Redberry: a computer algebra system designed for tensor manipulation

Stanislav Poslavsky

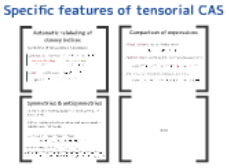
Institute for High Energy Physics,
Protvino, Russia

SRRC RF ITEP of NRC Kurchtov Institutes,
Moscow, Russia

Dmitry Bolotin

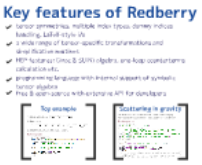
Institute of Bioorganic Chemistry of RAS,
Moscow, Russia

Outline

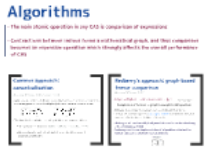


Specific features of tensorial CAS

Redberry's place among tensor software



Key features of Redberry & examples



Algorithms & performance

Specific features of tensorial CAS

Automatic relabeling of dummy indices

is critical for all CAS operations. Few examples:

Substitutions: substitute $R_{\alpha\beta} = R^{\mu}{}_{\alpha\mu\beta}$ into $R_{\rho\tau}R^{\tau\mu}$

without relabeling: $R^{\mu}{}_{\rho\mu\tau}R^{\mu\tau}{}_{\mu}{}^{\mu}$

the correct way: $R^{\alpha}{}_{\rho\alpha\tau}R^{\beta\tau}{}_{\beta}{}^{\mu}$

Expand: expand power $(p_\mu p^\mu + m^2)^2$

- gives: $p_\mu p^\mu p_\alpha p^\alpha + 2\,m^2\,p_\mu p^\mu + m^4$

Comparison of expressions

- Simplifications: reduce similar terms

$$A_\mu A^\mu + A_\nu A^\nu = 2\,A_\mu A^\mu$$

- Substitutions: matching both free and dummy indices

apply $F^{\alpha\beta}F_{\alpha\gamma} \to T^{\beta}{}_{\gamma}$ to $F^{\beta\gamma}F^{\mu}{}_{\gamma}F_{\beta\mu}$ $\Rightarrow$ $T^{\gamma}{}_{\mu}F^{\mu}{}_{\gamma}$

- Functions: matching both arguments and indices

apply $f_{\mu\nu}(x_{\alpha\beta}) = x_\mu{}^\alpha x_{\alpha\nu}$ to $f_{\alpha\beta}(y_\alpha y_\beta)$ $\Rightarrow$ $y_\alpha y_\mu y^\mu y_\beta$

Symmetries & antisymmetries

- Ability to define arbitrary symmetries is a key feature of tensorial CAS
- Both symmetries and antisymmetries must be considered in matching and simplification

For example, if $R_{abcd} = R_{cdba} = -R_{bacd}$

$$\Rightarrow\ R^{abcd}R_{cfdc}R^{ef}{}_{ab} + R_{rc}{}^{df}R_{ab}{}^{rc}R_{fd}{}^{ba} = 0$$

Another example: if $W_{abcde} = W_{ebcda} = W_{cbade}$

$$\Rightarrow (W_{bdc}{}^{ij}+W_{bdc}{}^{ji}+W_{bcd}{}^{ij}+W_{dbc}{}^{ji}+W_{dc}{}^{i}{}_{b}{}^{j})(W_{cfhji}+W_{chfji}+W_{cjhfi}+W_{fchij}+W_{fchji}) -$$
$$(W_{bdc}{}^{ij}+W_{bdc}{}^{ji}+W_{bcd}{}^{ji}+W_{dbc}{}^{ij}+W_{dc}{}^{i}{}_{b}{}^{j})(W_{cfhij}+W_{chfij}+W_{cihfj}+W_{fchij}+W_{fchji}) = 0$$

...

Automatic relabeling of dummy indices

is critical for all CAS operations. Few examples:

Substitutions: substitute $R_{\alpha\beta} = R^{\mu}{}_{\alpha\mu\beta}$ into $R_{\rho\tau}R^{\tau\mu}$

without relabeling: $R^{\mu}{}_{\rho\mu\tau}R^{\mu\tau}{}_{\mu}{}^{\mu}$

the correct way: $R^{\alpha}{}_{\rho\alpha\tau}R^{\beta\tau}{}_{\beta}{}^{\mu}$

Expand: expand power $(p_\mu p^\mu + m^2)^2$

- gives: $p_\mu p^\mu p_\alpha p^\alpha + 2\,m^2\,p_\mu p^\mu + m^4$

Comparison of expressions

- Simplifications: reduce similar terms

$$A_\mu A^\mu + A_\nu A^\nu = 2\, A_\mu A^\mu$$

- Substitutions: matching both free and dummy indices

apply $F^{\alpha\beta}\, F_{\alpha\gamma} \to T^{\beta}{}_{\gamma}$ to $F^{\beta\gamma}\, F^{\mu}{}_{\gamma}\, F_{\beta\mu}$ $\Rightarrow$ $T^{\gamma}{}_{\mu}\, F^{\mu}{}_{\gamma}$

- Functions: matching both arguments and indices

apply $f_{\mu\nu}(x_{\alpha\beta}) = x_{\mu}{}^{\alpha} x_{\alpha\nu}$ to $f_{\alpha\beta}(y_\alpha y_\beta)$ $\Rightarrow$ $y_\alpha y_\mu y^\mu y_\beta$

Symmetries & antisymmetries

- Ability to define arbitrary symmetries is a key feature of tensorial CAS

- Both symmetries and antisymmetries must be considered in matching and simplification

For example, if $R_{abcd} = R_{cdba} = -R_{bacd}$

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

Another example: if $W_{abcde} = W_{ebcda} = W_{cbade}$

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

Specific features of tensorial CAS

Automatic relabeling of dummy indices

is critical for all CAS operations. Few examples:

Substitutions: substitute $R_{\alpha\beta} = R^{\mu}{}_{\alpha\mu\beta}$ into $R_{\rho\tau}R^{\tau\mu}$

without relabeling: $R^{\mu}{}_{\rho\mu\tau}R^{\mu\tau}{}_{\mu}{}^{\mu}$

the correct way: $R^{\alpha}{}_{\rho\alpha\tau}R^{\beta\tau}{}_{\beta}{}^{\mu}$

Expand: expand power $(p_\mu p^\mu + m^2)^2$

- gives: $p_\mu p^\mu p_\alpha p^\alpha + 2\,m^2\,p_\mu p^\mu + m^4$

Comparison of expressions

- Simplifications: reduce similar terms

$$A_\mu A^\mu + A_\nu A^\nu = 2\,A_\mu A^\mu$$

- Substitutions: matching both free and dummy indices

apply $F^{\alpha\beta}F_{\alpha\gamma} \to T^{\beta}{}_{\gamma}$ to $F^{\beta\gamma}F^{\mu}{}_{\gamma}F_{\beta\mu}$ $\Rightarrow$ $T^{\gamma}{}_{\mu}F^{\mu}{}_{\gamma}$

- Functions: matching both arguments and indices

apply $f_{\mu\nu}(x_{\alpha\beta}) = x_\mu{}^\alpha x_{\alpha\nu}$ to $f_{\alpha\beta}(y_\alpha y_\beta)$ $\Rightarrow$ $y_\alpha y_\mu y^\mu y_\beta$

Symmetries & antisymmetries

- Ability to define arbitrary symmetries is a key feature of tensorial CAS
- Both symmetries and antisymmetries must be considered in matching and simplification

For example, if $R_{abcd} = R_{cdba} = -R_{bacd}$

$$\Rightarrow\ R^{abcd}R_{cfdc}R^{ef}{}_{ab} + R_{rc}{}^{df}R_{ab}{}^{rc}R_{fd}{}^{ba} = 0$$

Another example: if $W_{abcde} = W_{ebcda} = W_{cbade}$

$$\Rightarrow (W_{bdc}{}^{ij}+W_{bdc}{}^{ji}+W_{bcd}{}^{ij}+W_{dbc}{}^{ji}+W_{dc}{}^{i}{}_{b}{}^{j})(W_{cfhji}+W_{chfji}+W_{cjhfi}+W_{fchij}+W_{fchji}) -$$
$$(W_{bdc}{}^{ij}+W_{bdc}{}^{ji}+W_{bcd}{}^{ji}+W_{dbc}{}^{ij}+W_{dc}{}^{i}{}_{b}{}^{j})(W_{cfhij}+W_{chfij}+W_{cihfj}+W_{fchij}+W_{fchji}) = 0$$

...

Redberry's place among tensor software

- Mathematica packages: xAct, Tensorial, Ricci, etc.
- Maple packages: Maple Physics, GRTensorII, etc.
- Standalone tools: Cadabra, Reduce etc.

Non free

- **Redberry is aimed at solution of large scale problems with millions of tensorial terms in a reasonable time**

E. g. calculation of one-loop counterterms of gravitational field requires to process ~ 700 000 tensorial terms with ~ 8 multipliers and ~ 10 indices per term takes in Redberry less than 8 minutes (on this Mac Book Air)

- **The performance is main objective**

Example: assuming that $W_{abcde} = W_{ebcda} = W_{cbade}$ check

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

Timing:

Redberry	Cadabra	xAct	Maple Physics
2ms	180ms	180ms	200ms

Key features of Redberry

- ✔ tensor symmetries, multiple index types, dummy indices handling, LaTeX-style i/o
- ✔ a wide range of tensor-specific transformations and simplification routines
- ✔ HEP features: Dirac & SU(N) algebra, one-loop counterterms calculation etc.
- ✔ programming language with internal support of symbolic tensor algebra
- ✔ free & open-source with extensive API for developers

Toy example

1. Setup symmetries $R_{abcd} = R_{cdba} = -R_{bacd}$
2. Substitute

$$R_{abcd} = (2R_{abcd} + R_{acbd} - R_{adbc})/3 \quad \text{in} \quad 2R_{abcd}R^{acbd} - R_{abcd}R^{abcd}$$

3. Simplify and check that result is zero

Redberry code:

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

Scattering in gravity

Calculate matrix element of scalar particle scattering by exchanging one massive Fierz-Pauli graviton in D dimensions

Redberry code:

```
// Auxiliary tensor
P = 'P_ab[p_a] = g_ab + p_a*p_b/M**2'.t
// Fierz-Pauli graviton propagator
D = '''D_{mn ab}[p_a] = ((P_ma[p_a] * P_nb[p_a] + P_mb[p_a] * P_na[p_a])/2
        - P_mn[p_a] * P_ab[p_a]/(D-1))/(p_a*p^a + M**2)'''.t
// Scalar-graviton vertex
V = 'V_{mn}[p_a, k_a] = p_m*k_n + p_n*k_m - (1/2)*g_mn*(p_a - k_a)*(p^a - k^a)'.t
// Matrix element
M = 'V_{ab}[p1_a, k1_a]*D^{ab cd}[p1_a - k1_a]*V_cd[p2_a, k2_a]'.t
// Apply substitutions
M = (D & V & P) >> M
// Set up Mandelstam variables
mShell = setMandelstam([p1_a: 'm', p2_a: 'm', k1_a: 'm', k2_a: 'm'])

// Simplifications of matrix element
M = (ExpandAll & EliminateMetrics & 'd^i_i = D'.t) >> M
M = (mShell & 'u = 4*m**2 - s - t'.t & Factor) >> M
println M
```

$$> \frac{1}{4(D-1)M^4(M^2+t)}\left(16(D-2)m^4M^4 - 8m^2\left(M^4(2(D-1)s+(D-4)t) - 2M^2t^2\right)\right.$$
$$\left. + M^4\left(4(D-1)s^2+4(D-1)st+(D-10)t^2\right) + 4(D-4)M^2t^3 + 4(D-2)t^4\right)$$

Toy example

1. Setup symmetries $R_{abcd} = R_{cdba} = -R_{bacd}$

2. Substitute

$$R_{abcd} = (2R_{abcd} + R_{acbd} - R_{adbc})/3 \quad \text{in} \quad 2R_{abcd}R^{acbd} - R_{abcd}R^{abcd}$$

3. Simplify and check that result is zero

Redberry code:

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

Scattering in gravity

Calculate matrix element of scalar particle scattering by exchanging one massive Fierz-Pauli graviton in D dimensions

Redberry code:

```
// Auxiliary tensor
P = 'P_ab[p_a] = g_ab + p_a*p_b/M**2'.t
// Fierz-Pauli graviton propagator
D = '''D_{mn ab}[p_a] = ((P_ma[p_a] * P_nb[p_a] + P_mb[p_a] * P_na[p_a])/2
            - P_mn[p_a] * P_ab[p_a]/(D-1))/(p_a*p^a + M**2)'''.t
// Scalar-graviton vertex
V = 'V_{mn}[p_a, k_a] = p_m*k_n + p_n*k_m - (1/2)*g_mn*(p_a - k_a)*(p^a - k^a)'.t
// Matrix element
M = 'V_{ab}[p1_a, k1_a]*D^{ab cd}[p1_a - k1_a]*V_cd[p2_a, k2_a]'.t
// Apply substitutions
M = (D & V & P) >> M
// Set up Mandelstam variables
mShell = setMandelstam([p1_a: 'm', p2_a: 'm', k1_a: 'm', k2_a: 'm'])

// Simplifications of matrix element
M = (ExpandAll & EliminateMetrics & 'd^i_i = D'.t) >> M
M = (mShell & 'u = 4*m**2 - s - t'.t & Factor) >> M
println M
```

$$> \frac{1}{4(D-1)M^4\left(M^2+t\right)}\left(16(D-2)m^4M^4-8m^2\left(M^4(2(D-1)s+(D-4)t)-2M^2t^2\right)\right.$$
$$\left.+M^4\left(4(D-1)s^2+4(D-1)st+(D-10)t^2\right)+4(D-4)M^2t^3+4(D-2)t^4\right)$$

Algorithms

- The main atomic operation in any CAS is comparison of expressions
- Contractions between indices forms a mathematical graph, and thus comparison becomes an expensive operation which strongly affects the overall performance of CAS

Common approach: canonicalisation

(Rodionov & Taranov 1989, Portugal 1999)

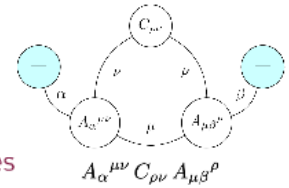
- Main idea: put indices of all expressions into "canonical" form (rename dummies, sort using symmetries, sort multipliers), then comparison becomes trivial:

$$R^{abcd}R_{efdc}R^{ef}{}_{ab} \quad \to \quad -R^{ab}{}_{cd}R_{abef}R^{efcd}$$
$$R_{rc}{}^{df}R_{ab}{}^{rc}R_{fd}{}^{ba} \quad \to \quad R^{ab}{}_{cd}R_{abef}R^{efcd}$$

- The algorithm is equivalent to finding double coset representatives
  - × NP-hard problem in general (Butler 1984, Luks 1993, Holt 2005)
  - × Works only with products of simple tensors (need to expand if product contains sum)

Redberry's approach: graph-based tensor comparison

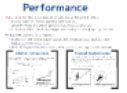
(Rodionov & Taranov 1989)



Graph: multipliers - vertexes, dummies - edges

Comparison of tensors = graph isomorphism (GI) problem

- Redberry does not perform expensive "canonicalisation" of terms
- It searches for isomorphisms (mappings of indices) between tensors and does not rely on canonical form of expressions
- **Althought, GI is at least NP, in all practical cases it can be solved very efficiently (McKay, 1980)**
- **Redberry uses its own implementation of GI problem optimized for typical expressions arising in real calculations**



Common approach: canonicalisation

(Rodionov & Taranov 1989, Portugal 1999)

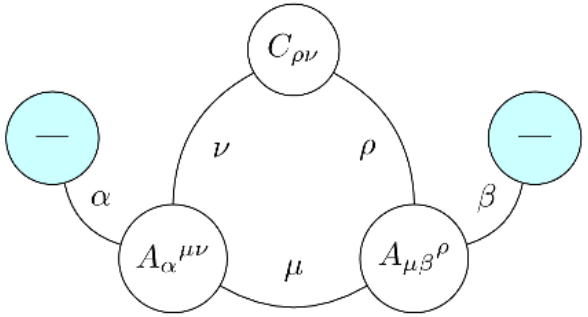
- Main idea: put indices of all expressions into "canonical" form (rename dummies, sort using symmetries, sort multipliers), then comparison becomes trivial:

$$R^{abcd}R_{efdc}R^{ef}{}_{ab} \quad \rightarrow \quad -R^{ab}{}_{cd}R_{abef}R^{efcd}$$
$$R_{rc}{}^{df}R_{ab}{}^{rc}R_{fd}{}^{ba} \quad \rightarrow \quad R^{ab}{}_{cd}R_{abef}R^{efcd}$$

- The algorithm is equivalent to finding double coset representatives
  - ✖ NP-hard problem in general (Butler 1984, Luks 1993, Holt 2005)
  - ✖ Works only with products of simple tensors (need to expand if product contains sum)

Redberry's approach: graph-based tensor comparison

(Rodionov & Taranov 1989)

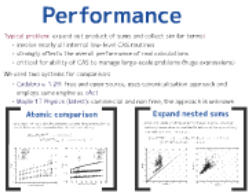


Graph: multipliers - vertexes, dummies - edges

Comparison of tensors = graph isomorphism (GI) problem

- Redberry does not perform expensive "canonicalisation" of terms
- It searches for isomorphisms (mappings of indices) between tensors and does not rely on canonical form of expressions

- **Althought, GI is at least NP, in all practical cases it can be solved very efficiently (McKay, 1980)**
- **Redberry uses its own implementation of GI problem optimized for typical expressions arising in real calculations**



Performance

Typical problem: expand out product of sums and collect similar terms:

- involve nearly all internal low-level CAS routines
- strongly affects the overall performance of real calculations
- critical for ability of CAS to manage large-scale problems (huge expressions)

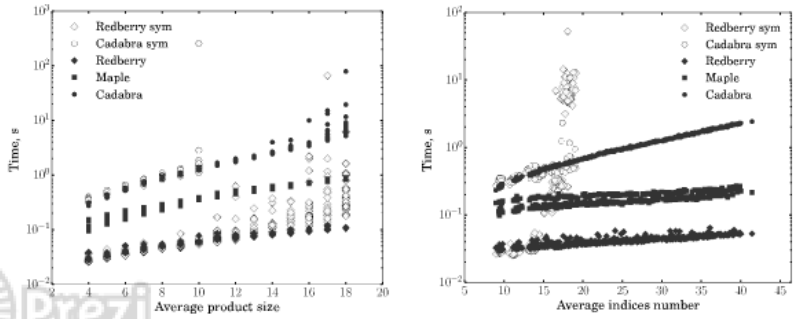
We used two systems for comparison:

- Cadabra v. 1.29: free and open-source, uses canonicalisation approach and employs same engine as xAct
- Maple 17 Physics (latest): commercial and non free, the approach is unknown

Atomic comparison

- Generate sum of 100 random different terms, rearrange indices (rename dummies and shuffle indices within symmetries) and subtract from itself, e. g. :

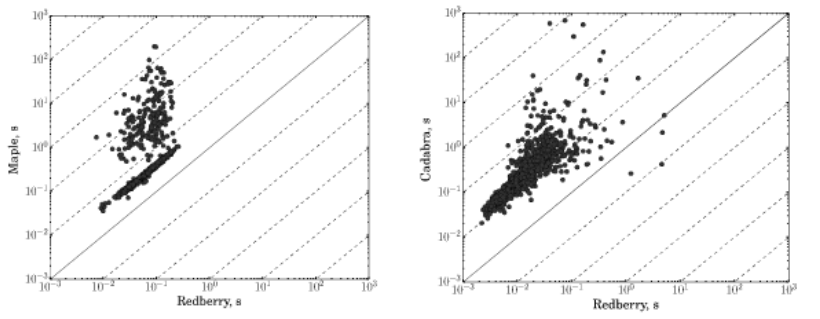
$$\left(F_{\mu\nu\alpha}T^{\nu}{}_{\beta}W^{\alpha\beta} + \ldots \textit{(99 terms)}\right) - \left(T^{\alpha}{}_{\rho}W^{\nu\rho}F_{\mu\alpha\nu} + \ldots \textit{(99 terms)}\right) = 0$$



Expand nested sums

- Generate random nested products of sums, expand, rearrange indices (rename dummies and shuffle indices within symmetries), and subtract from itself, e. g. :

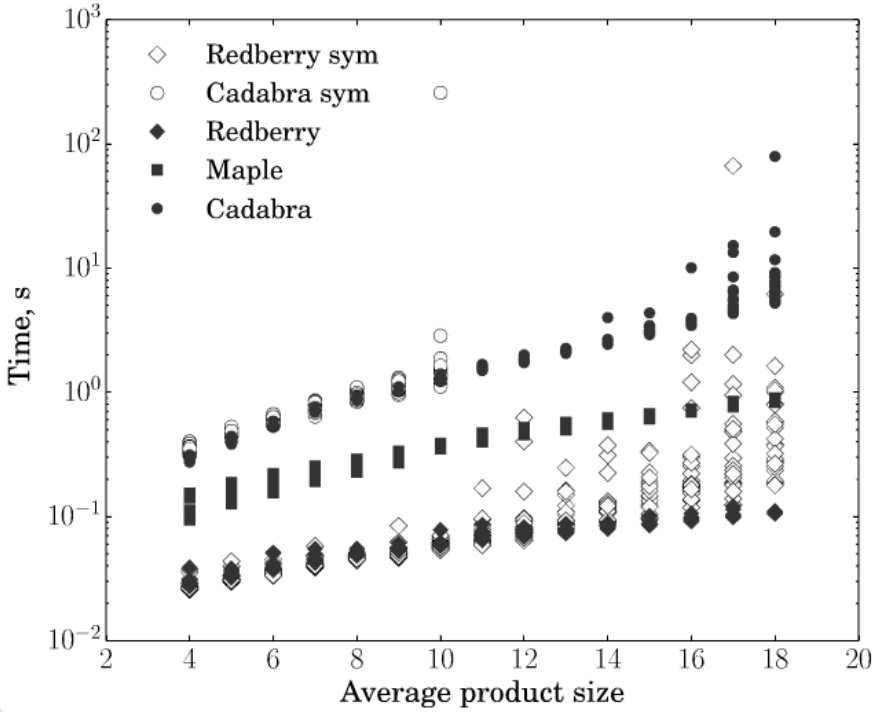
$$\left(F_{\mu\nu}\left(T^{\mu\alpha}R^{\nu\beta} + \ldots\right) + \ldots\right)\left(R_{\alpha\rho}\left(F^{\rho}{}_{\beta}R_{\tau\gamma} + \ldots\right) + \ldots\right) \times (\ldots) - \left(F_{\nu\mu}T^{\nu\beta}R^{\mu\alpha}R_{\beta\rho}F^{\rho}{}_{\alpha}R_{\tau\gamma} + \ldots\right) = 0$$

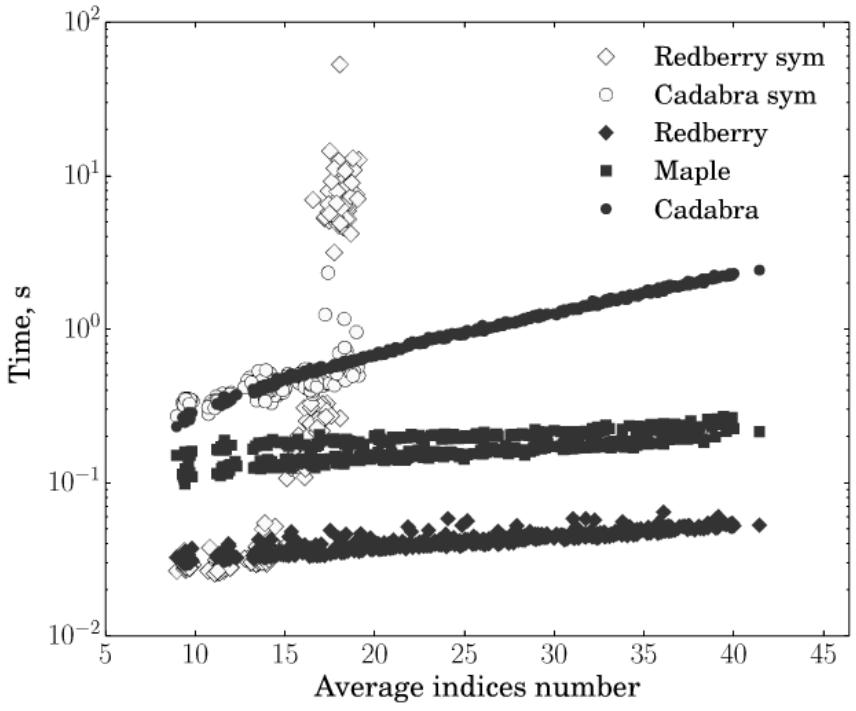


Atomic comparison

- Generate sum of 100 random different terms, rearrange indices (rename dummies and shuffle indices within symmetries) and subtract from itself, e. g. :

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

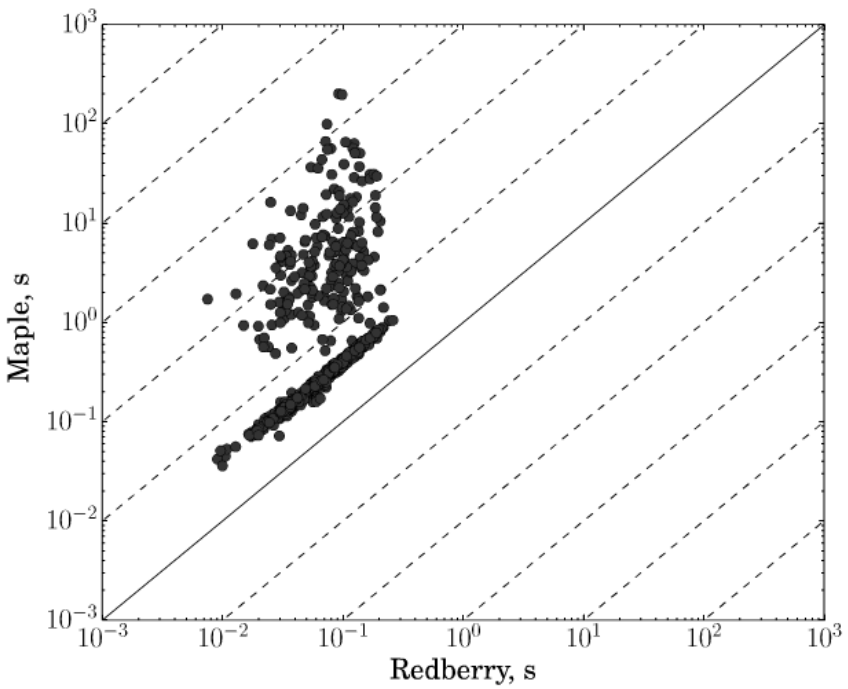


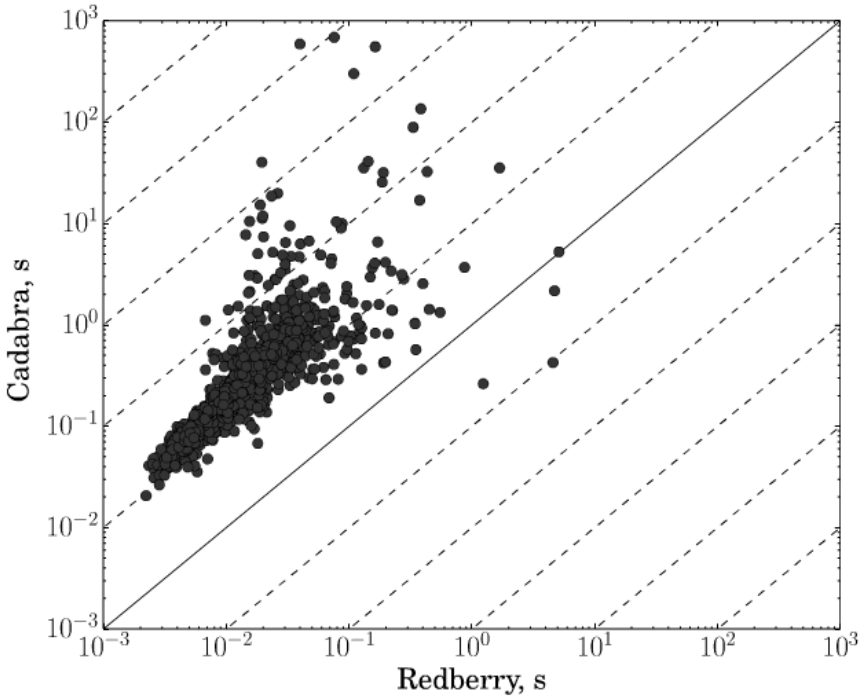


Expand nested sums

- Generate random nested products of sums, expand, rearrange indices (rename dummies and shuffle indices within symmetries), and subtract from itself, e. g. :

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





Algorithms

- The main atomic operation in any CAS is comparison of expressions
- Contractions between indices forms a mathematical graph, and thus comparison becomes an expensive operation which strongly affects the overall performance of CAS

Common approach: canonicalisation

(Rodionov & Taranov 1989, Portugal 1999)

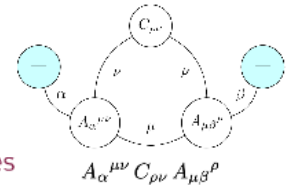
- Main idea: put indices of all expressions into "canonical" form (rename dummies, sort using symmetries, sort multipliers), then comparison becomes trivial:

$$R^{abcd}R_{efdc}R^{ef}{}_{ab} \quad \to \quad -R^{ab}{}_{cd}R_{abef}R^{efcd}$$
$$R_{rc}{}^{df}R_{ab}{}^{rc}R_{fd}{}^{ba} \quad \to \quad R^{ab}{}_{cd}R_{abef}R^{efcd}$$

- The algorithm is equivalent to finding double coset representatives
  - × NP-hard problem in general (Butler 1984, Luks 1993, Holt 2005)
  - × Works only with products of simple tensors (need to expand if product contains sum)

Redberry's approach: graph-based tensor comparison

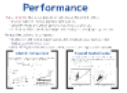
(Rodionov & Taranov 1989)



Graph: multipliers - vertexes, dummies - edges

Comparison of tensors = graph isomorphism (GI) problem

- Redberry does not perform expensive "canonicalisation" of terms
- It searches for isomorphisms (mappings of indices) between tensors and does not rely on canonical form of expressions
- **Althought, GI is at least NP, in all practical cases it can be solved very efficiently (McKay, 1980)**
- **Redberry uses its own implementation of GI problem optimized for typical expressions arising in real calculations**



Technical details

- Programming language: Java, Groovy (for user interface)
- Operating system: any Linux, Windows, Mac
- Lines of code: 132 148
- User interface: IntelliJ IDEA (syntax highlighting, code completion), command line is also available
- License: GNU GPL v3 (free and open source)
- Requirements: Java 7+, Groovy 2+

Full overview and comprehensive documentation:

http://redberry.cc

Thank you!

References

- D. A. Bolotin, S. V. Poslavsky, "Introduction to Redberry: a computer algebra system designed for tensor manipulation", arXiv:1302.1219 [cs.SC]
- J. M. Martín-García et.al, "xAct: Efficient tensor computer algebra for Mathematica", http://xact.es/
- Kasper Peeters, "Introducing Cadabra: a symbolic computer algebra system for field theory problems", hep-th/0701238
- A. Ya. Rodionov, A. Yu. Taranov, "Combinatorial aspects of simplification of algebraic expressions", Eurocal '87, Lecture Notes in Computer Science Volume 378, 1989, pp 192-201
- R. Portugal, "Algorithmic simplification of tensor expressions", J. Phys. A 32 (1999) 7779-7789
- G. Butler, "On Computing Double Coset Representatives in Permutation Groups", in Computational Group Theory, ed. M. D. Atkinson, Academic Press (1984), 283--290
- Eugene M. Luks, "Permutation groups and polynomial-time computation", In Finkelstein and Kantor [FK93], pages 139– 175
- Derek F. Holt, Bettina Eick, Eamonn A. O'Brien, "Handbook Of Computational Group Theory", Chapman and Hall/CRC, 2005

1. Setup symmetries $R_{abcd} = R_{cdba} = -R_{bacd}$

2. Check that $R^{ef}{}_{ab} R_{efdc} R^{abcd} + R_{rc}{}^{df} R_{ab}{}^{rc} R_{fd}{}^{ba} = 0$

Redberry code:

```
// Setup symmetries
addSymmetries 'R_abcd', [[0, 2], [1, 3]].p, -[[1, 0]].p
// Input expression
expr = 'R^abcd*R_efdc*R^ef_ab + R_rc^df*R_ab^rc*R_fd^ba'.t
println expr //prints zero
```

```
> 0
```

Mappings of indices

- The result of comparison is not just logical "true" or "false" but a complicated mapping:

$$F_{ab}G^{bc} \xrightarrow[\text{maps to}]{} F_{iq}G^{qj} \;=\; \left\{ \begin{array}{l} a \to i \\ c \to j \end{array} \right\}$$

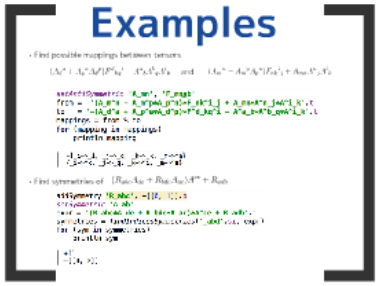
- Several mappings can be found for a pair of tensors. E.g., if R_{ab} is antisymmetric, then

$$R_{ab}A_c + R_{bc}A_a \xrightarrow[\text{maps to}]{} R_{ab}A_c + R_{bc}A_a \;=\; +\left\{ \begin{array}{l} a \to a \\ b \to b \\ c \to c \end{array} \right\} \text{ and } -\left\{ \begin{array}{l} a \to c \\ b \to b \\ c \to a \end{array} \right\}$$

- When mapping tensor onto itself, we obtain permutational symmetries of its indices, so

⇒ **Finding symmetries of tensors = graph automorphism (GA) problem**

Examples



Examples

- Find possible mappings between tensors

$$-(A_d{}^a + A_p{}^a A_d{}^p)F^d{}_{kq}{}^i - A^a{}_b A^b{}_q A^i{}_k \quad \text{and} \quad (A_m{}^n - A_m{}^p A_p{}^n)F_{nk}{}^i{}_j + A_{mn}A^n{}_j A^i{}_k$$

```
setAntiSymmetric 'A_mn', 'F_mnab'
from =  '(A_m^n - A_m^p*A_p^n)*F_nk^i_j + A_mn*A^n_j*A^i_k'.t
to   = '-(A_d^a + A_p^a*A_d^p)*F^d_kq^i - A^a_b*A^b_q*A^i_k'.t
mappings = from % to
for (mapping in mappings)
    println mapping
```

```
-{_i->_i, _j->_q, _k->_k, _m->^a}
{_i->^k, _j->_q, _k->^i, _m->^a}
```

- Find symmetries of $(R_{abc}A_{de} + R_{bde}A_{ac})A^{ce} + R_{adb}$

```
addSymmetry 'R_abc', -[[0, 1]].p
setSymmetric 'A_ab'
expr = '(R_abc*A_de + R_bde*A_ac)*A^ce + R_adb'.t
symmetries = findIndicesSymmetries('_abd'.si, expr)
for (sym in symmetries)
    println sym
```

```
+[]
-[[0, 2]]
```