Redberry: a computer algebra system designed for tensor manipulation

Stanislav Poslavsky

Institute for High Energy Physics, Protvino, Russia
SRRC RF ITEP of NRC Kurchatov Institute, Moscow, Russia

E-mail: stvlpos@mail.ru

Dmitry Bolotin

Institute of Bioorganic Chemistry of RAS, Moscow, Russia

E-mail: bolotin.dmitriy@gmail.com

Abstract. In this paper we focus on the main aspects of computer-aided calculations with tensors and present a new computer algebra system Redberry which was specifically designed for algebraic tensor manipulation. We touch upon distinctive features of tensor software in comparison with pure scalar systems, discuss the main approaches used to handle tensorial expressions and present the comparison of Redberry performance with other relevant tools.

1. Introduction

General-purpose computer algebra systems (CASs) have become an essential part of many scientific calculations. Focusing on the area of theoretical physics and particularly high energy physics, one can note that there is a wide area of problems that deal with tensors (or more generally — objects with indices). This work is devoted to the algebraic manipulations with abstract indexed expressions which forms a substantial part of computer aided calculations with tensors in this field of science. Today, there are many packages both on top of general-purpose systems (Maple Physics [1], xAct [2], Tensorial etc.) and standalone tools (Cadabra [3, 4], SymPy [5], Reduce [6] etc.) that cover different topics in symbolic tensor calculus. However, it cannot be said that current demand on such a software is fully satisfied [7].

The main difference of tensorial expressions (in comparison with ordinary indexless) lies in the presence of contractions between indices. This additional structure must be reflected in computer representation of tensorial expressions. It makes the existing pure list-based CASs not a good basis for implementing algorithms for tensor manipulation [4]. Because of contractions, implementation of even such a fundamental atomic operation like expression comparison becomes much more complicated, which in turn complicates implementation of general operations such as substitutions, reduction of similar terms etc. Since solution of a real-world problem comprises a long sequence of such common operations, their performance is critical for obtaining of result in a reasonable time.

Here we present Redberry — a computer algebra system for algebraic manipulations with tensorial expressions. A comprehensive documentation accompanied by lots of examples can be

found on the Redberry website http://redberry.cc and in Ref. [8]. This paper gives a very brief overview of features specific for tensorial software, approaches used in the field and describes Redberry in comparison with other systems putting a particular stress on performance.

2. Specific features of tensorial CAS

The presence of indices brings an additional complexity to tensorial expressions. It makes the algorithms used in pure scalar CASs inapplicable and thus requires fundamentally different approach. Let's consider few noteworthy aspects which set tensorial CAS apart from scalar ones.

One of the most frequent operations arising in majority of computer-aided calcualtions with tensors is automatic relabeling of dummy indices. Consider a trivial substitution:

$$R_{\alpha\beta} = R^{\boldsymbol{\mu}}{}_{\alpha\boldsymbol{\mu}\beta} \quad \text{into} \quad R_{\rho\tau}\, R^{\tau\boldsymbol{\mu}}$$

$$R^{\boldsymbol{\mu}}{}_{\rho\boldsymbol{\mu}\tau}\, R^{\boldsymbol{\mu}\rho}{}_{\boldsymbol{\mu}}{}^{\boldsymbol{\mu}} \quad \text{(incorrect ✗)} \quad , \quad R^{\boldsymbol{\alpha}}{}_{\rho\boldsymbol{\alpha}\tau}\, R^{\boldsymbol{\beta}\rho}{}_{\boldsymbol{\beta}}{}^{\boldsymbol{\mu}} \quad \text{(correct ✓)}$$

One can see that substitution of r.h.s. without appropriate relabeling of dummy index μ leads to an indices clash. Thus correctness of CAS operations depends on its ability to automatically resolve such conflicts (as shown).

Another nontrivial atomic operation is comparison or matching of tensorial expressions (this operation is straightforward for scalar expressions). Consider a common simplification routine — reduction of similar terms:

$$(1) \quad A_\mu\, A^\mu + A_\nu\, A^\nu = 2\, A_\mu\, A^\mu \qquad (2) \quad F_{abcd}\, F^{acef}\, F_{ef}{}^{bd} - F_{bfed}\, F^{cafd}\, F^{be}{}_{ca} = 0$$

In both cases (and second one is really non trivial) CAS has to match dummy indices in order to check that the summands are equal. The problem of comparison of indexed expressions is very similar to a well known *pattern matching problem* from scalar CASs, while here it arises even in simplest calculations.

Additional aspect of comparison emerges in substitutions where both free and dummy indices must be matched. Consider for example a substitution:

$$F^{\alpha\beta}\, F_{\alpha\gamma} = T^\alpha{}_\gamma \quad \to \quad F^{\beta\gamma}\, F^\mu{}_\gamma\, F_{\beta\mu}, \quad \text{which gives} \quad T^\gamma{}_\mu\, F^\mu{}_\gamma.$$

The ability to define arbitrary permutational symmetries and antisymmetries of tensors is required for nearly all calculations. The presence of symmetries significantly complicates issues mentioned above. For example, supposing that $R_{abcd} = R_{cdab} = -R_{bacd}$ one can check that

$$R^{abcd}\, R_{efdc}\, R^{ef}{}_{ab} + R_{rc}{}^{df}\, R_{ab}{}^{rc}\, R_{fd}{}^{ba} = 0.$$

Checking of this equality requires both matching of indices and considering information about symmetries. Here is a more complicated example (suppose $W_{abcde} = W_{ebcda} = W_{cbade}$):

$$\begin{aligned}(W_{bde}{}^{ij} + W_{bde}{}^{ji} + W_{bed}{}^{ij} + W_{dbe}{}^{ji} + W_{de}{}^{i}{}_{b}{}^{j})(W_{cfhji} + W_{chfji} + W_{cjhfi} + W_{fchij} + W_{fchji}) - \\ (W_{bde}{}^{ij} + W_{bde}{}^{ji} + W_{bed}{}^{ji} + W_{dbe}{}^{ij} + W_{de}{}^{i}{}_{b}{}^{j})(W_{cfhij} + W_{chfij} + W_{cihfj} + W_{fchij} + W_{fchji}) = 0\end{aligned} \tag{3}$$

This identity can be proved by using information about contractions of indices and symmetries of tensor W_{abide} without expansion of brackets.

3. A brief overview of Redberry

Today there are a lot of packages written both on top of general CASs and standalone systems that cover different topics in computer computations with tensors. The most relevant packages that are actively supported are `xAct` [2] (open-source, written on top of Mathematica), `Maple Physics` [1] (commercial, not open-source) and `Cadabra` [3] (open-source, standalone). These packages well cover most of the features described above.

On the other hand, many modern calculations in physics involve extremely huge expressions, thus performance of CAS becomes critical. This aspect is not well addressed in the existing software. Redberry is aimed at solution of large scale problems with millions of tensorial terms in a reasonable time. For example, calculation of one-loop counterterms of gravitational field[1] requires processing of about 700 000 tensorial terms with about 8 multipliers and about 10 indices per term; Redberry solves this problem in less than 8 minutes (on a standard laptop).

Another example that one can easily check using mentioned packages is the identity (3). Time used by different systems to prove it is the following: Redberry — 2 ms, Cadabra — 180 ms, xAct — 180 ms, Maple Physics — 200 ms. In this example Redberry shows nearly 100 fold higher performance. Moreover, Redberry does not perform expansion of brackets while other systems do.

Along with high performance Redberry provides a comprehensive set of both general-purpose and HEP-specific tools for tensor handling which include: native dummy indices handling (particularly automatic resolving of clashes), support of arbitrary permutational symmetries, extensive tools for simplification of tensorial expressions, multiple index types, LaTeX-style input/output and a wide range of other tensor-specific transformations. Out of the box Redberry provides instruments for computations in high energy physics including tools for Feynman diagrams calculation (Dirac & SU(N) traces, simplification of Levi-Civita tensors etc.) and for calculation of one-loop counterterms in quantum field theory in curved space-time.

Redberry allows to use general purpose programming language equipped with a dozen of computer algebra and particularly tensor-specific syntax constructions which facilitates common usage as well as implementation of custom functionality for particular problems.

In order to demonstrate Redberry syntax let us consider a "toy" example. The following example proves that $\left(2\,R_{abcd}\,R^{acbd} - R_{abcd}\,R^{abcd}\right) = 0$ where R_{abcd} is a Riemann tensor by using the identity $R_{abcd} = \left(2\,R_{abcd} + R_{acbd} - R_{adbc}\right)/3$:

```
1 // Setup symmetries
2 addSymmetries 'R_abcd', [[0, 2], [1, 3]].p, -[[1, 0]].p
3 // Define substitution
4 subs = 'R_abcd = (2*R_abcd + R_acbd - R_adbc)/3'.t
5 // Input expression
6 expr = '2*R_abcd*R^acbd - R_pqrs*R^pqrs'.t
7 // Apply substitution and expand
8 expr = (subs & Expand) >> expr
9 println expr //prints zero
```

```
▷ 0
```

In line 2 we define symmetries by providing two generators of corresponding permutation group written in disjoint cycle notation (`.p` converts list of integers into a `Permutation` object). In line 4 and 6 we input substitution and target expression (`.t` converts string into a computer object). Finally in line 8 we apply substitution and expand out brackets. The result is zero since Redberry automatically reduces similar terms taking into account all symmetries.

[1] for details on one-loop calculations in Redberry see [8] and online documentation at http://redberry.cc

One can find lots of examples (including physical examples such as Feynman diagrams and one-loop counterterms calculation) along with comprehensive documentation on the Redberry website (http://redberry.cc).

4. Algorithms

Since the most frequently invoked atomic operation during any calculation is comparison of expressions, its performance makes the main contribution to the overall performance of CAS. This operation arises in such common simplification as reduction of similar terms. The algorithms of tensor comparison (or matching) is the main topic in the field of tensorial CAS.

The existing systems (known to the authors) are based on the so-called *indices canonicalization approach* [9, 10]. This procedure brings indices of products into unique canonical order using the information about symmetries of its multipliers; when indices of tensors are brought into canonical form, it becomes trivial to test whether two expressions are equal. As it is shown in [9, 10], this problem is equivalent to the problem of *double coset enumeration* which is known to be $\mathcal{NP}$-hard (see Sec. 4.6.8 in [11]). Unfortunately, no satisfactory algorithm to solve this problem has been found to date (see Sec. 4.6.8 in [11]). Importantly, indices canonicalization works only with products of simple tensors, e.g. if product contains a sum one need to expand out brackets first.

Redberry uses another approach to the problem which is based on the idea that contractions between indices in product constitutes a graph (see for example Fig. 1). From such point of view the problem of testing whether two expressions are equal is equivalent to problem of testing whether corresponding graphs are isomorphic (this approach was discussed in [9], but was not implemented in any CAS). While no worst-case polynomial-time algorithms are known for the general Graph Isomorphism (GI) problem, there are several algorithms which solve it very effectively in all practical cases (see e.g. [12] and overview given in [13]). The GI algorithm implemented in Redberry is specifically optimized for typical problems arising in physics (i.e. when one needs to compare huge number of small graphs; see below).

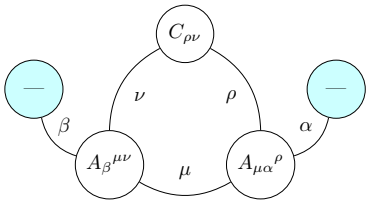

Figure 1: Graph representation of tensor $A_\beta{}^{\mu\nu}\, C_{\rho\nu}\, A_{\mu\alpha}{}^{\rho}$

In order to find out how Redberry performance compares to existing approaches we carried out performance evaluation of Redberry, Cadabra (which is open source and uses the same canonicalisation engine as Mathematica xAct) and Maple Physics (commercial, implementation is unknown). We performed two types of benchmarks.

In our first benchmark we measured performance of comparison operation. For this purpose, we randomly generated sums of products of tensors[2] and measured time needed by each system to obtain zero when subtracting it from itself rewritten in equivalent form by renaming dummies, rearranging terms and shuffling indices according to symmetries. For example:

$$\left(F_{\mu\nu\alpha}T^{\nu}{}_{\beta}W^{\alpha\beta} + \ldots \textit{(99 terms)}\right) - \left(T^{\nu}{}_{\beta}T^{\alpha}{}_{\rho}W^{\nu\rho}F_{\mu\alpha\nu} + \ldots \textit{(99 terms)}\right) = 0$$

The results of our benchmarks are shown on Fig. 2. We analysed cases without and with symmetries (symmetric or antisymmetric with respect to all indices of particular simple tensor; in case with symmetries Maple failed to obtain zero on majority of examples, so it was excluded from the benchmark). We found that in case without symmetries, both Redberry and Maple have polynomial dependence on size of products (Fig. 2 left), while Cadabra has exponential; in case with symmetries, both Cadabra and Redberry have exponential dependence in worst case, but Redberry has better performance on larger problems. The dependence on the number of

[2] all expressions used for benchmarks can be found at http://redberry.cc/documentation:benchmarks

indices (Fig. 2 right) in case without symmetries is also polynomial for Redberry and Maple and exponential for Cadabra; on the other hand, the presence of symmetries almost does not change the behaviour of Cadabra, while Redberry fails to simplify large expressions in a reasonable time.

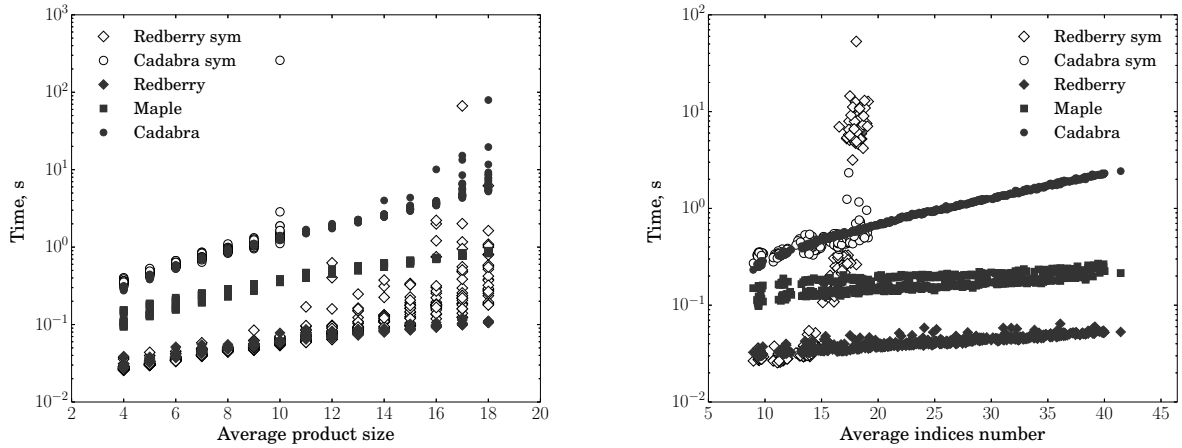

Figure 2: **Dependence of time spent in collect routine on product size (left) and on average number of indices in products (right)**. For these benchmarks we generated sums (100 summands) of products of simple tensors, then subtracted it from itself (in equivalently rewritten form: shuffled summands and multipliers, renamed dummy indices and interchanged indices of simple tensors according to their symmetries) and measured time needed to obtain zero. For Redberry and Cadabra we performed benchmarks both with (marked as "sym" in the legend) and without symmetries of simple tensors, while Maple was unable to simplify such expressions in case with symmetries.

In the second benchmark we measured performance of the most common simplification procedure — expand out product of sums and reduce similar terms (*expand and collect*). For this benchmark we used sums of products of sums of products[2]. We applied *expand and collect*, subtracted result from the initial expression and measured time needed by each system to obtain zero. For example:

$$\Big(F_{\mu\nu}\Big(T^{\mu\alpha}R^{\nu\beta}\ +\ldots\Big)+\ldots\Big)\Big(R_{\alpha\rho}\Big(F^{\rho}{}_{\beta}R_{\tau\gamma}\ \ +\ldots\Big)+\ldots\Big)\times\Big(\ldots\ \Big)$$
$$-\Big(F_{\nu\mu}T^{\nu\beta}R^{\mu\alpha}R_{\beta\rho}F^{\rho}{}_{\alpha}R_{\tau\gamma}\ +\ldots\Big)\ =\ 0$$

Fig. 3 shows the comparison of time spent in *expand and collect* operation for the same input expressions for Redberry vs. Maple and for Redberry vs. Cadabra. On average in these benchmarks Redberry is 38 times faster than Maple and 62 times faster than Cadabra.

5. Conclusions

In this paper we considered the main features of algebraic computer-aided manipulations with tensors. We discussed two main approaches used for implementation of symbolic tensor algebra and presented a new computer algebra system Redberry based on the alternative to common approach for tensor handling. We demonstrated that Redberry outperforms other tools (Maple Physics and Cadabra) in typical operations on large tensorial expressions. Further information about Redberry including comprehensive documentation accompanied by lots of examples can be found on the Redberry website http://redberry.cc and in Ref. [8]

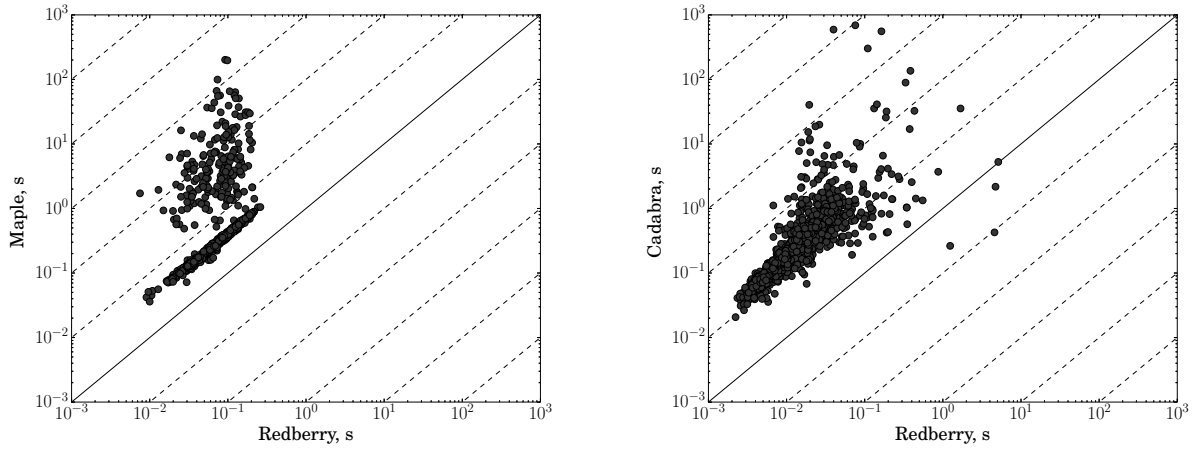

Figure 3: **Comparison of time spent in expand and collect routine for randomly generated expressions.** Each input problem plotted as filled circle. Each plot represents comparison of execution times needed to simplify input expression to zero between two systems (left for Maple versus Redberry, right for Cadabra versus Redberry). Solid line corresponds to equal execution times, dashed lines shows 10x, 100x, 1000x, etc. execution time ratio. For benchmark presented here, Maple failed to obtain zero for more then 16% of input expressions, this points are not presented on the left plot. On average in these benchmarks Redberry is 38 times faster than Maple and 62 times faster then Cadabra.

Acknowledgements

The authors would like to thank A. Kataev and V. Ilyin for valuable and stimulating discussions. The work of S.P. was supported by the RFBR (grant #14-02-00096 A), grant of SAEC "Rosatom" and Helmholtz Association. The work of D.B. was financially supported by RSCF (grant #14-14-00533).

References

[1] Maple Inc *Maple Physics homepage: http://www.maplesoft.com*

[2] J Martin-Garcia *xAct homepage: http://www.xact.es/*

[3] Peeters K *arXiv:hep-th/0701238*

[4] Peeters K 2007 *Comput.Phys.Commun.* **176** 550–558 arXiv:cs/0608005 [cs.SC]

[5] SymPy Development Team *SymPy homepage http://www.sympy.org*

[6] Anthony Hearn *REDUCE homepage: http://www.reduce-algebra.com*

[7] Korol'kova A, Kulyabov D and Sevast'yanov L 2013 *Programming and Computer Software* **39** 135–142 ISSN 0361-7688

[8] Bolotin D A and Poslavsky S V *arXiv:1302.1219 [cs.SC]*

[9] Rodionov A and Taranov A 1989 *Eurocal '87* (*Lecture Notes in Computer Science* vol 378) ed Davenport J (Springer Berlin Heidelberg) pp 192–201 ISBN 978-3-540-51517-3 URL `http://dx.doi.org/10.1007/3-540-51517-8_113`

[10] Manssur L R U, Portugal R and Svaiter B F 2002 *International Journal of Modern Physics C* **13** 859–879

[11] Holt D, Eick B and O'Brien E 2005 *Handbook of Computational Group Theory* Discrete Mathematics and Its Applications (Taylor & Francis) ISBN 9781420035216

[12] McKay B D 1981 *Congressus Numerantium* **30** 4587 10th. Manitoba Conference on Numerical Mathematics and Computing (Winnipeg, 1980)

[13] McKay B D and Piperno A *arXiv:1301.1493 [cs.DM]*