ .\ B))$	$(A\ .\ (C\ .\ B))$
$(CONS\ '(A\ \ B)\ '(C\ \ D))$	$((A\ \ B)\ \ C\ \ D)$

Function: CAR

Number of arguments: 1

Type of arguments: any dotted pair

Parameter passing: pre-evaluated

Explanation: Returns the sexpr that is the right element of the dotted pair.

Example:

input	*output*
$(CAR\ '(A\ .\ B))$	A
$(CAR\ '(A\ \ B))$	A
$(CAR\ '((A\ \ B)\ \ (C\ \ D))$	$(A\ \ B)$

Function: CDR

Number of arguments: 1

Type of arguments: any dotted pair

Parameter passing: pre-evaluated

Explanation: Returns the sexpr that is the left element of the dotted pair.

Example:

input	*output*
$(CDR\ '(A\ .\ B))$	B
$(CDR\ '(A))$	NIL
$(CDR\ '(A\ \ B))$	(B)
$(CDR\ '((A\ \ B)\ \ (C\ \ D))$	$((C\ \ D))$

One shall note the difference of acting with CDR on a list and a dotted pair. The CDR of a list will never be an atom, except the case where the list has only one element, and therefore NIL is returned.

In LISP programming very frequently it is the case that various combinations of CAR and CDR operations have to be successively applied. To provide an easy handle to this, LISP provides functions for all possible combinations. Given a combination, we select in left-to-right order, the relevant A's and D's in the CAR's and CDR's. Then we sandwich this string of A's and D's between a left-hand C and a right-hand R and obtain the name for this combination. An example will be clarifying: $(CDDADR\ \ \triangle)$ means

$(CDR\ (CDR\ (CAR\ (CDR\ \triangle))))$

Function: LIST

Number of arguments: ≥ 1

Type of arguments: any sexpr

Parameter passing: pre-evaluated

Explanation: After the pre-evaluation has taken place a list of these evaluated arguments is formed and returned

Example:

input	*output*
$(LIST\ '\!A)$	(A)
$(LIST\ (CAR\ '\!(A\ B))\ '\!C)$	$(A\ C)$

Function: SET

Number of arguments: 2

Type of arguments: First shall evaluate to an identifier atom, second is any sexpr

Parameter passing: pre-evaluated

Explanation: This function is the main assignment function, the identifier atom is assigned the value of the second argument after pre-evaluation, the result returned is also this value

Example:

input (in sequence)	*output*
$(SET\ '\!A\ '\!(C\ D))$	$(C\ D)$
A	$(C\ D)$
$(CAR\ A)$	C
$(SET\ '\!A\ '\!(D))$	(D)
$(CAR\ A)$	D

Function: SETQ

Number of arguments: 2

Type of arguments: First shall be an identifier atom, second is any sexpr

Parameter passing: first non-pre-evaluated, second pre-evaluated

Explanation: Just the similar action with SET; the only difference is one has not to $QUOTE$ the first argument.

Example:

input (in sequence)	*output*
$(SETQ\ \ A\ \ '(C\ \ D))$	$(C\ \ D)$
A	$(C\ \ D)$

Function: COND

Number of arguments: ≥ 1

Type of arguments: Lists of two sexprs

Parameter passing: non-pre-evaluated

Explanation: This is the conditional statement. In other words, the **if** statement of this language. The first element of the list which is the first argument is evaluated. If the result is non-NIL then the second element of the first argument list is evaluated and the value is returned. If the first element evaluates to NIL then the same algorithm is applied to the second argument. If non of these arguments provide a first argument that evaluates to a non-NIL value then NIL is returned as the result.

Example:

input	*output*
$(COND\ \ ((CDR\ \ '(A))\ \ 'B)\ \ ((CAR\ \ '(A))\ \ 'C)\ \ (T\ \ 'D))$	C

Function: EQUAL

Number of arguments: 2

Type of arguments: sexpr

Parameter passing: pre-evaluated

Explanation: Test whether the two arguments (after evaluation) are the same or not

Example:

input	*output*
$(EQUAL\ \ (CAR\ '(A\ \ B))\ \ 'A)$	T
$(EQUAL\ \ (CDR\ '(A\ \ B))\ \ 'B)$	NIL

Function: ATOM

Number of arguments: 1

Type of arguments: sexpr

Parameter passing: pre-evaluated

Explanation: Checks whether the argument is an atom or not

Example:

input	*output*
$(ATOM\ \ 'A)$	T
$(ATOM\ \ '(A))$	NIL
$(ATOM\ \ (CAR\ '(A)))$	T

The above introduced functions are just a small subset of all the functions available which are over 150 in number.

II.5.4 How to introduce new functions

LISP uses a unique notation, called the **λ-notation** for the function representation. The λ-notation is derived from the λ-calculus, a formalism invented by the logician Alonzo Church to model functions which are describable by algorithms. The λ-calculus is useful for discussing the concepts of function and function application. Since many algorithms compute functions and since function applications are simulated by procedure calls, the calculus is well suited for a purified discussion of procedures in programming languages.

In order to discuss the λ-calculus formalism we start with an example writing

$$f(x,y) \stackrel{\triangle}{=} x \times y + y$$

as the definition of the function f. With this we supposed to convey the following intention: f is the name of a function or rule; whenever f is supplied with two numeric arguments it is supposed to multiply those arguments add the result to the second. The resulting sum is the answer. Although on the left hand side of the definition, $f(x,y)$ is stated, actually what we meant with this is that we will use x and y as dummy arguments in the definition and f is the actual entity defined. To reflect these **stand alone** property functions λ-calculus introduces the abstract operator λ. With the aid of it the function f would be defined as:

$$f \stackrel{\triangle}{=} \lambda(x,y)(x \times y + y)$$

Now if we want to evaluate $f(1,2)$ we write

$$\underbrace{[\lambda(x,y)(x \times y + y)]}_{f}\ (1,2)$$

In LISP λ is represented by the built in atom $LAMBDA$, so, if $f(1,2)$ is to be calculated we write the following LISP expression:

$$((LAMBDA\ \ (X\ \ Y)\ \ (PLUS\ \ (TIMES\ \ X\ \ Y)\ \ Y))\ \ 1\ \ 2)$$

The value after LISP evaluates this sexpr will be 4.

It is possible to assign a name to a $LAMBDA$ expression: so, if this atom is utilized as the first element of a list which is to be evaluated then the associated $LAMBDA$ is recalled and used for the evaluation. It is also possible to declare these user defined functions to receive their arguments pre-evaluated or non-pre-evaluated. There exist many other details on the way of evaluation on which the user can have a control, but we consider these out of the scope of this work and instruct the reader who has further interest, to consult a LISP text book; an extensive reference list can be found in the reference section. Before we finish this section on LISP we will give a standard example: How to define the factorial function? Recall the recursive definition of the factorial:

$$\begin{aligned} n! &\stackrel{\triangle}{=} n \cdot (n-1)! \quad , \quad n > 0 \\ 0! &\stackrel{\triangle}{=} 0 \end{aligned}$$

```
(PUTD 'FACTORIAL 'EXPR
      '(LAMBDA (N)
        (COND ((EQUAL N 0) 1)
              (T (TIMES N (FACTORIAL (PLUS N -1)))))))
```

Here $PUTD$ is the function that takes three arguments which are all pre-evaluated:

1. An sexpr that shall evaluate to an identifier atom that will be used as the name of the function.

2. An sexpr that shall evaluate to the type of the function. That means how the arguments shall be passed. $EXPR$ stands for pre-evaluation.

3. An sexpr that shall evalute to a λ-expression which is the definition of the function.

LISP has an additional feature, which makes it unique: The property list. It is possible to associate **any** sexpr with an "indicator" (which has to be an atom) to an atom. To do this there is no need for a pre-allocation. In most LISPs this mechanism is governed by a **hashing** system which enables instant acquisition. The functions related are PUT and GET. It is also possible to put a true T under an indicator, for Boolean purposes. LISP provides for this an easy way, namely, $FLAG$ and its query function $FLAGP$.

II.6 An Inside Look to REDUCE

As it is understood from the naming of this section our intention is **not** to make an introduction how to use REDUCE as an end user. For this purpose serves the REDUCE 3.3 manual and interested readers are kindly referred to read it. This section will cover the internal architecture of REDUCE on which no documentation is available. This knowledge is necessary to understand technically how an extension to REDUCE was written.

II.6.1 The Programing Language Used and the Source Organization

REDUCE is a huge program written in RLISP. As you may already have discovered, LISP is not very readable and comfortable while programming with it (*because of its structure which adopts a prefix syntax and a grouping with parenthesis*). Therefore, a language with a syntax that is more familiar was developed. The semantics of this language is exactly LISP. Therefore, one can (*and shall*) think of RLISP as a translational tool. In fact RLISP is actually implemented as a translator which is written in LISP. The syntax of RLISP is ALGOL like, which makes it very friendly looking. You can find the description of this syntax in BNF notation as an appendix. Since REDUCE incorporates this syntax, while bootstraping REDUCE, first there exist a primitive kernel of an RLISP translator and then REDUCE replaces this with a more sophisticated and property-list driven version. This is the first module of the REDUCE source code. The source of REDUCE consists of the following modules:

1. RLISP
2. ALG1 + ALG2
3. MATRIX
4. HEP (*High Energy Physics*)
5. FACTOR (*Factorisation*)
6. INT (*Integration*)

The first two items are compulsory to have a minimal REDUCE working. The remaining parts are optional. There exist other utility modules too. Among these are modules for

- Tracing

- Cross reference generation
- Fancy outputing (*like operand suppression, sub-index printing, etc.*)
- Exterior algebra calculation
- REDUCE output to TEXformat conversion
- Differential equation solving
- Integral equation solving
- Symbolic error term calculation (in series approximation)

II.6.2 The Environments REDUCE is Providing

As it was said in the previous section that LISP does not make a difference between the "program" and the "data". This makes it possible to use REDUCE to modify or enhance itself. REDUCE provides the user with two environments, namely the **ALGEBRAIC** and the **SYMBOLIC** environments. In the ALGEBRAIC mode only certain types of input are understood, the READ-EVAL loop is based on an algebraic evaluation scheme. Here, for example, an atom which was not predefined stands for itself (*ie. evaluates to itself*), and an input like

$$A + A;$$

makes sense, the result is $2 * A$. The main user group will remain in this environment forever and will be satisfied. The complete language recognized in this environment is explained in the "REDUCE 3.x Users Manual". In the SYMBOLIC environment or so called SYMBOLIC mode, however, it is possible to extend REDUCE's syntax, semantic, add facilities to handle new algebraic data types (*for example vectorial quantities and operations on them*) and improve the i/o handling. Mainly all the source code of REDUCE is a part of this environment, that means all the functions, variables that make up REDUCE are reachable only in this environment. The environment that the user is in can be set by evaluting the keyword expressions `ALGEBRAIC;` and SYMBOLIC;. It is also possible to shift from the RLISP kind of syntax to the plain LISP syntax while remaining in the SYMBOLIC environment. To do this one has to evaluate the keyword expression `END;`. The way to return is evaluating the LISP expression `(BEGIN)`. This will take the user into the RLISP type syntax, **but always** back to the ALGEBRAIC environment. From now

on, unless otherwise said the reader shall assume the RLISP syntax when any of the two environments (modes) of REDUCE is mentioned.

II.6.3 Input

In REDUCE (both in the SYMBOLIC and the ALGEBRAIC mode) an input may be one of the following ended with a terminator.

- An atom : *like* `A, RES, X, 1`
- A functional expression : *like* `TRON(), FACTORIAL(M**2+1)`
- An infix expression : *like* `A+2*B, R:=(2+SIN(X))**3`
- A special syntax expression which starts with a keyword : *like* `IF, WHILE, FOR, OUT, ON, OFF`

A **terminator** is either a (`;`) semicolon or a (`$`) Dollar sign. The semicolon has the meaning "evaluate the input, find the result and do display (or print) the result". The Dollar sign means "evaluate the input find but do not display the result". Especially in the ALGEBRAIC mode if the output is long and just an intermediate result the user has the chance to avoid the time consumption for the printing process.

All the inputs are translated to their LISP equivalents; the first three kinds of items will be translated according to the RLISP ⇒ LISP translation rules (see appendix). The fourth item is somewhat different. When the REDUCE's input parsing function `XREAD()` encounters a keyword which has under the indicator `STAT` a property (lets say `F`) then `F` is expected to be the function that takes the parsing over. Immediately the function (in this case `F`) takes over and with the aid of two other functions of REDUCE it can carry out the parsing process. The sexpr that shall be returned as the result of that specific expression is totally generated by this function. The two functions which are provided to have an access to the token stream are:

- `SCAN()`
- `XREAD()`

The first returns the first untouched token in the parsed expression (the so called token stream) and removes it from the token stream. Also this token becomes available as the value of the global atom `CURSYM!*`. The second (surprisingly) is the `XREAD()` function

itself. LISP is a recursive language and it is quite common that a function calls itself. This enables nesting of expressions. So, it is possible to parse an input like

```
A := K + IF K>0 THEN (1-K)**2 ELSE K**3;
```

In this example `XREAD()` is invoked several times recursively. While it is at `K>0`, `(1-K)**2` and at `K**3` a new level of `XREAD()` is entered. `XREAD()` has to know where to stop. We know already that a ultimate stop is the terminator. But if that would be the only one the second level which is entered while parsing `K>0` will attempt to parse the line to its end (ie. up to the semicolon). After the 0 without knowing anything about a top level IF he would parse the token `THEN` and since `K>0 THEN` by itself is not a sensible expression the second call of `XREAD()` would terminate with an error. Therefore, these special (intermediate) keywords where `XREAD()` has to stop at (the so called delimeters) are flagged with the indicator `DELIM` and any level of `XREAD()` only parses up to an unconsumed delimiter or a terminator. REDUCE was (as it was said before) written incremental. This caused some of its functions to be coded *ad hoc*, `XREAD()` seems to be among those. `XREAD()` takes a single argument which can be `NIL` `T` or some of the keywords which tell it what the father `XREAD()` that has initialized is parsing (*eg. a parsing of* `IF`*,* `FOR`*,* `MAT`*, Grouping,* `PROC`).

If a sequence of characters is not going to be translated as it is and shall form a single token (*like for* <<,<, :=) then they are introduced to the system by a `NEWTOK()` function call, an example shall clarify:

```
NEWTOK '((!:  !=)    SETQ    ! !:!=!  )
```

as seen `NEWTOK()` has only one argument which is a list of three members. The first is a list of individual characters which shall be matched in the input-character-stream, the second member is the atom that shall be returned as the token, the third is optional, it is the characters that shall be used for print out (*in the example above the assignment sign, when printed, is declared to have a blank before and after it*).

There exist many infix operators. These are recognized by an indicator `INFIX` on their property list with a property which is an integer, standing for the precedence. All of the infix operators are in the list which remains as the value of the global atom `PRECLIS!*`, in the order of precedence (lower precedence first, higher last). So, if someone is going to introduce an infix operator by itself this list shall be updated. Infix operators can be flagged according to their arity, associativity `NARY`, `RIGHT`, `ALT`. For details dig through the source code. The statements `INFIX` and `PRECEDENCE` aid the user in defining his own infix operators.

II.6.4 The Evaluation

Before explaining the sophisticated scheme of evaluation we would like to introduce the data structure that is used in the interior of REDUCE (the average user knows nothing about it and also has not to!). REDUCE has two internal formats on which it performs the manipulations

- The prefix representation
- Standard quotient representation

The **prefix representation** is in total conformity with the LISP functional call. With this we mean a list which has a first member representing the function (or operation) and the remaining members of the list are the arguments which are subject to that operation. These arguments can also be lists. Consider the algebraic expression

$$7X^2 + 24$$

The prefix expression that corresponds to it, which is generated after the parsing process is

```
(PLUS (TIMES 7 (EXPT X 2)) 24)
```

Although this is much more convenient to work with, compared to the infix input expression it does not form the best basis for algebraic manipulation. The atoms which represent the operations have to be introduced; the variables and coefficients are not distinguished; adopt a list type of data structure which is more expansive in memory units compared to a dotted pair structure; etc. Before giving a more detailed explanation, first, let us put down the representation of the example above in the second internal data structure form, the so called **standard quotient**:

```
((((X . 2) .  7)  .  24)  .  1)
```

In this representation an expression in n variables $f(x_1, x_2, \ldots, x_n)$ is expressed as a power series in **one** variable whose coefficients are functions of the remaining $n-1$ variables.

$$f(x_1, x_2, \ldots, x_n) = \sum_{i=0}^{m_1} f_i(x_2, \ldots, x_n) x_1^i$$

Here m_1 is, as expected, the polynomial degree of the function f in the variable x_1. Of course the coefficient functions are expanded applying the same scheme until the coefficients are integers. This is called in REDUCE the **standard form**.The dotted expression

of two standard forms, forms the **standard quotient** which represents the division (a rational function) of two multivariate polynomials.

Since there exist functional representations like $\cos(x)$ or any user defined closed function, REDUCE should provide a handle to incorporate this. This is done. In the definition of a standard quotient in the Backus-Naur form is stated below:

<standard-form> ::= () | <integer>| (<standard-term> . <standard-form>)

<standard-term> ::= (<standard-power> . <standard-form>)

<standard-power> ::= (<kernel> . <non-zero positive integer>)

<kernel> ::= () | <variable>| <standard-form> |

(<operator> . <list of simplified arguments>)

One has to pay attention to keep track of which stage of evaluation is expected to return what kind of internal representation. That means a prefix expression or a standard quotient (from now on will be abbreviated as **SQ**). There are constructor and selector functions (usually implemented as SMACRO's) which serve to construct primitive SQ structures and reach the leading coefficient, reductum, degree etc. A table of all these functions is given on the following page.

S E L E C T O R S

Operator	Domain	Result	Explanation	Performed Func.
DENR	SQ	SF	Denominator	CDR
NUMR	SQ	SF	Numerator	CAR
LC	SF	SF	Leading Coefficient	CDAR
LDEG	SF	integer	Leading degree	CDAAR
LPOW	SF	SP	Leading power	CAAR
MVAR	SF	kernel	Main variable	CAAAR
RED	SF	SF	Reductum	CDR
TC	ST	SF	Coefficient	CDR
TDEG	ST	integer	Degree	CDAR
TPOW	ST	SP	Power part	CAR
PDEG	SP	integer	Degree	CDR

C O N S T R U C T O R S

Infix Oper.	Left Domain	Right Domain	Result	Explanation	Perf. Func.
.+	ST	SF	SF	Add a term to (polynomial)	CONS
./	SF	SF	SQ	Divide two (polynomials) and get a (rational)	CONS
.*	SP	SF	ST	Multiply a power by a coefficient form to produce a term	CONS

In addition to these there exist also conversion functions between various forms. Functions which upgrade all types to SFs and SQs are present. A mnemonically nice convention is adopted: An abbreviation in which `K`, `T`, `F`, `Q` stand for kernel, standard term, standard form, standard quotient respectively are used. Any conversion function starts with a `!*` and the numeral `2` stands for the word **to**. The conversion function is named according the concatenation of `!*`◯`2`△ where ◯ and △ are the abbreviation letters for the structures the conversion takes places **from** and **to** respectively. As an example, `!*P2Q` is the function which converts a standard power to a standard quotient.

The functions which perform algebraic operations on this internal structure can be found in the source code, these parts are comparably better documented.

The main evaluation function is `SIMP()` but if it is called for the first time, that means if you know that no prior call has occurred then use `SIMP!*()`. These functions take a prefix expression, evaluate it and return a SQ result. There exists a function `PREPSQX()` which converts a SQ into a prefix expression. The combined effect (without mentioning some other works which is performed) can be obtained by a call to the function `REVAL()`. `SIMP()`, the main evaluation function makes a case discrimination looking on various flags and properties of the first member of the list (the prefix expression) namely the function. This is done before any arguments are touched with a pre-evaluation purpose. If this atom has a property under an indicator `SIMPFN` then without doing anything that property (which is expected to be the name of a defined function) is called receiving as argument the list of the actual arguments. If the atom is also flagged `FULL` then the name of the function is also included (that means exactly the prefix expression which was going to be evaluated is passed to the function. It is possible to define the atom (the first element of the prefix expression) to be the name of the evaluating function. There is a short cut provided for this purpose, if that atom has under the indicator `SIMPFN` the property `SIMPIDEN` then automatically a function that has the name of that atom is invoked. It is important to stress on the fact that any function call due to such a `SIMPFN` *take over* has to return a SQ as a result.

Another possible case is that the first element of the prefix expression is flagged `OPFN`, which is the case for all operators declared in the ALGEBRAIC mode. Then automatically the rest of the members are algebraically evaluated (with calls to `REVAL()`) then a check for a possible pattern matching is performed.

There exist additional case discriminations for array element evaluation, matrix algebra which we will not enter here in details.

II.6.5 Output

The printing of an internal algebraic structure is done by the function `MATHPRINT()`. This function takes a single argument, an prefix algebraic expression, and prints it on the current output channel. This channel can be the users terminal screen, the printer or any file. How the output is redirected and how the user can control the way the expressions are printed is explained in details in the users manual.

`MATHPRINT()` immediately initiates a call to the function `MAPRINT()` which has two arguments. This function is a highly recursive one and provides a handle to pass the control over to a function while a prefix expression is printed. The first of its arguments is the prefix expression to be printed, the second is the precedence number of the call. This second argument is the way to inform any child function call of the father call's operator precedence. So, if a lower precedence operation is going to be printed then it is encapsuled in parenthesis. A greater number in the second argument means a higher precedence. To resume control over a prefix expression printing the first argument (which has to be an atom) shall have a function name in its property list under the indicator `PRIFN`. When this is the case this function takes over the printing post and receives the prefix expression as the argument. It is a bug of REDUCE that the precedence number is not passed forward. Also it is worth to mention that in the preceding versions of REDUCE the indicator was not `PRIFN` but `SPECPRN` and the argument passed was the `CDR` part of the prefix expression (the first element was removed). One can think of the REDUCE output media as a 2-dimensional cartesian-like coordinated system. There exist some functions, flags, global variables which serve for the output process to this hypothetical media. The actual printing is carried out by more complicated printing procedures which are based on a buffered output mechanism, but normally, in the case of extending the printing abilities of REDUCE one does not have to worry about this core behavior. In order to print an atom (to this hypothetical page) one calls the function `PRIN2!*()` which serves the same purpose as `PRIN2()` but performs also a bookkeeping. Setting the flag `OBRKP!*` to `NIL` disables line breaks in the case of an `ON LIST` if the `PLUS` or `DIFFERENCE` is inside an operator argument. The value of the global atom `YCOORD!*` controls on which line (or row) the printing is done. So, for example, in the case of a subscript this value shall be

decreased by one, and in the case of an superscript it shall be increased by one. There is (theoretically) no limitation on the value `YCOORD!*` can have but while `YCOORD!*` is being increased or decreased relatively the values `YMAX!*` and `YMIN!*` shall be checked and if necessary updated. `POSN!*` holds the first empty position to print in the line. The function `TERPRI!*()` is very similar to the ordinary `TERPRI()` but takes an argument. If it is `NIL` then after the lines are printed an additional blank line is printed else not. In both cases all the lines that are in the buffer, that means the lines between `YMIN!*` and `YMAX!*`, are printed. The global `ORIG!*` holds the value where `POSN!*` shall start after a `TERPRI!*()` has occurred.

CHAPTER III

PHYSICAL APPLICATIONS

III.1 A Study of Stationary, Axially Symmetric Space-time Geometries Satisfying Modified Double Duality Equations Using the Exterior Calculus Package $\mathbf{X^TR}$ for REDUCE [1]

The gravitational fields in four dimensional space-time are described in terms of a Lorentzian metric tensor g, and an independent metric compatible connection ω that is a rule for parallelly transporting tensorial quantities along curves in space-time. The field equations satisfied by g and ω are obtained from a locally Lorentz invariant action

$$\int_{M_4} L(g, w)$$

by a well defined variational principle. Einstein's gravitational theory is based on the single non-trivial curvature invariant linear in curvature components. Thus we consider the Einstein-Hilbert 4-action

$$I_0 = -\frac{1}{2\kappa^2} \int_{M_4} R_{ab} \wedge {}^*(e^b \wedge e^b) \tag{III.1}$$

where $\kappa^2 = 8\pi G/c^3$ is the universal gravitational constant. Independent connection variations of I_0 imply that the connection is Levi-Civita. Then the source-free Einstein's equations obtained by coframe variations of I_0 involve at most second order partial derivatives of the components of the metric tensor.

Various types of gravitational theories that generalise Einstein's theory were being discussed during the recent years. Conceptually the simplest way to generalise Einstein's

[0] A part of this section is published as [31]

theory is to write down the gravitational action using higher order curvature invariants. A generalisation that received a lot of attention was due to Stephenson, Kilmister, Yang and several others. [12, 13, 14] In this theory the gravitational action is written in analogy with Yang-Mills type gauge theories, so that the field equations are obtained by varying a quadratic curvature invariant. A generalised theory of gravity of this fashion is described by the 4-action [21, 22]

$$I_1 = k \int_{M_4} R_{ab} \wedge {}^*R^{ab} \tag{III.2}$$

Here k is a dimensionless coupling constant. The field equations obtained by varying I_1, provided the space-time torsion is constrained to be zero, involves at most fourth order partial derivatives of the metric tensor components. Nevertheless, it is still an interesting theory. Firstly because it is motivated by an analogy with Yang-Mills theory, so that the gauge structure is manifest. Secondly, in a perturbative approach to field quantisation it leads to a renormalisable, albeit in general, non-unitary quantum gravity. [15] But even at the classical level the theory has the problem of admitting non-physical solutions along with the physically admissible ones. [16, 17] Therefore, in an attempt to constrain the quadratic theory further it was suggested to add on the Einstein-Hilbert action. [18] If it is further allowed to add a cosmological constant, then some remarkable simplifications follow. For a definite value of the cosmological constant, there is a set of modified double dual curvature equations whose integrability conditions give precisely the variational field equations. [19, 20] Thus, static spherically symmetric geometries with dynamical torsion are determined by solving the modified double duality equations. These geometries also solve the field equations of the Poincaré gauge field theory of Hehl and his collaborators, [21, 22] as well as the Bach-Weyl equations. [23]

In this section we study stationary, axially symmetric solutions to the modified double duality equations. [24] We also start from the Kerr-de Sitter metric, however, we aim in particular to determine geometries that in the limit of vanishing rotation parameter would go to the static, spherically symmetric solutions given in reference [19]. The long and tedious algebraic manipulations that led to a system of coupled ordinary differential equations are performed using $\mathrm{X^TR}$.

III.1.1 Stationary, axisymmetric double dual curvatures

We consider solutions described by the Kerr de Sitter metric [25]

$$
\begin{aligned}
g &= -\frac{\delta_r^2}{r^2+a^2\cos^2\theta}\left[\frac{dt-a\sin^2\theta d\varphi}{1-\frac{1}{3}ka^2}\right]^2 \\
&\quad +(r^2+a^2\cos^2\theta)\left\{\frac{dr^2}{\delta_r^2}+\frac{d\theta^2}{1-\frac{1}{3}ka^2\cos^2\theta}\right\} \\
&\quad +\sin^2\theta\left(\frac{1-\frac{1}{3}ka^2\cos^2\theta}{r^2+a^2\cos^2\theta}\right)\left[\frac{adt-(r^2+a^2)d\varphi}{1-\frac{1}{3}ka^2}\right]^2
\end{aligned}
\tag{III.3}
$$

expressed in terms of Boyer-Lindquist coordinates (t, r, θ, φ). [26] We define the following for later convenience:

$$
\begin{aligned}
(T_1)^2 &\triangleq 1-\frac{k}{3}a^2 \\
(T_2)^2 &\triangleq r^2+a^2\mu^2 \\
(T_3)^2 &\triangleq 1-\frac{k}{3}a^2\mu^2 \\
M^2 &\triangleq 1-\mu^2 \\
(\delta_r)^2 &\triangleq \frac{k}{3}(r^4+a^2r^2)+r^2-2mr+a^2
\end{aligned}
$$

where $\mu=\cos\theta$. m shows the Schwarzschild mass, a the rotation, and k is a real parameter. We take the orthonormal co-frames

$$
\begin{aligned}
e^0 &= \frac{\delta_r}{(T_1)^2T_2}\,dt-\frac{aM^2\delta_r}{(T_1)^2T_2}\,d\varphi \\
e^1 &= \frac{aMT_3}{(T_1)^2T_2}\,dt-\frac{MT_3(a^2+r^2)}{(T_1)^2T_2}\,d\varphi \\
e^2 &= \frac{T_2}{\delta_r}\,dr \\
e^3 &= \frac{T_2}{MT_3}\,d\mu
\end{aligned}
\tag{III.4}
$$

Then the Levi-Civita connection 1-forms are found as follows:

$$
\begin{aligned}
\dot{\omega}^0{}_1 &= \frac{arMT_3}{(T_2)^3}\,e^2+\frac{a\mu\delta_r}{(T_2)^3}\,e^3 \\
\dot{\omega}^0{}_2 &= \frac{1}{3(T_2)^5\delta_r}[a^2\mu^2(3a^2\mu^2\delta_r\frac{\partial\delta_r}{\partial r}+ka^2r^3-3a^2r+3kr^5+3r^3)+3mr^4 \\
&\qquad\qquad -3a^2r^3+kr^7]\,e^0 \\
&\quad +\frac{MT_1ar}{(T_2)^3}\,e^1 \\
\dot{\omega}^0{}_3 &= \frac{a^2\mu MT_3}{(T_2)^3}\,e^0-\frac{a\mu\delta_r}{(T_2)^3}\,e^1 \\
\dot{\omega}^1{}_2 &= -\frac{arMT_3}{(T_2)^3}\,e^0+\frac{r\delta_r}{(T_2)^3}\,e^1
\end{aligned}
\tag{III.5}
$$

$$\dot{\omega}^1{}_3 = \frac{a\mu\delta_r}{(T_2)^3}\, e^0 + \frac{\mu}{3MT_3(T_2)^3}[ka^2(\mu^2(T_2)^2 - M^2r^2) - 3(a^2+r^2)]\, e^1$$
$$\dot{\omega}^2{}_3 = \frac{a^2\mu MT_3}{(T_2)^3}\, e^2 - \frac{r\delta_r}{(T_2)^3}\, e^3$$

We will construct solutions such that the full connection 1-forms are of the following form:

$$\begin{aligned}
\omega_{01} &= \Omega_1 e^2 + \Omega_2 e^3 \qquad \text{(III.6)}\\
\omega_{02} &= \Omega_3 e^0 + \Omega_4 e^1\\
\omega_{03} &= \Omega_5 e^0 + \Omega_6 e^1\\
\omega_{12} &= \Omega_7 e^0 + \Omega_8 e^1\\
\omega_{13} &= \Omega_9 e^0 + \Omega_{10} e^1\\
\omega_{23} &= \Omega_{11} e^2 + \Omega_{12} e^3
\end{aligned}$$

The modified double duality equations are

$$^*(R^{ab} + \frac{\lambda}{2}\, e^a \wedge e^b) = -(R^{ab} + \frac{\lambda}{2}\, e^a \wedge e^b)^* \qquad \text{(III.7)}$$

where λ is an arbitrary real parameter. These equations may also be written in the form

$$^*R_{ab} + R^*_{ab} = \lambda\, ^*(e_a \wedge e_b) \qquad \text{(III.8)}$$

Manipulating the structure equations to construct the left-hand-side of these equations we obtain:

$$\begin{aligned}
^*R_{01} + R^*_{01} &= S_1\, e^0 \wedge e^1 + S_2\, e^2 \wedge e^3 \qquad \text{(III.9)}\\
^*R_{02} + R^*_{02} &= S_3\, e^0 \wedge e^2 + S_4\, e^0 \wedge e^3 + S_5\, e^1 \wedge e^2 + S_6\, e^1 \wedge e^3\\
^*R_{03} + R^*_{03} &= S_7\, e^0 \wedge e^2 + S_8\, e^0 \wedge e^3 + S_9\, e^1 \wedge e^2 + S_{10}\, e^1 \wedge e^3\\
^*R_{12} + R^*_{12} &= -S_{10}\, e^0 \wedge e^2 + S_9\, e^0 \wedge e^3 - S_8\, e^1 \wedge e^2 + S_7\, e^1 \wedge e^3\\
^*R_{13} + R^*_{13} &= S_6\, e^0 \wedge e^2 - S_5\, e^0 \wedge e^3 + S_4\, e^1 \wedge e^2 - S_3\, e^1 \wedge e^3\\
^*R_{23} + R^*_{23} &= S_2\, e^0 \wedge e^1 - S_1\, e^2 \wedge e^3
\end{aligned}$$

The functions $S_1, \ldots, S_{10}$ will be given explicitly below. In terms of these functions the modified double duality equations (III.8) read

$$S_2 = -S_6 = S_9 = \lambda \qquad \text{(III.10)}$$

$$S_1 = S_3 = S_4 = S_5 = S_7 = S_8 = S_{10} = 0$$

These functions classify into two distinct sets. Six of them, namely $S_1, S_2, S_4, S_5, S_8, S_9$ are of the generic form:

$$\frac{1}{(T_2)^3}\Big\{MT_3(a^2\mu+(T_2)^2\partial_\mu)\Omega_A+\delta_r(r+(T_2)^2\partial_r)\Omega_B \\ +(T_2)^3[\Omega_C\Omega_D+\Omega_E\Omega_F+\Omega_G\Omega_H+\Omega_I\Omega_J]\Big\} \qquad \text{(III.11)}$$

The actual expressions we give as a table:

	A	B	C·D	E·F	G·H	I·J
S_1	-1	2	$8\cdot 9$	$-7\cdot 10$	$6\cdot 3$	$-5\cdot 4$
S_2	-11	12	$8\cdot 3$	$-7\cdot 4$	$-6\cdot 9$	$5\cdot 10$
S_4	9	-4	$-8\cdot 1$	$-7\cdot 12$	$-6\cdot 11$	$5\cdot 2$
S_5	3	10	$-8\cdot 11$	$7\cdot 2$	$6\cdot 1$	$5\cdot 12$
S_8	-7	-6	$4\cdot 11$	$-3\cdot 2$	$-1\cdot 10$	$-12\cdot 9$
S_9	5	-8	$-4\cdot 1$	$-3\cdot 12$	$2\cdot 9$	$-11\cdot 10$

As an example the first horizontal line in the table means:

$$S_1 = \frac{1}{(T_2)^3}\Big\{-MT_3(a^2\mu+(T_2)^2\partial_\mu)\Omega_1+\delta_r(r+(T_2)^2\partial_r)\Omega_2 \\ +(T_2)^3[\Omega_8\Omega_9-\Omega_7\Omega_{10}+\Omega_6\Omega_3-\Omega_5\Omega_4]\Big\}$$

The second set of functions S_3, S_6, S_7, S_{10} are of the generic form

$$\frac{1}{(T_2)^3T_3}\ \{\ MT_3[-r\delta_r+(T_2)^2\partial_r(\delta_r)+(T_2)^2\delta_r\partial_r]\Omega_B+ \\ [\mu(\tfrac{1}{3}kr^2a^2(2\mu^2-1)-(r^2+a^2)+\tfrac{1}{3}ka^4\mu^4)+M^2(T_3)^2(T_2)^2\partial_\mu]\Omega_A \\ +2aMT_3[\mu\delta_r\Omega_C+rMT_3\Omega_D] \\ +(T_2)^3T_3\ [\Omega_E\Omega_F+\Omega_G\Omega_H+\Omega_I\Omega_J+\Omega_K\Omega_L]\ \}$$

The actual expressions are read from the following table:

	A	B	C	D	E·F	G·H	I·J	K·L
S_3	4	9	-3	-10	$8\cdot 2$	$-7\cdot 11$	$6\cdot 12$	$5\cdot 1$
S_6	10	-3	-9	4	$-8\cdot 12$	$-7\cdot 1$	$6\cdot 2$	$-5\cdot 11$
S_7	6	-7	-5	8	$-4\cdot 12$	$-3\cdot 1$	$2\cdot 10$	$-11\cdot 9$
S_{10}	-8	-5	7	6	$-4\cdot 2$	$3\cdot 11$	$-1\cdot 9$	$-12\cdot 10$

III.1.2 The Solution

The aim now is to find a particular solution family of the equations S_i given in the previous subsection.

We seek to find a solution family where we have the even indiced Ω_i's identically set equal to zero. That means

$$\Omega_2 \equiv \Omega_4 \equiv \Omega_6 \equiv \Omega_8 \equiv \Omega_{10} \equiv \Omega_{12} \equiv 0$$

III.1.2.1 The First type of S_i

Under this assumption the first type of the S_i's $_{i=1,2,4,5,8,9}$ are of the following form:

$$\frac{MT_3}{T_2^3}(a^2\mu + T_2^2\partial_\mu)\Omega_B \tag{III.12}$$

For S_1, S_4, S_5, S_8 which are equal to zero the integration is easily done:

$$\Omega_a = f_1 F_a(r) \qquad\qquad {}_{a=1,3,7,9} \tag{III.13}$$

provided that $T_2, T_3 \neq 0$ and $f_1 \stackrel{\triangle}{=} T_2^{-1}$.

For S_2, S_9 the RHS is not zero but λ. The solution is obtained in terms of two elliptic integrals:

$$\Omega_5 = f_1 F_5(r) - f_2 \tag{III.14}$$

$$\Omega_{11} = f_1 F_{11}(r) + f_2 \tag{III.15}$$

Where

$$\begin{aligned} f_2 &\stackrel{\triangle}{=} -\frac{\lambda}{T_2}(r^2 E_{Elliptic_1} + a^2 E_{Elliptic_2}) \\ E_{Elliptic_1} &\stackrel{\triangle}{=} \int \frac{1}{\sqrt{(1-\mu^2)(1-\frac{k}{3}a^2\mu^2)}} \\ E_{Elliptic_2} &\stackrel{\triangle}{=} \int \frac{\mu^2}{\sqrt{(1-\mu^2)(1-\frac{k}{3}a^2\mu^2)}} \end{aligned}$$

III.1.2.2 The Second type of S_i

With the assumption made, the second type of the S_i's become

$$f_3\Omega_B + f_4\frac{\partial\Omega_B}{\partial r} + f_5\Omega_C + f_6[\Omega_G\Omega_H + \Omega_K\Omega_L] \tag{III.16}$$

For sake of convenience we defined the following:

$$\begin{aligned} f_3 &\triangleq -r\delta_r + T_2^2(\delta_r)_r \\ f_4 &\triangleq T_2^2\delta_r \\ f_5 &\triangleq 2a\mu\delta_r \\ f_6 &\triangleq \frac{T_2^3}{M} \\ f_7 &\triangleq -\frac{\lambda T_2^3}{M} \end{aligned} \tag{III.17}$$

For S_3, S_7, S_{10} the RHS of (III.16) is equal to zero and for S_6 it is equal to f_7.

Now the scheme is to plug in the results for the Ω_a's which were obtained in the previous subsection into these equation and try to find a solution for the arbitrary functions $F_a(r)$. This will complete the solution. In order to simplify the equations we will make a restricting assumption and set two of the arbitrary functions to be $F_1(r) \equiv 1$ and $F_{11}(r) \equiv 0$. So there remains four functions namely F_3, F_5, F_7, F_9 to be determined. But, unfortunately, this attempt fails, even if we consider the limit $a = 0$. So it is **not** possible to find a global solution which is valid for all the space, with the assumed ansatz. Therefore we sacrifice the idea of "globality" and restrict ourself to "strips" on which μ is constant. This will have the consequence that we will not have the right to inspect the solution for the μ variation. **From now on we fix μ to be a parameter and not a free variable**.

We define the followings

$$G_1 \triangleq F_3 + F_7$$

$$G_3 \triangleq F_3 - F_7 \tag{III.18}$$

$$G_2 \triangleq F_5 + F_9$$

$$G_4 \triangleq F_5 - F_9 \tag{III.19}$$

$$\begin{aligned} h_1 &\triangleq g_1 + g_4 \\ h_2 &\triangleq g_2 + g_3 \\ h_3 &\triangleq g_4 - g_1 \\ h_4 &\triangleq g_2 - g_3 \\ h_5 &\triangleq g_8 - g_9 + g_5 \\ h_6 &\triangleq g_6 + g_7 + g_3 \\ h_7 &\triangleq g_8 - g_9 - g_5 \\ h_8 &\triangleq g_6 + g_7 - g_3 \end{aligned} \tag{III.20}$$

Where

$$
\begin{aligned}
g_1 &= -\frac{r}{r^2+A_1^2} + \frac{A_2r^5+A_3r^3+A_4r^2+A_5r-A_6}{(r^2+A_1^2)[A_2r^4+A_7r^2-2A_4r+a^2]} \\
g_2 &= \frac{A_8}{r^2+A_1^2} \\
g_3 &= A_9\frac{A_{10}r^2+A_{11}}{[A_2r^4+A_7r^2-2A_4r+a^2]^{\frac{1}{2}}} \\
g_4 &= \frac{A_{12}}{[A_2r^4+A_7r^2-2A_4r+a^2]^{\frac{1}{2}}} \\
g_5 &= \frac{A_8+A_{13}}{r^2+A_1^2}[A_{10}r^2+A_{11}] \\
g_6 &= A_{13}\frac{[A_2r^5+A_3r^3+A_4r^2+A_5r-A_6]\cdot[A_{10}r^2+A_{11}]}{(r^2+A_1^2)[A_2r^4+A_7r^2-2A_4r+a^2]} \\
g_7 &= A_{13}\left\{-\left(\frac{r}{r^2+A_1^2}\right)\cdot[A_{10}r^2+A_{11}]+2A_{10}r\right\} \\
g_8 &= A_{14}\frac{[A_{10}r^2+A_{11}]^2}{[A_2r^4+A_7r^2-2A_4r+a^2]^{\frac{1}{2}}} \\
g_9 &= -\frac{A_{15}(r^2+A_1^2)}{[A_2r^4+A_7r^2-2A_4r+a^2]^{\frac{1}{2}}}
\end{aligned}
$$

All A_i's are independent of the variable r and are defined as:

$$
\begin{aligned}
A_1^2 &\triangleq a^2\mu^2 & A_2 &\triangleq \tfrac{1}{3}k \\
A_3 &\triangleq \tfrac{2}{3}ka^2\mu^2 & A_4 &\triangleq m \\
A_5 &\triangleq a^2[\mu^2(\tfrac{1}{3}ka^2+1)-1] & A_6 &\triangleq a^2\mu^2m \\
A_7 &\triangleq \tfrac{1}{3}ka^2+1 & A_8 &\triangleq 2a\mu \\
A_9 &\triangleq -\frac{\lambda}{\sqrt{1-\mu^2}} & A_{10} &\triangleq E_{Elliptic_1} \\
A_{11} &\triangleq a^2E_{Elliptic_2} & A_{12} &\triangleq \frac{1}{\sqrt{1-\mu^2}} \\
A_{13} &\triangleq -\lambda & A_{14} &\triangleq \frac{\lambda^2}{\sqrt{1-\mu^2}} \\
A_{15} &\triangleq \sqrt{\frac{\lambda}{1-\mu^2}}
\end{aligned}
$$

It is easily verified that with the above definitions the equations mentioned boil down to two non-homogeneous linear differential equation pairs of first order with non-constant coefficients, namely:

$$
G_1' = -h_1G_1 - h_2G_2 + h_5
$$

$$G_2' = h_2G_1 - h_1G_2 + h_6 \tag{III.21}$$

$$G_3' = h_3G_3 + h_4G_4 + h_7$$

$$G_4' = -h_4G_3 + h_3G_4 + h_8 \tag{III.22}$$

These coupled differential equations are of the form:

$$X'(r) = \xi(r)X(r) + \eta(r)Y(r) + \rho(r) \tag{III.23}$$

$$Y'(r) = -\eta(r)X(r) + \xi(r)Y(r) + \sigma(r) \tag{III.24}$$

which addmits a solution:

$$\begin{aligned} X &= R\cos\phi\left[K_1 + \int \Lambda_1\, dr\right] + R\sin\phi\left[K_2 + \int \Lambda_2\, dr\right] \\ Y &= -R\sin\phi\left[K_1 + \int \Lambda_1\, dr\right] + R\cos\phi\left[K_2 + \int \Lambda_2\, dr\right] \end{aligned} \tag{III.25}$$

Where

$$R \triangleq \exp(\int \xi\, dr) \qquad \phi \triangleq \int \eta\, dr$$

and K_1 and K_2 are arbitrary integration constants.

$$\Lambda_1 \triangleq \frac{1}{R}(\rho\cos\phi - \sigma\sin\phi) \qquad \Lambda_2 \triangleq \frac{1}{R}(\sigma\cos\phi + \rho\sin\phi)$$

When we attempt to calculate R and ϕ with the functions encountered in (III.21) and (III.22) we have:

{*Here and below for* $\pm$ *take* $+$ *sign for pair (III.21) and* $-$ *sign for (III.22)* }

$$\begin{aligned} R &= \exp(-\int g_1 \pm g_4\, dr) \\ \phi &= -(\int g_3 \pm g_2\, dr) \end{aligned}$$

Performing partial fraction decomposition on g_1 and then integrating one obtains:

$$I_{g_1} \triangleq \int g_1\, dr \tag{III.26}$$

$$= \ln\left[K\,(r^2 + A_1^2)^{\frac{\alpha_1 - A_2}{2A_2}} \cdot (r - r_{11})^{\frac{\alpha_3}{A_2}} \cdot (r - r_{12})^{\frac{\alpha_4}{A_2}} \cdot (r - r_{21})^{\frac{\alpha_5}{A_2}} \cdot (r - r_{22})^{\frac{\alpha_6}{A_2}}\right] + \frac{\alpha_2}{A_1 A_2}\arctan\left(\frac{r}{A_1}\right) \tag{III.27}$$

Here K is the arbitrary integration constant, $r_{11}, r_{12}, r_{21}, r_{22}$ are the roots of the denominator of the LHS expression of the below given partial fraction decomposition and the α_i are are defined on the RHS.

$$\frac{A_2r^5 + A_3r^3 + A_4r^2 + A_5r - A_6}{(r^2 + A_1^2)[A_2r^4 + A_7r^2 - 2A_4r + a^2]} \equiv \frac{1}{A_2}\left[\frac{\alpha_1 r + \alpha_2}{r^2 + A_1^2} + \frac{\alpha_3}{r - r_{11}} + \frac{\alpha_4}{r - r_{12}} + \frac{\alpha_5}{r - r_{21}} + \frac{\alpha_6}{r - r_{22}}\right]$$

The second integral term in the calculation of R, namely $\int g_4\, dr$ is unfortunately elliptic, we will name it as

$$I_{Elliptic_1} \triangleq \int g_4\, dr \tag{III.28}$$

$$I_{g_2} \triangleq \int g_2\, dr = \frac{A_8}{A_1} \arctan\left(\frac{r}{A_1}\right) \tag{III.29}$$

The integral $\int g_3\, dr$ is elliptic too:

$$I_{Elliptic_2} \triangleq \int g_3\, dr \tag{III.30}$$

So with the result of (III.27), (III.29), (III.30),(III.28) we have

$$\begin{aligned} R &= \exp(-[I_{g_1} \pm I_{Elliptic_1}]) \\ \phi &= -(I_{Elliptic_2} \pm I_{g_2}) \end{aligned}$$

This completes the solution for the G_a's. F_a's are obtained by solving the set of two equations in two unknowns (III.18) and (III.19). Since Λ_1 and Λ_2 which are integrated in the solutions (III.25) contain elliptic integrals it is not possible no carry the calculation further in order to obtain a solution in finite terms. We put down the final solutions for Ω_i (i=1,3,5,7,9,11):

$$\boxed{\begin{aligned} \Omega_1 &= f_1 \\ \Omega_3 &= f_1 F_3 \\ \Omega_5 &= f_1 F_5 - f_2 \\ \Omega_7 &= f_1 F_7 \\ \Omega_9 &= f_1 F_9 \\ \Omega_{11} &= f_2 \end{aligned}} \tag{III.31}$$

III.1.3 The Torsion Ansatz

Now everything is in hand to put down the torsion 2-forms T^a. The torsion forms are determined by the contortion forms K_{ab}. As it is well known:

$$\begin{aligned} K_{ab} &= \omega_{ab} - \dot{\omega}_{ab} \\ T^a &= K^a_b \wedge e^b \end{aligned}$$

Using the definitions of $\dot{\omega}_{ab}$ in terms of $\dot{\Omega}_a$ stated previously and the results that we have right obtained we get:

$$\begin{aligned}
T^0 &= (\dot{\Omega}_3 - f_1 F_3)\, e^0 \wedge e^2 + (f_2 - f_1 F_5 + \dot{\Omega}_5)\, e^0 \wedge e^3 + (f_1 - \dot{\Omega}_1 + \dot{\Omega}_4)\, e^1 \wedge e^2 \\
&\quad - 2\dot{\Omega}_2\, e^1 \wedge e^3 \\
T^1 &= (f_1 F_7 + f_1 - 2\dot{\Omega}_1)\, e^0 \wedge e^2 + f_1 F_9\, e^0 \wedge e^3 - \dot{\Omega}_8\, e^1 \wedge e^2 - \dot{\Omega}_{10}\, e^1 \wedge e^3 \\
T^2 &= (\dot{\Omega}_1 - \dot{\Omega}_4 - f_1 F_7)\, e^0 \wedge e^1 + (f_2 + \dot{\Omega}_5)\, e^2 \wedge e^3 \\
T^3 &= -f_1 F_9\, e^0 \wedge e^1 + \dot{\Omega}_8\, e^2 \wedge e^3
\end{aligned}$$

III.2 Kaluza-Klein Reduction of Generalised Theories of Gravity and Non-minimal Gauge Couplings [2]

A different type of generalization of Einstein's gravitational theory was pioneered by Kaluza and Klein who, being motivated by a desire to find a formal unification among the fundamental long-ranged forces of nature, considered the Einstein-Hilbert action over a 5-dimensional space-time manifold. [28, 29] They discovered that the Einstein-Hilbert 5-action can be reduced to the coupled Einstein-Maxwell system over the actual 4-dimensional space-time manifold.

Electromagnetism is the only other long-ranged interaction in nature that is described by a classical field theory. The source-free Maxwell's equations are obtained by a variational principle from the 4-action

$$I_M = -\frac{1}{2e^2}\int_{M_4} F \wedge {}^*F \tag{III.32}$$

where $F = dA$ and e is the electric charge. The minimal coupling of electromagnetic fields to gravity defined by the 4-action $I_0 + I_M$ provides a good description of the observed phenomena. Suppose we further add on higher derivative gravitational interactions by considering $I_0 + I_1 + I_M$. In this case the minimal coupling rule fails. For instance, the external field of an isolated, static, spherically symmetric charge distribution can no longer be asymptotically flat. Can the electromagnetic fields be consistently coupled to generalised theories of gravity? This is our motivation for studying non-minimal electromagnetic couplings to higher derivative gravitational theories. In general at this level of generalization we should have taken into account all possible curvature and $U(1)$ gauge invariants that yield field equations involving at most fourth order partial derivatives of the component fields. This would indeed be a very complicated theory. Is there any way to delineate some of these non-minimal couplings?

In this section we apply the dimensional reduction technique of Kaluza-Klein to a higher derivative theory of gravitation and exhibit the non-minimal coupling of electromagnetic fields to gravity thus induced in 4-dimensional actual space-time.

III.2.1 Kaluza-Klein reduction

Let M_5 denote the 5-dimensional space-time manifold with topology $M_4 \times S^1$ where M_4 is the actual space-time and S^1 is a compact internal space. The radius of S^1 is usually

[1] The results of this section are published as [43]

assumed to be of the order of the Planck length. A global Killing vector denoted by K whose closed integral curves coincide with S^1 exists. Then the U_1 algebra generated by the action of K is related to the electromagnetic gauge invariance. The metric tensor on M_5 is given by

$$G = \eta_{AB}\mathbf{e}^A \otimes \mathbf{e}^B \tag{III.33}$$

where $\eta_{AB} = diag(\ -\ +\ +\ +\ +\ )$. The notation and conventions are defined in the appendix. We work in a coordinate chart $x^M : (x^\mu, y)$ which is adapted to the K-isometry of (M_5, G) so that $K = \frac{\partial}{\partial y}$. Then the hypersurfaces $y = constant$ are identified with M_4. The lower case Latin indices $_{a,b=0,1,2,3}$ refer to orthonormal frames on M_4. The $SO(1,4)$ Lie-algebra valued connection 1-forms $\{\mathbf{\Omega}^A{}_B\}$ over M_5 are labelled so as to satisfy $\mathbf{\Omega}_{AB} = -\mathbf{\Omega}_{BA}$. The following choice of the orthonormal basis 1-forms

$$\begin{aligned} \mathbf{e}^a(x,y) &= e^a(x) \quad , \quad {}_{a=0,1,2,3} \\ \mathbf{e}^5(x,y) &= \phi(x)(dy + A(x)) \end{aligned} \tag{III.34}$$

is consistent with the K-isometry of the 5-metric. The substitution of (III.34) into (III.33) gives

$$G = g + \phi^2 A \otimes A + \phi(A \otimes dy + dy \otimes A) + \phi^2 dy \otimes dy \tag{III.35}$$

from which we identify the 4-metric $g = \eta_{ab}e^a \otimes e^b$, the electromagnetic potential 1-form $A = A_a e^a$, and a scalar field $\phi(x)$ on M_4. In fact ϕ is the variable norm of the Killing vector K. Given the coframe expression (III.34), the Levi-Civita connection 1-forms are uniquely determined by solving the Cartan-Maurer equations with torsion $\mathbf{T}^A = 0$. We find

$$\begin{aligned} \mathbf{\Omega}_{ab} &= \omega_{ab} - \tfrac{1}{2}\phi F_{ab}\mathbf{e}^5 \\ \mathbf{\Omega}_{5a} &= -\mathbf{\Omega}_{a5} = \tfrac{1}{2}\phi F_{ab}e^b + \frac{\partial_a \phi}{\phi}\mathbf{e}^5 \end{aligned} \tag{III.36}$$

Then the curvature 2-forms are given by

$$\begin{aligned} \mathbf{R}_{ab} &= \pi_{ab} + \tau_{ab} \wedge \mathbf{e}^5 \\ \mathbf{R}_{5a} &= -\mathbf{R}_{a5} = \rho_a + \sigma_a \wedge \mathbf{e}^5 \end{aligned} \tag{III.37}$$

where

$$\pi_{ab} = R_{ab} - \tfrac{1}{2}\phi^2 F_{ab}F - \tfrac{1}{4}\phi^2 F_{ac}F_{bd}e^c \wedge e^d \tag{III.38}$$

$$\tau_{ab} = -D(\tfrac{1}{2}\phi^2 F_{ab}) + \tfrac{1}{2}F_{ab}d\phi + \tfrac{1}{2}(\partial_a\phi\ F_{bc} - \partial_b\phi\ F_{ac})e^c$$

$$\rho_a = D(\tfrac{1}{2}\phi F_{ab}e^b) + \partial_a\phi\ F$$

$$\sigma_a = \frac{D(\partial_a\phi)}{\phi} + \tfrac{1}{4}\phi^2 F_{ac}F^c{}_b e^b$$

We also need the following Hodge duality relations

$$^{\#}(\mathbf{e}^a \wedge \mathbf{e}^b) = {}^*(e^a \wedge e^b) \wedge \mathbf{e}^5 \tag{III.39}$$

$$^{\#}(\mathbf{e}^a \wedge \mathbf{e}^5) = {}^*e^a$$

where $^*: E^p(M_4) \to E^{4-p}(M_4)$ is the Hodge map defined with respect to the 4-metric g.

III.2.2 Non-minimal gauge couplings

We consider the higher derivative theory described by the 5-action [30]

$$I = \int\limits_{M_5} \left\{ k\mathbf{R}_{AB} \wedge {}^{\#}\mathbf{R}^{AB} - \frac{1}{2\kappa^2}\mathbf{R}_{AB} \wedge {}^{\#}(\mathbf{e}^A \wedge \mathbf{e}^B) \right\} \tag{III.40}$$

Few comments on the nature of variational field equations are in order. In general in any number of dimensions, the independent coframe and connection variations of a second order curvature invariant leads to a system of field equations which allows a dynamical space-time torsion. In the present case we obtain the field equations

$$-\frac{1}{2\kappa^2}\mathbf{R}^{BC} \wedge {}^{\#}(\mathbf{e}_A \wedge \mathbf{e}_B \wedge \mathbf{e}_C) + \tfrac{1}{2}k(\iota_A\mathbf{R}_{BC} \wedge {}^{\#}\mathbf{R}^{BC} - \mathbf{R}_{BC} \wedge \iota_A{}^{\#}\mathbf{R}^{BC}) = 0 \tag{III.41}$$

by coframe variations and

$$-\frac{1}{2\kappa^2}{}^{\#}(\mathbf{e}_A \wedge \mathbf{e}_B \wedge \mathbf{e}_C) \wedge \mathbf{T}^C + 2k\mathbf{D}^{\#}\mathbf{R}_{AB} = 0 \tag{III.42}$$

by connection variations. It is apparent from those equations that the space-time torsion need not necessarily vanish. Then it is possible for dimensional reduction to prescribe independent torsion degrees of freedom, provided they satisfy K-isometry conditions

$$\mathcal{L}_K(\mathbf{T}^A \otimes \mathbf{X}_A) = 0 \tag{III.43}$$

where $\mathcal{L}_K$ denotes Lie derivative with respect to K, and $\{\mathbf{X}_A\}$ are the frame fields such that

$G(\mathbf{X}_A, \mathbf{X}_B) = \eta_{AB}$. Here we consider only the case of vanishing space-time torsion, and we constrain our variations to preserve this choice. This can be achieved, for instance, by

the method of Lagrange multipliers and the field equations we obtain from (III.40) are given as

$$
\begin{aligned}
&-4k\mathbf{D}(\iota^B\mathbf{D}^{\#}\mathbf{R}_{AB}) + 2k\mathbf{e}_A \wedge \mathbf{D}(\iota^C\iota^B(\mathbf{D}^{\#}\mathbf{R}_{BC})) \\
&+ \tfrac{1}{2}k(\iota_A\mathbf{R}_{BC} \wedge {}^{\#}\mathbf{R}^{BC} - \mathbf{R}_{BC} \wedge \iota_A{}^{\#}\mathbf{R}^{BC}) \\
&- \frac{1}{2\kappa^2}\mathbf{R}^{BC} \wedge {}^{\#}(\mathbf{e}_A \wedge \mathbf{e}_B \wedge \mathbf{e}_C) = 0
\end{aligned}
\tag{III.44}
$$

subject to the constraint that the connection is Levi-Civita.

Our remaining task is to substitute the curvature 2-forms (III.37) together with Hodge duality relations (III.39) into the 5-action (III.40) and obtain a reduced 4-action density defined by

$$
L_5 = dy \wedge L_4(g, \phi, A) \tag{III.45}
$$

Again we performed the algebraic manipulations necessary for dimensional reduction using $\mathrm{X^TR}$. We obtained the following expression:

$$
\begin{aligned}
L_4 \;=\;& \phi R_{ab} \wedge {}^*R^{ab} + \tfrac{1}{2}\alpha\phi R_{ab} \wedge {}^*e^{ab} - \tfrac{1}{4}\alpha\phi^3 F \wedge {}^*F - \alpha d\,{}^*d\phi && \text{(III.46)} \\
&- \tfrac{3}{2}\phi^3 R_{ab} \wedge F^{ab}{}^*F && \text{(III.47)} \\
&+ \tfrac{3}{16}\phi^5(F_{ab})^4{}^*1 + \tfrac{5}{16}\phi^5 F_{ab}F^{bc}F_{cd}F^{da}{}^*1 \\
&+ \tfrac{1}{4}\phi D(\phi F_{ab}) \wedge {}^*D(\phi F^{ab}) + \tfrac{1}{2}\phi D(\phi F_{ab}e^b) \wedge {}^*(D(\phi F^a{}_c e^c)) \\
&+ \frac{1}{2\phi}D(\partial_a\phi) \wedge {}^*D(\partial^a\phi) + \tfrac{7}{2}F \wedge {}^*F(\partial_a\phi)^2\phi \\
&- \tfrac{1}{2}\phi F_{ab}F^b{}_c\partial^a\phi\;\partial^c\phi \\
&+ \tfrac{1}{2}\phi^2 F^{ab}d\phi \wedge {}^*DF_{ab} + \phi F_{ac}e^c \wedge {}^*D(\phi F^{ab})\partial_b\phi + 2\,{}^*F \wedge D(\phi F_{ab}e^b)\partial^a\phi\;\phi^2 \\
&+ \phi^2 F_{ab}F^b{}_c e^c \wedge {}^*D(\partial^a\phi)
\end{aligned}
$$

where we set the coupling constant $\alpha = -\frac{1}{2k\kappa^2}$. Note that the kinetic terms for the photon and the scalar boson fields implicitly contain couplings to curvature components. In order to make these couplings manifest, we must write these terms in a way independent of the choice of frame fields. The simplifications leading to this goal involve, first, the use of gravitational and gauge Bianchi identities, and second, partial differentiations resulting in closed forms that do not affect the variational field equations. We proved the identities (modulo closed forms)

$$
\begin{aligned}
D(\phi F^{ab}) \wedge {}^*D(\phi F^{ab}) \;=\;& -2\,d\,{}^*(\phi F) \wedge {}^*(d\,{}^*(\phi F)) + 2\,\phi^2 R_{ab} \wedge F^{ab}{}^*F \\
&- 2\,F_{ac}e^c \wedge {}^*D(\phi F^{ab})\partial_b\phi + F_{ab}\,d\phi \wedge {}^*D(\phi F^{ab}) \\
&- 4\,\phi^2 P^a \wedge (\iota_a F \wedge {}^*F - F \wedge \iota_a{}^*F) - \phi^2 Q\; F \wedge {}^*F \quad ,
\end{aligned}
\tag{III.48}
$$

and

$$D(\phi F^{e}_{ab} b) \wedge {}^*D(\phi F^{a}{}_{c} e^{c}) = \tfrac{1}{2} D(\phi F^{ab}) \wedge {}^*D(\phi F^{ab}) - F_{ac} e^{c} \wedge {}^*D(\phi F^{ab}) \partial_b \phi + \tfrac{1}{2} F_{ab}\, d\phi \wedge {}^*D(\phi F^{ab}) \quad . \tag{III.49}$$

Here $P_a = \mathcal{R}_{ba} e^b$ are the Ricci 1-forms and $Q = \eta^{ab} \mathcal{R}_{ab}$ is the curvature scalar. We also made use of the identity

$$D(\partial_a \phi) \wedge {}^*D(\partial^a \phi) = -d\,{}^*d\phi \wedge {}^*(d\,{}^*d\phi) - \tfrac{1}{2} P^a \wedge (\partial_a \phi\, {}^*d\phi + d\phi \wedge \iota_a {}^*d\phi) - \tfrac{1}{2} Q\, d\phi \wedge {}^*d\phi \quad . \tag{III.50}$$

Substituting the above identities in (III.46) and re-organising terms we reach the following expression for the reduced action density:

$$\begin{aligned} L_4 = {} & \phi R_{ab} \wedge {}^*R^{ab} + \tfrac{1}{2} \alpha \phi R_{ab} \wedge {}^*e^{ab} - \phi d\,{}^*(\phi F) \wedge {}^*(d\,{}^*(\phi F)) - \tfrac{1}{4} \alpha \phi^3 F \wedge {}^*F \\ & - \tfrac{11}{8} \phi^5 (F \wedge {}^*F)\, {}^*(F \wedge {}^*F) + \tfrac{5}{16} \phi^5 (F \wedge F)\, {}^*(F \wedge F) \\ & - 2\, \phi^3 P^a \wedge (\iota_a F \wedge {}^*F - F \wedge \iota_a {}^*F) - \tfrac{1}{2} \phi^3 Q\, F \wedge {}^*F - \tfrac{1}{2} \phi^3 R_{ab} \wedge F^{ab} {}^*F \\ & - \frac{2}{\phi} d\,{}^*d\phi {}^*(d\,{}^*d\phi) - \alpha d\,{}^*d\phi \\ & - P^a \wedge \frac{(\partial_a \phi\, {}^*d\phi + d\phi \wedge \iota_a {}^*d\phi)}{\phi} - Q\, \frac{d\phi \wedge {}^*d\phi}{\phi} \\ & + 4\, (F_{ab})^2 \phi\, d\phi \wedge {}^*d\phi + 4 F^{a}{}_{b} F_{ac} \partial^b \phi\, \partial^c \phi\, \phi\, {}^*1 + 3\, \phi^2 d\phi \wedge F_{ab} {}^*DF^{ab} \end{aligned} \tag{III.51}$$

We can now read-off from (III.51) various types of interactions between the (spin-2) graviton field g, (spin-1) photon field F, and (spin-0) scalar boson field ϕ. The obvious ground state of the above system is fixed by setting $g = \eta$, $F = 0$ and $\phi = 1$. Then different types of interactions are distinguished as follows:

For the case $F = 0$, $\phi = 1$ we get pure gravitational interactions:

$$L_g = R_{ab} \wedge {}^*R^{ab} + \tfrac{1}{2} \alpha\, R_{ab} \wedge {}^*e^{ab} \tag{III.52}$$

For the case $g = \eta$, $\phi = 1$ we get pure electromagnetic interactions :

$$L_F = -d\,{}^*F \wedge {}^*(d\,{}^*F) - \tfrac{1}{4} \alpha\, F \wedge {}^*F - \tfrac{11}{8} (F \wedge {}^*F) {}^*(F \wedge {}^*F) + \tfrac{5}{16} (F \wedge F) {}^*(F \wedge F) \tag{III.53}$$

The first two terms correspond to a generalised theory of electromagnetism considered by Bopp [32] and Podolsky. [33] The last two terms are a particular combination of quartic Maxwell invariants.

For the case $g = \eta$, $F = 0$ we get the kinetic term for the scalar boson:

$$L_\phi = -\frac{2}{\phi} d\,{}^*d\phi\, {}^*(d\,{}^*d\phi) - \alpha d\,{}^*d\phi \tag{III.54}$$

The graviton-photon interaction terms are found by setting $\phi = 1$:

$$L_{g-F} = -\tfrac{1}{2} R_{ab} \wedge F^{ab}\, {}^*F - 2\, P^a \wedge (\iota_a F \wedge {}^*F - F \wedge \iota_a {}^*F) - \tfrac{1}{2} Q\, F \wedge {}^*F \qquad \text{(III.55)}$$

The first term corresponds to the direct curvature-electromagnetic field coupling studied by Prassana [34] and Buchdahl. [35] The other terms involve the coupling of the Ricci tensor to the electromagnetic stress-energy-momentum tensor and are new. The graviton-scalar boson interactions, found by setting $F = 0$, have a similar form:

$$L_{g-\phi} = -P^a \wedge \frac{(\partial_a \phi\, {}^*d\phi + d\phi \wedge \iota_a {}^*d\phi)}{\phi} - Q\, \frac{d\phi \wedge {}^*d\phi}{\phi} \qquad \text{(III.56)}$$

Finally the photon-scalar boson interactions are determined by setting $g = \eta$:

$$L_{\phi-F} = 4\, (F_{ab})^2 \phi\, d\phi \wedge {}^*d\phi + 4 F^a{}_b F_{ac} \partial^b \phi\, \partial^c \phi\, \phi\, {}^*1 + 3\, \phi^2 d\phi \wedge F_{ab} {}^*DF^{ab} \qquad \text{(III.57)}$$

III.3 Direct Curvature-Yang-Mills Field Couplings Induced by the Kaluza-Klein Reduction of Euler Form Actions in Seven Dimensions [3]

In four dimensions, experimental evidence show that Einstein's field equations

$$G_{ab} = -\kappa^2 T_{ab} \tag{III.58}$$

adequately describe the observed phenomena. Here T_{ab} are the components of the symmetrised stress-energy-momentum tensor of the matter fields, κ^2 is the universal gravitational coupling constant, and G_{ab} denotes the covariant components of the 2 $^{\text{nd}}$rank symmetric Einstein's tensor. The latter has the unique property of being covariantly constant, and involving at most second order partial derivatives of the metric tensor components. We had stated in the previous section that the Einstein's tensor follow from the local variations of the Einstein-Hilbert action which is linear in curvature components:

$$I_E = -\frac{1}{2\kappa^2} \int\limits_{M_4} R_{ab} \wedge {}^*(e^b \wedge e^b) \tag{III.59}$$

A formal unification of long ranged interactions of nature can be achieved by constructing gravitational field theories in higher dimensional space-times. [29] The types of force fields and the form of interactions among them are determined by the dimensionally reduced theory in physical four dimensional space-time. In space-time dimensions $D > 4$, there are other higher order curvature invariants, which when used in an action, contribute to gravitational field equations partial derivatives of metric components of order no higher than two. [37] In fact, these higher order curvature invariants can be given in terms of dimensionally continued Euler forms [38]

$$L_D^{(n)} = R_{a_1 b_1} \wedge R_{a_2 b_2} \wedge \ldots \wedge R_{a_n b_n} {}^*(e^{a_1} \wedge e^{b_1} \wedge \ldots \wedge e^{a_n} \wedge e^{b_n}) \tag{III.60}$$

in dimensions $D > 2n, \qquad n = 0, 1, \ldots, [D/2]$. For the case $D = 2n$, $L_{2n}^{(n)}$ is an exact form. Its integral over the space-time manifold M_{2n} is proportional to the topological Euler-Poincaré characteristic $\chi(M_{2n})$.

Dimensional reduction of the 2 $^{\text{nd}}$order Euler–Poincaré action and some of its physical consequence are already discussed in the literature. [36, 39] In general, it was shown that [40] the Kaluza-Klein reduction of $L_D^{(n)}$ in $D = 2n + 3$ dimensions ($n > 1$) down to $D = 4$ dimensions gives a vanishing cosmological term besides the Einstein-Yang-Mills term, disregarding all other higher order contributions to the reduced action. It was suggested

[2] The results of this section are published as [48]

earlier that in $D > 2n$ dimensions, a gravitational action consisting of a linear combination of all non-trivial Euler forms should be used: [41]

$$I = \sum_{n=0}^{[D/2]} k_n \int\limits_{M_4} L_D^{(n)} \quad . \tag{III.61}$$

In fact this idea was motivated by a study of string induced gravitational models. [42] However, it should be noted that in those models the actual gauge fields can be related with degrees of freedom other than the space-time metric. In the previous section, [30, 43] we showed by explicit calculation that gravitational and gauge field interactions induced by the Kaluza-Klein reduction of higher dimensional gravitational theories based on quadratic curvature invariants can be quite complicated. There exist in the literature explicit calculations displaying several types of gravitational and gauge field couplings induced by the dimensional reduction of Euler form actions. [36, 44] But a complete Kaluza-Klein reduction of (III.61) has not yet been given. In this paper we consider the case $D = 7$ and dimensionally reduce 7-action (III.61) on $M_4 \times S^3$. As S^3 is the group manifold of $SO(3)$, the off-diagonal components of the Kaluza-Klein 7-metric are identified with the Yang-Mills potentials. The actual gravitational and Yang-Mills field interactions are then determined from the reduced 4-action. Straightforward but tedious generalisations, such as those involving an arbitrary gauge group G, or those including a scalar field that describes a variable radius for S^3, will not be attempted here.

III.3.1 Kaluza-Klein reduction on $M_4 \times S^3$

We will work in a coordinate chart $x^M : (x^\mu, y^m)$ adopted to the isometries of space-time. Here (x^μ), μ=0,1,2,3, are the coordinates of the 4-space-time, and (y^m), m=5,6,7, constitute a coordinate system for S^3. Suppose $\{e^\alpha\}$ is a set of orthonormal basis 1-forms on S^3. Then the metric on the 3-sphere

$$g_{S^3} = \delta_{\alpha\beta} e^\alpha \otimes e^\beta = g_{mn}(y) dy^m \otimes dy^n \tag{III.62}$$

and the dual frame vectors $\{X_\alpha\}$ such that $e^\alpha(X_\beta) = \delta^\alpha{}_\beta$ generate the Lie algebra of $SO(3)$:

$$[X_\alpha, X_\beta] = \epsilon_{\alpha\beta}{}^\gamma X_\gamma \quad . \tag{III.63}$$

We have the structure equations

$$de^\alpha = \tfrac{1}{2} \epsilon^\alpha{}_{\beta\gamma} e^\beta \wedge e^\gamma \quad . \tag{III.64}$$

The Kaluza-Klein ansatz for the basis 1-forms read

$$\begin{aligned} \mathbf{e}^a &= e^a(x) \quad , \quad a{=}0,1,2,3 \\ \mathbf{e}^\alpha &= e^\alpha(y) + A^\alpha(x) \quad , \quad a{=}5,6,7 \quad . \end{aligned} \tag{III.65}$$

The actual space-time 4-metric

$$g = \eta_{ab} e^a \otimes e^b \tag{III.66}$$

and the Levi-Civita 4-connections are found by solving the structure equations

$$de^a + \omega^a{}_b \wedge e^b = 0 \quad . \tag{III.67}$$

We identify $A^\alpha = A^\alpha{}_a e^a$ with the Yang-Mills potential 1-forms so that the Yang-Mills field 2-forms

$$F^\alpha = dA^\alpha + \tfrac{1}{2}\epsilon^\alpha{}_{\beta\gamma} A^\beta \wedge A^\gamma \equiv \tfrac{1}{2} F^\alpha_{ab} e^a \wedge e^b \quad . \tag{III.68}$$

They satisfy the gauge Bianchi identity

$$D_A F^\alpha \equiv dF^\alpha + \epsilon^\alpha{}_{\beta\gamma} A^\beta \wedge F^\gamma = 0 \quad . \tag{III.69}$$

We substitute the Kaluza-Klein ansatz (III.65) in (A.2) (with vanishing torsion) and solve for the connection 1-forms:

$$\mathbf{\Omega}_{ab} = \omega_{ab} - \tfrac{1}{2} F^\alpha_{ab} \mathbf{e}_\alpha \tag{III.70}$$

$$\mathbf{\Omega}_{\alpha\beta} = \tfrac{1}{2}\epsilon_{\alpha\beta\gamma}(e^\gamma - A^\gamma) \tag{III.71}$$

$$\mathbf{\Omega}^\alpha{}_a = -\mathbf{\Omega}_a{}^\alpha = \tfrac{1}{2} F^\alpha_{ab} e^b \tag{III.72}$$

Then putting the connection 1-forms given above into (A.3), we find the curvature 2-forms

$$\mathbf{R}_{ab} = \pi_{ab} + \tau^\alpha_{ab} \wedge \mathbf{e}_\alpha + \Sigma_{ab}{}^{\alpha\beta} \mathbf{e}_\alpha \wedge \mathbf{e}_\beta \tag{III.73}$$

$$\mathbf{R}_{\alpha\beta} = \Psi_{\alpha\beta} + \tfrac{1}{4} \mathbf{e}_\alpha \wedge \mathbf{e}_\beta \tag{III.74}$$

$$\mathbf{R}^\alpha{}_a = -\mathbf{R}_a{}^\alpha = \rho_a{}^\alpha + \sigma_a{}^\alpha{}_\beta \wedge \mathbf{e}^\beta \tag{III.75}$$

where

$$\pi_{ab} = R_{ab} - \tfrac{1}{2} F^\alpha_{ab} F^\alpha - \tfrac{1}{4} F^\alpha_{ac} e^c \wedge F_{\alpha bd} e^d \tag{III.76}$$

$$\tau^\alpha_{ab} = -\tfrac{1}{2} D_A F^\alpha_{ab} \tag{III.77}$$

$$\Sigma_{ab}{}^{\alpha\beta} = -\tfrac{1}{4}\epsilon_{\alpha\beta\gamma} F^\gamma_{ab} + \tfrac{1}{4} F^\alpha_{ac} F^{\beta c}{}_b \tag{III.78}$$

$$\Psi_{ab} = -\tfrac{1}{2}\epsilon_{\alpha\beta\gamma} F^\gamma + \tfrac{1}{4} F^\alpha_{ac} F^{\beta c}{}_b e^a \wedge e^b \tag{III.79}$$

$$\rho_a{}^\alpha = \tfrac{1}{2}(D_A F^\alpha_{ab}) \wedge e^b \tag{III.80}$$

$$\sigma_a{}^\alpha{}_\beta = \tfrac{1}{4}\epsilon^\alpha{}_{\beta\gamma} F^\gamma_{ab} e^b + \tfrac{1}{4} F^\alpha_{cb} F_\beta{}^b{}_a e^c \quad . \tag{III.81}$$

For the purpose of dimensional reduction, we further need the following decomposition of the Hodge map:

$$^{\#}1 = {}^{*}1 \wedge \mathbf{e}^5 \wedge \mathbf{e}^6 \wedge \mathbf{e}^7 \tag{III.82}$$

where * : $E^p(M) \to E^{4-p}(M)$ denotes the Hodge map with respect to the 4-metric g. Then a dimensionally reduced 4-action density will be defined from the identity

$$L_7 = L_4 \wedge \mathbf{e}^5 \wedge \mathbf{e}^6 \wedge \mathbf{e}^7 \quad . \tag{III.83}$$

We substitute the curvature 2-forms (III.73–III.75) in the dimensionally continued Euler densities (III.60) for the cases $n = 1, 2, 3$ and find the following reduced 4-action densities. We checked our calculations on computer using the exterior calculus package $\mathrm{X^TR}$ in REDUCE. [31]

$$L_4^{(1)} = \pi_{ab} \wedge {}^{*}(e^a \wedge e^b) - 2\sigma_a{}^{\alpha}{}_{\alpha} \wedge {}^{*}e^a + \tfrac{3}{2}{}^{*}1 \tag{III.84}$$

$$\begin{aligned} L_4^{(2)} = {} & \epsilon^{abcd}\pi_{ab} \wedge \pi_{cd} - 4(\rho_a{}^{\alpha} \wedge \tau_{bc\alpha} + \sigma_a{}^{\alpha}{}_{\alpha} \wedge \pi_{bc}) \wedge {}^{*}(e^a \wedge e^b \wedge e^c) \\ & + 3\pi_{ab} \wedge {}^{*}(e^a \wedge e^b) + 4\Sigma_{ab}{}^{\alpha\beta}\Psi_{\alpha\beta} \wedge {}^{*}(e^a \wedge e^b) - 4\sigma_a{}^{\alpha}{}_{\alpha} \wedge \sigma_b{}^{\beta}{}_{\beta} \wedge {}^{*}(e^a \wedge e^b) \\ & + 4\sigma_a{}^{\alpha}{}_{\beta} \wedge \sigma_b{}^{\beta}{}_{\alpha} \wedge {}^{*}(e^a \wedge e^b) - 2\sigma_a{}^{\alpha}{}_{\alpha} \wedge {}^{*}e^a \end{aligned} \tag{III.85}$$

$$\begin{aligned} L_4^{(3)} = {} & \tfrac{9}{2}\epsilon^{abcd}\pi_{ab} \wedge \pi_{cd} + 12\epsilon^{abcd}\pi_{ab} \wedge \Sigma_{cd}{}^{\alpha\beta}\Psi_{\alpha\beta} + 6\epsilon^{abcd}\tau_{ab}^{\alpha} \wedge \tau_{cd}^{\beta} \wedge \Psi_{\alpha\beta} \\ & - 24\epsilon^{abcd}\Sigma_{ab\alpha\beta}\rho_c{}^{\alpha} \wedge \rho_d{}^{\beta} - 24\epsilon^{abcd}\tau_{ab\alpha} \wedge \rho_c{}^{\alpha} \wedge \sigma_d{}^{\beta}{}_{\beta} \\ & + 24\epsilon^{abcd}\tau_{ab\alpha} \wedge \rho_c{}^{\beta} \wedge \sigma_d{}^{\alpha}{}_{\beta} \\ & - 12\epsilon^{abcd}\pi_{ab} \wedge \sigma_c{}^{\alpha}{}_{\alpha} \wedge \sigma_d{}^{\beta}{}_{\beta} + 12\epsilon^{abcd}\pi_{ab} \wedge \sigma_c{}^{\alpha}{}_{\beta} \wedge \sigma_d{}^{\beta}{}_{\alpha} \\ & + 6\pi_{ab} \wedge {}^{*}(e^a \wedge e^b \wedge e^c) \wedge \sigma_c{}^{\alpha}{}_{\alpha} + 6\tau_{ab\alpha} \wedge {}^{*}(e^a \wedge e^b \wedge e^c) \wedge \rho_c{}^{\alpha} \\ & + 12\epsilon_{\alpha\beta\gamma}\epsilon^{\alpha'\beta'\gamma'}\Sigma_{ab\alpha'\beta'}{}^{*}(e^a \wedge e^b \wedge e^c) \wedge \sigma_c{}^{\alpha}{}_{\gamma'} \wedge \Psi^{\beta\gamma} \\ & + 8\epsilon_{\alpha\beta\gamma}\epsilon^{\alpha'\beta'\gamma'} \wedge \sigma_a{}^{\alpha}{}_{\alpha'} \wedge \sigma_b{}^{\beta}{}_{\beta'} \wedge \sigma_c{}^{\gamma}{}_{\gamma'}{}^{*}(e^a \wedge e^b \wedge e^c) \quad . \end{aligned} \tag{III.86}$$

Our final task is to substitute the expressions (III.76–III.81) in the above formulas. The dimensionally reduced Einstein-Hilbert action density needs no further discussion: [45]

$$L_4^{(1)} = R_{ab} \wedge {}^{*}(e^a \wedge e^b) - \tfrac{1}{2}F_{\alpha} \wedge {}^{*}F^{\alpha} + \tfrac{3}{2}{}^{*}1 \quad . \tag{III.87}$$

Here the cosmological constant is proportional to the $SO(3)$ group space volume. Introducing back physical units, the value of the cosmological constant induced by dimensional

reduction turns out to be unacceptably large. The dimensional reduction of the 2 $^{\rm nd}$order Euler-Poincaré density gives (modulo a closed form): [36, 44]

$$\begin{aligned} L_4^{(2)} &= 3R_{ab}\wedge{}^*(e^a\wedge e^b) - \tfrac{5}{2}F_\alpha\wedge{}^*F^\alpha \\ &\quad - \tfrac{3}{2}\epsilon_{\alpha\beta\gamma}F^\alpha_{\ a}{}^b F^\beta_{\ b}{}^c F^\gamma_{\ c}{}^a{}^*1 \\ &\quad - 5R_{ab}F_\alpha{}^{ab}\wedge{}^*F^\alpha + 5P^a\wedge(\iota_a F^\alpha\wedge{}^*F_\alpha - F^\alpha\wedge\iota_a{}^*F_\alpha) \\ &\quad - 2(F_\alpha\wedge{}^*F_\alpha)^*(F^\beta\wedge{}^*F^\beta) + 2(F_\alpha\wedge{}^*F_\beta)^*(F^\alpha\wedge{}^*F^\beta) \\ &\quad + \tfrac{1}{2}(F_\alpha\wedge F_\beta)^*(F^\alpha\wedge F^\beta) - \tfrac{1}{8}(F_\alpha\wedge F_\alpha)^*(F^\beta\wedge F^\beta) \quad . \end{aligned} \tag{III.88}$$

The above expression does not contain any higher derivative couplings just as we expected. At the lowest order of approximation, except for unusual scale factors, it consists of the Einstein-Yang-Mills 4-action with vanishing cosmological constant. In the next order of approximation we find a F^3-term. [46] The presence of direct curvature couplings to Yang-Mills fields of generic form RF^2 has already been pointed out. The other direct curvature-Yang-Mills field couplings induced by the Kaluza-Klein reduction of $L_7^{(3)}$ are new, and here we wish to concentrate on those. Going back to expression (III.86) and checking the form of the functions (III.76–III.81), we observe that the generic types of interactions we would get in the reduced 4-action density will be RF^2, F^3, F^4, RF^3, RF^4,F^5 and F^6. In fact we find terms of the type $(DF)^2F$, but these yield upon partial differentiation and using gauge and gravitational Bianchi identities, terms of the generic types RF^3 and F^4. In a similar way, terms of the type $(DF)^2F^2$ will be replaced by terms of the type RF^4 and F^5. The RF^2 couplings thus induced from the reduction of $L_7^{(3)}$ are found to be

$$\tfrac{21}{2}R_{ab}F_\alpha{}^{ab}\wedge{}^*F^\alpha + 12P^a\wedge(\iota_a F^\alpha\wedge{}^*F_\alpha - F^\alpha\wedge\iota_a{}^*F_\alpha) + \tfrac{3}{2}\mathcal{Q}F^\alpha\wedge{}^*F_\alpha \tag{III.89}$$

Similarly the F^3 term found by reducing $L_7^{(3)}$ is

$$3\epsilon_{\alpha\beta\gamma}F^\alpha_{\ a}{}^b F^\beta_{\ b}{}^c F^\gamma_{\ c}{}^a{}^*1 \tag{III.90}$$

These together with F^4-type coupling obtained from $L_7^{(3)}$, may be added on to the expression (III.88), thus giving rise to shifts in the corresponding coupling coefficients.

III.3.2 RF^3 - type couplings

The simplest non-trivial direct curvature-Yang-Mills field couplings induced by the Kaluza-Klein reduction of the 3 $^{\rm rd}$order Euler-Poincaré density are of the generic type RF^3. The

derivation of the actual form of these couplings is rather involved, so we give some details. Explicit RF^3 couplings come from the following terms in (III.86):

$$12\epsilon^{abcd}\pi_{ab}\wedge\Sigma_{cd}{}^{\alpha\beta}\Psi_{\alpha\beta}-12\epsilon^{abcd}\pi_{ab}\wedge\sigma_c{}^{\alpha}{}_{\alpha}\wedge\sigma_d{}^{\beta}{}_{\beta}+12\epsilon^{abcd}\pi_{ab}\wedge\sigma_c{}^{\alpha}{}_{\beta}\wedge\sigma_d{}^{\beta}{}_{\alpha}$$

Substituting from (III.76–III.81) and keeping only the RF^3–type terms wee see that the reduced 4-action density will get the contribution

$$-6R^*_{ab}\wedge F^{\alpha a}{}_c e^c\wedge H_\alpha{}^b{}_d-3R_{ab}\wedge F^{\alpha ab}H_\alpha-6R_{ab}\wedge H^{\alpha ab}F_\alpha \tag{III.91}$$

where $R^*_{ab}=\frac{1}{2}\epsilon_{ab}{}^{cd}R_{cd}$ and we defined the 2-forms

$$H^\alpha=\tfrac{1}{2}\epsilon^\alpha{}_{\beta\gamma}F^\beta_{ac}F^{\gamma c}{}_b e^a\wedge e^b\equiv\tfrac{1}{2}H^\alpha_{ab}e^a\wedge e^b \quad . \tag{III.92}$$

Next we make use of the gravitational Bianchi identity $R^a{}_b\wedge e^b=0$ and cast the expression (III.91) into the form

$$-24R^*_{ab}\wedge H^{\alpha ab}F_\alpha \quad . \tag{III.93}$$

This may as well be written in an equivalent way

$$6R^{ab}\wedge H^\alpha_{ab}{}^*F_\alpha+24\mathcal{R}^{ab}F^\alpha_{ac}H_\alpha{}^c{}_b{}^*1+12\mathcal{Q}H^\alpha\wedge{}^*F_\alpha \tag{III.94}$$

where $\mathcal{R}_{ab}e^b=\iota^bR_{ba}$ are the Ricci 1-forms and $\mathcal{Q}=\iota^a\iota^bR_{ba}$ is the curvature scalar.
The remaining RF^3 couplings are implicit in $(DF)^2F$ type interactions which come from the following terms in (III.86) :

$$\begin{aligned}&6\epsilon^{abcd}\tau^\alpha_{ab}\wedge\tau^\beta_{cd}\wedge\Psi_{\alpha\beta}-24\epsilon^{abcd}\Sigma_{ab\alpha\beta}\rho_c{}^\alpha\wedge\rho_d{}^\beta-24\epsilon^{abcd}\tau_{ab\alpha}\wedge\rho_c{}^\alpha\wedge\sigma_d{}^\beta{}_\beta\\&+24\epsilon^{abcd}\tau_{ab\alpha}\wedge\rho_c{}^\beta\wedge\sigma_d{}^\alpha{}_\beta \quad .\end{aligned}$$

Substituting above from the expressions (III.76–III.81) we find the relevant terms to be given by

$$\begin{aligned}&\tfrac{3}{4}\epsilon^{abcd}D_AF^\alpha_{ab}\wedge D_AF^\beta_{cd}\wedge\epsilon_{\alpha\beta\gamma}F^\gamma \qquad\qquad (III.95)\\&-\tfrac{3}{2}\epsilon^{abcd}\epsilon_{\alpha\beta\gamma}F^\alpha_{ab}D_AF^\beta_{cc'}\wedge D_AF^\gamma_{dd'}\wedge e^{c'}\wedge e^{d'}\\&+\tfrac{3}{2}\epsilon^{abcd}\epsilon_{\alpha\beta\gamma}D_AF^\alpha_{ab}\wedge D_AF^\beta_{cc'}\wedge e^{c'}\wedge F^\gamma_{dd'}e^{d'} \quad .\end{aligned}$$

We differentiate each term by parts and use the basic identity

$$D^2_AF^\alpha_{ab}=R_a{}^cF^\alpha_{cb}+R_b{}^cF^\alpha_{ac}+\epsilon^\alpha{}_{\beta\gamma}F^\beta F^\gamma_{ab} \quad . \tag{III.96}$$

Neglecting irrelevant terms, throwing away closed forms and making use of gravitational Bianchi identities we finally bring (III.95) to the form

$$-\tfrac{3}{2}\epsilon^{abcd}\epsilon_{\alpha\beta\gamma}F^\alpha_{ab}R_c{}^{a'}\wedge(F^\beta_{a'd}F^\gamma+F^\beta_{a'b'}e^{b'}\wedge F^\gamma_{dd'}e^{d'}) \quad . \tag{III.97}$$

After a few manipulations this can be rewritten as

$$3\mathcal{R}^{ab}F^{\alpha}_{ac}H_{a}{}^{c}{}_{b}{}^{*}1 + 3\mathcal{Q}H^{\alpha} \wedge {}^{*}F_{\alpha} \quad . \tag{III.98}$$

Therefore, summing (III.94) and (III.98), we write the Lagrangian density that determines the RF^3–type direct curvature-Yang-Mills field couplings as follows:

$$L_{RF^3} = 6R_{ab} \wedge H^{ab*}_{\alpha}F^{\alpha} + 27\mathcal{R}^{ab}F^{\alpha}_{ac}H_{\alpha}{}^{c}{}_{b}{}^{*}1 + 15\mathcal{Q}H^{\alpha} \wedge {}^{*}F_{\alpha} \quad . \tag{III.99}$$

III.3.3 Concluding Comments

To conclude we would like to emphasize once again that there seems to be no reason to use solely the Einstein-Hilbert action in higher dimensional space-times to describe the dynamics of gravitational fields. The action (III.61) given by a linear combination of the non-trivial dimensionally continued Euler forms leads also to field equations that involve at most 2 $^{\text{nd}}$order partial derivatives. To be specific, we considered in this section a space-time of dimension $D = 7$, and discussed the Kaluza-Klein reduction of the dimensionally continued Euler forms assuming the product topology $M_4 \times S^3$. The leading terms in Yang-Mills fields we found in the Kaluza-Klein reduced action coincide with the standard Einstein-Yang-Mills 4-action. We saw that the Kaluza-Klein reduction of the action (III.61) induces particular Yang-Mills self interaction terms of the generic types F^3, F^4, F^5 and F^6; as well as terms describing direct coupling of the space-time curvature to Yang-Mills fields. RF^2–type couplings were already known and discussed. There are terms of the type RF^3 and RF^4 which are new. We worked out RF^3 couplings induced by the Kaluza-Klein reduction explicitly. It is known that the dimensional reduction procedure selects a particular subset of all possible invariants of the generic type RF^2. [47] It would be interesting to see whether the RF^3–terms have a similar structure. The physical consequences of these direct curvature-Yang-Mills field couplings ought to be further studied and understood.

CHAPTER IV

CONCLUSION

In this thesis we introduced a symbolic algebraic manipulation software to perform exterior calculus and dimensional reduction. The software is developed for REDUCE, therefore a closer look to the interior of REDUCE is also given. Some problems of implementation are underlined. With the help of the developed software we concentrated on three original problems in the field of relativistic theories of gravitation.

One of these was looking for a torsional solution of the modified double duality equations, which are field equations different than Einstein's field equations. The metric was chosen to be the Kerr-de-Sitter metric with a torsion ansatz, expressed in Boyer-Lindquist coordinates. The reduced field equations are obtained. Since no global solution existed for the assumed type of ansatz, a strip solution for which one angular coordinate is treated parametrically is obtained.

In the second of the problems, the dimensional reduction technique of Kaluza-Klein is applied to Stephenson-Kilmister-Yang theory of gravity in $D = 5$ and the non-minimal couplings of electromagnetic fields to gravity thus induced in 4-dimensional actual space-time are exhibited.

Thirdly, we considered a space-time of $D = 7$, and discussed the Kaluza-Klein reduction of the dimensionally continued Euler forms assuming the product topology $M_4 \times S^3$. The leading terms in Yang-Mills fields coincide with the standard Einstein-Yang-Mills 4-action. It is observed that higher order Yang-Mills self interaction terms of generic types F^3, F^4, F^5 and F^6 are present. Terms coupling the space-time curvature directly to Yang-Mills fields are observed, too. RF^2 type of couplings were already known, but RF^3 and RF^4 type couplings are new. RF^3 couplings are explicitly worked out.

LIST OF REFERENCES

[1] A. Petermann, *Helv. Phys. Acta* **30** (1957) 407

[2] J. A. Campbell, A. C. Hearn *J. Comp. Phys* **5** (1970) 280

[3] T. Sasaki *J. Comp. Phys* **22** (1976) 189

[4] J. Kahane *J. Math. Phys* **9** (1968) 1732

[5] A. Deprit,et al. *Science* **168** (1970) 1569

[6] T. Soma *Optik* **49** (1977) 255

[7] J. B. Hayter, J. Penfold *Molecular Phys.* **42** (1981) 109

[8] J. A. Van Hultzen *SIGSAM* **14**/2 (1980) 36

[9] J. Calmet,J. A. van Hulzen in *Computer Algebra* edited by B. Buchberger et al. (Springer-Verlag, 2.ed., 1983) 247

[10] E. Kaltofen in *ibid.* 95

[11] H. Cartan*Differential Forms* (Hermann, paris, 1970)
H. Flanders *Differential Forms* (Academic Press, 1963)
M. Schreiber*Differential Forms* (Springer-Verlag, 1977)
C. Nash,S. Sen *Topology and Geometry for Physicists* (Academic Press, 1983)
A. Trautman *Differential Geometry for Physicists* (Stony Brook Lectures, Bibliopolis, 1984)

[12] G. Stephenson, *Nuo. Cim.* **2** (1958) 263

[13] C. W. Kilmister and D. J. Newman, *Proc. Camb. Phil. Soc.* **57** (1961) 851

[14] C. N. Yang, *Phys. Rev. Lett.* **33** (1974) 445

[15] K. S. Stelle, *Phys. Rev.* **D16** (1977) 953

[16] R. Pavelle, *Phys. Rev. Lett.* **34** (1975) 1114

[17] A. H. Thompson, *Phys. Rev. Lett.* **35** (1975) 320

[18] G. Debney, E. E. Fairchild and S. T. C. Siklos, *G. R. G.* **9** (1978) 879

[19] I. M. Benn, T. Dereli and R. W. Tucker, *G. R. G.* **13** (1981) 581

[20] P. Baekler, F. W. Hehl and E. W. Mielke in *Proceedings, 2nd Marcel Grossmann Meeting*, edited by R. Ruffini (North-Holland, 1982) 413

[21] E. W. Mielke *Geometrodynamics of Gauge Fields* (Akademie-Verlag, Berlin, 1987)

[22] I. M. Benn, T. Dereli and R. W. Tucker, *J. Phys.* **A15** (1982) 849

[23] T. Dereli and R. W. Tucker, *J. Phys.* **A15** (1982) L27

[24] J. D. McCrea, P. Baekler and M. Gürses, *Nuo. Cim.* **B99** (1987) 171

[25] B. Carter, in *Black Holes*, Les Houches, 1972, edited by C. DeWitt and B. S. DeWitt (Gordon&Breach, 1973)

[26] R. H. Boyer and R. W. Lindquist, *J. Math. Phys.* **8** (1967) 265

[27] A. C. Hearn *REDUCE 3, User Manual* Rand Corp., (1983)

[28] T. Dereli, R. W. Tucker, *Nucl. Phys.* **B209** (1982) 217

[29] T. Appelquist, A. Chodos, P. G. O. Freund *Modern Kaluza-Klein Theories* (Addison-Wesley, 1987) and references therein.

[30] T. Dereli, G. Sülap, in *Proc. of 10th International Conference on General Relativity and Gravitation*, edited by B. Bertotti, T. De Felice, A. Pascolini (Padova, 1983) 501

[31] T. Dereli, G. Üçoluk, *J. Comp. Phys.* (to be published)

[32] F. Bopp, *Ann. d. Phys.* **38** (1940) 345

[33] B. Podolsky, Phys. Rev. **62** (1941) 68

[34] A. R. Prassana, *Phys. Lett.* **37A** (1971) 331

[35] H. A. Buchdahl, *J. Phys.* **A12** (1979) 1037

[36] F. Müller-Hoissen, *Phys. Lett.* **201B** (1988) 325

[37] D. Lovelock, *J. Math. Phys.* **12** (1971) 498

[38] B. Zumino, *Phys. Rep.* **137** (1986) 109

[39] J. Madore, *Phys. Lett.* **110A** (1985) 289
F. Müller-Hoissen, *Phys. Lett.* **163B** (1985) 106
C. Wetterich, M. Reuter, *Nucl. Phys.* **B304** (1988) 653

[40] M. Arık, T. Dereli, *Phys. Lett.* **189B** (1987) 96
M. Arık, T. Dereli, *Phys. Rev. Lett.* **62** (1989) 5

[41] F. Müller-Hoissen, *Class. Q. Grav.* **3** (1986) 665

[42] B. Zwiebach, *Phys. Lett.* **156B** (1985) 315
D. G. Boulware, S. Deser, *Phys. Rev. Lett.* **55** (1985) 2656

[43] T. Dereli, G. Üçoluk, *Kaluza-Klein reduction of generalised theories of gravity and non-minimal gauge couplings.* (submitted to *Class. Q. Grav.*)

[44] F. Müller-Hoissen, *Phys. Lett.* **201B** (1988) 325
F. Müller-Hoissen, *Class. Q. Grav.* **5** (1988) L35

[45] R. Kerner, *Ann. Inst. H. Poincaré* **9A** (1968) 143
W. Mecklenburg, *Phys. Rev.* **D21** (1980) 2149

[46] A. I. Alekseev, B. A. Arbuzov, *Theo. Mat. Phys.* **59** (1984) 372

[47] G. Horndeski, *J. Math. Phys.* **17** (1976) 1980

[48] T. Dereli, G. Üçoluk, *Direct Curvature-Yang-Mills Field Couplings Induced by the Kaluza-Klein reduction of Euler Form Actions in Seven Dimensions* (submitted to *Class. Q. Grav.*)

Appendix A

NOTATION AND CONVENTIONS

The gravitational fields in an n-dimensional space-time are described by a Lorentzian metric

$$\mathbf{G} = \eta_{AB}\mathbf{e}^A \otimes \mathbf{e}^B \tag{A.1}$$

where $\eta_{AB} = diag(-\underbrace{+\ldots+}_{n-1})$ and a set of independent but metric compatible connection 1-forms $\{\mathbf{\Omega}^A{}_B\}$. The indices $A,B,\ldots = 0,1,\ldots,n$ refer to an orthonormal set of frame vectors $\{\mathbf{X}_A\}$ [1]. They are raised and lowered by η^{AB} and η_{AB}. $\{\mathbf{e}^A\}$ are the orthonormal basis 1-forms dual to the frame fields, i.e., $\mathbf{e}^A(\mathbf{X}_B) = \delta^A{}_B$. They satisfy the structure equations

$$d\mathbf{e}^A + \mathbf{\Omega}^A{}_B \wedge \mathbf{e}^B = \mathbf{T}^A \tag{A.2}$$

$$d\mathbf{\Omega}^A{}_B + \mathbf{\Omega}^A{}_C \wedge \mathbf{\Omega}^C{}_B = \mathbf{R}^A{}_B \tag{A.3}$$

where $\mathbf{T}^A = \mathbf{T}_{BC,}{}^A\ \mathbf{e}^B \wedge \mathbf{e}^C$ are the torsion 2-forms, $\mathbf{R}^A{}_B = \frac{1}{2}\mathbf{R}_{CD,}{}^A{}_B\ \mathbf{e}^C \wedge \mathbf{e}^D$ are the curvature 2-forms of space-time. The integrability conditions for the structure equations yield the Bianchi identities

$$d\mathbf{T}^A + \mathbf{\Omega}^A{}_B \wedge \mathbf{T}^B = \mathbf{R}^A{}_B \wedge \mathbf{e}^B \tag{A.4}$$

$$d\mathbf{R}^A{}_B + \mathbf{\Omega}^A{}_C \wedge \mathbf{R}^C{}_B + \mathbf{\Omega}_B{}^C \wedge \mathbf{R}^A{}_C = 0 \quad . \tag{A.5}$$

The following linear operators on forms are defined:

$d \equiv e^a \nabla_{X_a}$: $E^p(M) \to E^{p+1}(M)$ is the exterior derivative,

[1] If no dimensional reduction is going to be performed at all then we use ω instead of $\mathbf{\Omega}$ and lower-case Latin indices instead of upper-case. This will also be the case for the physical quantities after dimensional reduction.

$\iota_A \quad : \quad E^p(M) \to E^{n-1}(M)$ are the interior product operators such that $\iota_A(\mathbf{e}^B) = \delta_A{}^B$. We define also the Hodge map [2] $\#$: $E^p(M) \to E^{n-p}(M)$ defined so that the invariant volume element

$$\#1 = \mathbf{e}^0 \wedge \mathbf{e}^1 \wedge \ldots \wedge \mathbf{e}^n \quad .$$

A * to the right of a 2 $^{\text{nd}}$rank antisymmetric tensor denotes its dual. For instance, in 4 dimensions

$$R^*_{ab} = \frac{1}{2!}\epsilon_{ab}{}^{cd}\, R_{cd} \tag{A.6}$$

[2] Again, if no dimensional reduction is performed then a * to the left of a form will denote its Hodge dual.

Appendix B

The RLISP Syntax

In this appendix a formal scheme for the translation of the RLISP syntax to standard LISP is presented[1]

A rule has a name in brackets <...> by which it is known and is defined by what follows the meta symbol ::=. Each rule of the set consists of one or more "alternatives" separated by the | meta symbol, being the different ways in which the rule will be matched by source text. Each alternative is composed of a "recognizer" and a "generator" separated by the ⇒ meta symbol. This symbol can be interpreted as a kind of equivalence representation *RLISP* ⇒ *LISP*. The recognizer is a concatenation of any of three different forms.

1. **Terminals:** Upper case lexemes and punctuation which is not part of the meta syntax represent items which must appear as is in the source text for the rule to succeed.

2. **Rules:** Lower case lexemes enclosed in <...> are names of other rules. The source text is matched if the named rule succeeds.

3. **Primitives:** Lower case singletons not in brackets are names of primitives or primitive classes of Standard LISP.

The recognizer portion of the following rule matches an extended syntax procedure:

<function> ::=

ftype *PROCEDURE* id (<id list>) ; <statement> ; ⇒

[1] The content is a revision of the appendix A of the standard LISP report

A function is recognized as an "ftype" (one of the tokens $EXPR$, $FEXPR$, etc.) followed by the keyword $PROCEDURE$, followed by an "id" (the name of the function), followed by an "<id list>" (the formal parameter names) enclosed in parentheses. A semicolon terminates the title line. The body of the function is a <statement> followed by a semicolon. For example:

$EXPR\ PROCEDURE\ NULL(X)\ ;\quad EQ(X, NIL)\ ;$

satisfies the recognizer, causes the generator to be activated and the rule to be matched successfully.

The generator is a template into which generated items are substituted. The three syntactic entities have corresponding meanings when they appear in the generator portion.

1. **Terminals:** These lexemes are copied as is to the generated text.

2. **Rules:** A rule has succeeded in the recognizer section then the value of the rule is the result of the generator portion of that rule.

3. **Primitives:** When primitives are matched the primitive lexeme replaces its occurrence in the generator.

If more than one occurrence of an item would cause ambiguity in the generator portion this entity appears with a subscript. Thus:

<conditional statement> ::=

IF <expression> $THEN$ <statement$_1$> $ELSE$ <statement$_2$> ...

has occurrences of two different <statement>s. The generator portion uses the subscripted entities to reference the proper generated value.

The <function> rule appears in its entirety as:

<function> ::=

ftype $PROCEDURE$ id (<id list>) ; <statement> ;

$\Rightarrow$ ($PUTD$ ($QUOTE$ id) ($QUOTE$ ftype)

($QUOTE$ ($LAMBDA$ (<id list>) <statement>)))

If the recognizer succeeds (as it would in the case of the $NULL$ procedure example) the generator returns:

$(PUTD\ (QUOTE\ NULL)\ (QUOTE\ EXPR)$
$\quad (QUOTE\ (LAMBDA\ (X\ (EQ\ X\ NIL))))$

The identifier in the template is replaced by the procedure name NULL, <id list> by the single formal parameter X, the <statement> by $(EQ\ X\ NIL)$ which is the result of the <statement> generator. $EXPR$ replaces ftype, the type of the defined procedure.

The Extended Syntax Rules

<function> ::=
ftype $PROCEDURE$ id (<id list>) ; <statement> ;
$\Rightarrow$ $(PUTD\ (QUOTE$ id) $(QUOTE$ ftype)
$(QUOTE\ (LAMBDA$ (<id list>) <statement>)))

<id list> ::=
id $\Rightarrow$ id
| id, <id list> $\Rightarrow$ id <id list>

<statement> ::=
<expression> $\Rightarrow$ <expression>
| <proper statement> $\Rightarrow$ <proper statement>

<proper statement> ::=
<assignment statement> $\Rightarrow$ <assignment statement>
| <conditional statement> $\Rightarrow$ <conditional statement>
| <while statement> $\Rightarrow$ <while statement>
| <compound statement> $\Rightarrow$ <compound statement>

<assignment statement> ::=
id := <expression>
$\Rightarrow$ $(SETQ$ id <expression>)

<conditional statement> ::=

IF <expression> *THEN* <statement$_1$> *ELSE* <statement$_2$>

⇒ (*COND* (<expression> <statement$_1$>)

(*T* <statement$_2$>))

| *IF* <expression> *THEN* <statement>

⇒ (*COND* (<expression> <statement>))

<while statement> ::=

WHILE <expression> *DO* <statement>

⇒ (*PROG NIL*

LBL (*COND* ((*NULL* <expression>) (*RETURN NIL*)))

<statement list>

(*GO LBL*))

<compound statement> ::=

BEGIN SCALAR <id list> ; <program list> *END*

⇒ (*PROG* (<id list>) <program list>)

| *BEGIN* <program list> *END*

⇒ (*PROG NIL* <program list>)

| << <statement list> >>

⇒ (*PROGN* <statement list>)

<program list> ::=

<full statement> ⇒ <full statement>

| <full statement> <program list>

⇒ <full statement> <program list>

<full statement> ::=

<statement> ⇒ <statement>

| id: ⇒ id

<statement list> ::=

<statement> ⇒ <statement>

| <statement> ; <statement list>

⇒ <statement> <statement list>

<expression> ::=

<expression$_1$> . <expression$_2$>

⇒ ($CONS$ <expression$_1$> <expression$_2$>)

| <expression$_1$> = <expression$_2$>

⇒ ($EQUAL$ <expression$_1$> <expression$_2$>)

| <expression$_1$> EQ <expression$_2$>

⇒ (EQ <expression$_1$> <expression$_2$>)

| ′ <expression> ⇒ ($QUOTE$ <expression>)

| id <expression> ⇒ (id <expression>)

| id(<argument list>) ⇒ (id <argument list>)

| number ⇒ number

| id ⇒ id

<argument list> ::=

() ⇒

| <expression> ⇒ <expression>

| <expression> , <argument list>

⇒ <expression> <argument list>

Appendix C

SOURCE OF X^TR

```
%*****************************************************************;
%                       ---------------                           ;
%                          X   T   R                              ;
%                       ---------------                           ;
%                                                                 ;
%                                                                 ;
%             A Package For Exterior Calculus                     ;
%                          and                                    ;
%                Dimensional Reduction                            ;
%                                                                 ;
%                                                                 ;
%  Author : Gokturk Ucoluk                                        ;
%  Date   : 1984 - 1989                                           ;
%  Place  : Middle East Tech. Univ., Physics Dept. Ankara, Turkey;
%                                                                 ;
%                                                                 ;
%                                                                 ;
%  Version: 3.15                                                  ;
%                                                                 ;
%                                                                 ;
%*****************************************************************;

%*****************************************************************;
%  VERSION HISTORY:                                               ;
%  3.0 : HodgeDrift added... (May 89)                             ;
%  3.1 : Flag DIMRED & relavant code in WEDGESIMP added..(28 May);
%  3.15: Changes of 3.1 revised ... since indexkind does not nec.;
%        gives means on which space the creature lives.. (1 June);
%                                                                 ;
%                                                                 ;
%*****************************************************************;


global '(!*forms
         xspcdim!*
         !*coords
         !*baseone
         baseone!*table
         index!*span
         !*derexp
         !*inbase
         !*unkinprd
         signature!*
         !*killsuper
         !*letpreval
```

```
          !*drifthodge
          dimredspc1!*
          dimredspc2!*
          !*dimred );

% General Utility Procedures ;

symbolic procedure fdegree u; %radical change: no more nil ret. ;
    if atom u then if (u:=get(u,'fdeg)) then u else 0
     else if get(car u,'fdeg) then get(car u,'fdeg)
     else if car u eq 'd then fdegree cadr u+1
     else if car u eq 'inprod then fdegree caddr u-1
     else if car u eq 'lieder then fdegree caddr u
     else if car u memq '(times wedge)
      then eval ('plus.for each x in cdr u collect fdegree x)
     else if car u eq 'quotient then fdegree cadr u
     else if car u memq '(plus difference minus) then fdegree cadr u
     else if car u eq 'hodge then xspcdim!* - fdegree cadr u
     else if get(car u,'fdegreefn) then
         apply(get(car u,'fdegreefn),list fdegree cadr u)
%    else if get(car u, 'simpfn) then fdegree cadr u
     else 0;

symbolic procedure scatteredfdegree x;
% Needed for DIMRED.. returns the fdegrees of the form product term;
% It will be called only in wedgesimp, and after getzfout is performed;
% Therefore no PLUS or simular odd terms may be present ;
% the result is a two element list where the first element represents;
% the the fdegree of the part of x which lives in the 1st dim.red. spc;
% and the second the corresponding quatity of the second part;
% a nil in a slot means the corrsponding part does not exist ;
% if both slots are nil then a meaningful  discrimination couldn't ;
% be made. (so it doesnot mean that the fdegree is zero);
% This is for example the case for the 'd' operator which has an     ;
% operant of mixed life style (has in both spaces non nil fdegree part;
   if atom x then
        if flagp(x,'liveson1) then list2(get(x,'fdeg),nil)
        else if flagp(x,'liveson2) then list2(nil,get(x,'fdeg))
        else list2(get(x,'fdeg1),get(x,'fdeg2))
   else if flagp(car x,'liveson1) then list2(fdegree x,nil)
   else if flagp(car x,'liveson2) then list2(nil,fdegree x)
   else if get(car x,'fdeg1) or get(car x,'fdeg2) then
                  list2(get(car x,'fdeg1),get(car x,'fdeg2))
   else if flagp(car x,'indexedlstyle) then
          if get(cadr x,'dimredkind)=1 then list2(fdegree x,nil)
          else if get(cadr x,'dimredkind)=2 then list2(nil,fdegree x)
          else '(nil nil)
   else
     begin scalar s; % We've an oper.  which allows hybrit arguments..;
          s := scatteredfdegree(cadr x); %Only first arg..nary not imp;
          return
            if car s and null cadr s then list2(fdegree x,nil)
            else if cadr s and null car s then list2(nil,fdegree x)
            else '(nil nil)
     end;

symbolic procedure negonflg(u,v); if u then negsq v else v;

symbolic procedure removelast u;
    if cddr u then removelast cdr u else rplacd(u,nil);

symbolic procedure ascend(a1,a2); a1>a2;

symbolic procedure oddp u; if numberp u and remainder(u,2)=1 then t;
```

```
symbolic procedure demcd u;
    caadr u
     .for each x in cdadr u collect reval list('quotient,x,caddr u);

symbolic procedure demcdp u;
    pairp cadr u and caadr u memq '(plus difference);

symbolic procedure free(u,v);
    if atom u then null depends(u,v)
     else if get(car u,'independent) eq v then t
     else if get(car u,'dependent) eq v then nil
     else null depends(u,v);

symbolic procedure newdepends(u,v);
    if atom u then depends(u,v)
     else if get(car u,'independent) eq v then nil
     else if get(car u,'dependent) eq v then t
     else depends(u,v);

symbolic procedure dependsl(u,v);
    v and (newdepends(u,car v) or dependsl(u,cdr v));

symbolic procedure ldepends(u,v);
    u and (newdepends(car u,v) or ldepends(cdr u,v));

symbolic procedure desq u;
    if null u then nil ./ 1
     else if atom u then !*k2q u
     else if car u eq '!*sq then cadr u
     else u;

symbolic procedure allbaseonep l;
    if l then eqcar(car l,'e) and allbaseonep cdr l else t;

symbolic procedure permpp(u,v);
    if null u then t
     else if car u=car v then permpp(cdr u,cdr v)
     else not permpp(cdr u,subst(car v,car u,cdr v));



symbolic procedure antiderdist(f,l,w,c);
    begin scalar s,l1,l2,p;
       l2 := l;
     l:
       s :=
        list3('times,
              list3('expt,'(minus 1),'plus.p),
              w.append(l1,append(list apply(f,list car l2),cdr l2)))
         .s;
       l1 := aconc(l1,car l2);
       p := apply(c,list car l2).p;
       if l2 := cdr l2 then go to l;
       s := reverse s;
       return 'plus.(caddr car s.cdr s)
    end;

symbolic procedure getzfout(u,fl);
    begin scalar nz,z;
       nz := getzfout1(u,fl);
       if null nz then return nil;
       z := car nz;
       nz := cdr nz;
       nz := if nz then list (if cdr nz then car u.nz else car nz);
       z := 'times.z;
       return append(z,nz)
```

```
    end;

symbolic procedure getzfout1(u,fl);
    begin scalar z,nz;
       for each x in cdr u do if fdegree x=0
                               then <<z := x.z;
                                      if fl then nz := 1 . nz>>
                              else if eqcar(x,'quotient)
                               then <<z :=
                                       list('quotient,1,caddr x).z;
                                      if eqcar(cadr x,'times)
         then for each y in cdadr x do if fdegree y=0 then z := y.z
                                          else nz := y.nz
          else nz := cadr x.nz>>
                              else if eqcar(x,'times)
         then for each y in cdr x do if fdegree y=0 then z := y.z
                                       else nz := y.nz
          else nz := x.nz;
       return if null z then nil else reverse z.reverse nz
    end;

symbolic procedure let1(x,y); let2(x,y,nil,t);

symbolic procedure hasfree u;
   if atom u then if u memq FRLIS!* then t  else nil
   else hasfree car u or hasfree cdr u;

switch killsuper, letpreval, dimred;
% ON killsuper : forms with fdegree>spcdim will be killed.    ;
% ON letpreval : in def. of LET rules the LHS will be reval as usual;
%                if its OFF then LHS of the LET rule stays as it is.;
% ON dimred    : shall have a redspacedims before it will kill higher;
%                form degrees looking in which reduced space they live;
!*letpreval := T;

% LINEARIZATION ..... Eventually What The REDUCE people haven't DO!;
fluid '(numf numd denf dend);

symbolic procedure lnrtimes(u,v);
    begin scalar numf,numd,denf,dend,d,n,r;
       n := cadr u;
       d := caddr u;
       if eqcar(n,'times) and eqcar(cadr n,'minus)
         then <<numf := '((minus 1));
                n := 'times.(cadadr n.cddr n)>>;
       if eqcar(n,'times)
         then for each x in cdr n do if free(x,v) then numf := x.numf
                                     else if eqcar(x,'expt)
                                               and (r :=
                                                     lnrexpt(x,v))
                                      then <<numf := car r.numf;
                                             numd := cdr r.numd>>
                                     else numd := x.numd
        else if eqcar(n,'expt) and (r := lnrexpt(n,v))
         then <<numf := car r.numf; numd := cdr r.numd>>
        else numd := list n;
       if eqcar(d,'times)
         then for each x in cdr d do if free(x,v) then denf := x.denf
                                     else if eqcar(x,'expt)
                                               and (r :=
                                                     lnrexpt(x,v))
                                      then <<denf := car r.denf;
                                             dend := cdr r.dend>>
                                     else dend := x.dend
        else if eqcar(d,'expt) and (r := lnrexpt(d,v))
         then <<denf := car r.denf; dend := cdr r.dend>>
        else dend := list d;
```

```
      if null numf and null denf then return nil;
      if null numf then numf := '(1);
      if null numd then numd := '(1);
      for each x in '(numf
                      numd
                      denf
                      dend) do if r := eval x
                                  then if cdr r
                                         then set(x,'times.reverse r)
                                        else set(x,car r);
      n := if denf then list('quotient,numf,denf) else numf;
      d := if dend then list('quotient,numd,dend) else numd;
      return n.d
    end;


symbolic procedure lnrexpt(u,v); expt!-separate(u,v);

symbolic procedure formlnexpt(op,u,v);
    begin scalar r;
      return if car u eq 'expt and (r := lnrexpt(u,v))
                then list('times,car r,formlnr list(op,cdr r,v))
               else list(op,u,v)
    end;

symbolic procedure linearize(u,v);
    begin scalar op,body,y,z,p,r;
      op := car u;
      body := cdr u;
      for each x in body do if free(x,v) then z := x.z
                            else y := x.y;
      if z
        then return 'times
                    .nconc(reverse z,
                           if null y or null cdr y then y
                            else list linearize(op.reverse y,v));
     l:
      y := car body;
     i:
      if atom y then <<z := y.z; go to b>>;
     q:
      if car y eq 'plus or car y eq 'difference
        then <<z := reverse z;
               z :=
                for each x in cdr y collect
                  linearize(op.append(z,append(list x,cdr body)),v);
               z := car y.z;
               go to fin>>
       else if car y eq 'times
        then <<r := nil;
               for each x in cdr y do if free(x,v) then p := x.p
                                      else if eqcar(x,'expt)
                                       then begin scalar dum;
                                              dum := lnrexpt(x,v);

                                              if dum
                                                then <<p :=
                                                         car dum.p;

                                                       r :=
                                                        cdr dum.r>>
                                               else r := x.r
                                            end
                                      else r := x.r;
               if cdr r then z := ('times.reverse r).z
                else z := car r.z;
               go to b>>
```

```
        else if car y eq 'minus
         then <<p := '(minus 1).p; z := cadr y.z; go to b>>
        else if car y neq 'quotient then go to s;
       if cadr y neq 1 and free(cadr y,v)
         then <<p := cadr y.p; y := list('quotient,1,caddr y)>>;
       if free(caddr y,v)
         then <<p := list('quotient,1,caddr y).p;
                y := cadr y;
                go to i>>;
       if atom cadr y and atom caddr y then go to n;
       if eqcar(cadr y,'minus)
         then <<y := list('quotient,cadadr y,caddr y);
                p := '(minus 1).p>>;
       if demcdp y then <<y := demcd y; go to q>>;
       if (eqcar(cadr y,'times) or eqcar(caddr y,'times))
            and (r := lnrtimes(y,v))
         then <<p := car r.p; y := cdr r; go to n>>
        else if eqcar(cadr y,'expt) and (r := lnrexpt(cadr y,v))
         then <<p := car r.p; y := list('quotient,cdr r,caddr y)>>;
       if eqcar(caddr y,'expt)
         then if r := lnrexpt(caddr y,v)
                then <<p := list('quotient,1,car r).p;
                       y := list('quotient,cadr y,cdr r);
                       go to n>>
              else go to n;
     s:
       if car y eq 'expt and (r := lnrexpt(y,v))
         then <<p := car r.p; y := cdr r>>;
     n:
       z := y.z;
     b:
       body := cdr body;
       if body then go to l;
       z := op.reverse z;
     fin:
       p := reverse p;
       return if p then aconc('times.p,z) else z
    end;


symbolic procedure narylin(u,f);
    begin scalar y,z;
       u := revop1 u;
       if null subfg!* then go to a
        else if (z := linearize(u,get(car u,'ghostdep))) neq u
         then return simp z;
     a:
       return if opmtch u then simp u else apply(f,list u)
    end;

symbolic procedure unarylin(u,f);
    begin scalar z;
       u :=
        revop1 (if cddr u then u
                  else append(u,list get(car u,'ghostdep)));
       if null subfg!* then go to a
        else if (z := formlnr u) neq u then return simp z;
     a:
       removelast u;
       return if opmtch u then simp u else apply(f,list u)
    end;

% The Redefinition of DIFFP of alg1 ..to have a better chain ruling ;

global '(!*derexp); %update to have chain-rule exp. if der is unknown;
```

```
switch derexp;

symbolic procedure diffp(u,v);
   %u is a standard power, v a kernel.
   %value is the standard quotient derivative of u wrt v;
   begin scalar w,x,y,z,key; integer n;
        n := cdr u;     %integer power;
        u := car u;     %main variable;
        if u eq v and (w := 1 ./ 1) then go to e
         else if atom u then go to f
         %else if (x := assoc(u,dsubl!*)) and (x := atsoc(v,cdr x))
%              and (w := cdr x) then go to e   %deriv known;
            %dsubl!* not used for now;
         else if (not atom car u and (w:= difff(u,v)))
                  or (car u eq '!*sq and (w:= diffsq(cadr u,v)))
          then go to c  %extended kernel found;
         else if (x:= get!*(car u,'dfn)) then nil
         else if car u eq 'plus and (w:=diffsq(simp u,v))
          then go to c
         else go to h;  %unknown derivative;
        y := x;
        z := cdr u;
    a:  w := diffsq(simp car z,v) . w;
        if caar w and null car y then go to h;  %unknown deriv;
        y := cdr y;
        z := cdr z;
        if z and y then go to a
         else if z or y then go to h;  %arguments do not match;
        y := reverse w;
        z := cdr u;
        w := nil ./ 1;
    b:  %computation of kernel derivative;
        if caar y
          then w := addsq(multsq(car y,simp subla(pair(caar x,z),
                                                  cdar x)),
                          w);
        x := cdr x;
        y := cdr y;
        if y then go to b;
    c:  %save calculated deriv in case it is used again;
        %if x := atsoc(u,dsubl!*) then go to d
        %else x := u . nil;
        %dsubl!* := x . dsubl!*;
    d:  %rplacd(x,xadd(v . w,cdr x,nil,t));
    e:  %allowance for power;
        %first check to see if kernel has weight;
        if (x := atsoc(u,wtl!*))
          then w := multpq('k!* to (-cdr x),w);
        return if n=1 then w else multsq(!*t2q((u to (n-1)) .* n),w);
    f:  %check for possible unused substitution rule;
        if not depends(u,v)
           and (not (x:= atsoc(u,powlis!*))
                 or not smember(v,simp cadddr x))
          then return nil ./ 1;
        w := list('df,u,v);
        go to j;
    h:  %final check for possible kernel deriv;
        y := nil;
        if car u eq 'df then key:=t;
        w := if key then 'df . cadr u . derad(v,cddr u)
             else list('df,u,v);
        y := cddr u;
        w := if (x := opmtch w) then simp x
             else if not depends(cadr w,caddr w) then nil ./ 1
             else if !*derexp then
```

```
                    begin
                        if atom cadr w then return mksq(w,1);
                        w := nil ./ 1;
                        for each m in cdr(if key then cadr u else u) do
                          w := addsq(multsq(
                                 if (x := opmtch (z :=
                                  'df . if key then (cadr u.derad(m,y))
                                         else list(u,m) )) then
                                  simp x else mksq(z,1),
                                  diffsq(simp m,v)),
                                  w);
                        return w
                    end
              else mksq(w,1);
         go to e;
    j:   w := if x := opmtch w then simp x else mksq(w,1);
         go to e
   end;



symbolic procedure remcoordinate u;
    begin scalar msg;
       msg := !*msg;
       begin !*msg := nil end;
       for each x in u do <<depend1(x,'form,nil);
                            remprop(x,'dependent);
                            remprop(x,'fdeg);
                            remflag(x,'coord)>>;
       !*forms := setdiff(!*forms,u);
       !*coords := setdiff(!*forms,u);
       !*msg := msg
    end;

put('remcoordinate,'stat,'rlis);

% The ( REMBASEONE ) statement... ;


symbolic procedure rembaseone;
    <<remprop('e,'dependent);
      remprop('e,'fdeg);
      remprop('e,'simpfn)>>;

put('rembaseone,'stat,'endstat);

% The ( FORMDEGREES ) statement... ;


symbolic procedure formdegrees u;
    if car u eq 'all
      then <<for each x in !*forms do wfdgr x;
             if get('e,'fdeg)=1
               then begin
                       terpri();
                       prin2 '"e(1 .. ";
                       prin2 xspcdim!*;
                       prin2 '") : orthonormal base 1-forms";
                       return nil
                    end>>
     else for each x in u do wfdgr x;

put('formdegrees,'stat,'rlis);

symbolic procedure wfdgr x;
    if null get(x,'fdeg)
```

```
      then begin
              prin2 x;
              prin2 '" : is not a form...";
              terpri();
           end
     else begin
             prin2 x;
             prin2 '" : ";
             prin2 get(x,'fdeg);
             prin2 '"-form";
             if flagp(x,'coord) then prin2 " & coordinate";
             terpri();
          end;


% The ( REMFORM ) statement... ;

put('remform,'stat,'rlis);

symbolic procedure remform u;
    begin scalar msg;
       msg := !*msg;
       begin !*msg := nil end;
       for each x in u do <<remprop(x,'fdeg);
                            remprop(x,'dependent);
                            depend1(x,'form,nil)>>;
       !*forms := setdiff(!*forms,u);
       !*msg := msg
    end;


% The  ( SPACEDIM ) Operator...;

flag('(spacedim),'opfn);

symbolic procedure spacedim n; xspcdim!* := reval n;


% The ( BASE ) switch;

global '(!*inbase);

switch inbase;

put('inbase,'simpfg,'((nil (shifttocoord)) (t (shifttobase))));

symbolic procedure shifttocoord;
    begin scalar x,msg;
       msg := !*msg;
       !*msg := nil;
       rmsubs();
       for i:=0:sub1(xspcdim!*) do
                   <<x := list('e,i);
                     let1(list('d,x),getv(get('e,'dasfnofcoord),i));
                     let1(x,getv(get('e,'fnofcoord),i))>>;
       for each y in !*coords do <<let2(list('d,y),nil,nil,nil);
                                   let2(list('d,y),nil,t,nil)>>;
       !*msg := msg
    end;

symbolic procedure shifttobase;
    begin scalar x,msg;
       msg := !*msg;
       !*msg := nil;
       if null get('e,'dasfnofbase) then return nil;
       rmsubs();
```

```
       for i:=0:sub1(xspcdim!*) do
                   <<x := list('e,i);
                     let2(x,nil,nil,nil);
                     let2(x,nil,t,nil);
                     let1(list('d,x),getv(get('e,'dasfnofbase),i))>>;
       for each y in !*coords do let1(list('d,y),
                                      get(y,'dasfnofbase));
       !*msg := msg
    end;


% The ( SIGNATURE ) statement ..... ;

put('signature,'stat,'rlis);

symbolic procedure signature u; signature!* := eval ('plus.u);


% The ( GENERATE )  procedure .....  ;

flag('(generate),'opfn);

symbolic procedure generate u;
    begin scalar z,v,a,b,baseone;
       put('invierbein,'matrix,'matrix);
       setk('invierbein,aeval list('quotient,1,'vierbein));
       z := for each x in !*coords collect !*k2q list('d,x);
       v :=
        for each x in cdr
             get('vierbein,'rvalue) %rvalue is new:former 'matrix;
          collect scalprod(for each y in x collect desq y, z);
       a := mkvect xspcdim!*;
       b := mkvect xspcdim!*;
       for i := 0 : xspcdim!* - 1 do
           << putv(b,i, simp list('d,putv(a,i,prepsq nth(v,i+1))));
              baseone := list('e,i).baseone >>;
       put('e,'fnofcoord,a);
       put('e,'dasfnofcoord,b);
       baseone := reversip baseone;
       z := for each x in baseone collect !*k2q x;
       v :=
        for each x in cdr get('invierbein,'rvalue)
          collect scalprod(for each y in x collect desq y, z);
       for each x in !*coords do <<put(x,'dasfnofbase,prepsq car v);
                                   v := cdr v>>;
       z :=
        for each x in !*coords collect list('d,x).get(x,'dasfnofbase);
       a := mkvect xspcdim!*;
       for i := 0 : xspcdim!* - 1 do
           << putv(a,i,prepsq quotsq(subf(car getv(b,i),z),
              subf(cdr getv(b,i),z)));
              putv(b,i,prepsq getv(b,i)) >>;
       put('e,'dasfnofbase,a);
       if !*inbase then  shifttobase() else shifttocoord()
    end;


% WEDGE .... ( ^ )  The Exterior Product Operator;

put('wedge,'ghostdep,'form);
put('wedge,'simpfn,'wedgesimpfn);
newtok '( (!&)  wedge  ! !&!  );
precedence wedge,difference;
flag('(wedge),'right);
flag('(wedge),'nary);
switch drifthodge;
```

```
!*drifthodge := T;

symbolic procedure wedgesimpfn u;
    narylin('wedge.u,function wedgesimp);

symbolic procedure wedgesimp u;
    begin scalar x,y,s,d1,d2;
       if hasfree u or not !*letpreval then return mksq(u,1);
       x := getzfout(u,nil);
       if x then return simp x;
       u := wedgeflat u;
       x := repeats u;
     a:
       if null x then go to c;
       if oddp fdegree car x then return nil ./ 1;
       x := cdr x;
       go to a;
     c:
       if (fdegree u>xspcdim!*) and !*killsuper then return nil ./ 1;
       if !*dimred then
         <<
           d1 := d2 := 0;
           for each x in cdr u do
             << y := scatteredfdegree x;
                if car y  then d1 := d1 + car y;
                if cadr y then d2 := d2 + cadr y >>;
           if (d1>dimredspc1!*)or(d2>dimredspc2!*) then return nil ./ 1
         >>;
       for each z in cdr u do if fdegree z=0 then x := z.x
                              else s := z.s;
       if x
         then return simp ('times
                           .nconc(x,
                                  if null s then nil
                                   else if cdr s
                                    then list ('wedge.reverse s)
                                   else s));
       x := wedgeord cdr u;
       x := ('wedge.car x).cdr x;
       if null cdr x then << y:= 0; go to b >>;
       y := for each z in cdr x collect 'times.list(car z,cdr z);
       y := eval ('plus.y); % If symbolic fdegree, should be recoded;
     b:
       x := car x;
       s := nil;
       if !*drifthodge then s:=drifthodge(cdr x);
       if null s then s:=mksq(x,1);
       return if oddp y then negsq s else s;
    end;

fluid '(xex);

symbolic procedure wedgeord u;
    begin scalar xex; return word u.xex end;

symbolic procedure word u;
    if null u then nil
     else if null cdr u then u
     else if null cddr u then word2(car u,cadr u)
     else wordad(car u,word cdr u);

symbolic procedure wordad(a,u);
    if null u then list a
     else if wordp(a,car u) then a.u
     else <<xex := (fdegree a.fdegree car u).xex;
            car u.wordad(a,cdr u)>>;
```

```
symbolic procedure word2(u,v);
    if wordp(u,v) then list(u,v)
     else <<xex := (fdegree u.fdegree v).xex; list(v,u)>>;



symbolic procedure wordp(u,v);
    if null u then v
     else if atom u
      then if atom v
             then if numberp u then numberp v and not u>v
                   else if numberp v then t
                   else orderp(u,v)
            else if flagp(car v,'frozen) then wordp(u,cadr v)
            else t
     else if atom v
      then if flagp(car u,'frozen) then wordp(cadr u,v) else nil
     else if car u=car v then wordp(cdr u,cdr v)
     else wordp(car u,car v);

unfluid '(xex) ;



symbolic procedure wedgeflat u;
    for each x in u conc if atom x then x.nil
                          else if car x memq '(wedge times)
                           then wedgeflat cdr x
                          else x.nil;

% The drifthodge function is after XTR ver. 3.0 ;
% It does two things ;
% In its first phase : (... & *P & ... & *Q & ...) are converted;
%                  to   (... &  P & ... &  Q & ...) provided that;
%                       fdegree(P)+fdegree(Q)=XSPCDIM!*         ;
%                                                               ;
% In its second phase: (... & *P & ... &  Q & ...) are converted;
%                  to   (... &  P & ... & *Q & ...) provided that;
%                       fdegree(P)=fdegree(Q)                   ;
% If both phase attempts failed so NIL is returned.             ;
symbolic procedure drifthodge U;   % U is expected without WEDGE;
   begin scalar V,S,SIGN,DEG,SEEK,M,TOUCHED,MEM,PIV,ONEFORM;
        V := U;
        SIGN := T;
      A:
        if null V then go to PHASE2;
        if not (DEG:=peep car V) then go to B;
        SEEK := XSPCDIM!* - DEG;
        S := cdr V;
        M := 0;
      C:
        if S then
          if not (peep car S = SEEK) then
            << M := M + fdegree car S;
               S := cdr S;
               go to C >>
          else
            << TOUCHED := T;
               SIGN := SIGN eq signchange1p(M,DEG);
               if onep cadar V  or  onep cadar S then ONEFORM:=T;
               rplaca(V,cadar V);
               rplaca(S,cadar S) >>;
      B:
        V := cdr V;
```

```
        go to A;
 PHASE2:
        V := U;
      D:
        if null V then go to FIN;
        if not(DEG:=peep car V) then go to E;
        S := cdr V;
        M := 0;
        PIV := nil;   % since S can be nil at this stage ;
      F:
        if S then
          << if (SEEK:=fdegree car S)=DEG then
               << PIV := S;
                  MEM := M >>;
             M := M + SEEK;
             S := cdr S;
             go to F >>;
        if PIV then
          << TOUCHED := T;
             SIGN := SIGN eq signchange2p(MEM);
             rplaca(V,cadar V);
             rplaca(PIV,list('HODGE,car PIV)) >>;
      E:
        V := cdr V;
        go to D;
     FIN:
        if null TOUCHED then return nil;
        U := 'WEDGE . U;
        U := if ONEFORM then simp U else mksq(U,1);
        return if SIGN then negsq U else U
   end;

symbolic procedure signchange1p(X,P);
   oddp(XSPCDIM!* *(X+P) - P*P + (XSPCDIM!*-SIGNATURE!*)/2);

symbolic procedure signchange2p(X);
   oddp(XSPCDIM!* * X);

symbolic procedure peep X;
   if eqcar(X,'HODGE) then fdegree cadr X;



% The INTERIOR PRODUCT ... ( .*. ) ... Operation  ;

put('inprod,'ghostdep,'form);
put('inprod,'simpfn,'inprodsimpfn);
newtok '( ( !. !* !.)  inprod ! !.!*!.!  );
precedence inprod, wedge;

switch unkinprd;

symbolic procedure inprodsimpfn u;
    secondarylin('inprod.u,function inprodsimp);

symbolic procedure inprodsimp u;
    begin scalar v;
       if hasfree u or not !*letpreval then return mksq(u,1);
       v := getzfout1(cdr u,nil);
       return if null v then pureinprsimp u
              else if null cdr v then nil ./ 1
              else simp append('times.car v,
                               list ('inprod.(cadr u.cdr v)))
    end;

 fluid '(v);
```

```
symbolic procedure pureinprsimp u;
    begin scalar v,f;
       v := cadr u;
       f := caddr u;
       return if eqcar(f,'wedge)
                then simp antiderdist(function (lambda x;
                                                  list('inprod,v,x)),
                                      cdr f, 'wedge,
                                      function fdegree)
              else if eqcar(f,'d) and v=cadr f then 1 ./ 1
              else if eqcar(f,'inprod)
               then begin scalar p,q;
                     l:
                       p := cadr f.p;
                       if car p eq v then return nil ./ 1;
                       f := caddr f;
                       if eqcar(f,'inprod) then go to l;
                       q := prepsq pureinprsimp list('inprod,v,f);
                       return if q neq f then nil ./ 1
                              else inprodord(f,reverse p)
                    end
              else if numberp v and eqcar(f,'e)
               then if v=cadr f then 1 ./ 1 else nil ./ 1
              else if !*unkinprd then mksq(u,1)
              else nil ./ 1
    end;

unfluid '(v);

symbolic procedure inprodord(w,s);
    begin
       if null cdr s then return mksq(pack(s,w),1);
       s := reverse s;
       s := (not permp(s,s := ordn s)).s;
       w := mksq(pack(reverse cdr s,w),1);
       return negonflg(car s,w)
    end;

symbolic procedure pack(vlist,fr);
    if null vlist then fr
     else pack(cdr vlist,list('inprod,car vlist,fr));

symbolic procedure encode u;
    begin
       put('!*aptal!*,'ghostdep,get(car u,'ghostdep));
       return append('!*aptal!*.cddr u,list get(car u,'ghostdep))
    end;

symbolic procedure decode(op,u);
    if atom u then u
     else if car u eq '!*aptal!*
      then append(op,removelast decode(op,cdr u))
     else decode(op,car u).decode(op,cdr u);

symbolic procedure secondarylin(u,f);
    begin scalar z;
       u := revop1 u;
       if null subfg!* then go to a
        else if (z := decode(list(car u,cadr u),formlnr encode u))
                  neq u
         then return simp z;
    a: return if opmtch u then simp u else apply(f,list u)
    end;


% The HODGE .... ( # ) .... ;
```

```
put('hodge,'ghostdep,'form);
put('hodge,'simpfn,'hodgesimpfn);
put('!#,'newnam,'hodge);
put('hodge,'prifn,'hodgeprint);
flag('(hodge),'frozen);   % Very necessary for drifthodge ;


symbolic procedure hodgesimpfn u;
    unarylin('hodge.u,function hodgesimp);


symbolic procedure hodgesimp u;
    begin scalar body,s;
       if hasfree u or not !*letpreval then return mksq(u,1);
       if car u=1 then return mksq('wedge.baseone!*table,1);
       body := getzfout1(u,nil);
       if body
         then return simp ('times
                           .nconc(car body,
                                  list list('hodge,
                                            if cdr body
                                              then cadr body
                                             else 1)));
       body := cadr u;
       return if atom body then mksq(u,1)
              else if car body eq 'hodge
               then <<body := cadr body;
                      s :=
                       fdegree body*(xspcdim!*-fdegree body)
                        +(xspcdim!*-signature!*)/2;
                      if oddp s then negsq mksq(body,1)
                       else mksq(body,1)>>
              else if car body eq 'wedge and allbaseonep cdr body
               then begin
                       u := cdr body;
                       body := setdiff(baseone!*table,u);
                       if null body then return 1 ./ 1;
                       for each x in u do s :=
                                          s neq (eta cadr x=(-1));
                       return if s
                                   eq permpp(baseone!*table,
                                             append(u,body))
                                then negsq mksq('wedge.body,1)
                               else mksq('wedge.body,1)
                    end
              else if allbaseonep list body
               then <<u := list body;
                      body := setdiff(baseone!*table,u);
                      s := eta cadar u=(-1);
                      if s eq permpp(baseone!*table,append(u,body))
                        then negsq mksq('wedge.body,1)
                       else mksq('wedge.body,1)>>
              else mksq(u,1)
    end;

symbolic procedure hodgeprint u;
  <<prin2!* "#";
    if pairp cadr u and get(caadr u,'infix) then maprint(cadr u,1000)
    else maprin cadr u >>;

% The Right HODGE operator ... ( rhodge ) ....;

put('rhodge,'simpfn,'rhodgesimp);
```

```
symbolic procedure rhodgesimp u;
    begin scalar r,s;
       u := car u;
       if repeats cdr u then return nil ./ 1;
       r := setdiff(index!*span,cdr u);
       for each x in cdr u do s := s neq (eta x=(-1));
       return if s eq permpp(index!*span,append(r,cdr u))
                then simp (car u.r)
               else negsq simp (car u.r)
    end;



% The Exterior Derivative .... ( D ) ....;

put('D,'newnam,'d);
put('d,'prifn,'dprint);
put('d,'simpfn,'dsimpfn);
put('d,'ghostdep,'form);
flag('(d),'frozen);


symbolic procedure dsimpfn u; unarylin('d.u,function ddsimp);

symbolic procedure ddsimp u;
    begin scalar body,degre;
       if hasfree u or not !*letpreval then return mksq(u,1);
       body := cadr u;
       degre := fdegree body;
       return if body=1 or degre=xspcdim!* then nil ./ 1
               else if atom body or get(car body,'fdeg) then mksq(u,1)
               else if car body eq 'd then nil ./ 1
               else if degre=0 then nullfd body
               else if degre := getzfout1(u,nil)
                then begin scalar f;
                       if cddr degre
                         then begin
                                terpri();
                                prin2 '"more than one nero-form:";
                                return prin2 degre
                              end;
                       f := cadr degre;
                       degre :=
                        if cdar degre then 'times.car degre
                         else caar degre;
                       return simp list('plus,
                                        list('wedge,
                                             prepsq nullfd degre,
                                             f),list('times,
                                                     degre,
                                                     list('d,f)))
                     end
               else if car body memq '(times wedge)
                then simp antiderdist(function (lambda x;
                                                   list2('d,x)),
                                      cdr body,'wedge,
                                      function fdegree)
               else if not car body memq '(inprod lieder hodge)
                then nullfd body
               else mksq(u,1)
    end;

symbolic procedure dftoform u;
    if atom u then u
```

```
     else if car u eq 'df and caddr u eq 'form then list('d,cadr u)
     else dftoform car u.dftoform cdr u;

symbolic procedure nullfd u; dftoform simpdf list(u,'form);

symbolic procedure dprint u;
    <<prin2!* (if null !*nat or !*raise then 'd!  else 'd);
      maprin cadr u>>;


symbolic procedure forms u;
    begin scalar msg,z;
       msg := !*msg;
       begin !*msg := nil end;
       z :=
        for each x in u collect <<put(cadr x,'fdeg,caddr x);
                                  put(cadr x,'dependent,'form);
                                  depend1(cadr x,'form,t);
                                  cadr x>>;
       !*forms := union(!*forms,z);
       !*msg := msg
    end;

put('forms,'stat,'rlis);


% The ( COORDINATE ) declaretion... ;


symbolic procedure coordinate u;
    begin scalar msg,v;
       msg := !*msg;
       !*msg := nil;
       flag(u,'coord);
       xspcdim!* := length u;
       baseone!*table := nil;
       index!*span := nil;
       for i:=sub1(xspcdim!*) step (-1) until 0 do
           <<baseone!*table := list('e,i) . baseone!*table;
                     index!*span := i . index!*span>>;
       put('e,'fdeg,1);
       put('e,'dependent,'form);
       depend1('e,'form,t);
       mkop 'e;
       for each x in u do <<put(x,'fdeg,0);
                            put(x,'dependent,'form);
                            depend1(x,'form,t)>>;
       !*forms := union(!*forms,'e . u);
       !*coords := union(!*coords,u);
       !*msg := msg
    end;

put('coordinate,'stat,'rlis);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%;
% In this section the software tool is   introduced to declare in the ;
% algebraic mode an operator to be linear over the form structures.   ;
% That means :                                                        ;
%              op( <any alg. expr. of forms and scalars> )            ;
% Will automatically simplify to :                                    ;
%                                                                     ;
%  ----                                                               ;
%  \                                                                  ;
%  /   <alg.expr of scalars & zero-forms>  * op(<non-zero-form expr> );
```

```
%  ----                                    i                              i ;
%   i                                                                       ;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%;
%  Author : Gokturk Ucoluk                                                  ;
%  Date   : May 1989                                                        ;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%;


symbolic procedure xtrop u;
   for each x in u do
     << put(x,'ghostdep,'form);
        put(x,'simpfn,
            subst(x,'x,'(lambda (u)
              (unarylin (cons (quote x) u) (function xtrsimpiden)))))
     >>;
 symbolic procedure fdegreefn u;
    if null cdr u then rederr "Use as: FDEGREEFN <function>,<xtrop>"
    else for each x in cdr u do put(x,'fdegreefn,car u);

put('fdegreefn,'stat,'rlis);
put('xtrop,'stat, 'rlis);

symbolic procedure xtrsimpiden u;
   begin scalar s;
        if hasfree u or not !*letpreval then return mksq(u,1);
        s := getzfout1(u,nil);
        return
          if s then
            simp('times.nconc(car s,list list(car u,
                      if cdr s then cadr s else 1)))
          else mksq(u,1);
   end;



symbolic procedure eta n; if n=0 then (-1) else 1;  %That's common;

%*********************************************************************;
%   This  part handles dimensional reduction.                         ;
%   What it does is exactly taking a simplified tensoral expression   ;
%   possibly with contracted indices and if these indices were        ;
%   SPLIT defined, beforehand then the expression will be split       ;
%   into the an expression which has the new indices                  ;

%   Author: Gokturk UCOLUK                                            ;
%   Date  : May 1989                                                  ;
%*********************************************************************;

symbolic procedure dimred(u,newspcdim);
 <<xspcdim!* := newspcdim;
   if atom u then u
    else if car u eq 'QUOTIENT then
           list('QUOTIENT,drplusorterm cadr u,caddr u)
    else drplusorterm u >>;

flag ('(dimred),'opfn);

symbolic procedure drplusorterm u;
   if car u eq 'PLUS then
     'PLUS . mapcan(cdr u,'dimredterm)
    else replus dimredterm u;

symbolic procedure dimredterm u;
   % u is a member of a PLUS or a term itself, the result is a LIST of;
   % possibly splited new terms. Even if no spliting happens a list of;
```

```
   % the original term is returned ;
   begin scalar s;
        u := fixexpt u;  %Performs (EXPT <any> 2)->(TIMES <any> <any>);
        s := fishing u;  % The list of the split indces is generated  ;
        if null u then return u.nil; % Sorry,no split,but we rtrn LIST;
        s := checkrepeat s;  % Split indices must be pairs in the list;
        s := for each x in s collect
                for each p in get(x,'split)
                   collect x.p;   % Form assoc. list;
        s := cartesprod s; % A list of a all possible subst. is formed;
        return for each a in s collect sublist(a,u) ;
   end;

symbolic procedure fixexpt u;
   if atom u then u else
    if car u eq 'EXPT and caddr u = 2 then list('times,cadr u,cadr u)
   else fixexpt car u . fixexpt cdr u;

symbolic procedure sublist(a,u);
   % Functionaly equivalent of sublis, but more efficient for our case;
   if atom u then
     begin scalar s;
          s := assoc(u,a);
          return if s then cdr s else u
     end
    else sublist(a,car u) . sublist(a,cdr u);



symbolic procedure checkrepeat u;
   % Expects a list of atoms where each atom appears exactly two times;
   % The result is an error if that is not hold, else a generated list;
   % in which the atoms appear only once;
   if u then
     if car u memq cdr u then car u . checkrepeat delete(car u, cdr u)
   else rederr list(u,"is SPLIT declared but appears as free index",
                      " ... that is not allowed");

fluid '(basket);

symbolic procedure fishing u;
   % u is any sxpr, fishing returns a list of all atoms appearing in  ;
   % u at any level & was declared SPLIT, the entries will be present ;
   % in the returned list as much as they appeared in the sexpr.      ;
   % This will serve for a check of the proper usage of Einstein's    ;
   % repeated index summation convention                              ;
   begin scalar basket;
        fishing1 u; % This will modify basket, can cause prob.in comp.;
        return reversip basket;
   end;

symbolic procedure fishing1 u;
   if pairp u then  << fishing1 car u; fishing1 cdr u >>
   else if get(u,'split) then basket := u . basket;


unfluid '(basket) ;  % The compiler knows what this mean ;

symbolic procedure cartesprod u;
   % u:  a list of lists (which can be consired as sets) the result is;
   % the cartesian product of these sets. So for example :            ;
   % cartesprod '((a b c) (x y)) wil evaluate to the list :           ;
   % ((a x) (a y) (b x) (b y) (c x) (c y))                           ;
   if cdr u then
    begin scalar s;
         s := cartesprod cdr u;
```

```
         return
           mapcan(car u,
                  function (lambda x;
                            mapcar(s, function (lambda y; x.y))));
    end
   else mapcar(car u, function (lambda x; x.nil));

symbolic procedure split u;
   begin scalar s;
        put(car u,'split,cdr u);
        put(car u,'dimredkind,0);  %Not a bad idea , Ehhh ;
        s := length cdr u;
        for i:=1:s do
          << u := cdr u;
             put(car u,'dimredkind,i) >>
   end;

symbolic procedure remsplit u;
   for each x in u do remprop(x,'split);

rlistat '(split remsplit);

symbolic procedure indexkind u; get(u,'dimredkind);

symbolic procedure splitp u; if get(u,'split) then t;

operator splitp,indexkind;

symbolic procedure putindexkind u;
   for each x in cdr u do put(x,'dimredkind, car u);

symbolic procedure remindekind u;
   for each x in u do remprop(x,'dimredkind);

rlistat '(putindexkind remindexkind);

symbolic procedure freeofindex(u,x);
   % Checks whether the expression (u) is does not contain ;
   % any dependency on a term which has a indexkind of (x).;
   if atom u then
     if null u then t
     else not eqn(get(u,'dimredkind),x)
   else freeofindex(car u,x) and freeofindex(cdr u,x);

operator freeofindex;   % So we can use it in algebraic mode;

symbolic procedure redspacedims u;
<<
  if not (length u = 2) then
     rederr list("Only S -> M x N reduction allowed");
  u := for each x in u collect reval x;
  if not ( numberp car u and numberp cadr u) then
    rederr list("The Dimensions must be numeric, sorry");
  dimredspc1!* := car u;
  dimredspc2!* := cadr u
>>;

symbolic procedure liveson1 u;
   << remflag(u,'liveson2);
      flag(u,'liveson1) >>;

symbolic procedure liveson2 u;
   << remflag(u,'liveson1);
      flag(u,'liveson2) >>;

symbolic procedure indexedlstyle u;
```

```
   << remflag(u,'liveson1);
      remflag(u,'liveson2);
      flag(u,'indexedlstyle) >>;

symbolic procedure lifestyle u;
   for each x in u do
     << if (not pairp x) or  not(length x = 3) then
          rederr "entry not of form: form(dim1,dim2)";
        if not get(car x,'fdeg) then
          rederr list(car x," is not form");
        if not (get(car x,'fdeg)=(cadr x+caddr x)) then
          rederr list(car x,"was declared to be ",get(car x,'fdeg),
                      "-form. Sum of dim.s does not agree");
        remflag(car x.nil,'liveson1);
        remflag(car x.nil,'liveson2);
        put(car x,'fdeg1,cadr x);
        put(car x,'fdeg2,caddr x) >>;

rlistat '(redspacedims liveson1 liveson2 indexedlstyle lifestyle);


%*****************************************************************;
%             Source code for XTR  ends here !                    ;
%*****************************************************************;
END$
```

```
%*****************************************************************;
%  The code below is a simple index printing facility. It is      ;
%  adopted from the INDEX PRINTING FACILITY by van Hultzen.       ;
%  The flag NOCOMMA serves for the purpose of ommiting the        ;
%  print of a comma between indices.                              ;
%  Although this code is not a part of the source of XTR we       ;
%  we include it for its usefulness.                               ;
%*****************************************************************;

SYMBOLIC;

GLOBAL '(!*NOARG FARGLIST!*, !*NOCOMMA);

SWITCH NOARG;
 %if ON,operators are printed without arg.,if they were DONOARG decl;
SWITCH NOCOMMA;
 %if ON,subindices are printed without a comma, if a DOINDEX was decl;

!*NOARG := T;
FARGLIST!* := NIL;

SYMBOLIC PROCEDURE REMARG(L);
begin scalar OP;
     OP := CAR L;
     L := CDR L;
     IF !*NAT
       THEN
        <<IF NULL(ASSOC(OP,FARGLIST!*))
           THEN FARGLIST!* := (OP . L) . FARGLIST!*;
           PRIN2!*(OP)>>
     ELSE
      <<PRIN2!*(OP);
        PRIN2!* "(";
        OBRKP!* := NIL;
        INPRINT('!*COMMA!*,0,L);
        OBRKP!* := T;
        PRIN2!* ")" >>;
end;

SYMBOLIC PROCEDURE FLATSIZEC1(U);
IF ATOM(U)
THEN LENGTHC(U)
ELSE FLATSIZEC1(CAR U) + FLATSIZEC2(CDR U) + 2;

SYMBOLIC PROCEDURE FLATSIZEC2(U);
IF NULL(U)
THEN 0
ELSE
  IF CDR(U)
  THEN
   << if !*nocomma then
        FLATSIZEC1(CAR U) + FLATSIZEC2(CDR U)
      else
        FLATSIZEC1(CAR U) + 1 + FLATSIZEC2(CDR U) >>
  ELSE FLATSIZEC1(CAR U);

SYMBOLIC PROCEDURE FARG;
BEGIN SCALAR L,X;
  L := FARGLIST!*;
  PRIN2!* "THE OPERATORS HAVE THE FOLLOVING ARGUMENTS";
  TERPRI!* T;
  WHILE L DO
  <<PRIN2!*(CAAR L);
    PRIN2!*("=");
    X := REMPROP(CAAR L,'PRIFN);
    MAPRIN(CAR L);
    IF X
```

```
    THEN PUT(CAAR L,'PRIFN,X);
    TERPRI!* T;
    L := CDR(L);
  >>
END;

PUT('FARG,'STAT,'ENDSTAT);

SYMBOLIC PROCEDURE CLFARG;
FARGLIST!* := NIL;

PUT('CLFARG,'STAT,'ENDSTAT);

SYMBOLIC PROCEDURE DONOARG(L);
PUTPRSP(L,'REMARG);

PUT('DONOARG,'STAT,'RLIS);

SYMBOLIC PROCEDURE DOINDEX(L);
PUTPRSP(L,'SUBIND);

PUT('DOINDEX,'STAT,'RLIS);

SYMBOLIC PROCEDURE PUTPRSP(L,F);
   FOR EACH ID IN L DO
     IF ATOM(ID) THEN PUT(ID,'PRIFN,F)
      ELSE LPRIM LIST(ID,"NOT AN ATOM");

SYMBOLIC PROCEDURE OFFNOARG(L);
OFFINDEX(L);

PUT('OFFNOARG,'STAT,'RLIS);

SYMBOLIC PROCEDURE OFFINDEX(L);
   FOR EACH ID IN L DO
     IF ATOM(ID)
      THEN REMPROP(ID,'PRIFN)
      ELSE LPRIM LIST (ID,"NOT AN ATOM");

PUT('OFFINDEX,'STAT,'RLIS);

SYMBOLIC PROCEDURE SUBIND(L);
BEGIN SCALAR X,S,OP;
  OP := CAR L;      % CHANGE FOR REDUCE 3.3 ;
  L := CDR L;
  IF YCOORD!*<=0 AND !*NAT AND NUMBCH(L)
  THEN
  <<IF FLATSIZEC(OP)+FLATSIZEC2(L)>LINELENGTH(NIL)-spare!*-POSN!*
    THEN TERPRI!* T;
    PRIN2!*(OP);
    S := POSN!*;
    YCOORD!* := YCOORD!* - 1;
    IF YMIN!* > YCOORD!*
    THEN YMIN!* := YCOORD!*;
    IF L
    THEN
        if !*NOCOMMA then
          for each x in L do prin2!*(x)
        else INPRINT('!*COMMA!*,0,L);
    YCOORD!* := YCOORD!* + 1;
    RETURN S;
  >>
  ELSE
  <<X := REMPROP(OP,'PRIFN);
    MAPRIN(OP . L);
    PUT(OP, 'PRIFN,X);
    RETURN POSN!*;
  >>;
```

```
END;


SYMBOLIC PROCEDURE NUMBCH(L);   %Modified on  purpose;
NULL(L) OR (ATOM(CAR L) AND NUMBCH(CDR L));


ALGEBRAIC;

END$
```

The below given program is used for the calculations involved in the second physical problem considered in this thesis.

```
% 5 DIM -> 4 x 1 Reduction (Phi field present)           ;
% Author:G. Ucoluk,    Date:May 1989,    Place:Karlsruhe ;

symbolic operator fdegree;
symbolic procedure allbaseonep u;nil;
algebraic;
forms e=1, RR=2,R=2,FF=2,DD=0; %Just to get formdeps ok. 0 will go;
spacedim 5; lisp;superspcdim!* := 5;algebraic;
signature -1,1,1,1 ;
operator R,F,DEL,DD,E,RR;   symbolic remprop('dd,'fdeg);
doindex R,F,E;
on nocomma; off letpreval,killsuper;
antisymmetric R,F$
split AA,A,v;
split BB,B,v;
split CC,C,v;
split GG,D,v;   % Not used DD , since DD stands for the cov. deriv.;
putindexkind 1,  g,h,p,q;
xtrop VHODGE;
fdegreefn ADD1,DD;
fdegreefn VHODGEDEGFN,VHODGE;  symbolic procedure VHODGEDEGFN u;
                                    superspcdim!* - u;
for all a,b,c,d  such that indexkind a=1 and indexkind b=1
  let RR(a,b,c,d) = R(a,b) - phi**2/2*F(a,b)*FF
           - phi**2/4*F(a,c)*F(b,d)*(e c & e d)
           - 1/2*DD(phi*F(a,b)) & e v
           -1/2*(F(a,b)*DEL(c,phi)-DEL(a,phi)*F(b,c)+DEL(b,phi)*F(a,c))
               *(e c & e v);
for all a,b,c such that indexkind a=1
     let RR(v,a,b,c) = 1/2*DD(phi*F(a,b)*e b) + DEL(a,phi)*FF
                       + (DD(DEL(a,phi)) & e v)/phi
                       + phi**2/4*F(a,c)*F(c,b)*(e b&e v),
         RR(a,v,b,c) = -RR(v,a,b,c);

 for all x,a,b let RR(x,x,a,b)=0;  %it is antisymmetric;


 for all a such that freeofindex(a,2) and freeofindex(a,0)
     let       VHODGE a = # a & e v,
               VHODGE(e v& a) =  # a,
               VHODGE(a & e v) = -# a;

for all a,b such that freeofindex(a,2) and freeofindex(a,0) and
                      freeofindex(b,2) and freeofindex(b,0)
    let       VHODGE(a & e v & b) = (-1)**(fdegree a) * #(a & b);

let VHODGE VHODGE e v = - e v; % Carefully done with spacedim=5;

on letpreval;

%Here we go on with the Lagrangian;
dimred(
          RR(AA,BB,c,d) & VHODGE RR(AA,BB,g,h)
        + alpha/2*RR(AA,BB,c,d) & VHODGE(e AA & e BB)
      ,4);    % 4:new space dim;

end$
```

VITA

Göktürk Üçoluk was born in 1958 in Istanbul as a Turkish citizen. He graduated from Bosphorus University, Istanbul, in 1980 receiving a BS degree in Physics. At the same university he completed his MS study in 1982. His MS thesis was on *"Symbolic Algebraic Computations in Physics"*. Since 1982 he is working as a research assistant at METU, Ankara. Göktürk Üçoluk is married and has one daughter.