An Algorithm to Simplify Tensor Expressions

R. Portugal[1]
Department of Applied Mathematics, University of Waterloo, Waterloo, Ontario, N2L 3G1 - Canada.

Abstract

The problem of simplifying tensor expressions is addressed in two parts. The first part presents an algorithm designed to put tensor expressions into a canonical form, taking into account the symmetries with respect to index permutations and the renaming of dummy indices. The tensor indices are split into classes and a natural place for them is defined. The canonical form is the closest configuration to the natural configuration. In the second part, the Gröbner basis method is used to simplify tensor expressions which obey the linear identities that come from cyclic symmetries (or more general tensor identities, including non-linear identities). The algorithm is suitable for implementation in general purpose computer algebra systems. Some timings of an experimental implementation over the Riemann package are shown.

1 Introduction

Recently, some attempts to describe algorithms for simplifying tensor expressions have appeared in the literature.[1][2] The tensor symmetry properties and the presence of dummy indices make the problem complex. Fulling et al.[1] present a method to put monomials containing scalars or tensors formed from the Riemann tensor and its covariant derivative into normal forms. They use the representation theory of the symmetric, general linear and orthogonal groups to classify and to present bases for the possible expressions. Ilyin and Kryukov[2] represent tensor expressions as vectors in a linear space of dimension $n!$, where n is the number of indices of the expression. They formulate an algorithm to simplify tensor expressions based on the group algebra of the permutation group. Though the method is elegant from the mathematical point of view, it is inefficient, since the complexity of the algorithm is $\mathrm{O}(n!)$.

In this work, the problem is addressed in two parts. In the first, the tensor expressions are put into a canonical form taking into account the symmetries with respect to index permutations and renaming of dummy indices. In the second part, the Gröbner basis method (see Geddes at al.[3] and refs. therein) is used to address the linear identities that come from cyclic symmetries (or more general tensor identities). The algorithm is suitable for implementation in general purpose computer algebra systems, since many functionalities of these systems (besides the Gröbner basis implementation) are required.

[1]On leave from Centro Brasileiro de Pesquisas Físicas, Rua Dr. Xavier Sigaud 150, Urca, Rio de Janeiro, Brazil. CEP 22290-180. Email: portugal@cat.cbpf.br

It is known that the names of the dummy indices have no intrinsic meaning, and the presence of two or more of them in a tensor expression creates symmetries with respect to renaming. This leads to algorithms of complexity O(n!) where n is the number of dummy indices. Dummy indices difficult the definition of tensor rules and the use of pattern matching. One way to solve the dummy index problem is to rename them in terms of the index positions and the name of the tensors (see step 9 of the algorithm). However, the tensor symmetries may change the index positions, invalidating the process. This kind of renaming method is invariant if there is a prescribed method to put the indices into a canonical position.

In section 2, an algorithm to put tensor expressions into a canonical form is described. The canonical position of the indices is based on definitions 3 to 6. Some parts of the definitions of this section involve conventions that can be changed without losing the "canonization" character of the algorithm. The ideal format for the canonical tensor is

$$T^{F^+\ S_{II}\ A^+\ B^+\ C^+\ S_I}{}_{C^-\ B^-\ A^-\ F^-}$$

where F^+, S_{II}, A^+, B^+, C^+, S_I, C^-, B^-, A^- and F^- represent sequences of indices whose meanings are defined in step 7 of the algorithm. The indices of class S_I can have contravariant or covariant character. They are placed in the midst of the classes that have the character fixed. This ideal format is only achieved in some circumstances, as when the tensor T is totally symmetric or antisymmetric. The canonical position of the indices is the one closest to this format, where the notion of "closest" is precisely defined.

In section 3, some timings of an experimental implementation of the algorithm over the Riemann package[4] are presented. Polynomials built from the Riemann tensor are challenges for any algorithm to simplify tensor expressions. I believe that special techniques can improve the timings for the kind of symmetries of the Riemann tensor, but none has been implemented for the demonstration of this section. The implementation in the Maple system[5] can easely handle products of three Riemann tensors.

In section 4, the problem of the simplification of tensor expressions under the presence of side tensor identities, which come generally from cyclic symmetries, is addressed. The Gröbner basis implementation of the Maple system is used to accomplish the full simplification.

2 The algorithm

Consider the definition of the normal and canonical functions given in Geddes et al.[3]. Let E be a set of expressions which admits a canonical function. Consider the operations of multiplication, addition and contraction of tensors as defined in the tensor algebra.[6][7][8] If a coordinate system has been selected, the tensor algebra can be performed through the tensor components. In this work, a tensor expression is any expression written in terms of non-assigned tensor components obeying the rules of the tensor algebra whose coefficient factors are members of the set E. Consider Riemannian spaces, in which exists a fundamental metric tensor which establish a relation between the covariant and contravariant tensor indices.

2.1 Definitions

Hypothesis 1: Tensor expressions do not obey any side tensor identities except the symmetries with respect to index permutations or the symmetries with respect to renaming or the inversion of

character of dummy indices.

"Symmetries with respect to index permutations" means that the tensors obey one or more equations of the kind:

$$T^{\,i_1\,\ldots\,i_n} = \epsilon\, T^{\,\pi(i_1\,\ldots\,i_n)}$$

where $\epsilon = 1$ or $\epsilon = -1$ and $\pi(i_1 \ldots i_n)$ is a permutation of $i_1 \ldots i_n$.

Definition 1: *Induced symmetry* of a sub-set of indices of a tensor.

The *induced symmetries* of a sub-set of indices of a tensor are the symmetries that the sub-set inherits from the symmetries of the whole set of indices. Pairs of dummy indices are treated as independent free indices.

For example, the *induced symmetry* of the first two indices of the Riemann tensor is the skew symmetry. The second and third indices have no *induced symmetry* regardless any contraction between the first and fourth indices.

Definition 2: *Equivalent* index configurations.

Two index configurations[2] of a tensor T extracted from a tensor product are said to be *equivalent* if one configuration can be put into the other by the use of any of the following properties:

1. Character inversion of the dummy indices,
2. Renaming the dummy indices,
3. Index permutation allowed by the symmetries of the tensors of the tensor product.

Definition 3: Suppose the tensor T has n indices and let (λ_1, ..., λ_p) be a partition of n where p is a positive integer less than n. The indices of T can be grouped in disjoint classes C_1 ..., C_p where a generic class C_i has λ_i indices. The indices of each class can be substituted with numbers in such way that the indices of class C_i run from $(\sum_{j=1}^{i-1} \lambda_j) + 1$ to $\sum_{j=1}^{i} \lambda_j$. Consider all index configurations of the tensor T and let the indices be substituted with the corresponding numbers. The configurations are in one-to-one correspondence with the elements of the symmetric group G_n.[9][10] The following criteria put in order of decreasing configurations with respect to classes C_1 to C_p:

a. Smaller value of the position of the first index of class C_1 in the tensor T. If the positions are equal, consider the position of the next index of class C_1. If the positions of all indices of class C_1 are equal, consider the positions of the indices of the next classes until C_p.

b. Smaller value of the first index member of class C_1 that appears in the tensor T. If the first indices of class C_1 that appear in all configurations are equal, consider the next index member of class C_1. If the indices of class C_1 appear in the same order, consider the order of the indices of the next classes until C_p.

Definition 3a alone compares the position of the classes, while definition 3b alone compares the order of the indices. Given a set of *equivalent* configurations of a tensor T, definition 3 allows one to select the smallest configuration of the set with respect to a given partition of the number of indices. The smallest configuration is unique. If definition 3b is not applied, or if it is applied for some but not all classes, instead of having one smallest configuration, one may have a sub-set of smallest configurations.

[2]The index configuration is the list of indices of the tensor.

Definition 4: *Character normal configurations.*

Let C^+ and C^- be the classes of the contravariant and covariant indices respectively of a tensor T extracted from a tensor product. Consider the set of all *equivalent* index configurations of T. The *character normal configurations* consist of the sub-set of smallest index configurations according to definition 3a with respect to classes C^+ and C^-.

Definition 5: *Index normal configurations.*

Consider the definition of group I and II given in step3 and the definition of classes F^+ , S_{II} , A^+ , B^+ , C^+ , S_I,[3] C^- , B^- , A^- and F^- given in step 7a. The present criteria applies when a tensor (T) of group I is extracted from a tensor product. Consider the set of all *equivalent* index configurations of T. Let the indices be relabelled as described in step 7c. The *index normal configurations* of the tensor T consist of the sub-set of smallest configurations that are *character normal configurations*, and satisfy definition 3a for classes F^+ , S_{II} , A^+ , B^+ , C^+ , S_I , C^- , B^- , A^- and F^- and definition 3b for classes F^+ , S_{II} , B^+ , C^+ , C^- , B^- and F^-.

The order of the indices of classes A^+ and A^- and the order of the indices of the sub-classes of S_I are not considered in the definition of the *index normal configurations.*

Definition 6: *Index canonical configuration.*

The *index canonical configuration* is the only element of the set of *index normal configurations* which fully satisfies definition 3 with respect to classes F^+ , S_{II} , A^+ , B^+ , C^+ , S_I , C^- , B^- , A^- and F^- in this order.

Lemma 1: **a)** The set of *index normal configurations* has one element if and only if classes A and S_I have less than two elements.

b) Consider a tensor product, and let all equivalent tensor products be generated by all possible *equivalent* configurations of the indices of the tensors. There is either one or two products with the tensors in the *index canonical configuration.* If there are two products, they are equal but have opposite signs, yielding a null expression.

Proof: **a)** If class A or class S_I have at least two elements, it is possible to generate more than one *index normal configuration* by index permutation. If classes A and S_I have less than two elements, definition 3b (applied to the other classes) selects only one configuration.

b) The criteria of definition 3 select only one configuration from a set of configurations of indices. Two tensor products with the *index canonical configuration* may be generated when there are antisymmetric dummy indices. In this case, the tensor product is zero.

Rule 1: Consider a tensor product. The character of the first (from left to right) index of a pair of summed indices (if some exists) is contravariant, and the character of the second is covariant. The same rule applies for the summed indices within a single tensor.

[3] S_I is a collection of sub-classes, which have the same status of the other classes in this definition. The order of the sub-classes is described in step 7c.

2.2 The algorithm

The algorithm is divided into steps grouped by types of action that must be followed in increasing order unless otherwise stated. The goal is to put the indices into a canonical position. The canonical position involves the relative position of the indices inside the tensor as well as their character. All dummy indices are renamed and the original names are thrown away. The renaming of the dummy indices takes into account their position inside the tensors and the order of the tensors. Therefore the renaming process leads to canonical names only after the dummy indices have been put into a canonical position and an ordering for the tensors has been established.

Step 1.(Expanding) Expand the tensor expression modulo the coefficients. Select the tensor factors of the first term.[4] From now on, consider how to put a tensor product into a canonical form. Steps 1 to 9 are applied to all terms of the expanded expression, one at a time.

Step 2.(Raising or lowering indices) If there are any metric tensor with indices contracted with other tensors, the corresponding indices are raised or lowered according to the contraction character. The same goes for other special tensors built from the metric.

Step 3.(Splitting in group I and II) Split the product in two groups. Group I consists of tensors that have symmetries. Group II consists of tensors with no symmetries. Tensors of group I are placed in the left positions and tensors of group II in the right positions using the commutativity property of the product.

Step 4a.(Merging equal tensors) Tensors of group II with the same name and same number of indices and with no free indices merge into a new tensor. Suppose that each original tensor has n indices and that there are N equal tensors, then the new tensor has nN indices and is totally symmetric under the interchange of the groups of the n indices. After the merging, the resulting tensors are incorporated into group I. The information about the relation with original tensors is stored, since it will be used at the end of the algorithm to substitute the new tensor with the original tensor names.

Step 4b. Tensors of group I with the same name, same number of indices and same number of free indices merge to form a new tensor. The new tensor has the same symmetry under the interchange of group of indices as described in step 4a, and each group of indices inherits the symmetries of the original tensors. In step 4a, only tensors of group II with no free indices are merged. In this step, the tensors may have free indices.

Step 5a.(Sorting the tensor names) The tensors are lexicographically sorted inside each group. Tensors with same name but with different number of indices are different. They are sorted according to the number of indices.

Step 5b. Tensors with the same name and same number of indices are sorted according to the number of free indices.

[4]For expressions consisting of one tensor, only steps 6 and 9 are performed if the tensor has no symmetries; steps 7 and 9 are performed if the tensor has symmetries. Classes S_I and S_{II} are empty is this case.

Step 5c. Tensors of group II with the same name, same number of indices, and same number of free indices are sorted according to first free index. Notice that the free indices have fixed positions for tensors of group II.

Step 6.(Fixing the index character of group II) Consider the tensors of group II. The summed indices that are contracted within the tensors or those that are contracted with other tensors of group II may have their character changed in order to obey rule 1. All other summed indices (the ones contracted with tensors of group I) are put covariant. The free indices remain untouched.[5].

Step 7a.(Splitting in classes) Step 7 is performed for all tensors of group I. Consider the first tensor of group I (the current tensor). It has symmetries that can involve all indices or only some of them. The discussion that follows applies only for the indices involved in the symmetries or contracted with indices involved in the symmetries. The pairs of summed indices not involved in the symmetries must obey rule 1 and the free indices not involved in the symmetries remain untouched throughout the whole algorithm. Consider the indices that can have their positions affected by the symmetry. They are split into classes:

Class F^+: contravariant free indices

Class S_{II}: summed indices contracted with tensors of group II.

Classes A^+ , B^+ , C^+ , A^- , B^- , C^-: summed indices inside the tensor

Class S_I: summed indices contracted with tensors of group I.

Class F^-: covariant free indices

Let f^+ , s_{II} , a , b^+ , c^+ , s_I , F^+ , A , B^+ , C^+ , S_I , C^- , B^- and F^- respectively. Classes A^+ and A^- have the same size.

The summed indices inside the tensor are members of classes A^+ , B^+ , C^+ , A^- , B^- or C^-. When both indices of a pair of summed indices are involved in the symmetries, the contravariant index is a member of class A^+, and the covariant is a member of class A^-. If a index of a pair of summed indices is involved with the symmetries while the corresponding one is not, there are four cases. Let us call i^+ the index involved in the symmetries and i^- the corresponding index not involved in the symmetries. These indices are equal but have different characters (here the signs + and - do not describe the character). Suppose that the relative position of i^+ with respect to i^- cannot be inverted due to the symmetries. If i^+ is at the right of i^-, then i^+ is a member of class B^+; if i^+ is at the left, then i^+ is a member of class B^-. If the relative positions of i^+ and i^- can change, then i^+ is a member of class C^+ or C^- depending on whether i^+ is contravariant or covariant.

The indices of class S_I are contracted with the indices of the tensors of group I. Some of the latter indices may not be involved in the symmetries. They form sub-class ${S_I}^0$, which is further split into ${S_I}^{0^+}$ and ${S_I}^{0^-}$, corresponding to the indices of ${S_I}^0$ contracted with the tensor to the right or to the left of the current tensor respectively. The remainder indices of S_I are split into sub-classes. The indices of the current tensor contracted with the next (to the right) tensor of group I form the first sub-class (${S_I}^1$); the indices contracted with the next tensor form the second sub-class (${S_I}^2$), and so on until the last tensor of group I. If the current tensor is the first of group I, the sub-division in classes is complete; otherwise the sub-division continues and the next sub-class consists of the indices the current tensor contracted with the first tensor of group I. The following sub-class consists of the indices contracted with the second tensor of group I and so on until the last tensor of group I that

[5]This step can be displaced and performed together with step 9

has not been considered yet.

Step 7b.(Fixing the index character of group I) In order to obey rule 1, the indices of class S_{II} are put contravariant. The corresponding indices of tensors of group II have already been put covariant in step 6. The indices of class S_I are put contravariant if they are contracted with the tensor at the right, or covariant if they are contracted with the tensor at the left of the current tensor. The indices of class B^+ are put contravariant and the corresponding ones are put covariant. The indices of class B^- are put covariant and the corresponding ones are put contravariant. The character of the indices of classes F^- is maintained throughout the whole algorithm. The character of classes A^+ , C^+ , A^- and C^- may still change.

Step 7c.(Ordering and the numbering of the indices) This step establishes the order of the indices of classes F^+ , S_{II} , B^+ , C^+ , S_I , C^- , B^- and F^- to be used in step 7e. Also, it specifies the numbers each index receives in order for definition 3 to be applied. To begin with, let us discuss the order of the indices of classes F^+ and F^-. First, sort the free indices of class F^+ using their original names. The first sorted free index is substituted with the number 1 , the second by 2 , and so on until the f^+ th index, which is substituted with f^+ . The same process is performed for the indices of class F^- , which receive the numbers $f^+ \,..\, a^- + 1$, ..., $f^+ \,..\, f^-$.[6]

Class S_{II} consists of summed indices contracted with the tensors of group II. At this point, the tensors of group II are ordered. The positions of the indices of these tensors are used as a reference to order the indices of class S_{II}. The first summed index in group II (from left to right) that is a member of class S_{II} is the first index of S_{II}. It receives the number $f^+ + 1$. The second receives the number $f^+ + 2$ and so on until $f^+ + s_{II}$.

The indices of classes B^+ , B^- , C^+ and C^- are contracted with indices that have no symmetries; therefore, they have an ordering reference. They follow the same method used for the indices of class S_{II}.

Class S_I can have its sub-classes ordered. The first sub-class is ${S_I}^{0^+}$, followed by sub-classes ${S_I}^1$, ${S_I}^2$ and so on, and the last sub-class is ${S_I}^{0^-}$. The indices of the classes ${S_I}^{0^+}$ and ${S_I}^{0^-}$ can be ordered since they are contracted with indices not involved in the symmetries. They follow the same method used for the indices of class S_{II}. The indices of the other sub-classes cannot be ordered at this point. Also, the indices of class A cannot be ordered at this point.

Here follows, explicitly, the numbers reserved for each class:

Class F^+ : 1 , 2, ..., f^+

Class S_{II} : $f^+ + 1$, $f^+ + 2$,..., $f^+ + s_{II}$

Class A^+ : $f^+ + s_{II} + 1$, ..., $f^+ \,..\, a^+$

Class B^+ : $f^+ \,..\, a^+ + 1$, ..., $f^+ \,..\, b^+$

Class C^+ : $f^+ \,..\, b^+ + 1$, ..., $f^+ \,..\, c^+$

Class S_I : $f^+ \,..\, c^+ + 1$, ..., $f^+ \,..\, s_I$

Class C^- : $f^+ \,..\, s_I + 1$, ..., $f^+ \,..\, c^-$

Class B^- : $f^+ \,..\, c^- + 1$, ..., $f^+ \,..\, b^-$

Class A^-: $f^+ \,..\, b^- + 1$, ..., $f^+ \,..\, a^-$

Class F^- : $f^+ \,..\, a^- + 1$, ..., $f^+ \,..\, f^-$

[6]The notation $f^+ \,..\, f^-$ means $f^+ + s_{II} + a^+ + b^+ + c^+ + s_I + c^- + b^- + a^- + f^-$. The variables a^+ and a^- are equal to a.

Step 7d.(Generating the character configurations) Both indices of a pair from class A can change their positions due to the symmetries, but the relative position of some pairs may be fixed. The characters of the pairs, that cannot invert the relative position, can be chosen such that they obey rule 1. The characters of the remainder indices of class A (let a_0 be the number of pairs of class A that can invert their relative position) and the characters of indices of class C cannot be chosen a priori. Each pair has two states which are contravariant-covariant and covariant-contravariant, corresponding to the characters of the pair. The algorithm generates $2^{(a_0+c)}$ possible character configurations by changing the states of pairs that can invert their relative position. For each character configuration, step 7d' is performed to verify whether the *induced symmetries* of classes A^+ and A^- cancel the tensor.

If the *induced symmetry* of class A is totally symmetric or antisymmetric, it is not necessary to generate the 2^{a_0} character configurations of pairs of class A. The symmetry takes care of the pair inversion when necessary. If there are antisymmetric contracted indices, the tensor is null.

Step 7d'.(Verifying cancellation) Consider a character configuration generated in step 7d. Classes A^+ and A^- have *induced symmetries* which allow the indices of each class be interchanged. If the interchange of any two indices of class A^+ produces a change of sign while the interchange of the same indices in class A^- does not produce a change of sign, or vice-versa, then the current tensor product described in step 1 is zero. If so, the algorithm returns to step 2 for the next term.

Step 7e.(Applying the symmetries) At this point, more than one *equivalent* configuration may have been generated. The symmetries are applied[7] to all configurations in order to perform the following tasks. The contravariant indices are pushed to the left positions as far as possible and the covariant indices to right positions as far as possible. A sign change may be generated if there are antisymmetric indices. The state configurations that are members of the set of *character normal configurations* of the current tensor are selected. The symmetries are applied to the selected configurations again in order to put classes F^+ , S_{II} , A^+ , B^+ , C^+ , S_I , C^- , B^- , A^- and F^- as closely as possible in the smallest position as prescribed in definition 3a, and after that, the indices of classes F^+ , S_{II} , B^+ , C^+ , C^- , B^- , and F^- are put as closely as possible in their order as prescribed in definition 3b (see step 7c for relabelling). One configuration that is a member of the set of *index normal configurations* is selected. It does not matter which one is selected. At this point, the character and the positions of all classes have been determined, except the order of the indices of class A and of the indices of the sub-classes of S_I (not including ${S_I}^0$).

Step 7f.(Generating *equivalent* configurations of classes A and S_I) At this point, the character and the position of classes A^+ and A^- have been determined, but not the order of the indices. If the *induced symmetry* of class A^+ or A^- is totally symmetric or antisymmetric, the indices can be sorted taking the corresponding ones as reference. The antisymmetric indices may generate a sign change that must be considered. For the general kinds of *induced symmetries*, the selected configuration is submitted to all possible re-orderings of classes A^+ and A^- allowed by the *induced symmetry* of each class. The smallest configuration under the criteria of definition 3b among all these re-orderings is

[7]The application of the symmetries is a straightforward procedure that can be implemented for each kind of symmetry. In the case of the Maple system, tensors can be represented by tables and the symmetries by indexing functions, which can perform the tasks described in step 7e.

selected. The number of terms generated in this step is the product of the number of permutations allowed by the *induced symmetries* of classes A^+ and A^-.

Now consider sub-class ${S_I}^1$. It involves two tensors. The same set of indices appears twice - contravariantly in the first tensor, and covariantly in the second tensor. Both the contravariant and covariant indices have *induced symmetries*. The current tensor (in a normal form) is submitted to all possible re-orderings allowed by the *induced symmetry* of sub-class ${S_I}^1$, and the corresponding tensor is submitted to all possible re-orderings allowed by the corresponding *induced symmetry*. Notice that this process is equal to the one described for indices of class A . Therefore, one can define a new tensor totally contracted, having sub-class ${S_I}^1$ as the contravariant indices and having the corresponding indices as the covariant indices. By a recursive call, the indices of the new tensor will be in classes A or B and will be ordered by the method described for this class. The result is used to order sub-class ${S_I}^1$ and the corresponding indices. The same method is performed for sub-class ${S_I}^2$ and so on.

Step 7g.(Selecting the *index canonical configuration*) The indices of all configurations are substituted with their correspondent numbers (step 7c), and the *index canonical configuration* with respect to classes F^+ , S_{II} , A^+ , B^+ , C^+ , S_I , C^- , B^- , A^- and F^- is selected.

Step 8.(Recovering merged tensors) The symmetric (by group of indices) tensors that have been formed by merging tensors with equal names, equal number of indices and equal number of free indices in step 4a and 4b are converted back by the inverse process to a product of tensors with the original names but with the new indices generated by the previous steps.

Step 9.(Renaming the dummy indices) At this stage of the algorithm, the indices are in their final position. The dummy indices are renamed following the rules: There are two cases. The first occurs when the whole pair of dummy indices resides inside a tensor. If the tensor name is A and the number of indices is m , then the dummy index name will be $A\_m\_i\_j$, where i is the position of the contravariant index and j is the position of the covariant index and _ is some separator.[8] If the same name appears in other tensors of the product, no name conflict is generated. The second case occurs when the dummy indices involve two tensors. Suppose that the first tensor has the name A with m indices and the second has the name B with n indices, then the dummy index will be renamed $A\_m\_i\_B\_n\_j$, where i is the position of the contravariant index inside the tensor A and j is the position of the covariant index inside the tensor B . This renaming process proceeds from left to right. If a second dummy index receives the same name, then the number 1 is appended in its name $A\_m\_i\_B\_n\_j\_1$, for example. If a third summed index receives the same name, then the number 2 is appended in its name: $A\_m\_i\_B\_n\_j\_2$, and so on.[9]

Step 10.(Collecting equal tensor terms) After performing steps 1 to 9 for all terms, collect equal tensor products and put the coefficient factors into the canonical form.

Step 11.(Sorting the addition)[10] Each tensor product can be converted to a string by concatenating

[8]The separator is a symbol not present in tensor expressions.

[9]The method of appending a number to the repeated dummy index names can be fully avoided if step 8 is performed after step 9.

[10]In general, this step is unnecessary, for practical purposes, in the implementation of this algorithm in the multiple

with separators the names of the tensors and the indices in the order they appear in the product. These strings are sorted. The order of the terms in the sum is rearranged to be in the same order as the sorted concatenated strings.

2.3 Proof that the algorithm is a canonical function

Theorem: Under hypothesis 1, the algorithm described is a canonical function when applied to tensor expressions.
Proof: It is clear from the sorting uniqueness that the position of the tensors of group II in a tensor product is unique after the application of the algorithm. It is easy to verify that the characters of the indices of tensors of group II and the indices of classes F^+ , S_{II} , B^+ , S_I , B^- and F^- of tensors of group I also go to a unique character configuration, since this is a matter of convention. Therefore, all tensors of group II go to a unique index configuration and position in the tensor products.

The proof that tensors of group I (including the merged tensors of step 4) come to a unique configuration is a consequence of lemma 1 and the fact that the algorithm runs over all index character configurations of the indices of classes A and C and over all allowed index position configurations of classes A and S_I .

The algorithm generates all index configurations that cannot be fixed by convention, and lemma 1 ensures that only one configuration is selected as the canonical configuration.

2.4 Examples

Example 1: Consider the tensor expression $R^{i}{}_{b\,a\,i}$ (contraction of the Riemann tensor). Step 7a establishes that classes A and F^- are the only non empty classes in this case. Class A^+ is $[i^+]$, class A^- is $[i^-]$ and class F^- is $[a,\ b]$. Step 7c establishes that the contravariant index i receives the number 1, the covariant index i receives the number 2, and the indices a and b receive the numbers 3 and 4 in this order, since a precedes b. Step 7d establishes that there are two character configurations to be considered: $R^{i}{}_{b\,a\,i}$ and $R_{i\,b\,a}{}^{i}$. Since there is only one summed index, there are no *induced symmetries*, and step 7d' does not apply in this case. The symmetries are applied (step 7e) in order to put these configurations into the equivalent ones $R^{i}{}_{b\,i\,a}$ and $R^{i}{}_{a\,i\,b}$. Both are selected, since both are members of the set of *character normal configurations*. No further changes are generated by the symmetries, so the configurations that are members of the set of *index normal configurations* must be selected. The indices are substituted with their respective numbers, yielding [1,4,2,3] and [1,3,2,4] respectively. Definition 3a with respect to the partition (1,1,2) – corresponding to classes [1], [2] and [3,4] – selects both configurations. Definition 3b with respect to class [3,4] selects [1,3,2,4] as the only member of the set of *index normal configurations*. Since there are no *induced symmetries*, no more index configurations are generated. The *index canonical configuration* is [1,3,2,4] . Therefore, the canonical form is $R^{i}{}_{a\,i\,b}$, where i is *R_4_1_3*, as prescribed in step 9.

Example 2: Consider the tensor expression $T^{i}\,S^{l\,k}\,T^{j}\,R_{j\,k\,i\,l}$, where $S^{l\,k}$ is a totally antisymmetric tensor and $R_{i\,k\,j\,l}$ is the Riemann tensor. Step 3 splits the expression into group I: $[S^{l\,k},\ R_{j\,k\,i\,l}]$

purpose computer algebra systems.

and group II: $[T^{i}, T^{j}]$. Step 4b merges T^{i} and T^{j} into one totally symmetric tensor (let be called $T\_T^{i\,j}$) and adds to group I. Group II is empty now. Step 5a establishes the order for group I: $[R_{j\,k\,i\,l}, S^{l\,k}, T\_T^{i\,j}]$. Step 7a establishes that the only non-empty class in this case is S_I. For the first tensor we have ${S_I}^{1} = [k, l]$ and ${S_I}^{2} = [j, i]$. The order of these indices has not yet been established. Step 7b fixes the character as: $[R^{j\,k\,i\,l}, S_{l\,k}, T\_T_{i\,j}]$. Step 7e converts $R^{j\,k\,i\,l}$ into $R^{k\,j\,l\,i}$. Concerning class ${S_I}^{1}$, step 7f generates a new tensor; let us call $N^{k\,l}{}_{l\,k}$, antisymmetric in the last two indices. The algorithm is called recursively, and $N^{k\,l}{}_{l\,k}$ is converted to $-N^{k\,l}{}_{k\,l}$. Therefore, tensor $S_{l\,k}$ is converted to $-S_{k\,l}$. A similar situation happens with class ${S_I}^{2}$, and the tensor $T\_T_{i\,j}$ is converted to $T\_T_{j\,i}$. After step 8 we have: $[R^{k\,j\,l\,i}, -S_{k\,l}, T_{j}, T_{i}]$. Step 9 establishes that canonical form for the expression is $-R^{k\,j\,l\,i}S_{k\,l}T_{j}T_{i}$ where $k = R\_4\_1\_S\_2\_1$, $j = R\_4\_2\_T\_1\_1$, $l = R\_4\_3\_S\_2\_2$ and $i = R\_4\_4\_T\_1\_1$.

3 An experimental implementation

Algorithms to simplify tensor expressions have been implemented in some computer algebra systems.[11][12][13][14] Most implementations use pattern matching, which requires a big database of tensor rules. The underlying method used by the Ricci package[11] seems similar to the method presented here. All these implementations have not solved the dummy index problem, therefore the main simplifier spends a long time or cannot simplify tensor expressions with 8 or more dummy indices.

In this section, I present an experimental implementation of the algorithm described in section 2 over the Riemann package.[4]. The new package can be obtained from web sites,[11] and my purpose is to supersede the Riemann package in the near future with the new functionalities introduced to deal with tensor components abstractly.

The function *normalform* uses the algorithm to put tensor expressions into normal forms. No attempt has been made to put the output into the canonical form, that is, ordered with respect to the sum and to the product of terms, since this last step is unnecessary for the purpose of any general computer algebra system.

In the examples below, contravariant indices have positive signs and covariant indices have negative signs. Tensors are indexed variables and their symmetries are declared by the command *definetensor*. Some tensors are pre-defined: Christoffel symbols, Riemann tensor, Ricci tensor and Ricci scalar are pre-defined with the names Γ (Christoffel symbols) and R (Riemann, Ricci and Ricci scalar). More details can be found in the new help pages for the commands *normalform*, *definetensor* and *symmetrize*, and in the help pages of the Riemann package.

Expressions that are zero due to tensor symmetries are readily simplified:[12]

```
> readlib(showtime)():
> definetensor(T[i,j,k,l], sym[1,2] and asym[3,4]);
```

$$T^{i\,j\,k\,l}$$

[11] See the addresses: http://www.cbpf.br/~portugal/Riegeom.html or http://daisy.uwaterloo.ca/~rportuga/Riegeom.html

[12] All calculations have been performed in a Pentium 120 MHz with 32 Mb of RAM, running Maple V release 5 over Windows 95.

time = 0.01, bytes = 13478
> expr1 := printtensor(T[i,j,k,l]*T[-k,-l,m,n]);

$$T^{i\,j\,k\,l}\,T_{k\,l}{}^{m\,n}$$

time = 0.02, bytes = 4854
> normalform(expr1);

$$0$$

time = 0.10, bytes = 91586
> expr2 := printtensor(T[i,j,k,l]*V[-i]*V[-j]+V[b]*V[a]*T[-a,-b,l,k]);

$$T^{i\,j\,k\,l}\,V_{i}\,V_{j}+V^{b}\,V^{a}\,T_{a\,b}{}^{l\,k}$$

time = 0.01, bytes = 8042
> normalform(expr2);

$$0$$

time = 0.30, bytes = 338050
> off;

Polynomials constructed with the Riemann tensor exemplify the algorithm's worst performance. No special techniques have been implemented for the kind of symmetries of the Riemann tensor. In the next example, all possible ways to write the scalars formed by the product of two Riemann tensors are generated. The 40.320 expressions reduce to 4 independent non-null forms if the cyclic identity of the Riemann tensor is not considered. In the next section, we can verify that the cyclic identity reduces to 3 independent scalars (cf. ref. [1]).[13]
> S := {op(combinat[permute]([a,b,c,d,-a,-b,-c,-d]))}:
> nops(S);

$$40320$$

> readlib(showtime)():
> S1 := map(x->abs(normalform(R[op(1..4,x)]*R[op(5..8,x)])), S);

$$\{0,\ \left|R^{R1R1\ R2R2\ R3R3\ R4R4}\,R_{R1R1\ R2R2\ R3R3\ R4R4}\right|,\ \left|R^{R1R1\ R2R3\ R3R2\ R4R4}\,R_{R1R1\ R3R2\ R2R3\ R4R4}\right|,$$
$$\left|R_{R1R1\ R2R2}\,R^{R1R1\ R2R2}\right|,\ |(R)|^{2}\}$$

time = 35094.97, bytes = 21215820858

The program spends less than one second per expression on average. Next, one example involving the product of three Riemann tensors is provided:
> normalform(R[-c,-d,m,n]*R[a,b,d,c]*R[-n,-m,-b ,-a]);

$$-R^{R1R3\ R2R4\ R3R1\ R4R2}\,R^{R1R3\_1\ R2R4\_1}{}_{R1R3\ R2R4}\,R_{R3R1\ R4R2\ R1R3\_1\ R2R4\_1}$$

time = 26.41, bytes = 6303158
> off;

[13]Simplified names for the dummy indices are being using for this demonstration.

4 Simplification of tensor expressions

The algorithm of section 2 achieves a full simplified form of the tensor expressions if the tensors do not obey side identities. To accomplish the simplification of tensor expressions obeying side relations, the Gröbner basis method is used. The general strategy is to put the tensor expression and the side relations into a canonical form of the algorithm of section 2, and simplify the new tensor expression with respect to the new side relation.

Here I present an example of how the Gröbner basis method is used to simplify the expression $R^{a\,b\,c\,d}\,R_{a\,c\,b\,d} - 1/2\,R^{a\,b\,c\,d}\,R_{a\,b\,c\,d}$.

```
> expr := printtensor(R[a,b,c,d]*R[-a,-c,-b,-d]-1/2*R[a,b,c,d]*R[-a,-b,-c,-d]);
```

$$expr := R^{a\,b\,c\,d}\,R_{a\,c\,b\,d} - \frac{1}{2}\,R^{a\,b\,c\,d}\,R_{a\,b\,c\,d}$$

```
> EXPR := normalform(expr);
```

$$EXPR := R^{R1R1\ R2R3\ R3R2\ R4R4}\,R_{R1R1\ R3R2\ R2R3\ R4R4} - \frac{1}{2}\,R^{R1R1\ R2R2\ R3R3\ R4R4}\,R_{R1R1\ R2R2\ R3R3\ R4R4}$$

```
> side_rel := printtensor(R[a,b,c,d]*symmetrize(R[-a,-b,-c,-d],cyclic[b,c,d]));
```

$$side\_rel := R^{a\,b\,c\,d}\,(\frac{1}{3}\,R_{a\,b\,c\,d} + \frac{1}{3}\,R_{a\,c\,d\,b} + \frac{1}{3}\,R_{a\,d\,b\,c})$$

```
> SR := normalform(side_rel);
```

$$SR := \frac{1}{3}\,R^{R1R1\ R2R2\ R3R3\ R4R4}\,R_{R1R1\ R2R2\ R3R3\ R4R4} - \frac{2}{3}\,R^{R1R1\ R2R3\ R3R2\ R4R4}\,R_{R1R1\ R3R2\ R2R3\ R4R4}$$

```
> simplify(EXPR,{SR=0});
```

$$0$$

From this example we notice that the scalars $R^{a\,b\,c\,d}\,R_{a\,c\,b\,d}$ and $R^{a\,b\,c\,d}\,R_{a\,b\,c\,d}$ are not independent. More complex examples are available in the Riegeom web site (see footnote 11).

5 Conclusion

An algorithm to simplify tensor expressions based on computable definitions have been described. It has two parts. In the first part, the expression is put into a canonical form, taking into account symmetries with respect to index permutations and the renaming of dummy indices. The definition of the canonical form involves some conventions that can be changed without disqualifying the definition. The conventions are based on the implementation simplicity and on a readable display for tensor expression. In the second part, cyclic identities or more general kinds of tensor identities are addressed through the Gröbner basis method. The expression and the side relations are both put into a canonical form for the Gröbner method to work successfully. In this work, a precise definition of the canonical form for tensor expressions is provided.

No restriction is imposed on the kind of symmetries that the tensors can obey. For most of the symmetries that occur in practise, the algorithm is very fast. The symmetries of the Riemann

tensor reveal the algorithm's worst performance, but even in in this case it is useful for practical applications.

The invariant renaming of the dummy indices plays an important role in the efficiency of the algorithm, since it neutralizes the symmetries that come from dummy index renaming. This is a solution for the dummy indices problem mentioned in the introduction.

An experimental implementation of the algorithm over the Riemann package[4] is available free from web sites.[14] All calculations and timings presented in this work can be reproduced in the Maple system.

Acknowledgements:

The author thanks Ray McLenaghan and Keith Geddes for pointing out references and for the kind hospitality at the University of Waterloo. This work was made possible by a post-doctoral fellowship from CAPES/Ministério da Educação e do Desporto/Brazil.

References

[1] S. A. Fulling, R. C. King, B. G. Wybourne and C. J. Cummins, *Normal forms for tensor polynomials: I. The Riemann tensor*, Class. Quantum Grav. 9 (1992) 1151-1197.

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

[3] K. O. Geddes, S. R. Czapor and G. Labahn, *Algorithms for Computer Algebra*, Kluwer Academic Publisher, 1992.

[4] Portugal, R. and Sautú , S., *Applications of Maple to General Relativity*, Computer Physics Communications 105, 233-253 (1997)

[5] Waterloo Maple, Inc., 450 Phillip Street, Waterloo, Ontario, N2L 5J2, Canada. See the web pages: http://www.maplesoft.com/

[6] D. Lovelock and H. Rund, *Tensors, Differential Forms and Variational Principles*, John Wiley & Sons, 1975.

[7] L. P. Eisenhart, *Riemannian Geometry*, Princeton University Press, 1966.

[8] T. Y. Thomas, *Tensor Analysis and Differential Geometry*, Academic Press, 1965.

[9] D. E. Littlewood, *The Theory of Group Characters and Matrix Representations of Groups*, Oxford University Press, 1950.

[10] B. L. van der Waerden, *Algebra*, Frederick Ungar Publishing Co., 1970.

[14] See footnote 11.

[11] J. M. Lee, D. Lear and J. Roth, *Ricci - A Mathematica package for doing tensor calculations in differential geometry* (User's Manual version 1.2) 1995. See the web pages http://www.math.washington.edu/~lee/Ricci.

[12] L. Parker and S. M. Christensen, *MathTensor, A System for Doing Tensor Analysis by Computer*, Addison-Wesley, 1994.

[13] R. Bogen et al., *Macsyma Reference Manual*, vol II, chapter V2-3, Laboratory for Computer Science, MIT, 1983.

[14] M. Kavian, R. G. McLenaghan and K. O. Geddes, *MapleTensor: Progress Report on a New System for Performing Indicial and Component Tensor Calculation using Symbolic Computation*, ISSAC'96, Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation (ETH 1996), Y. N. Lakshman, Ed., pp.204.