Hiding canonicalisation in tensor computer algebra

Dominic Price, Kasper Peeters and Marija Zamaklar

dominic.t.price@durham.ac.uk, kasper.peeters@durham.ac.uk, marija.zamaklar@durham.ac.uk

Department of Mathematical Sciences
Durham University
South Road
DH1 3LE Durham
United Kingdom

August 25, 2022

Abstract

Simplification of expressions in computer algebra systems often involves a step known as "canonicalisation", which reduces equivalent expressions to the same form. However, such forms may not be natural from the perspective of a pen-and-paper computation, or may be unwieldy, or both. This is, for example, the case for expressions involving tensor multi-term symmetries. We propose an alternative strategy to handle such tensor expressions, which hides canonical forms from the user entirely, and present an implementation of this idea in the Cadabra computer algebra system.

1 Introduction

A key part of any symbolic computer algebra system is the ability to detect equivalence of two mathematical expressions. The common way to achieve this is to define either a "canonical form", such that all expressions which are equivalent have the same canonical form, or the weaker concept of "normal form", such that all expressions equivalent to zero are represented by zero (see e.g. [1]). There is a long history of canonicalisation algorithms, which of course depend strongly on the types of expressions one deals with. For any given class of expressions, it is common to find multiple different canonical or normal forms in the literature.

For tensor polynomial expressions, which form the topic we deal with in this paper, there is a fairly extensive literature on canonicalisation using so-called "mono-term symmetries". These are index permutation symmetries of the expression which take a single term to a single other term. A naïve method of finding these symmetries is by considering all the valid permutations of indices, but most implementations of this are based on the more efficient Butler-Portugal algorithm [2, 3] which is based on the same concept but scales much better as the number of indices and terms grow, although still displaying $O(n!)$ behaviour in cases of total symmetry. Implementations of this algorithm can be found in Canon [4] and xPerm [5]. An improved version which uses the same method but with a better $O(n^{1.36})$ complexity in cases of total symmetry, or large totally symmetric subsets of indices, has been given in [6].

As mentioned above, these algorithms or their implementations do not necessarily lead to the same canonical form. This is also true for "multi-term symmetries", which map one term to a *sum* of others (e.g. Bianchi identities). For canonicalisation subject to such symmetries, there is far

less literature available. Most approaches treat these multi-term relations as side-relations [7, 8], and determine equivalence of expressions subject to these relations using e.g. Gröbner bases techniques or by defining an explicit canonical index order for individual tensors [7, 9]. Very few complete (and maintained) implementations exist.

Many computer algebra systems consider canonicalisation as essentially the same thing as "simplification", that is, the reduction of an expression to "the simplest form". That this is not always helpful becomes manifest in the case of tensor multi-term symmetries. Canonical forms will always require that certain simple expressions will be re-written as more complicated ones with more terms. Many simple expressions which would appear naturally in a pen-and-paper computation will turn into much more complicated ones after multi-term canonicalisation. And even if that does not happen, different people may not agree on the same definition of "simplest".

The present paper builds on the idea that one should let go of the idea that canonicalisation is the same thing as simplification, and consider them separately. The question then arises whether canonicalisation is something which the user should be confronted with *at all*. We will argue that, in particular for expressions subject to multi-term symmetries, it is more useful to use canonicalisation only "under the hood", and provide a simplification algorithm which, instead, aims for a representation of the user's input which is as compact *and* as close as possible in structure to the original. We will explain that this idea has the advantage that canonicalisation is now allowed to produce ugly or very lengthy forms even for expressions which can be written in a simple way, and one can thus use algorithms which may not have anything to do with what one would with pen-and-paper. We discuss such an algorithm for expressions subject to tensor multi-term symmetries based on Young projectors, and present a fully-automatic implementation of it in the computer algebra system Cadabra [10].

The outline of this paper is as follows. In section 2.1-2.3 we summarise the general formalism behind mono- and multi-term symmetries. We then describe a generic canonicalisation method based on projection operators in section 2.4. The main new contribution is formed by sections 2.5 and 2.6, where we describe a simplification method based on this underlying canonicalisation method, and show how it works at the level of explicit examples. Implementation and complexity details are discussed in sections 2.7 and 2.8. We end with some discussion of extensions of the algorithm in section 3.

2 Simplifying generic tensor expressions

2.1 Review of tensor canonicalisation methods

The level of support for simplifying expressions based on tensor symmetries in existing computer algebra software is limited mainly to the mono-term symmetries of an object, while consideration of the multi-term symmetries of an object is usually only possible for some special cases and is generally provided as packages instead of core functionality.

For mono-term symmetries, defining a canonical form is simple and straightforward [2]: given an ordered set of indices, the canonical representation of an object is that which has the lexicographically least permutation of its indices which can be reached using its symmetries. In the case of a tensor with only free indices this can be mapped to the group theory problem of finding a representative for the term in the cosets of identical terms; this idea is extended in [3], where the symmetry of dummy indices requires the solution to be a double-coset representative. An alternative is to describe tensors as graphs (avoiding the problem of dummy index relabelling altogether), as is done e.g. in RedBerry [11].

Defining what a canonical representation means for an object with more complicated symmetries, such that one term can be equivalent to an arbitrary number of other terms, is more

complicated; proposals have been made in e.g. [7], although by this definition a single term may contain multiple terms in its "canonical" form. However, the main problem is that in general the computational expense of simplifying expressions using multi-term symmetries is very large and can be impractical even on modern hardware. There are, however, some approaches which can be taken to solve the problem in specific situations and there are implementations of these, mainly as packages for Mathematica and Maple, and mainly for the reduction of Riemann polynomials.

The method employed by the Invar package [8] (a package that runs on top of xTensor [5] or Canon[4]) is to create a rainbow table of possible relations and, after performing a mono-term canonicalisation, doing a lookup on the terms in the expression. The scope of this method is obviously somewhat limiting, typically only working for specific types of tensor monomials. The major benefit is that this provides a result very quickly for these common situations.

Another method is to accept a list of symmetries which completely describe the object and use group algebra techniques to simplify the expression, as demonstrated by the ATENSOR REDUCE program[12] and the Mathematica package "Tools of Tensor Calculus"[13]. Although this is a completely general method it suffers from both being far less efficient than the lookup method and also requiring the user to first calculate a symmetry basis for the object.

There are other papers describing theoretical approaches to the problem (such as [14, 15]) but without associated implementations.

2.2 Mono-term canonicalisation

Algorithms for tensor canonicalisation typically use the Butler-Portugal algorithm, which uses the generating set of an object's mono-term symmetries to find the lexicographically least permutation of indices [2]. For example, the symmetries of the Riemann tensor

$$\begin{aligned} R_{abcd} &\equiv -R_{bacd} \equiv -R_{abdc}\,, \\ R_{abcd} &\equiv R_{cdab}\,, \end{aligned} \tag{1}$$

can be described by the generating set

$$[\{-1,(0,1)\},\{1,(0,2)(1,3)\}]\,, \tag{2}$$

where the first element in each member of the set is the polarity of the symmetry and the second is the positions of the indices which are swapped. The relation $R_{abcd} \equiv -R_{abdc}$ is obtained by the application of both these symmetries and therefore does not need to be a member of this set. From this generating set the full list of equivalent permutations can be generated; in the case of R_{cdba} this is

$$-R_{abcd},\ R_{abdc},\ R_{bacd},\ R_{badc},\ -R_{cdab},\ R_{cdba},\ R_{dcab},\ -R_{dcba}\,, \tag{3}$$

from which, assuming no other ordering has been imposed, the least element is $-R_{abcd}$ and thus this is calculated as the canonical representation of R_{cdba}. The 16 remaining index permutations of the Riemann tensor similarly fall into two lists each containing 8 permutations and there are thus only 3 independent canonical representations of the Riemann tensor:

$$R_{abcd},\ R_{acdb},\ R_{adbc}\,. \tag{4}$$

By applying this algorithm to all terms in a sum of Riemann tensors it will therefore be reduced to the simplest possible representation containing at most 3 terms.

However when considering multi-term symmetries this method is no longer sufficient for detecting identical terms — take the simple example of the first Bianchi identity

$$R_{abcd} + R_{acdb} + R_{adbc} \equiv 0\,. \tag{5}$$

It is clear that application of the Riemann tensor mono-term symmetries will never yield (5) as these are the three independent permutations of the Riemann tensor from (4). Of course, examples such as this can be handled as special cases, but in the general case a more robust solution is required.

2.3 Tensor symmetries as Young diagrams

The "canonicalisation" routine we propose relies on the application of Young symmetrisers which are informally introduced here. There is a large number of textbooks on representations of the symmetric group; much of the following was inspired by [16] and [17] whilst a terser introduction to the subject can be found in [18].

The symmetries of objects which are invariant under permutation are described by the symmetric group S_n whose elements σ_i consist of the $n!$ possible permutations of the elements $1, 2, \ldots, n$. These permutations are commonly labelled in cycle notation, so that the label $i = (145)$ indicates the rearrangement $1 \to 4$, $4 \to 5$ and $5 \to 1$. Some permutations may be described by disjoint products of cycles such as $(12)(34)$. If a product of cycles is not disjoint, that is to say more than one cycle contains the same label, then conventionally the permutations are performed starting with the rightmost and working left, but it can also be rewritten as a product of one or more disjoint cycles by following where each cycle goes to, for example $(123)(24) \equiv (2413)$. Permutations can be divided into two categories depending on whether the number of swaps required to create it is even or odd; the permutation (23) only requires one swapping $(2 \leftrightarrow 3)$ and is an odd permutation, whilst (12345) is an even permutation requiring four swaps $(1 \leftrightarrow 2,\ 1 \leftrightarrow 3,\ 1 \leftrightarrow 4,\ 1 \leftrightarrow 5)$.

We can define representations ψ_i of S_n and it is these which define the symmetry of the object on which the group acts; the simplest example of a representation of S_n is the *trivial representation* which sends each element to the identity operator: $\psi_{\text{triv}}(\sigma) = \mathbf{1}$. Clearly this represents a completely symmetric object, for instance the metric tensor $g_{\mu\nu} \equiv g_{\nu\mu}$ has this representation on S_2. Another very common representation is the *alternating representation* defined by $\psi_{\text{alt}}(\sigma) = \text{sgn}(\sigma)$ where

$$\text{sgn}(\sigma) = \begin{cases} 1 & \text{if } \sigma \text{ is an even permutation}, \\ -1 & \text{if } \sigma \text{ is an odd permutation}, \end{cases} \tag{6}$$

so that $\psi_{\text{alt}}(\sigma_{(123)}) = 1$ and $\psi_{\text{alt}}(\sigma_{(45)}) = -1$. Objects with this representation are completely anti-symmetric, like the Levi-Civita symbol $\epsilon^{abc} \equiv -\epsilon^{bac} \equiv \epsilon^{bca} \equiv \ldots$ and the structure constants of many Lie algebras. As these two representations are both one-dimensional, together they form the complete set of irreps for $n = 2$: this is equivalent to stating that an object with two elements can be decomposed into a mixture of symmetric and antisymmetric parts, which is commonly seen in the classic decomposition of a matrix

$$M_{ab} \equiv \frac{1}{2}(M_{ab} + M_{ba}) + \frac{1}{2}(M_{ab} - M_{ba})\,. \tag{7}$$

Together the two brackets on the right hand side add up to M_{ab}, however, as the first bracket is manifestly symmetric and the second antisymmetric if the matrix M_{ab} possessed either of these symmetries one of the brackets would be identically zero.

When $n > 2$ these two representations no longer suffice to fully cover S_n, however, the construction of irreps is simplified by the use of Young tableaux, which are a diagrammatic representation of the irreps of S_n. The construction begins by drawing the *Young diagrams* for some n, and then by filling these diagrams with labels we will find the complete set of Young tableaux, each of which corresponds to an irrep. Each diagram λ corresponds to a partition of n,

commonly denoted $\lambda \vdash n$. In the case of $n = 3$ there are three partitions: $\lambda_1 = (3)$, $\lambda_2 = (2, 1)$ and $\lambda_3 = (1, 1, 1)$. A Young diagram is a left-justified rows of cells, where the length of each row is given by one of the members of the partition and the length of each row is always less than or equal to the row above. So for $n = 3$ the diagrams are

$$\lambda_1 = \ydiagram{3}, \quad \lambda_2 = \ydiagram{2,1}, \quad \lambda_3 = \ydiagram{1,1,1}, . \tag{8}$$

The set of all standard tableaux, and thus the irreps of S_n, is given by all the possible standard fillings of these diagrams, which is done by assigning the labels $1, 2, \ldots, n$ to each cell such that in each row the labels are strictly increasing left to right and in each column they are strictly increasing top to bottom:

$$\Lambda_{\text{triv}} = \begin{array}{|c|c|c|}\hline 1&2&3\\\hline\end{array}, \quad \Lambda_{\text{std}_1} = \begin{array}{|c|c|}\hline 1&2\\\hline 3\\\cline{1-1}\end{array}, \quad \Lambda_{\text{std}_2} = \begin{array}{|c|c|}\hline 1&3\\\hline 2\\\cline{1-1}\end{array}, \quad \Lambda_{\text{alt}} = \begin{array}{|c|}\hline 1\\\hline 2\\\hline 3\\\hline\end{array}. \tag{9}$$

Each diagram represents a symmetrization along each row and an antisymmetrization along each column, from which it is easy to see that Λ_{triv} corresponds to the trivial representation and Λ_{alt} the alternating representation. The other two tableaux are known as the *standard representations* and describe a mixed symmetry that is neither entirely symmetric nor entirely antisymmetric.

These tableaux can also be used to construct a decomposition, such as the one in (7), by constructing a *projection operator* for each tableau given by

$$P^+(\Lambda) = \frac{1}{N} \prod_{r \in \Lambda} S(r) \prod_{c \in \Lambda} A(c)\,. \tag{10}$$

Here r and c are the rows and columns of the tableau, S and A the symmetrization and antisymmetrization operators. N is a normalisation constant given by the product of the hook-lengths of each cell, the hook-length of a cell being the number of cells to the right and below a cell including the cell itself: in the diagram below each cell contains its hook length

$$\begin{array}{|c|c|c|c|c|}\hline 7&5&4&3&1\\\hline 5&3&2&1\\\cline{1-4} 1\\\cline{1-1}\end{array}\,, \tag{11}$$

and the normalisation would be $N = 7 \times 5^2 \times 4 \times 3^2 \times 2 \times 1^3 = 12600$. If the order of expanding out the columns and rows were reversed then we would obtain a different projector P^-; the symmetries become manifest under either formulation as long as the two types of projectors are not mixed, we will assume the convention that our projectors are of the form P^+.

For the four irreps of S_3 we find

$$\begin{aligned}
P\left(\begin{array}{|c|c|c|}\hline 1&2&3\\\hline\end{array}\right) T_{abc} &= \frac{1}{6}(T_{abc} + T_{acb} + T_{bac} + T_{bca} + T_{cab} + T_{cba})\,,\\
P\left(\begin{array}{|c|c|}\hline 1&2\\\hline 3\\\cline{1-1}\end{array}\right) T_{abc} &= \frac{1}{3}(T_{abc} + T_{bac} - T_{cba} - T_{bca})\,,\\
P\left(\begin{array}{|c|c|}\hline 1&3\\\hline 2\\\cline{1-1}\end{array}\right) T_{abc} &= \frac{1}{3}(T_{abc} + T_{cba} - T_{bac} - T_{cab})\,,\\
P\left(\begin{array}{|c|}\hline 1\\\hline 2\\\hline 3\\\hline\end{array}\right) T_{abc} &= \frac{1}{6}(T_{abc} - T_{acb} - T_{bac} + T_{bca} + T_{cab} - T_{cba})\,.
\end{aligned} \tag{12}$$

It can be easily calculated that, similarly to (7), the sum of these four projectors is T_{abc} and represents the decomposition of T_{abc} into symmetric parts.

The symmetry of a general tensor with n indices can be any combination of the irreps of S_n, and in order to determine which tableaux contribute to a tensor's symmetry this full decomposition can be written and any terms which do not satisfy the symmetry of the object discarded. However, while the complete set of Young projectors above sum up to the identity

$$\sum_{\Lambda \in \square^{\otimes n}} P(\Lambda) = \mathbf{1}\,, \tag{13}$$

the same is not true of other decompositions of products of Young tableaux as given by the Littlewood-Richardson rule (a derivation for which can be found in numerous places); examining the projections given in (12) one can see that

$$P\left(\begin{array}{|c|}\hline 1\\\hline 2\\\hline\end{array} \otimes \begin{array}{|c|}\hline 3\\\hline\end{array}\right) \neq P\left(\begin{array}{|c|c|}\hline 1&3\\\hline 2\\\cline{1-1}\end{array}\right) + P\left(\begin{array}{|c|}\hline 1\\\hline 2\\\hline 3\\\hline\end{array}\right)\,. \tag{14}$$

Another counter-intuitive property of these Young projectors is that they are not in general orthogonal, that is

$$P(\Lambda_i)P(\Lambda_j) \neq \delta_{ij}\,, \tag{15}$$

although this does hold in ≤ 4 dimensions (see e.g. [19]).

In this case a different construction of the projectors must be used. There are different constructions for which this and many other useful properties hold, including the Hermitian KS construction [20], their compact counterparts [19], and the original orthogonal construction given by Littlewood [21]. It should be noted that these constructions are far more computationally intensive than the simple construction described above, and while properties such as orthogonality and Hermiticity are desirable in other contexts, they are superfluous to the workings of our algorithm other than when a symmetry is described by a sum of irreps, in which case this will be noted.

2.4 Canonicalisation using projection operators

For any tensor symmetries, but in particular for multi-term ones, a simple way to canonicalise an expression is to project all tensors using their Young projection operators. The simplicity of this lies in the fact that it makes all symmetries manifest, and all multi-term relations will be satisfied identically after the projection. This has been used in concrete applications in the past (see e.g. [22]). This process is however cumbersome to the user and is often only practical if one has a reasonable idea of what the answer should be prior to the actual calculation. The huge number of terms which can be generated by a Young projection also mean that a naive approach to the problem can easily lead to space and time constraints becoming unmanageable.

To illustrate the robustness of this technique, consider the Bianchi identity from section 2.2 which the mono-term algorithms failed to spot. Fortunately this symmetry of the Riemann tensor can be made manifest by writing out the Young projection of each term. The decomposition of S_4 into irreps yields

$$\begin{array}{|c|c|c|c|}\hline 1&2&3&4\\\hline\end{array} \oplus \begin{array}{|c|c|c|}\hline 0&1&2\\\hline 3\\\cline{1-1}\end{array} \oplus \begin{array}{|c|c|c|}\hline 0&1&3\\\hline 2\\\cline{1-1}\end{array} \oplus \begin{array}{|c|c|c|}\hline 0&2&3\\\hline 1\\\cline{1-1}\end{array} \oplus \begin{array}{|c|c|}\hline 0&2\\\hline 1&3\\\hline\end{array} \oplus \begin{array}{|c|c|}\hline 0&1\\\hline 2&3\\\hline\end{array} \oplus \begin{array}{|c|c|}\hline 0&2\\\hline 1\\\cline{1-1} 3\\\cline{1-1}\end{array} \oplus \begin{array}{|c|c|}\hline 0&1\\\hline 2\\\cline{1-1} 3\\\cline{1-1}\end{array} \oplus \begin{array}{|c|c|}\hline 0&3\\\hline 1\\\cline{1-1} 2\\\cline{1-1}\end{array} \oplus \begin{array}{|c|}\hline 0\\\hline 1\\\hline 2\\\hline 3\\\hline\end{array}\,. \tag{16}$$

Clearly the only one of these terms which satisfies the symmetries of the Riemann tensor is

$$\begin{array}{|c|c|}\hline 0 & 2 \\ \hline 1 & 3 \\ \hline \end{array} \tag{17}$$

and so the projection of R_{abcd} consists of the 16 terms

$$\begin{aligned}&\frac{1}{12}\Big(R_{abcd} - R_{abdc} - R_{acdb} + R_{adcb} - R_{bacd} + R_{badc} + R_{bcda} - R_{bdca} \\ &- R_{cabd} + R_{cbad} + R_{cdab} - R_{cdba} + R_{dabc} - R_{dbac} - R_{dcab} + R_{dcba}\Big)\,.\end{aligned} \tag{18}$$

This expression manifestly exhibits all symmetries of the Riemann tensor. It is clearly not a very compact canonical form, but it does allow us to prove the Bianchi identity: by performing the Young projection of R_{acdb} and R_{adbc} as well, all the terms can be summed together to show that the result is indeed 0.

The Young-projected canonical form is thus far from a simplification, but it can be used to verify the equivalence of two expressions. The expression

$$R_{acdb} + R_{adbc} \tag{19}$$

can be projected and compared against the projection of R_{abcd} to find that they are identical up to a minus sign, so that the expression can be substituted for $-R_{abcd}$ as expected. In the remainder of this paper we discuss an alternative approach to simplification, which, similar to the example above, uses the messy and lengthy Young-projection canonicalisation under the hood to ensure that any given expressions never uses an over-complete basis of terms.

2.5 Hiding canonicalisation in simplification

The algorithm we propose based on this mechanism is implemented in Cadabra [10] with the name `meld`. The purpose of this algorithm is to provide an all-purpose routine which can be applied to any expression and ensure that only the minimal set of basis terms will appear in the output. This general logic applies to symmetries which go beyond those which can be achieved using Young projectors, and we will discuss a few of these in section 3, but the main focus of this paper will be on tensor expressions.

The Young projection routine in `meld` takes as an input a sum of terms which are identical in structure modulo index permutations. The process, visually depicted in fig. 1, is based on updating two stacks: `seen_terms` and `unseen_terms`. Terms from the input are added onto the `unseen_terms` stack and one-by-one popped off and Young projected to see if they can be written as a linear combination of terms in `seen_terms`. If such a linear combination exists, then the scalar factors of the terms in `seen_terms` are updated to include the contribution from the current term, else the current term is appended to `seen_terms`. The process continues until there are no more unseen terms.

From this the difference between "canonicalisation" and "simplification" is made evident: while a traditional canonicalisation routine will take each term in turn and replace it with a canonical representation so that at the end all identical terms can be collected, `meld` merely detects terms which are equivalent to each other and collects them together. An example of where this can result in radically different outputs is found in [7], where the definition of canonical form causes the expression

$$R^{abcd}R^{e\ f}_{\ a\ c}R_{bfde} \tag{20}$$

to be rewritten in the canonical form (modulo dummy index naming)

$$R^{abcd}R^{e\ f}_{a\ c}R_{bedf} - \frac{1}{4}R^{abcd}R^{ef}{}_{ab}R_{cedf}\,. \tag{21}$$

The `meld` algorithm would instead combine the second expression into the first.

2.6 Examples

2.6.1 Mono-term symmetries

Although specifically designed to act on expressions which can only be simplified by the examination of the multi-term symmetries of an object, `meld` is also capable of canonicalising expressions where only the mono-term symmetries need be considered. It must be noted, however, that whilst traditional canonicalisation routines, such as the `canonicalise` function provided by Cadabra which is based on the xPerm [5] canonicalisation algorithm, will return a lexicographically least form of the tensor, `meld` will only ever try to combine existing terms. A simple example is to look at a fully antisymmetric tensor A_{abc}: the following is the output of an interactive Cadabra session:

```
> A_{a b c}::AntiSymmetric;
 Property AntiSymmetric attached to A_{a b c}.
> ex := A_{b a c} + A_{c b a};
 A_{b a c} + A_{c b a}
> meld(ex);
 2A_{b a c}
> ex := A_{b a c} + A_{c b a};
 A_{b a c} + A_{c b a}
> canonicalise(ex);
 -2A_{a b c}
```

The salient point here is that while both algorithms produce a minimal representation of the input, as the two algorithms use different approaches to solving the problem they will often produce a different output. If the lexicographically canonical form is desired then a call to the far less computationally expensive `canonicalise` algorithm after `meld` can be used.

A more complex example is found in [23, 10.5.5] where it is shown through algebraic means that

$$\mathrm{Tr}(F\wedge A\wedge A) \equiv -\,\mathrm{Tr}(A\wedge F\wedge A)\,, \tag{22}$$

where A is a matrix-valued one-form and F is a matrix-valued two-form. Rewriting this in index notation makes the statement equivalent to the monomial identity:

$$F^{ab}{}_{\mu\sigma}A^{bc}{}_{\nu}A^{ca}{}_{\rho}\epsilon^{\mu\sigma\nu\rho} + A^{ab}{}_{\mu}F^{bc}{}_{\nu\sigma}A^{ca}{}_{\rho}\epsilon^{\mu\nu\sigma\rho} \equiv 0\,, \tag{23}$$

where $\epsilon^{\mu\nu\sigma\rho}$ is the fully antisymmetric Levi-Civita symbol. In this formulation it is within the scope of `meld` to show:

```
> \epsilon^{\mu \nu \rho \sigma}::AntiSymmetric;
 Property AntiSymmetric attached to \epsilon^{\mu \nu \rho \sigma}.
> ex:= F^{a b}_{\mu \sigma} A^{b c}_{\nu} A^{c a}_{\rho} \epsilon^{\mu \sigma \nu \rho}
      + A^{a b}_{\mu} F^{b c}_{\nu \sigma} A^{c a}_{\rho} \epsilon^{\mu \nu \sigma \rho}:
> sort_product(ex);
  A^{b c}_{\nu} A^{c a}_{\rho} F^{a b}_{\mu \sigma} \epsilon^{\mu \sigma \nu \rho}
+ A^{a b}_{\mu} A^{c a}_{\rho} F^{b c}_{\nu \sigma} \epsilon^{\mu \nu \sigma \rho}
> meld(ex);
 0
```

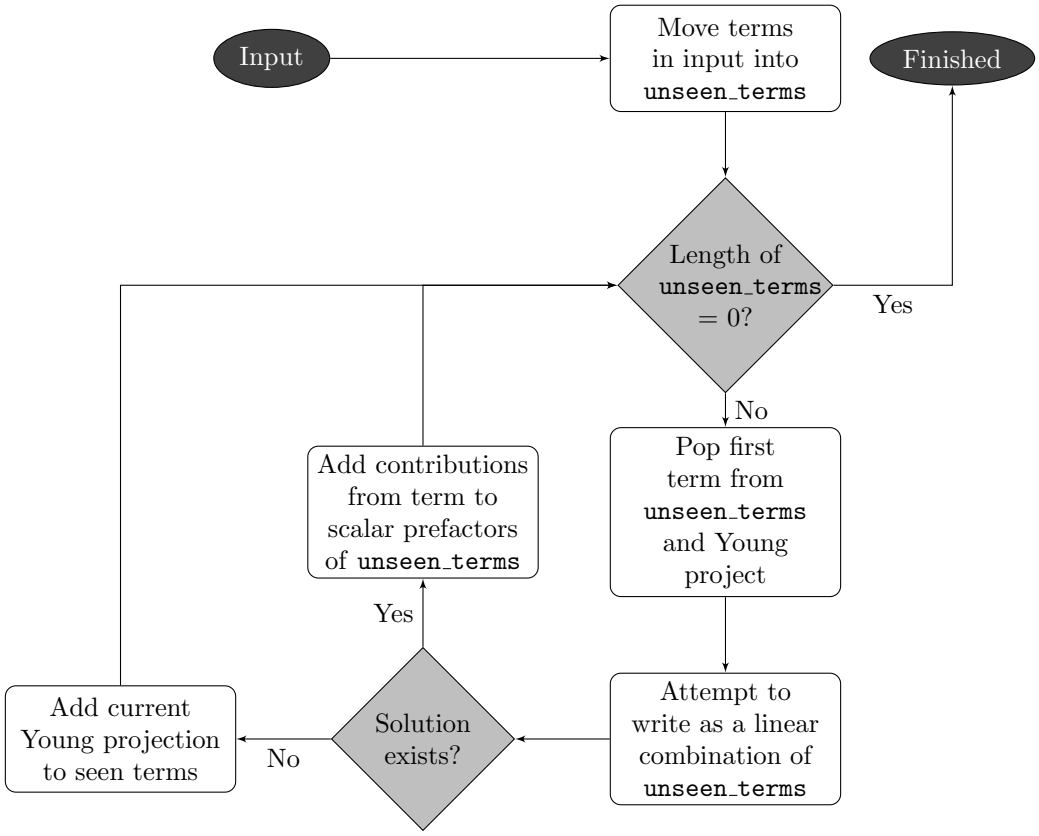


Figure 1: Visual representation of the logic in the `meld` algorithm.

Notice that although `meld` is able to take advantage of the fact that the $A^{ab}{}_{\mu}$ tensors are identical to symmetrise in these objects, `sort_product` must first be called to ensure that the structure of the two monomials matches.

2.6.2 Bianchi identities

As described above, the aspect of `meld` which makes it far more general than `canonicalise` is its ability to simplify expressions for which only considering the mono-term symmetries is not enough. An example shows the difference:

```
> R_{a b c d}::RiemannTensor.
 Property TableauSymmetry attached to R_{a b c d}.
> ex := R_{a b c d} + R_{a c d b} + R_{a d b c};
 R_{a b c d} + R_{a c d b} + R_{a d b c}
> canonicalise(ex);
 R_{a b c d}-R_{a c b d} + R_{a d b c}
> meld(ex);
 0
```

Here the `canonicalise` routine is only able to consider each term individually, making the most canonical form it can find for each, whilst `meld` recognises the Bianchi identity, although of course `meld` has no knowledge of specific identities but relies of the Young projection to make all such identities manifest. It can also take advantage of this if there are arbitrary scalar factors attached to each term

```
> ex := a_1 * R_{a b c d} + a_2 * R_{a c d b} + a_3 * R_{a d b c};
 a_{1} R_{a b c d} + a_{2} R_{a c d b} + a_{3} R_{a d b c}
> meld(ex);
 (a_{1}-a_{3}) R_{a b c d} + (a_{2}-a_{3}) R_{a c d b}
```

In this example the expression has been reduced to the fewest possible number of terms required to represent it making use of the Bianchi identity.

2.6.3 Tensor polynomials

The real convenience of `meld` comes when considering polynomials of objects which admit multi-term symmetries as the number of terms in their Young projection increases exponentially in the order of the polynomial, however this complexity can be hidden from the user. A good example of this is the following identity involving four Riemann tensors [24, 25],

$$R_{pqrs}R_{ptru}R_{tvqw}R_{uvsw} - R_{pqrs}R_{pqtu}R_{rvtw}R_{svuw} - R_{mnab}R_{npbc}R_{mscd}R_{spda} + \frac{1}{4}R_{mnab}R_{psba}R_{mpcd}R_{nsdc} \equiv 0\,. \quad (24)$$

Here there are 65,556 terms in the Young projection of each monomial, but this complexity is hidden in the implementation of the algorithm

```
> R_{m n p q}::RiemannTensor;
 Property RiemannTensor attached to R_{m n p q}.
> ex:= R_{p q r s} R_{p t r u} R_{t v q w} R_{u v s w}
| - R_{p q r s} R_{p q t u} R_{r v t w} R_{s v u w}
| - R_{m n a b} R_{n p b c} R_{m s c d} R_{s p d a}
| + (1/4) R_{m n a b} R_{p s b a} R_{m p c d} R_{n s d c}:
> meld(ex);
 0
```

It would also be possible to prove this identity by expanding each monomial separately and then using a traditional canonicalisation algorithm to simplify each bracket before distributing the result, however these extra steps are hidden by using `meld`. The simplification (to zero) is thus manifestly different from the internal (but hidden) canonicalisation process.

As mentioned above, `meld` attempts to combine terms together in the way in which they appear in the input, and therefore the order in which terms in a sum are placed can change the output. This is clearly visible in the two different lines below,

$$\begin{aligned} R_{ijkl}R_{ijkl} + R_{ijkl}R_{ikjl} &\to \frac{3}{2}R_{ijkl}R_{ijkl}\,, \\ R_{ijkl}R_{ikjl} + R_{ijkl}R_{ijkl} &\to 3R_{ijkl}R_{ikjl}\,. \end{aligned} \tag{25}$$

In both cases, `canonicalise` is unable to detect the relationship

$$R_{ijkl}R_{ijkl} \equiv 2R_{ijkl}R_{ikjl} \tag{26}$$

which, although initially appearing to be a mono-term relation, contains a hidden Bianchi identity.

Of course the logic of `meld` works equally well for expressions which contain multiple types of fields. The following example is taken from [26], which deals with monomials of the symbolic form H^2R^3, where H is a completely anti-symmetric field strength and R is the Riemann tensor. Table 3 of [26] shows an expansion of certain contractions in terms of a basis, given explicitly in the appendix of that paper. With `meld` we can construct a basis explicitly along the following lines:

```
> H_{a b c}::AntiSymmetric;
> t8t8H2R3:= t8_{m1 m2 m3 m4 m5 m6 m7 m8} t8_{n1 n2 n3 n4 n5 n6 n7 n8}
            H_{m1 m2 a} H_{n1 n2 a}
            R_{m3 m4 n3 n4} R_{m5 m6 n5 n6} R_{m7 m8 n7 n8};
> t8rule:= t8_{m1 n1 m2 n2 m3 n3 m4 n4} =
          -2 \delta_{n2 m1} \delta_{n1 m2} \delta_{n4 m3} \delta_{n3 m4}
          + ...
> substitute(t8t8H2R3, t8rule)
> distribute(_)
> eliminate_kronecker(_)
> meld(_)
 12 H_{m_4 m_3 a} H_{n_4 n_3 a}
    R_{m_3 m_4 n_3 n_4} R_{m_8 m_7 n_8 n_7} R_{m_7 m_8 n_7 n_8} + ...
```

This produces a minimal number of terms (six), which does not, however, use the same basis terms as in [26]. If one wanted to use the particular basis of [26], one could instead write down a generic linear combination of those basis terms, subtract from that the expression above, and run `meld` on the sum.

```
> expansion := c_1 R_{m n p s} R_{a b c d} R_{a b c d}
             + c_2 R_{m n p a} R_{s b c d} R_{a b c d} + ...;
> zero = expansion - t8t8H2R3;
> meld(zero);
 (c_1 + 1/8) H_{m n q} H_{p s q} R_{a b c d} R_{a b c d} R_{m n p s} + ...
```

The condition that this vanishes then fixes the coefficients c_i in agreement with the result of [26]. In this way `meld` can be used both to generate a basis and to verify an expansion in terms of an already determined basis.

2.6.4 Other complex tableaux

Whilst the most common example of an object with a multi-term is the Riemann tensor, there are of course many other possible tableaux shapes admitting a variety of identities. One example is the tensor T_{abcd} with the associated tableau

$$\begin{array}{|c|c|c|}\hline 0 & 1 & 3 \\ \hline 2 \\ \cline{1-1}\end{array} \tag{27}$$

from which using `meld` we can confirm also satisfies the Bianchi identity

$$T_{abcd} + T_{acdb} + T_{adbc} = 0\,. \tag{28}$$

Similarly other tableau shapes such as T_{abcde} described by

$$\begin{array}{|c|c|c|}\hline 1 & 3 & 0 \\ \hline 2 & 4 \\ \cline{1-2}\end{array}\,, \tag{29}$$

which is beyond the scope of most modern computer algebra systems to fully canonicalise can be shown to satisfy the relation

$$T_{abcde} + T_{abdec} + T_{baecd} + T_{cadbe} = 0\,. \tag{30}$$

Such "hook tableaux" are useful as they describe the symmetry of the covariant derivative of the Riemann tensor with a metric connection: this can be seen by computing the Littlewood-Richardson decomposition of the direct product of the two tableaux associated with ∇_e and R_{abcd}:

$$\begin{array}{|c|}\hline e \\ \cline{1-1}\end{array} \otimes \begin{array}{|c|c|}\hline a & c \\ \hline b & d \\ \cline{1-2}\end{array} = \begin{array}{|c|c|c|}\hline a & c & e \\ \hline b & d \\ \cline{1-2}\end{array} \oplus \begin{array}{|c|c|}\hline a & c \\ \hline b & d \\ \hline e \\ \cline{1-1}\end{array}\,. \tag{31}$$

The second Bianchi identity

$$\nabla_e R_{abcd} + \nabla_c R_{abdc} + \nabla_d R_{abec}\,, \tag{32}$$

is only satisfied by the first term in the decomposition and thus is the only term which describes the symmetry of $\nabla_e R_{abcd}$. Repeating this process [27, 28] reveals that the nth covariant derivative $\nabla_{e_1}\nabla_{e_2}\ldots\nabla_{e_n}R_{abcd}$ is described by

$$\overbrace{\begin{array}{|c|c|c|c|}\hline & & & \\ \hline & \\ \cline{1-2}\end{array}\cdots\begin{array}{|c|}\hline \\ \cline{1-1}\end{array}}^{n+2\text{ boxes}} \tag{33}$$

In fact `meld` can take advantage of this fact to canonicalise expressions involving covariant derivatives of Riemann tensors, which we illustrate with an example from [29]:

$$R^{abcd;e}{}_{a}R_{be}{}^{fg;hi}R_{cfgi;dh} = \frac{1}{8}R^{abcd;e}{}_{e}R_{ab}{}^{fg;hi}R_{cdfg;ih}\,, \tag{34}$$

which can be shown in Cadabra with

```
> D{#}::Derivative.
> R_{a b c d}::RiemannTensor.
> ex := D_{e}{D_{a}{R_{a b c d}}} D_{h}{D_{i}{R_{b e f g}}} D_{d}{D_{h}{R_{c f g i}}} -
| 1/8 D_{e}{D_{e}{R_{a b c d}}} D_{h}{D_{i}{R_{a b f g}}} D_{i}{D_{h}{R_{c d f g}}};
> meld(ex);
 0
```

2.7 Implementation details

While a high-level description of how the algorithm works has been presented in section 2.5, in this section some of the lower-level implementation details of the `meld` algorithm are given.

2.7.1 Data storage

In Cadabra, expressions are implemented as `Ex` objects which are trees consisting of nodes which contain a name, rational multiplier and information such as bracket type and parent relation which are only set for some node types. There are three main drawbacks to this storage format when considering storing Young projections. Most evidently, a collection of terms belonging to a Young projection will share the exact same structure except for index permutations. Thus the majority of the stored information is superfluous. Moreover, nodes are stored in a linked list which causes sibling traversal, such as iterating over the indices of an object, to be very likely to cause cache misses. Finally, dummy index naming is not irrelevant as T_{aa} and T_{bb} have different representations in this format even though they are identical.

In order to solve these problems a different storage format is introduced for the internals of the `meld` algorithm. This format stores the `Ex` object of a tensor monomial once per projection, alongside a list of *adjacency* representations of the index permutations in the projection, which is a mapping of the actual indices to a list of integers. To illustrate this with an example, consider the expression $R_{abcd}F_{cd}$ which contains two free indices a and b as well as two contractions. Internally, before projecting the term `meld` stores the `Ex` object representing this along with a single adjacency list

Slot						Weight
0	1	2	3	4	5	
-1	-2	4	5	2	3	1

The slots represent zero-based pointers into the index structure of the expression, so for the above expression slots 0-3 point to the four indices of R_{abcd} and slots 4-5 the two indices of F_{cd}. The negative integers represent unique free indices and the positive integers represent dummy indices by acting like pointers and indicating that the index in that slot is contracted with the index at the slot it "points" to: the value of 4 in slot 2 indicates that slots 2 and 4 are contracted representing the two c dummy indices in the expression. For consistency slot 4 contains the value 2 completing the contraction.

Each adjacency list is also assigned a weight, initially set to 1, which keeps track of the relative contributions each term has towards the overall object.

2.7.2 Symmetrization

Symmetrization is performed by decomposing the associated tableaux into columns and rows and symmetrizing along these. Assuming the tensor R_{abcd} has been defined with the tableau

$$\begin{array}{|c|c|}\hline 0 & 2 \\ \hline 1 & 3 \\ \hline \end{array}\,, \tag{35}$$

the algorithm successively applies the two antisymmetrisers $[0,1]$ and $[2,3]$ and the two symmetrisers $(0,2)$ and $(1,3)$ to all the adjacency lists:

(1) Antisymmetrise in (0,1)

Slot						Weight
0	1	2	3	4	5	
-1	-2	4	5	2	3	1
-2	-1	4	5	2	3	-1

(2) Antisymmetrise in (2,3)

Slot						Weight
0	1	2	3	4	5	
-1	-2	4	5	2	3	1
-2	-1	4	5	2	3	-1
-1	-2	5	4	3	2	-1
-2	-1	5	4	3	2	1

(3) Symmetrise in (0,2)

Slot						Weight
0	1	2	3	4	5	
-1	-2	4	5	2	3	1
-2	-1	4	5	2	3	-1
-1	-2	5	4	3	2	-1
-2	-1	5	4	3	2	1
4	-2	-1	5	0	3	1
4	-1	-2	5	0	3	-1
5	-2	-1	4	3	0	-1
5	-1	-2	4	3	0	1

(4) Symmetrise in (1,3)

Slot						Weight
0	1	2	3	4	5	
-1	-2	4	5	2	3	1
-2	-1	4	5	2	3	-1
-1	-2	5	4	3	2	-1
8 terms omitted...						
4	5	-1	-2	0	1	1
4	5	-2	-1	0	1	-1
5	4	-1	-2	1	0	-1
5	4	-2	-1	1	0	1

yielding the 16 terms as expected. Notice that the projectors are not normalised; this is in order to allow the weights to be combined using integer arithmetic instead of the more expensive rational arithmetic which would be required, and makes no difference to the result as all terms accumulate the same normalisation and so it only produces a global constant which can be factored out.

The `meld` algorithm can also accept objects which are defined with products or sums of tableaux. If products of tableaux, such as

$$\begin{array}{|c|}\hline 0 \\ \hline 1 \\ \hline \end{array} \otimes \begin{array}{|c|}\hline 2 \\ \hline 3 \\ \hline \end{array} \tag{36}$$

are provided, then the symmetrisers and antisymmetrisers are constructed and again applied in turn.

If a sum of tableaux is provided then the construction of Hermitian projectors given in [20] is

used to compute the symmetriser of each term, for example

$$\begin{aligned} H\left(\begin{array}{|c|c|}\hline 0&2\\\hline 1\\\cline{1-1}\end{array}\oplus\begin{array}{|c|}\hline 0\\\hline 1\\\hline 2\\\cline{1-1}\end{array}\right) &= H\left(\begin{array}{|c|c|}\hline 0&2\\\hline 1\\\cline{1-1}\end{array}\right)+H\left(\begin{array}{|c|}\hline 0\\\hline 1\\\hline 2\\\cline{1-1}\end{array}\right)\\ &= P\left(\begin{array}{|c|}\hline 0\\\hline 1\\\cline{1-1}\end{array}\right)P\left(\begin{array}{|c|c|}\hline 0&2\\\hline 1\\\cline{1-1}\end{array}\right)P\left(\begin{array}{|c|}\hline 0\\\hline 1\\\cline{1-1}\end{array}\right)+P\left(\begin{array}{|c|}\hline 0\\\hline 1\\\cline{1-1}\end{array}\right)P\left(\begin{array}{|c|}\hline 0\\\hline 1\\\hline 2\\\cline{1-1}\end{array}\right)P\left(\begin{array}{|c|}\hline 0\\\hline 1\\\cline{1-1}\end{array}\right)\end{aligned} \tag{37}$$

where H is a Hermitian projector and P the construction given in (10). In this instance the normalisation is relevant to ensure the correct mixing of terms, however it is still desirable to use integer arithmetic for efficiency and so each term is multiplied by the product of the all the normalisation constants so that

$$\frac{1}{N_1}P_1+\frac{1}{N_2}P_2+\cdots+\frac{1}{N_k}P_k \tag{38}$$

becomes

$$(N_2N_3\ldots N_k)P_1+(N_1N_3\ldots N_k)P_2+\cdots+(N_1N_2\ldots N_{k-1})P_k \tag{39}$$

where the brackets are guaranteed to be integer as each N_i is a product of integer hook-lengths.

2.7.3 Detection of linear dependence

The algorithm looks as each term in turn, calculating its Young projection as above and checking if it is a linear combination of previously encountered terms by solving the equation

$$L_1(c_{11}T_1+\ldots+c_{n1}T_n)+L_2(c_{12}T_1+\ldots+c_{n2}T_n)+\ldots=\lambda_1T_1+\ldots+\lambda_nT_n\,. \tag{40}$$

In this equation each bracket is a previously calculated projection of an input expression T_i which produces a sum of all the possible index permutations $T_1, T_2, \ldots, T_n$ with coefficients c_{ij} which may be zero, and the variables L_i are to be determined. The right hand side is the Young projection of the current term which we wish to express as a linear combination of the other projections.

As each T_i is linearly independent, by comparing coefficients this equation can be rearranged into the matrix equation $C\vec{L}=\vec{\lambda}$:

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1k}\\ c_{21} & \ddots & & \\ \vdots & & & \\ c_{n1} & & & c_{nk}\end{bmatrix}\begin{bmatrix}L_1\\L_2\\\vdots\\L_k\end{bmatrix}=\begin{bmatrix}\lambda_1\\\lambda_2\\\vdots\\\lambda_n\end{bmatrix}\,. \tag{41}$$

The matrix C is $n\times k$ and so the solution set depends on the relation between these two quantities. The columns of the matrix are always linearly independent as the vector $\vec{\lambda}$ is only appended to the matrix if it cannot be expressed as a linear combination of its columns. For $n=k$ there is therefore a unique solution and so k (the number of terms projected so far) can never get larger than n (the total number of possible index permutations). The initial matrix is constructed when $k=1$, and as all projections contain at least one term we therefore conclude $k\leq n$.

In fact, as n grows factorially with the number of indices it is normally much larger than k and the system is very over-constrained which makes solving the system computationally expensive. In order to mitigate this problem, the approach we take is to truncate the columns of C so that it

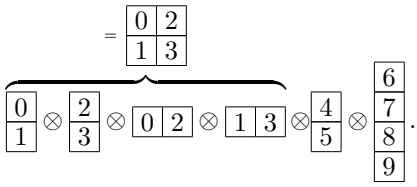
only contains k rows ensuring that linear independence between the columns is preserved. This is then solved to produce a potential solution $\vec{L}'$. To determine if it is an actual solution to (41), $\vec{L}'$ is then substituted back into (40) to ensure that it is a real solution; if not then the algorithm discards the solution $\vec{L}'$ and concludes that $\vec{\lambda}$ should be appended to C.

2.7.4 Optimisations

Even though the data structure we use reduces the memory footprint of the projections, their calculation is still an expensive process and various ways of minimising the complexity of this process have been considered. The current implementation focuses on two categories of optimisations: calculating independent symmetrisers and cancellation of similar symmetrisers. These are perhaps best illustrated by example. Consider

$$R^{\mu}{}_{\nu\rho\lambda} F_{\mu}{}^{\nu} \varepsilon^{ijlk} \,, \tag{42}$$

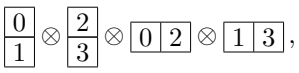
where $R^{\mu}{}_{\nu\rho\lambda}$ has the Riemann tensor symmetries and both F^{ν}_{μ} and ε^{ijkl} are totally antisymmetric. The projector in terms of index slots is represented by the product of symmetrisers



(43)

We first notice that two symmetrisers commute if they share no indices. This means that we can bring the antisymmetriser $\boxed{4}\boxed{5}^{T}$ to the front of the expression. However, the two slots it contains are contracted with the slots in the symmetriser $\boxed{0}\boxed{1}^{T}$ and therefore the application of one followed by the other is identical to only applying one allowing us to remove one of these symmetrisers from the expression. Note that this is only possible because we were able to pull both symmetrisers to the front of the expression allowing us to apply them both first, as if some other symmetriser were applied before these then the structure of the dummy indices would be broken and we could no longer rely on the indices in slots $(4, 5)$ being contracted with slots $(0, 1)$.

Further to the above, we can also see that the symmetriser $\boxed{6}\boxed{7}\boxed{8}\boxed{9}^{T}$ shares no slots with any other symmetriser and can be freely commuted through the expression, in addition to which its indices are not contracted with any other indices in the expression. It therefore has no interaction with any other symmetrisers and will add nothing more than an overall factor of ± 1 to the term, so instead of projecting it and increasing the total number of terms by 4! we simply sort the indices in these positions, multiply the weight of the initial term by -1 if this sorting corresponds to an odd permutation, and then drop the symmetriser from the expression. This reduces the total projector to



(44)

i.e. only the symmetry of the Riemann-like tensor needs to be calculated.

Other optimisations are possible, for instance the cancellation rules given in [19], which apply in particular to the combinations of symmetrisers which result from the construction of a Hermitian projector.

2.8 Complexity

In the literature on tensor canonicalisation, a lot of attention has been paid to the complexity of various algorithms. While that is certainly a worthwhile mathematical aim, the number of

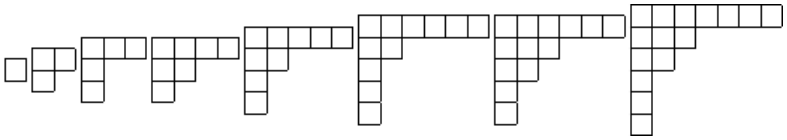


Figure 2: Examples of tableaux which minimise the number of terms in the Young projection for n indices.

practical research problems in which e.g. Riemann tensor monomials of order 10 or higher appear, is limited. Our approach has been that for many situations, it is more useful to have a slow but completely algorithmic solution than a fast solution which only covers a limited number of cases or needs tuning by hand. Nevertheless, we should make some comments about the complexity of our algorithm to put it in context.

The complexity of the algorithm is dependent on two main factors, the number of terms in the expression being acted on and the shape of the tableau associated with each term. The shape of a tableau Y can be described by a list $(r_1, r_2, \ldots, r_n)$ where each r_k is the number of boxes in the ith row of the tableau and thus represents and object with $n = \sum r_k$ indices. Similarly Y can be described by its columns $(c_1, c_2, \ldots, c_{r_1})$ where each c_k can be computed by the number of rows which are at least k cells long

$$c_k = \sum_{i=1}^{n} \Theta(r_i - k), \qquad \Theta(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 1 & \text{if otherwise} \end{cases} . \tag{45}$$

The total number of terms generated by the Young projection operator of Y is given by the product of the numbers of terms generated by the (anti)symmetrization of each row and column

$$N(Y) = \prod_{k=1}^{n} (r_k!) \prod_{k=1}^{r_1} (c_k!) \tag{46}$$

although this does not necessarily equal the total number of terms in the projection, for example in the case of a totally symmetric tensor S_{abab}, 4! operations are required to produce the resulting 2 terms $\frac{1}{2}(S_{abab} + S_{abba})$. Clearly the shape of the tableau plays a crucial role; some examples with a minimal number of terms in the projection are shown in fig. 2.

The complexity of the `meld` algorithm for an input consisting of T terms each of which has an associated tableau Y is of the order $O(TN(Y))$ as the algorithm computes the projection once per term. On top of this is a linear decomposition problem which involves inverting a $K \times K$ matrix where $K \leq T$ and checking a solution against each term in the projection, but as the constant factors for this part of the problem are far smaller we neglect this from the analysis. In fig. 5 the linear dependence on the number of terms is illustrated.

As with complexity analysis on other canonicalisation routines (see e.g. [4], [6]) we look at the performance as the number of indices in the input object increases. As there are many different possible tableaux for each number of indices n we will discuss this for various tableaux shapes. The worst case behaviour arises for fully (anti)symmetric tensors where the symmetry is represented by a single row or column S_n and has complexity $N(S_n) = n!$ (although in practice there are many situations where `meld` can optimise for this situation by simply sorting the indices).

On the other hand a box tableau B_n, such as the Riemann tensor, has $r_k = c_k = \sqrt{n}$ and thus $N(B_n) = \sqrt{n}!^{2\sqrt{n}}$. This difference that the tableau shape makes to the complexity is shown in fig. 6. As one gets closer to fully symmetric or fully anti-symmetric objects, the runtime

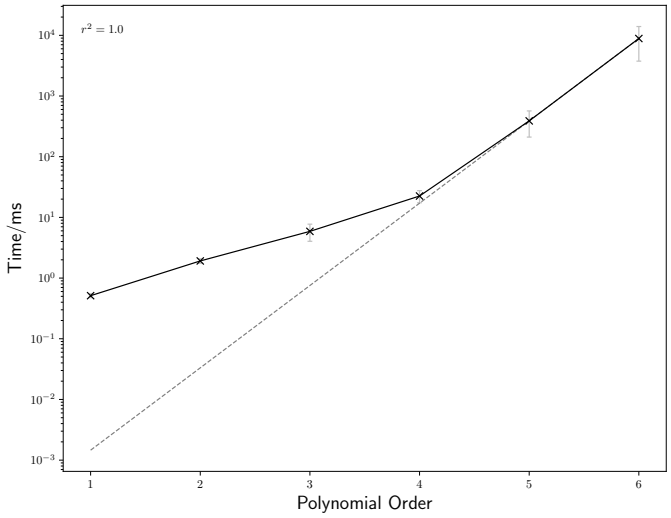


Figure 3: Run-time performance of `meld` as the number of terms in a tensor polynomial increases.

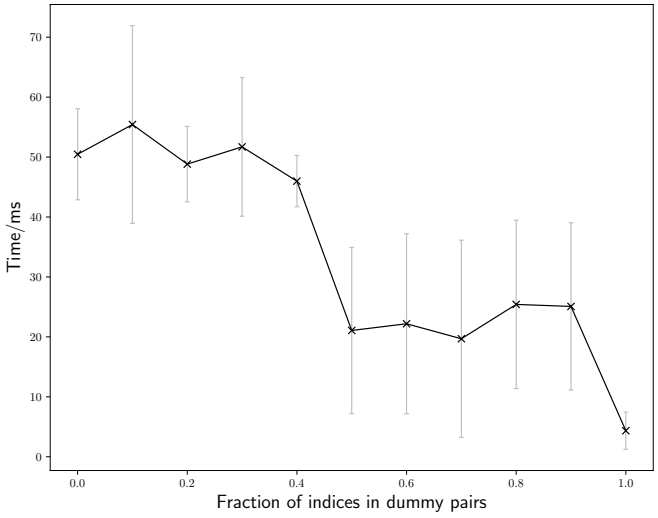


Figure 4: Run-time performance of `meld` as the number of dummy indices relative to free indices increases. Because of the fact that dummy index relabelling leaves the stored expression invariant, performance generically improves when a larger fraction of the indices is dummy.

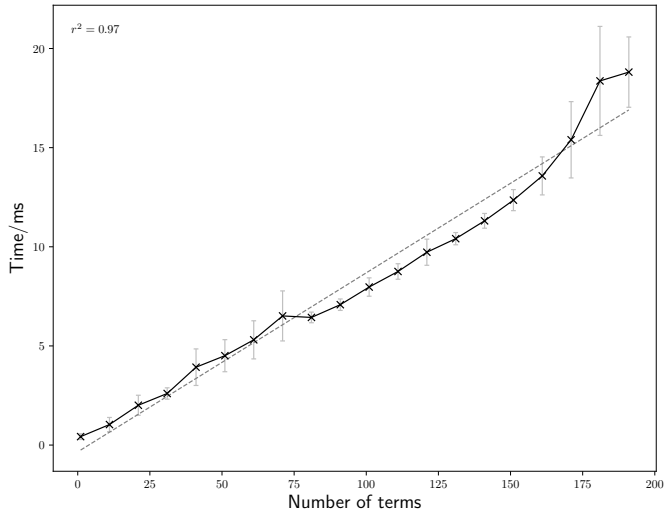


Figure 5: Run-time performance of `meld` as number of terms increases showing the linear relation.

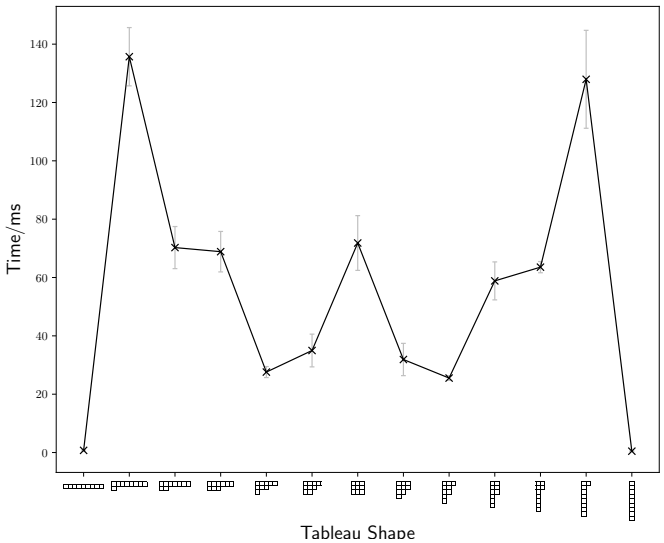


Figure 6: Run-time performance of `meld` as the shape of a tableau with 9 cells changes.

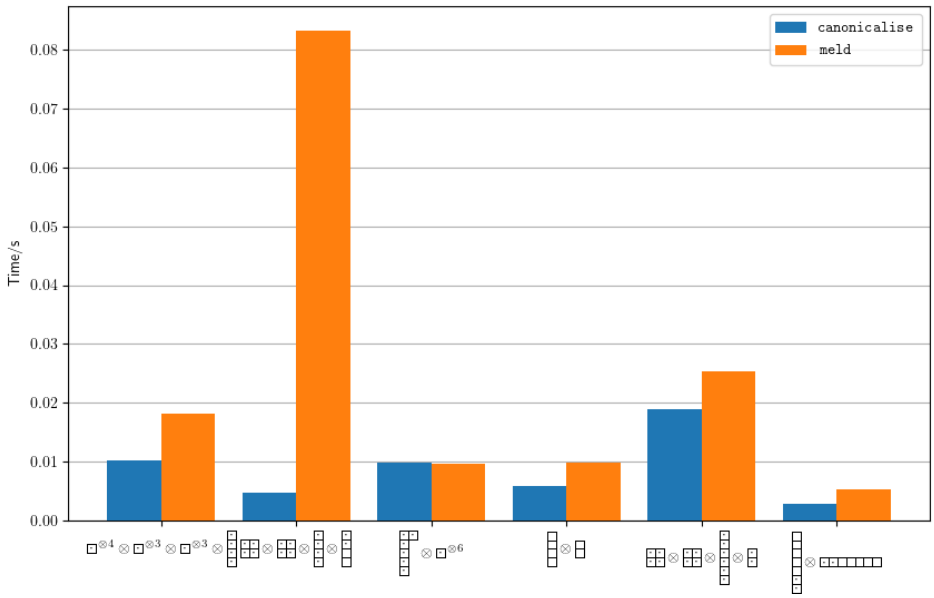


Figure 7: Comparison of `meld` and `canonicalise` against various symmetry types. Boxes containing a mark represent contracted index slots.

increases, and without the optimisation for those extreme cases as discussed in section 2.7.4, the difference between the shortest and longest runtime would increase to more than a factor 200.

We also examine the behaviour of the algorithm for various orders of monomials as this is a common situation in fields such as perturbative general relativity. Fig. 3 shows that scaling of the algorithm against a line representing the scaling $O(a^n)$ for monomial order n. The divergence from this at low orders is in accordance with the constant overhead of the algorithm.

The fourth scaling we present is the behaviour as the fraction of contracted index pairs varies. We briefly mentioned above the example of S_{abab} where the symmetrisation produces only two terms, as opposed to S_{abcd} where 24 terms are produced. The effect this has on the algorithm, presented in fig. 4, is harder to quantify, although in general a higher fraction of contracted index pairs results in a faster running time. Of the many factors which may contribute to this, we draw attention to the following: fewer allocations, more compact objects resulting in fewer cache misses, and the reduction of the number of terms required to iterate over in the calculation of linear dependence.

Obviously, this is not a behaviour which scales well for problems with very large number of indices, but the purpose is to provide a completely general solution for as many problems as possible rather than optimising for niche problems and reducing the overall generality.

Let us finally also present some practical considerations which result from the above discussion by comparing the runtime performance on expressions involving only mono-term symmetries, that is, for situations in which both `meld` and Cadabra's `canonicalise` algorithm (which uses xPerm [5] under the hood) can be used. This of course necessarily limits attention to cases for which `meld` was not primarly designed. The `meld` algorithm, while more brute-force, has the benefit of being relatively simple and thus cache friendly, and its implementation is such

that there is no need to deal with dummy index permutations. The `canonicalise` algorithm is more mathematically elegant, but because of that also more complex, and at least in its current implementation needs to consider a double-coset problem because dummy index relabelling needs to be taken into account. The results are shown in fig. 7 and show that in almost all scenarios the `canonicalise` algorithm is a more efficient approach. We have already mentioned this, but we wish to draw attention to the magnitude of the difference. In most cases the more complex `canonicalise` algorithm outperforms `meld` by a factor of less than two, which for everyday calculations may well be a minor expense to be paid. Even in the cases when `meld` scales very badly, the runtime of `meld` is still less than one-tenth of a second, and in more catastrophic cases (which are omitted from the graph to keep the other cases in scale) the actual runtime may only be a few seconds.

Therefore, while there are certainly occasions in which the use of `meld` is impractical, we find that as a general tool for detecting symmetries it is often efficient enough for real-world situations.

3 Extensions

The `meld` algorithm is, as previously mentioned, an algorithm designed to be a general routine which can detect many different symmetries without the user having to manually identify and apply individual relations. The ability to detect multi-term relations covers a large number of identities but by no means all possibilities. Various extensions which could be considered are presented here.

3.1 Reduction of traces

One such example which has been implemented is the cyclicity of the trace,

$$\mathrm{Tr}(ABC) = \mathrm{Tr}(BCA) = \mathrm{Tr}(CAB)\,, \tag{47}$$

where A, B and C do not necessarily commute. In the `meld` implementation, each term in the trace is split into commuting terms and non-commuting terms. The non-commuting terms are "cycled" until either they match against another term in the sum using a dummy-agnostic comparison, or are cycled back to their initial arrangement.

3.2 Dimension dependent identities

Another group of symmetries which fit into the objective of `meld`, but are not currently implemented, are the dimension-dependent identities which result from anti-symmetrisation of an object with n indices in $d < n$ dimensions,

$$F_{[\alpha_1\alpha_2\ldots\alpha_n]} = 0\,. \tag{48}$$

This is a result of the limited number of ways of labelling the n indices; easily seen with $n = 2, d = 1$ as the only way of labelling $F_{\alpha\beta} - F_{\beta\alpha}$ is $F_{xx} - F_{xx} = 0$. From this, many more identities can be derived [30] such as

$$R^a{}_b R^{bcde} R_{acde} = \frac{1}{4} R R^{abcd} R_{abcd} + 2R^{ac} R^{bd} R_{abcd} + 2R^a{}_b R^b{}_c R^c{}_a - 2R R^a{}_b R^b{}_a + \frac{1}{4} R^3\,, \tag{49}$$

which can be calculated by expanding

$$R^{ab}{}_{[ab} R^{cd}{}_{cd} R^e{}_{e]} = 0\,. \tag{50}$$

There exist general methods for generating such identities [31] which are tedious calculations to perform by hand and are excellent candidates for the types of relations a routine like `meld` should detect. An extension of the `meld` algorithm which deals with these identities will be reported on elsewhere.

3.3 Multi-term side relations

As well as simplifying expressions based on the index symmetries of tensors, another possibility for combining terms is by defining sets of side relations and using these to reduce the size of the set of basis terms. A motivating example for this is the commutator of covariant derivatives in the absence of torsion,

$$[\nabla_\mu, \nabla_\nu]V^\rho = R^\rho{}_{\sigma\mu\nu}V^\sigma \,. \tag{51}$$

In particular we might want to make use of

$$\nabla_\mu\nabla_\nu R^\rho{}_{\alpha\beta\gamma} = \nabla_\nu\nabla_\mu R^\rho{}_{\alpha\beta\gamma} + R^\rho{}_{\sigma\mu\nu}R^\sigma{}_{\alpha\beta\gamma} \,. \tag{52}$$

Given an arbitrary expression containing second order derivatives of Riemann tensors, it is a reasonable goal to use a basis of only two of the terms in this expression. By expressing the relation in a form which makes the side-relations manifest,

$$\begin{aligned}
\nabla_\mu\nabla_\nu R^\rho{}_{\alpha\beta\gamma} &\equiv \frac{1}{2}(\nabla_\mu\nabla_\nu R^\rho{}_{\alpha\beta\gamma} + \nabla_\nu\nabla_\mu R^\rho{}_{\alpha\beta\gamma} + R^\rho{}_{\sigma\mu\nu}R^\sigma{}_{\alpha\beta\gamma}) \,, \\
\nabla_\nu\nabla_\mu R^\rho{}_{\alpha\beta\gamma} &\equiv \frac{1}{2}(\nabla_\mu\nabla_\nu R^\rho{}_{\alpha\beta\gamma} + \nabla_\nu\nabla_\mu R^\rho{}_{\alpha\beta\gamma} - R^\rho{}_{\sigma\mu\nu}R^\sigma{}_{\alpha\beta\gamma}) \,, \\
R^\rho{}_{\sigma\mu\nu}R^\sigma{}_{\alpha\beta\gamma} &\equiv \frac{1}{2}(-\nabla_\mu\nabla_\nu R^\rho{}_{\alpha\beta\gamma} + \nabla_\nu\nabla_\mu R^\rho{}_{\alpha\beta\gamma} + R^\rho{}_{\sigma\mu\nu}R^\sigma{}_{\alpha\beta\gamma}) \,,
\end{aligned} \tag{53}$$

the similarity with the `meld` routine surfaces: each term in a sum is sequentially "projected" using the side-relations and an attempt made to write it as a linear combination of the previously projected terms.

4 Discussion and conclusion

We have argued that canonicalisation is not necessarily the most useful algorithm from the perspective of a computer algebra *user*, and will generally lead to expressions which are unnecessarily large. Instead of converting terms in an expression to canonical form, it may be more useful to meld them together (in the sense discussed in the main text), so that the form of the input expression is preserved as much as possible. We have shown how this can be made to work explitly for expressions involving tensors which exhibit multi-term symmetries, and have briefly commented on extensions to other related situations.

Our implementation uses a memory-efficient and dummy-name agnostic storage method, and shows that many practical computations can be handled in reasonable time. It is, however, not necessarily optimised for speed yet, and could easily be improved by combining it with a mono-term canonicaliser. Our focus has been on general applicability, simplicity of the algorithm, and easy-of-use in practical computations, for which the performance is certainly often good enough, as the examples in section 2.6 show.

References

[1] K. O. Geddes, S. R. Czapor, and G. Labahn, "Algorithms for computer algebra", Springer Science & Business Media, 1992.

[2] L. R. U. Manssur, R. Portugal, and B. F. Svaiter, "Group-theoretic approach for symbolic tensor manipulation", *Int. J. Mod. Phys.* **C13** (2002) 859–880.

[3] L. R. U. Manssur and R. Portugal, "Group-theoretic approach for symbolic tensor manipulation: II. Dummy indices", math-ph/0107032.

[4] L. R. U. Manssur and R. Portugal, "The Canon package: a fast kernel for tensor manipulators", *Comp. Phys. Commun.* **157** (2004) 173–180.

[5] J. M. Martín-García, "xPerm: Fast index canonicalization for tensor computer algebra", *Comp. Phys. Commun.* **179** (2008) 597–603, software available from http://xact.es.

[6] B. E. Niehoff, "Faster Tensor Canonicalization", *Comp. Phys. Commun.* **228** (2018) 123–145, arXiv:1702.08114.

[7] R. Portugal, "Algorithmic simplification of tensor expressions", *J. Phys.* **A32** (1999) 7779–7789.

[8] J. M. Martin-Garcia, R. Portugal, and L. R. U. Manssur, "The Invar Tensor Package", *Comp. Phys. Commun.* **177** (2007) 640–648, arXiv:0704.1756.

[9] R. Portugal, "An algorithm to simplify tensor expressions", *Comp. Phys. Commun.* **115** (1998) 215–230, gr-qc/9803023.

[10] K. Peeters, "Cadabra2: computer algebra for field theory revisited", *J. Open Source Softw.* **3** (2018), no. 32, 1118, software available from https://cadabra.science/.

[11] D. A. Bolotin and S. V. Poslavsky, "Introduction to Redberry: a computer algebra system designed for tensor manipulation", arXiv:1302.1219, software available from http://redberry.cc.

[12] V. A. Ilyin and A. P. Kryukov, "ATENSOR - REDUCE program for tensor simplification", *Comp. Phys. Commun.* **96** (1996) 36–52, arXiv:1811.05409.

[13] A. Balfagón, P. Castellví, and X. Jaén, "Tools of tensor calculus", gr-qc/9809022.

[14] B. Fiedler, "A use of ideal decomposition in the computer algebra of tensor expressions", *Zeitschrift für Analysis und ihre Anwendungen* **16** (1997), no. 1, 145–164.

[15] B. Fiedler, "Ideal decompositions and computation of tensor normal forms", math/0211156.

[16] P. Cvitanović, "Group Theory, Birdtracks, Lie's, and Exceptional Groups", Princeton University Press, 2008.

[17] M. Hamermesh, "Group Theory and its Application to Physical Problems", Cambridge, 2003.

[18] R. McNamara, "Irreducible representations of the symmetric group", *Research Experience for Undergraduates*, 2013.

[19] J. Alcock-Zeilinger and H. Weigert, "Compact Hermitian Young Projection Operators", *Journal of Mathematical Physics* **58** (2016), no. 5,.

[20] S. Keppeler and M. Sjodahl, "Hermitian Young Operators", *Journal of Mathematical Physics* **55** (2014), no. 2, arXiv:1307.6147.

[21] D. Littlewood, "The theory of group characters and matrix representations of groups", American Mathematical Society, 2nd edition ed., 1940.

[22] M. B. Green, K. Peeters, and C. Stahn, "Superfield integrals in high dimensions", *JHEP* **08** (2005) 093, hep-th/0506161.

[23] M. Nakahara, "Geometry, topology and physics", CRC Press, 2003.

[24] K. Peeters, P. Vanhove, and A. Westerberg, "Supersymmetric higher-derivative actions in ten and eleven dimensions, the associated superalgebras and their formulation in superspace", *Class. Quant. Grav.* **18** (2001) 843–889, hep-th/0010167.

[25] S. Frolov and A. A. Tseytlin, "R^4 corrections to conifolds and $G(2)$ holonomy metrics", *Nucl. Phys.* **B632** (2002) 69–100, hep-th/0111128.

[26] J. T. Liu and R. Minasian, "Higher-derivative couplings in string theory: five-point contact terms", *Nucl. Phys. B* **967** (2021) 115386, arXiv:1912.10974.

[27] S. A. Fulling, R. C. King, B. G. Wybourne, and C. J. Cummins, "Normal forms for tensor polynomials. I. The Riemann tensor", *Class. Quant. Grav.* **9** (1992), no. 5, 1151.

[28] B. Fiedler, "On the symmetry classes of the first covariant derivatives of tensor fields", *Seminaire Lotharingien de Combinatoire* **49** (2003) Article B49f.

[29] J. M. Martín-García, D. Yllanes, and R. Portugal, "The Invar tensor package: Differential invariants of Riemann", *Computer Physics Communications* **179** (2008), no. 8, 586–590, arXiv:0802.1274.

[30] X. Dianyan, "Two important invariant identities", *Phys. Rev. D* **35** Jan (1987) 769–770.

[31] S. B. Edgar and A. Höglund, "Dimensionally dependent tensor identities by double antisymmetrization", *Journal of Mathematical Physics* **43** Jan (2002) 659–677, gr-qc/0105066.