ATENSOR – REDUCE program for tensor simplification

V.A.Ilyin* and A.P.Kryukov†
Institute of Nuclear Physics, Moscow State University
119899, Moscow, Russia

April 10, 1996

PROGRAM SUMMARY

Title of program: ATENSOR

Catalogue number:

Program available from: CPC Program Library, Queen's University of Belfast, N. Ireland (see application form in this issue).

License provisions: none

Computer: Any computers where REDUCE [7] can be installed

Operation system: see above.

Programming language: REDUCE-3.4, 3.5

No. of bits in a word: 32

No. of lines in distributed program, including test data, etc.: 1963

Keywords: Tensor, symmetry, multiterm linear identities, dummy indices, simplification, permutation group, REDUCE, computer algebra.

Nature of physical problem: Simplification of tensor expressions with taking into account multiterm linear identities, symmetry relations and renaming dummy indices. This problem is important for the calculations in the gravity theory, differential geometry, other fields where indexed objects arise.

Method of solution: The group algebra technique for permutation group is applied to construct a canonical subspace and the effective algorithm for the corresponding projection.

Restriction on the complexity of the problem: Computer operative memory is the severest restriction.

Running time: It depends on the problem. For example the expression contained contraction of Riemann tensor with antisymmetric tensor of second order $\epsilon_{\mu\nu} * R_{\mu\nu\lambda\kappa} - \epsilon_{\alpha\beta} * R_{\lambda\kappa\beta\alpha}$ require about 82s CPU time on HP9000/735 with 8M for REDUCE.

References:

[1] V.A.Ilyin and A.P.Kryukov, in Proc. of the Int. Symp. of Symbolic. and Algebraic Computation (ISSAC'91), July 15-17, 1991, Bonn, Ed. by S.Watt, ACM Press (1991) 224.

[2] V.A.Ilyin and A.P.Kryukov, in "New Computing techniques in Physics Research II", Proc. of AIHENP-92, Ed. by D.Perret-Gallix, World Scientific, Singapore (1992) 639-348

*E-mail: ilyin@theory.npi.msu.su
†E-mail: kryukov@theory.npi.msu.su

LONG WRITE-UP

1 Introduction

Objects with indices are often used in mathematics and physics. Tensors are classical examples here [1, 2]. Indexed objects can have very complicated and intricated properties. For example the Riemann tensor has symmetry properties with respect to permutation of indices. Moreover it satisfies the cyclic identity [1]. There are a number of linear identities with many terms in the case of Riemann-Cartan geometry with torsion [2].

So there is a problem of reduction of expressions which contain indexed objects, called "tensor expressions" below.

Two reduction strategies can be used.

First, the corresponding expressions are expanded in terms of basic elements to resolve symmetries and identities in the explicit form. However, this way requires introducing a lot of objects of different types and rules for their management. In the Riemann tensor case, these are the Christofel symbol, the metric tensor, and their derivatives. As a rule, this leads to large intermediate expressions. Furthermore, such an approach fails, for instance, in the case of geometry with torsion.

Second strategy is to consider indexed objects as formal objects with some properties. Note that if we consider tensors which have only symmetries corresponding to permutation indices and renaming dummy ones then the problem can be solved in terms of double cosets of permutation group [3]. However, if linear identities with many terms (> 2) are present, this approach fails because the summation operator leaves the group framework.

Our approach to the problem of simplification of tensor expressions is based on the consideration of tensor expressions as vectors in some linear space. The preliminary version of this idea was proposed in [4]. The advanced version of the algorithm was presented at ISSAC'91 [8] and AIHENP-92 [9]. Here we present a program in which we implement the proposed algorithm in REDUCE and give a detailed description of the program. We generalized the algorithm to the case of tensor multiplication.

Complementary approach was developed in [11]. Here Young diagram technique was used to solve the simplification problem in some specific case. Namely, when dummy indices are allowed only between basic tensors for which generic symmetry relations and multiterm linear identities should be imposed. Although this algorithm is a powerful tool in 'indicial tensor calculations' of asymptotic expansions of heat kernels of differential operators it fails in general case.

From the user's point of view, there are three groups of tensor properties:

S - symmetry with respect to index permutations;

I - linear identities.

D - invariance with respect to renamings of dummy indices;

As an illustration, for the Riemann curvature tensor these properties are:

$$\begin{array}{ll} \mathbf{S:} & R_{abcd} = R_{cdab}, \qquad R_{abcd} = -R_{bacd}; \\ \mathbf{I:} & R_{abcd} + R_{acdb} + R_{adbc} = 0; \\ \mathbf{D:} & R_{abcd}R_{ckmn}R_{dlps} = R_{abdc}R_{dkmn}R_{clps}. \end{array}$$

Note that multiterm linear identities will produce many rewriting rules which can complicate the problem essentially.

The problem under investigation can be formulated as a question: *whether two tensor expressions are equal or not, taking into account S-I-D properties?* Then the problem of *simplest (shortest) canonical form for given expression arises as a central one.*

2 Basic tensors and tensor expressions

Let us give some definitions.

Under *basic tensors* we will understand the object with finite number of indices which can have such properties as *symmetry* and *multiterm linear identities*[1].

Then, under *tensor expression* we will understand any expression which can be obtained from basic tensors with the help of the following operations:

- summation with integer coefficients;
- multiplication (commutative) of basic tensors[2].

We assume that all terms in the tensor expression have the same number of indices. Some pairs of them are marked as dummy ones. The set of nondummy names have to be the same for each term in the tensor expression. The names of dummies can be arbitrary.

3 Algebraic approach

To start with, let us consider the case of one basic tensor and tensor expressions which are linear combinations of this basic tensor with integer coefficients (i.e. without multiplication of tensors).

Let a tensor F has indices $\mu_1, \ldots, \mu_n$. There are $n!$ formally different objects

$$F_{\pi(\mu_1,\ldots,\mu_n)}, \quad \pi = \left\{ \begin{array}{ccc} 1 & \cdots & n \\ \pi(1) & \cdots & \pi(n) \end{array} \right\} \in S_n,$$

where S_n is the group of permutations of the set $(1, \ldots, n)$ and

$$\pi(\mu_1, \ldots, \mu_n) \equiv (\pi(\mu_1), \ldots, \pi(\mu_n))$$

.

If F has symmetries with respect to index permutations it means that there is a subgroup $H \in S_n$ such that

$$F_{h\circ\pi} \; - \; d(h)\cdot F_\pi \; = \; 0, \qquad \forall h \in S_n;$$

$$d(h) \in R^1; \; h\circ\pi \equiv (\pi(h(\mu_1)), \ldots, \pi(h(\mu_n))).$$

Multiterm linear identities can be written in the following form

$$\sum_{\pi\in S_n} \alpha_\pi \cdot F_\pi = 0, \quad \alpha_\pi \in \mathbf{R}^1.$$

If some pairs of dummy indices are present without any loss of generality we may suppose that their names are already normalized in some way, i.e. they have fixed names. Therefore one can only exchange these names: 1) change names inside each pair and 2) change pairs of these names. With this restriction the exchanging of dummy indices means that a subgroup $M \in S_n$ exists such that[3]

$$F_{\pi\circ m(\mu_1,\cdots,\mu_n)} = F_\pi, \quad \forall\, m \in M, \pi \in S_n.$$

Let us consider now *the group algebra* of S_n [5]. This is a linear space $\mathbf{R}^{n!}$ in which unit vectors correspond to permutations:

$$\mathbf{R}^{n!} \ni e_\pi \leftrightarrow \pi \in S_n.$$

The vectors e_π are orthogonal to each other in the Euclidean metric.

[1]The symmetry relations are partial case of linear identities indeed

[2]Including contraction of indices.

[3]The difference in actions of H and M on S_n in the discussed relations ($h \circ \pi$ and $\pi \circ m$) is connected with the different nature of these transformations: the symmetry acts upon place of indices while dummy indices renaming acts upon the names.

So we have an explicit isomorphism between tensor expressions and points in $\mathbf{R}^{n!}$:

$$\sum_{\pi} \alpha_\pi \cdot e_\pi \leftrightarrow \sum_{\pi} \alpha_\pi \cdot F_\pi,$$

3.1 Subspace K

In terms of $\mathbf{R}^{n!}$, the left hand sides of the **S-I-D** relations correspond to the vectors

$$e_{h\circ\pi} - d(h) e_\pi, \quad e_{\pi\circ m} - e_\pi, \quad \sum_{k=1}^{n!} \alpha_k e_k, \quad (1)$$

$$h \in H,\ m \in M,\ \pi \in S_n.$$

These vectors span some subspace $\mathbf{K} \in \mathbf{R}^{n!}$. We denote its dimension as K.

Now we can split group algebra of S_n into orthogonal components in terms of the Euclidean metric

$$\mathbf{R}^{n!} = \mathbf{K} \oplus \mathbf{Q}.$$

It is obvious that all points of $\mathbf{R}^{n!}$ lying in $\mathbf{K}$ correspond to tensor expressions which are equal to zero due to the **S-I-D** properties. Thus, any two points A and A' lying on the plane parallel to $\mathbf{K}$ correspond to equal expressions:

$$(A - A') \leftrightarrow \sum_{\pi \in S_n} \alpha_\pi^{SID} \cdot F_\pi = 0.$$

The $(n! - K)$ dimensional subspace $\mathbf{Q}$ could be chosen as a set of canonical elements: a point A_Q can be determined as a canonical representative of the **S-I-D** equivalence class of A. Then one can conclude that the problem of comparing tensor expressions may be solved by comparing their canonical representatives. Such an approach was developed in [4] where the Gramm-Schmidt orthogonalization procedure was used as a main technical method. However, this procedure requires too much time and computer memory during executions.

In [8] the authors proposed an effective procedure where another subspace, denoted below as $\mathbf{L}$, had been considered as a set of canonical elements. In the next section we give a concise formulation of this procedure.

3.2 "Triangle" S-I-D basis

Let's designate vectors (1) as

$$V_k^0 = \sum_{j=1}^{n!} \alpha_{kj}^0 e_j,\ k = (1, ..., \tilde{K}). \quad (2)$$

Then we construct new vectors by recurrent applying (with steps $m = 1, \ldots, \tilde{K}$) of the following transformation:

— *if* $V_m^{m-1} \neq 0$ *then define* k_m *by first nonzero coefficient in* V_m^{m-1} *and let*

$$V_k^m \equiv V_k^{m-1}, \qquad k = 1, \ldots, k_m;$$

$$V_k^m \equiv V_k^{m-1} - \sum_{i=1}^{m} \frac{\alpha_{kj_i}^{m-1}}{\alpha_{k_i j_i}^{m-1}} V_{k_i}^{m-1},\ k > k_m; \quad (3)$$

— *if* $V_m^{m-1} = 0$ *then* $V_k^m \equiv V_k^{m-1}$.

As a result, we have a set of nonzero vectors $V_k^{(K)}$ which span the subspace $\mathbf{K}$. Note that all nonzero vectors $V_k^{(K)}$ are linearly independent. Indeed, let us reorder the unit vectors

$$(e_1, \ldots, e_{n!}) \longrightarrow (e_{j_1}, \ldots, e_{j_m}, \ldots).$$

Then the vectors $V_k^{(K)}$ with $k \geq k_m$ will have zero projections onto the unit vectors e_{j_i} $(i < m)$. In other words, the set $V_k^{(K)}$ has a "triangle" form in this reordered basis. It is evident that a number of nonzero vectors $V_k^{(K)}$ is equal to K - the dimension of the **S-I-D** subspace $\mathbf{K}$.

We have made some optimization in addition to the transformation (3):

on each step m all already constructed nonzero vectors $V^{(K)}_{k_i}$ $(i < m)$ are improved by the following transformation:

$$V^m_{k_i} \to V^m_{k_i} - \frac{\alpha^m_{k_i j_m}}{\alpha^m_{k_m j_m}} V^m_{k_m}, \quad i < m. \qquad (4)$$

Such optimization doesn't spoil the "triangle" structure and vectors V^k_i with $k < k_m$ will have zero projections onto the unit vectors e_j with $j = j_1, ..., j_{m-1}$. Thus, we conclude that each vector $V^{(K)}_k$ has not more then $(n! - K)$ nonzero components.

3.3 Subspace of canonical elements

Let us consider an arbitrary vector, designated below as

$$A^0 \equiv A = \sum_{j=1}^{n!} a^0_i \cdot e_j,$$

and recurrently apply $(m = 1, \ldots, K)$ the following transformation:

$$A^m \equiv A^{m-1} - \frac{a^{m-1}_{j_m}}{\alpha^{m-1}_{k_m j_m}} V^{(K)}_{k_m} = \sum_{j=1}^{n!} a^m_j e_j. \quad (5)$$

It is easy to see that this linear transformation has the following properties:

- it shifts the vector A along the plane parallel to $\mathbf{K}$. (*So A and A^{K+1} correspond to equal tensor expressions*);
- vector A^{K+1} has zero projections onto unit vectors e_j with $j = j_1, \ldots, j_K$;
- $A^{K+1} = 0 \quad \text{iff} \quad A \in \mathbf{K}$.

Thus, we conclude that $\forall A \in \mathbf{R}^{n!}$:

- $A^{K+1} = A'^{K+1}$ iff $(A - A') \in \mathbf{K}$;
- transformation (5) doesn't shift A^{K+1};
- image of the transformation (5) is some linear subspace $\mathbf{L}$ with dimension $(n! - K)$ and $\mathbf{L} \cap \mathbf{K} = \{0\}$[4];
- every vector in $\mathbf{L}$ has not more than $(n! - K)$ nonzero components.

We define the subspace L as a set of canonical representatives for S-I-D equivalence classes under consideration.

4 Multiplication

If a tensor expression is obtained by the multiplication of basic tensors then we directly generate the set of **S-I-D** relations as a *product* of basic ones.

Then the expression is elaborated, taking into account additional relations originated from the multiplication rule (in our case from the commutativity). Let us consider a tensor tt with two indices as an example. If we multiply it by itself, say as $tt(i,j) * tt(k,l)$, the additional symmetry appears:

$$tt(i,j) * tt(k,l) = tt(k,l) * tt(i,j).$$

We consider all such permutations and add the corresponding elements to the **S-I-D** relations. Then we perform the "triangle" procedure (3,4) to construct the full **K** subspace, and use the procedure (5) for the canonical representative calculation[5].

[4] Here $\{0\}$ means that this set has a single point - zero vector.

[5] It is clear how to generalize the algorithm to the case of a noncommutative finite algebra of basic tensors. In such cases, more complicated relations will appear instead of the simple symmetry written above. However, in any case they will be linear identities which can be elaborated naturally in the framework of our approach.

5 Some definitions

Let us give some definitions which we use in the following.

1. $\mathbf{K}$-*basis* is a general name for "triangle" set of linear independent vectors $V_k^{(\mathbf{K})}$ $(k = k_1, \ldots, k_K)$.
2. $\mathbf{K_0}$-*basis* is the $\mathbf{K}$-basis for a basic tensor considered as a separate tensor expression, taking into account its symmetries and linear identities only.
3. $\mathbf{K_M}$-*basis* is the $\mathbf{K}$-*basis* of the expression under consideration which arises from $\mathbf{K_0}$-*bases* of basic tensors and relations generated by their multiplication.
4. $\mathbf{K_D}$-*basis* is the completion of the $\mathbf{K_M}$-*basis* up to the full $\mathbf{K}$-*basis* of the expression under consideration, taking into account relations which arise from renamings of dummy indices. We shall also call this basis the *full* $\mathbf{K}$-*basis*.
5. To *sieve* some $\mathbf{S}-\mathbf{I}-\mathbf{D}$ vector means to do the step m of the "triangle" procedure (3).
6. To *rearrange* some $\mathbf{S}-\mathbf{I}-\mathbf{D}$ vector means to do the step m of the "triangle" procedure (4).
7. To *sieve* some vector A by a $\mathbf{K}$-*basis* means to apply the transformation (5).

6 Algorithm

There are two sets of operations. One is performed due to the introduction of new basic tensors, and the result of these operations is the construction of the new $\mathbf{K_0}$-bases. Another set of operations is connected with the simplification of tensor expressions itself.

The starting procedure for the construction of $\mathbf{K_0}$-bases is $TSYM$ (section 8.4), and the algorithm is:

1. Generate full list of **S-I** vectors (1): $L_{\mathbf{S-I}} = \{v\}$.
2. Let $\mathbf{K} = \mathbf{K_0}$ where $\mathbf{K_0}$ is initial basis (may be empty).
3. If the list $L_{\mathbf{S-I}}$ is empty then finish.
4. Take the next vector v from $L_{\mathbf{S-I}}$ and delete it from this list: $L_{\mathbf{S-I}} \rightarrow L_{\mathbf{S-I}}/\{v\}$.
5. Sieve the vector v through the $\mathbf{K}$-basis (transformation (3), procedure *sieve_pv*, section 7.7). The result is a new vector v'.
6. If $v' \neq 0$ then insert v' in the $\mathbf{K}$-basis and rearrange the basis (transformation (4), procedure *insert_pv*, section 7.7).
7. Repeat from step 3.

All $\mathbf{K_0}$-bases are stored. There is a possibility to delete any basic tensor from the list of tensors and so to delete its $\mathbf{K_0}$-basis (section 8.3).

The algorithm of simplification of tensor expressions is:

1. If there is no multiplication of tensors or there are no new **S-I** relations due to their multiplication then go to step '*4*'.
2. Expand the $\mathbf{K_0}$-bases involved up to the necessary rank of the permutation

group corresponding to the multiplication of basic tensors in the expression elaborated (procedure *t_upright*, section 7.5), and collect the obtained relations as the initial **K**-basis of the considered expression.

3. Complete the **K**-basis by additional vectors which arise from the multiplication (see section 4) by the algorithm of the **K**-basis construction (see above). The result is the $\mathbf{K_M}$-basis for the expression under simplification.

4. *Dummies.* Complete the $\mathbf{K_M}$-basis by vectors corresponding to the relations which arise from renamings of dummy indices (if they are present). The result is the $\mathbf{K_D}$-basis - the full **K**-basis for the expression under simplification.

5. Sieve the tensor expression through the $\mathbf{K_D}$-basis (transformation (5), procedure *sieve_t*, section 7.7). The result is the construction of the canonical representative for the expression.

Any $\mathbf{K_M}$ and $\mathbf{K_D}$-bases are not stored and are constructed each time an expression is simplified.

7 Program description

ATENSOR program consists of the following blocks:

- interface with REDUCE system;
- generator of permutations;
- p-vector arithmetics;
- tensor arithmetics;
- generator of the *multiplication* relations;
- generator of the *dummy* relations;
- utilities to work with **K**-bases.

7.1 Interface with REDUCE

Interface with REDUCE is implemented by defining of a domain [10]. This is a natural way to implement a new object in REDUCE. We should define the following set of procedures:

Operation	Internal proc.	Comment
minus	t_minus	Unary minus
plus	t_plus	Summ
times	t_times	Product
difference	t_difference	Substruction
zerop	t_zerop	Does tensor equal zero?
prepfn	t_prep	
prifn	t_pri	Print function
intequiv	tintequiv	Is tensor equivalent to integer

The following procedures must be define for completeness but can not be used as operations for tensors. These procedures produce an error message if called.

Operation	Internal proc.	Comment
expt	t_expt	Power
quotient	t_quotient	
divide	t_divide	
gcd	t_gcd	Great common divider
minusp	t_minusp	Is tensor negative?
onep	t_onep	Does tensor equal 1?
i2d	i2tensor	Transform integer to tensor

We also define the domain name $(TENSOR)$ and the tag $(!:TENSOR)$.

To complete the interface with REDUCE, we add the tag of the tensor domain to the global variable $DOMAINLIST!*$

This method supports the input process of tensor expressions and provides calls of the corresponding internal procedures automatically.

7.2 Permutations

This block implements generation of permutations of N order and includes some procedures for working with them. All procedures work with the *packed* and the *unpacked* form of permutations.

Unpacked form of a permutation p is a list of numbers[6]:

$$p = (d_1\ d_2\ ...\ d_k), \quad 1 \leq d_i \leq k \leq 99.$$

Packed form is the corresponding number:

$$p = d_1 d_2 ... d_k.$$

For example,

$$p = (1\ 2\ 3\ 4\ 5) \qquad \text{and} \qquad p = 12345.$$

The transformation of a permutation from one form to another is made automatically. The packed form is more economic with respect to computer memory, but requires more time to proceed. There is the global variable *!*ppacked*. If it is T (the default value) then all permutations are packed; if NIL then they are stored in the unpacked form.

We use the well known algorithm for generation of permutations [12]. We implement the following procedures:

- $mkunitp(n)$ – generates the unit element of S_n;
- $pfind(p_1, p_2)$ – returns x such that $p_2 = x \circ p_1$;
- $prev(p)$ – returns reverse permutation x such that $x \circ p = 1$;
- $psign(p)$ – returns $(-1)^k$, where k is the number of transpositions which are necessary to apply to the permutation p to get the identical permutation.
- $pmult(p_1, p_2)$ – returns the permutation $x = p_1 \circ p_2$;
- $pappl(p, l)$ – returns l with the elements permuted by p, so that l is replaced by $p(l)$.

There are some utilities to work with permutations:

- $pupright(p, d)$ – extends the permutation $p \in S_n$ to the right up to the element of S_{n+d} with the identical permutation of the extra indices (this utility is used for the elaboration of multiplication of basic tensors);
- $pupleft(p, d)$ – extends the permutation $p \in S_n$ to the left up to the element of S_{n+d} with the identical permutation of the extra indices (this utility is used for the elaboration of multiplication of basic tensors);
- $pappend(p_1, p_2)$ - concatenates the permutation $p_1 \in S_{n_1}$ with the permutation $p_2 \in S_{n_2}$. Returns the element of $S_{n_1+n_2}$;

[6] see section 9 for details.

- $pkp(p)$ – packs the permutation p;
- $unpkp(p)$ – unpacks the permutation p.

7.3 P-vectors

P-vectors are one of the main objects in the program. They represent the vector in $\mathbf{R^n}$ – the group algebra of S_n. In the program they are implemented as a REDUCE domain [10].

Internal structure of p-vector is:

$p\!-\!vector$ ::= $(!{:}pv\ .\ p\!-\!list)$
$p\!-\!list$ ::= $NIL|(coeff\ .\ perm)\ .\ p\!-\!list$
$coeff$ ::= $integer$

All the standard operations are defined for p-vectors because they form a domain.

The following procedures are used for tensor simplification:

- $pv\_sort(pv)$ – sorts the p-list so that all permutation will be ordered, e.g. $p_i > p_j\ \forall\ i < j$;
- $pv\_compress(pv)$ – removes all terms with zero coefficient;
- $pv\_renorm(pv)$ – reduces the first coefficient (in integer numbers), i.e. $pv \rightarrow pv/GCD(c_1, c_2, ...)$ where c_i are the coefficients.

Some utilities are available to work with p-vectors:

- $pappl\_pv(p, pv)$ – applies the permutation p to the p-vector pv.
  Returns $p' = \sum_i c_i\ pmult(p, p_i)$ where $pv = \sum_i c_i\ p_i$;
- $pv\_applp(pv, v)$ – applies the p-vector pv to the permutation p.
  Returns $p' = \sum_i c_i\ pmult(p_i, p)$ where $pv = \sum_i c_i\ p_i$;
- $pv\_upright(pv, d)$ - expands the p-vector pv to the right.
  Returns $p' = \sum_i c_i\ pupright(p_i, d)$ where $pv = \sum_i c_i\ p_i$;
- $pv\_upleft(pv, d)$ - extends the p-vector pv to the left.
  Returns $p' = \sum_i c_i\ pupleft(p_i, d)$ where $pv = \sum_i c_i\ p_i$.

7.4 Tensors

Tensors are the main objects in the program. They represent tensor expressions. In the program they are implemented as a REDUCE domain [10].

The internal structure of a tensor is:

$tensor$ ::= $(!{:}tensor\ i\!-\!tensor_1\ ...\ i\!-\!tensor_k)$
$i\!-\!tensor$ ::= $(t\!-\!header\ \ t\!-\!list)$
$t\!-\!list$ ::= $NIL\ |\ p\!-\!list\ .\ t\!-\!list$
$t\!-\!header$ ::= $(t\!-\!name\ \ i\!-\!list)$
$t\!-\!name$ ::= $(t_1\ \ t_2\ \ ...\ \ t_k)$
$i\!-\!list$ ::= $(i_1\ \ i_2\ \ ...\ \ i_l)$

where $t_1, t_2, ..., t_k$ are basic tensors identifiers, and $i_1, i_2, ..., i_l$ are indices (identifiers).

Let us consider an example tensor expression and its representation in the internal notations.

Let $tt(i, j)$ be a tensor of second order. The internal representation is

$$(!{:}tensor\ (((tt)\ (i\ \ j))\ (1\ .\ 12)))$$

Thus, the tensor expression

$$tt(i, j) + tt(j, i)$$

will have the internal representation

$$(!{:}tensor\ (((tt)\ (i\ \ j))\ (1\ .\ 21)\ (1\ .\ 12)))$$

The most important procedures from this block are described bellow in section 8.

The simplification of tensor expressions is performed by the function t_simp. The result of this procedure is the canonical form of the tensor expression, i.e. the sieved vector $t' \in \mathbf{L}$.

7.5 Tensor multiplications

The main procedures of the tensor multiplication block are the following:

- $t\_split(tt)$ - splits a term of the tensor expression into the list of basic tensors as factors;
- $t\_fuse(tf_1, tf_2)$ - combines tensor factors tf_1 and tf_2 into the product. This operation is reverse to the previous one;
- $addmultsym(t_1, t_2)$ - adds symmetry and multiterm linear identity relations generated by the multiplication to the **K**-basis.

Some utilities are available to work with tensors.

- $t\_upright(tt, th)$ - extends the tensor tt to the right with respect to the t-header th;
- $t\_upleft(tt, th)$ - extends the tensor tt to the left with respect to the t-header th;
- $t\_pri(tt)$ - outputs the tensor tt in the natural form.

7.6 Dummy indices

Dummy relations are created in the process of evaluation of a tensor expressions. Their number may be very large and unpredictable in advance. Therefore we do not save **D**-relations in contrast to **S** and **I** ones. This leads to the loss of time but saves the memory.

During simplification of tensor expressions we use internal names for indices. Original names are saved and used in the I/O process. Thus, if we have, for example, a dummy index i (really there are two such names in the expression considered) then it will be replaced with two internal names:

$$\_nn \quad \text{and} \quad \_mm,$$

where $mm = nn + 1$. The original name is stored as a special property of the new ones.

The dummy block produces relations generated by renamings of dummies. The main procedures are the following:

- $adddummy(tt)$ - adds the new relations to the **K**-basis;
- $dl\_get(il)$ - returns the list of dummy indices from the index list il;
- $il\_simp(il)$ - replaces original names of the dummy indices with their internal names;
- $mk\_dsym(t_1)$ - returns the list of tensor relations with changed dummy indices in each pair;
- $mk\_ddsym(t_1)$ - returns the list of tensor relations with permuted pairs of dummy indices.

7.7 Working with K-bases

This block contains the procedures for working with **K**-bases.

All **K**-bases for various tensor expressions are stored as lists in the global variable

$$!{*}basis ::= (k-basis_1 \; k-basis_2 \; ...)$$

The structure of the basis is:

$$k-basis ::= (t-header) \,.\, t-list$$

where the header t and the list t are defined above.

The main procedures are the following:

- $sieve\_pv(pv, b)$ – sieves the p-vector pv using the basis b. This procedure is used for the construction of the **K**-basis ("triangle" transformation (3,4)) and for the simplification of tensors expressions (projection to the canonical element (5)). This is the main step of the function $sieve\_t$.;
- $reduce\_pv(pv, qv)$ - reduces the p-vector pv with respect to the p-vector qv. This is the main step of the function $sieve\_pv$.;

- $insert\_pv(pv, b)$ - inserts the p-vector pv into the basis b. This procedure also rearranges b with respect to pv.
- $sieve\_t(tt)$ - sieves the tensor tt using the corresponding $\mathbf{K}$-basis. The first step of this procedure is generation of relations due to renamings of dummies and the corresponding completion of $\mathbf{K_M}$ up to $\mathbf{K_D}$-basis.

7.8 Global variables

In this section we describes the main global variables which allow a user to control the work. We show the default values in brackets.

- $!*ppacked(T)$ - are permutations stored in packed form?
- $!*debug(NIL)$ - switches the debug output on.

8 User's interface

To simplify the user interface, we restricted the number of additional commands. The names of these commands are very similar to the standard REDUCE ones used in similar cases.

8.1 *KBASIS*

The command $KBASIS$ prints the tensor $\mathbf{K}$-basis.

The number of vectors in the basis, i.e. the dimension of the corresponding subspace $\mathbf{K}$, is typed in the last line of the output. Format of this command is:

$$KBASIS\ \ tt_1, tt_2, ..., tt_n;$$

Here $tt_1, tt_2, ..., tt_n$ are tensor names.

To output the $\mathbf{K}$-basis in the case of the multiplication of two or more tensors, it is necessary to use the following format of the command:

$$KBASIS\ \ t_1(t_2, ..., t_k), \ldots;$$

Here $t_1, t_2, \ldots$ are the names of tensor factors.

If some names have not been declared as tensors the message is produced

****** basis1 *** Invalid as tensor: tt*

8.2 *TENSOR*

The command $TENSOR$ declares new tensors. Format of this command is:

$$TENSOR\ \ t_1, t_2, ..., t_n;$$

Here $t_1, t_2, ..., t_n$ are identifiers. The number of indices will be fixed during the first evaluation of a tensor expressions.

If some names have been declared as tensors already the message is produced:

+++ tt is already declared as tensor.

8.3 *TCLEAR*

The command $TCLEAR$ removes tensors from the list of tensors. Format of this command is:

$$TCLEAR\ \ t_1, t_2, ..., t_n;$$

Here $t_1, t_2, ..., t_n$ are the names of tensor (identifiers).

If some names have not been declared as tensors the message is produced

+++ xxx is not a tensor.

Note: All $\mathbf{K}$-bases where any of t_i is included as a factor will be lost.

8.4 *TSYM*

The command $TSYM$ defines symmetry relations of basic tensors. Format of this command is:

$$TSYM\ \ te_1, te_2, ..., te_k;$$

Here $te_1, te_2, ..., te_k$ are linear combinations of basic tensors with integer coefficients not containing without dummy indices.

All relations correspond to the left hand side of the symmetry equations. For example, for antisymmetric tensor we have the relation $AA(i,j) + AA(j,i) = 0$. Thus, the corresponding input format is:

$$TSYM \;\; AA(I,J) + AA(J,I);$$

8.5 Algebraic operations

The standard algebraic operations are available for tensors:

- $+$ sum of tensors;
- $-$ difference of tensors or negation;
- $*$ multiplication of tensors.

We assume that two indices with identical names means the summation over their values (the Einstein convention) - they are dummy ones. Thus, the multiplication of two tensors may be either a direct product, or it can contain contractions of dummy indices.

Examples:

$$t_1(i,j) * t_2(j,k);$$

$$t_1(i,j) + 2 * t_2(j,i);$$

8.6 Switch *DUMMYPRI*

This switch is controlled by the standard REDUCE commands ON/OFF. It controls the output process for tensor expressions. The default value is OFF.

$DUMMYPRI$ prints dummy indices with internal names – numbers. The general rule is:the index $(2k-1)$ is contracted with the index $(2k)$.

Examples[7]:

$TENSOR \;\; GG;$
$GG(M,M); \qquad \Rightarrow GG(M,M)$
$ON \;\; DUMMYPRI;$
$GG(M,M); \qquad \Rightarrow GG(M_{41}, M_{42})$

[7]Here and below REDUCE output is given after the arrow "$\Rightarrow$".

8.7 Switch *SHORTEST*

This switch is controlled by the standard REDUCE commands ON/OFF. It controls the output process for tensor expressions. The default value is OFF.

$SHORTEST$ prints tensor expression in shortest form that was produced during evaluation. Hoever, the sortest form may be noncanonical.

$TENSOR \;\; C;$
$TSYM \quad C(K,L,M) + C(L,M,K) + C(M,K,L);$
$C(K,L,M) + C(M,L,K);$
$\Rightarrow \;\; (-1) * C(L,M,K) + (-1) * C(M,K,L) + C(M,L,K)$
$ON \;\; SHORTEST;$
$C(K,L,M) + C(M,L,K);$
$\Rightarrow C(K,L,M) + C(M,L,K)$

9 Memory usage

Let us consider simplification of a tensor expressions with n indices. The rank of the corresponding permutation group is n and the dimension of its group algebra is $n!$. Let us consider two cases: when there are many $\mathbf{S}-\mathbf{I}-\mathbf{D}$ relations (so that the dimension of the $\mathbf{K}$ subspace is almost equal to $n!$) and when there are small number of $\mathbf{S}-\mathbf{I}-\mathbf{D}$ relations (so that the dimension of the $\mathbf{K}$ subspace is small).

In the first case, about $n! \cdot l \cdot k$ Lisp cells are necessary to store the full $\mathbf{K_D}$-basis. Here l is the number of cells needed to store a single term of a p-vector, and k is the average number of terms in vectors from this $\mathbf{K_D}$-basis. The number of terms in these vectors (k) is about $2-3$ in practical cases. The number of terms in the simplified expression is (in practical cases) $O(1)$, and can be omitted from this estimate.

In the case of a small set of symmetries and linear identities of basic tensors, the number of vectors in the full $\mathbf{K_D}$-basis is small enough, and can be omitted from this estimate. However, the number of terms in a canonical representative (the expression after simplification) will be about $n!$.

Anyway, we have to work with a practically full set of permutations, which contains $n!$ members.

Thus, the minimum computer memory necessary to store the elaborated expressions is not less then $n! \cdot l \cdot k$. The typical number of Lisp cells necessary to store a single term of a tensor expressions (basic tensor) is 4. The length of each cell is 8 Byte. The results of calculation for different ranks of the permutation group are collected in the table.

Rank of S_n	Number of Mcells	Memory in Mbyte
9	2.9	22.6
10	29.0	226.8
11	319.3	2494.8

2–3 times more memory is necessary in the intermediate steps of calculations. Modern computers usually equipped with up to 512 Mbyte memory can elaborate tensor expressions with not more then 10 indices with the help of the proposed algorithm. However, hardware development is very fast now, and it will be possible to solve problems with 11 indices with the help of our program. Finally, we note that it possible to modify the algorithm so that the memory limitations would be not so hard. However, this advantage is compensated by a significant increase of the execution time. To summarize this section, we conclude that the absolute limit for the group algebra approach developed in this work is 12 indices.

10 Examples

10.1 Symmetric and antisymmetric tensors

At the beginning, let us declare the names of basic tensors:

$tensor\ \ s2, a3, v1, v2, v3;$

By using $TSYM$ command we introduce the **S-I** relations of the basic tensors:

$$\begin{array}{lll} tsym & s2(i,j) - s2(j,i), & \text{\% Symmetric} \\ & a3(i,j,k) + a3(j,i,k), & \text{\% Antisymm.} \\ & a3(i,j,k) - a3(j,k,i); & \end{array}$$

Let us output the $\mathbf{K_0}$-bases constructed for the tensors a_2 and s_2

$kbasis\ \ s2, a3;$

$$\begin{array}{ll} \Rightarrow & s2(j,i) + (-1) * s2(i,j) \\ \Rightarrow & 1 \\ & \\ \Rightarrow & a3(k,i,j) + a3(j,i,k) \\ \Rightarrow & a3(k,j,i) + (-1) * a3(j,i,k) \\ \Rightarrow & a3(i,k,j) + (-1) * a3(j,i,k) \\ \Rightarrow & a3(i,j,k) + a3(j,i,k) \\ \Rightarrow & a3(j,k,i) + a3(j,i,k) \\ \Rightarrow & 5 \end{array}$$

Now we are ready to simplify tensor expressions. Some examples are:

$$\begin{array}{ll} s2(i,j) + s2(j,i); & \Rightarrow 2 * s2(i,j) \\ a3(i,j,k) * s2(i,j); & \Rightarrow 0 \\ a3(i,j,k) * v1(i) * v2(j) * v1(k); & \Rightarrow 0 \end{array}$$

Sometimes one can get a 'strange' output if one will not be careful with the input. For example,

$$\begin{array}{ll} x := s2(i,i); & \Rightarrow x := s2(i,i) \\ x * v1(i); & \Rightarrow s2(i,i) * v1(i) \end{array}$$

From the standard point of view, the second output is incorrect due to the fact that three indices with the same name are present. However, the input has not been recognized as an error. If one switches on the flag

$on\ \ dummypri;$

and then repeats the previous input then one gets the following output:

$$x * v1(i); \quad \Rightarrow s2(i_{23}, i_{24}) * v1(i)$$

Hence, the first two i's are dummies and the last one is a free index.

If a user would like to output the **K**-basis of the product of the tensors s_2 and a_3, the following format of the command $KBASIS$ has to be used:

$kbasis\ \ s2(a3);$

$$\Rightarrow \quad a3(j,i,k)*s2(i,j)+a3(k,i,j)*s2(j,i)$$
$$\Rightarrow \quad a3(j,i,k)*s2(j,i)+a3(k,i,j)*s2(j,i)$$
.....
$$\Rightarrow \quad 110$$

10.2 Riemann tensor

Let us introduce the Riemann tensor and the standard set of its **S-I** relations:
$tensor\ \ ri;$
$tsym\ \ ri(i,j,k,l)+ri(j,i,k,l);$
$tsym\ \ ri(i,j,k,l)+ri(i,j,l,k);$
$tsym\ \ ri(i,j,k,l)+ri(i,k,l,j)+ri(i,l,j,k);$

The $\mathbf{K_0}$-basis consists of 22 vectors (see TEST RUN OUTPUT) and the full vector space has $4! = 24$ dimensions. Thus, any expressions which are linear combinations of Riemann tensors with permuted indices can be simplified to expressions containing only two basic tensors[8].

This set of properties leads us to the very important symmetry property of Riemann tensor with respect to the exchange of pairs of indices:
$ri(i,j,k,l)-ri(k,l,i,j); \qquad \Rightarrow 0$

Let us consider some more examples,

$$ri(m,n,m,n)-ri(m,n,n,m)$$
$$\Rightarrow 2*ri(m,n,m,n).$$

Any tensors expressions consists of Rieman tensors may be expressed through summ of 2 ones:

$$ri(i,j,k,l) + ri(j,k,l,i) + ri(k,l,i,j) + ri(l,i,j,k);$$
$$\Rightarrow (-2)*ri(l,j,i,k)+4*ri(l,i,j,k)$$

A more complicated example with multiplication of the Riemann tensor and the antisymmetric tensor $a2$ is given in the section **TEST RUN OUTPUT**.

[8] This simplification has no relation to the number of independent components of the Riemann curvature tensor in space-time of various dimensions

Acknowledgements

The authors are grateful to A.Grozin for useful discussions.

This work was supported by Russian Foundation for Fundamental Research (grant 93-02-14428).

References

[1] L.D.Landau and E.M.Lifshitz, *Field Theory*, (Moscow, Nauka, 1973, in Russian); J.A.Schouten, *Tensor analysis for physicist*, (Oxford, 1951).

[2] A.Trautman, *Symp. Math.*, **12** (1973) 139; F.W.Hehl et al., *Rev. Mod. Phys.*, **48** (1976) 393; Yu.N.Obukhov and P.I.Pronin, *Acta Physica Polonica*, **B19** (1988) 341.

[3] A.Ya.Rodionov and A.Yu.Taranov, *Lecture Notes in Comp. Sci.* **378**, (Proceedins of EUROCAL'87, 1989) 192

[4] V.A.Ilyin et al., in: *Proc. of IV International conference on computer algebra in physical research (Dubna 1990, 22-26 May, USSR)* 190-194.

[5] M.A.Naimark, *Theory of group representation*, (Moscow, Nauka, 1976, in Russian).

[6] B.Buchburger, G.E.Collins and R.Loos, *Computer Algebra - symbolic and algebraic computation*, (second edition, Springer, 1983).

[7] A.C.Hearn, *REDUCE USER'S MANUAL, version 3.3*, (The Rand publication CP78, Rev. 7.87, 1987).

[8] V.A.Ilyin and A.P.Kryukov, in Proc. of the Int. Symp. of Symbolic. and Algebraic

Computation (ISSAC'91), July 15-17, 1991, Bonn, Ed. by S.Watt, ACM Press (1991) 224.

[9] V.A.Ilyin and A.P.Kryukov, in "New Computing techniques in Physics Research II", Proc. of AIHENP-92, Ed. by D.Perret-Gallix, World Scientific, Singapore, (1992) 639-648.

[10] R.J. Bradford, A.C. Hearn, J.A. Padget and E. Schrüfer, "Enlarging the *REDUCE* Domain of Computation," Proc. of SYMSAC '86, ACM, New York (1986) 100.

[11] S.A.Fulling, R.C.King, B.G.Wybourne and C.J.Cummins Class. Quantum Grav. **9** (1992) 1151-1197.

[12] D.E. Knuth, *Fundamental Algorithms, vol. 1 of the "Art of Computer Programming"* (Addison-Wesley, 1968).

TEST RUN OUTPUT

```
%*********************************************************************
%                          ATENSOR  TEST  RUN.
%
%                       V.A.Ilyin & A.P.Kryukov
%                 E-mail:   ilyin@theory.npi.msu.su
%                          kryukov@theory.npi.msu.su
%
%               Nucl. Phys. Inst., Moscow State Univ.
%                        119899 Moscow, RUSSIA
%*********************************************************************

% First of all we have to load the ATENSOR program using the one of the
% following command:
%       1) in "atensor.red"$         % If we load source code
%       2) load atensor$             % If we load binary (compiled) code.
load atensor;

(atensor)

% To control of total execution time clear timer:
showtime;

Time: 0 ms

% Switch on the switch TIME to control of executing time
% for each statement.
%on time$

% Let us introduce the antisymmetric tensor of the second order.
tensor a2;

% The antisymmetric property can be expressed as:
tsym a2(i,j)+a2(j,i);

% The K-basis that span K subspace is:
kbasis a2;

a2(i,j) + a2(j,i)
1

% Let us input very simple example:
a2(k,k);

0
```

```
% By the way the next two expressions looks like different ones:
a2(i,j);

a2(i,j)

a2(j,i);

a2(j,i)

% But the difference of them has a correct value:
a2(j,i)-a2(i,j);

2*a2(j,i)

% Next examples. For this purpose we introduce 3 abstract
% vectors - v1,v2,v3:
tensor v1,v2,v3;

% The following expression equal zero:
a2(i,j)*v1(i)*v1(j);

0

% It is interest that the result is consequence of the equivalence
% of the name of tensors.

% While the next one - not:
a2(i,j)*v1(i)*v2(j);

a2(i,j)*v1(i)*v2(j)

% Well. Let us introduce the symmetric tensor of the second order.
tensor s2;

tsym s2(i,j)-s2(j,i);


% Their K-basis look like for a2 excepted sign:
kbasis s2;

s2(j,i) + (-1)*s2(i,j)
1

% Of course the contraction symmetric  and antisymmetric tensors
% equal zero:
```

```
a2(i,j)*s2(i,j);

0

% By the way, the next example not so trivial for computer...
a2(i,j)*a2(j,k)*a2(k,i);

0

% Much more interesting examples we can demonstrate with the
% the tensor higher order. For example full antisymmetric tensor
% of the third order:
tensor a3;

% The antisymmetric property we can introduce through the
% permutation of the two first indices:
tsym a3(i,j,k)+a3(j,i,k);

% And the cyclic permutation all of them:
tsym a3(i,j,k)-a3(j,k,i);

% The K basis of a3 consist of 5 vectors:
kbasis a3;

a3(k,i,j) + a3(j,i,k)
a3(k,j,i) + (-1)*a3(j,i,k)
a3(i,k,j) + (-1)*a3(j,i,k)
a3(i,j,k) + a3(j,i,k)
a3(j,k,i) + a3(j,i,k)
5

% In the beginning some very simple examples:
a3(i,k,i);

0

a3(i,j,k)*s2(i,j);

0

% The full symmetric tensor of the third order may be introduce
% by the similar way:
tensor s3;

tsym s3(i,j,k)-s3(j,i,k);
```

```
tsym s3(i,j,k)-s3(j,k,i);

kbasis s3;

s3(k,j,i) + (-1)*s3(i,j,k)
s3(k,i,j) + (-1)*s3(i,j,k)
s3(j,k,i) + (-1)*s3(i,j,k)
s3(j,i,k) + (-1)*s3(i,j,k)
s3(i,k,j) + (-1)*s3(i,j,k)
5

% The next examples demonstrate some calculation with them:
s3(i,j,k)-s3(i,k,j);

0

s3(i,j,k)*a2(i,j);

0

a3(i,j,k)*s2(i,j);

0

s3(i,j,k)*a3(i,j,k);

0

% Now we consider very important physical case - Rieman tensor:
tensor ri;

% It has the antisymmetric property with respect to the permutation
% of the first two indices:
tsym ri(i,j,k,l) + ri(j,i,k,l);

% It has the antisymmetric property with respect to the permutation
% of the second two indices:
tsym ri(i,j,k,l) + ri(i,j,l,k);

% And the triple term identity with cyclic permutation the
% third of them:
tsym ri(i,j,k,l) + ri(i,k,l,j) + ri(i,l,j,k);

% The corresponding K basis consist of 22(!) vectors:
kbasis ri;
```

```
ri(l,k,i,j) + (-1)*ri(j,i,k,l)
ri(l,k,j,i) + ri(j,i,k,l)
ri(l,i,k,j) + (-1)*ri(j,k,i,l)
ri(l,i,j,k) + ri(j,k,i,l)
ri(l,j,k,i) + (-1)*ri(j,k,i,l) + ri(j,i,k,l)
ri(l,j,i,k) + ri(j,k,i,l) + (-1)*ri(j,i,k,l)
ri(k,l,i,j) + ri(j,i,k,l)
ri(k,l,j,i) + (-1)*ri(j,i,k,l)
ri(k,i,l,j) + (-1)*ri(j,k,i,l) + ri(j,i,k,l)
ri(k,i,j,l) + ri(j,k,i,l) + (-1)*ri(j,i,k,l)
ri(k,j,l,i) + (-1)*ri(j,k,i,l)
ri(k,j,i,l) + ri(j,k,i,l)
ri(i,l,k,j) + ri(j,k,i,l)
ri(i,l,j,k) + (-1)*ri(j,k,i,l)
ri(i,k,l,j) + ri(j,k,i,l) + (-1)*ri(j,i,k,l)
ri(i,k,j,l) + (-1)*ri(j,k,i,l) + ri(j,i,k,l)
ri(i,j,l,k) + (-1)*ri(j,i,k,l)
ri(i,j,k,l) + ri(j,i,k,l)
ri(j,l,k,i) + ri(j,k,i,l) + (-1)*ri(j,i,k,l)
ri(j,l,i,k) + (-1)*ri(j,k,i,l) + ri(j,i,k,l)
ri(j,k,l,i) + ri(j,k,i,l)
ri(j,i,l,k) + ri(j,i,k,l)
22

% So we get the answer for any expressions with 3 and more terms of
% Rieman tensors with not more then 2 terms. For example:
ri(i,j,k,l)+ri(j,k,l,i)+ri(k,l,i,j)+ri(l,i,j,k);

(-2)*ri(l,j,i,k) + 4*ri(l,i,j,k)

% This three identities leads us to very important symmetry property with
% respect to exchange of pairs indices:
ri(i,j,k,l)-ri(k,l,i,j);

0

% Let us start with simple example:
ri(m,n,m,n)-ri(m,n,n,m);

2*ri(m,n,m,n)

% Much more complicated example is:
a2(m,n)*ri(m,n,c,d) + a2(k,l)*ri(c,d,l,k);

0
```

```
% The answer is trivial but not so simple to obtain one.

% The dimension of the full space is 6! = 720.
% The K basis consists of 690 vectors (to reduce output we
% commented the last statement):
%kbasis ri(a2);

% One else nontrivial examples with Riemann tensors:
(ri(i,j,k,l)-ri(i,k,j,l))*a2(i,j);

 a2(i,j)*ri(i,j,k,l)
---------------------
          2

%***************** END OF TEST RUN ***********************
% The total execution time is:
showtime;

Time: 196940 ms  plus GC time: 10670 ms

$

END$
```