Tensor computer algebra in theoretical physics

Introducing new system: Redberry

D A Bolotin [1] **& S V Poslavsky** [2]
IBCh RAS, Moscow & IHEP, Protvino
[1] bolotin.dmitriy@gmail.com, [2] stvlpos@mail.ru

Problem

• assume all T are symmetric:

$$A = T_{abmj}T_{defi}T_{abc}T_{cdj}T_{inm}T_{efn} \quad (1a)$$
$$B = T_{mnij}T_{ebfa}T_{mic}T_{adb}T_{dne}T_{fcj} \quad (1b)$$
$$C = T_{abmj}T_{defi}T_{abc}T_{cef}T_{inm}T_{djn} \quad (1c)$$

how to check whether A = B or A = C or C = B ?

• take a substitution:

$$T_{abcd}T_{aeb}T_{ecj} \to X_{dj}$$

how to apply it to (1a-1c) ?

• let tensors have sophisticated symmetries, e.g. (R is Riemann):

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

how to automate simplification of such expressions ?

Graph-theoretical approach

Main idea: ***contractions of indices form a graph of multipliers*** *(similar to Penrose graphical notation).*

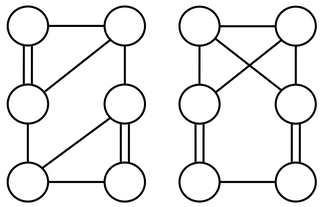

Figure 1: Graph representation of expressions (1a), (1b) (left) and (1c) (right). These graphs immediately shows that (1a) = (1b) ≠ (1c).

Computer algebra & Computational graph theory

Equality test	⟷	Graph Isomorphism
Find symmetries	⟷	Graph Automorphism
Substitutions (pattern matching)	⟷	Subgraph Isomorphism

• Although Graph Isomorphism is $\mathcal{NP}$-hard, there are very fast algorithms for all practical cases **[2]**

• Another approach for handling tensorial expressions (see **[3]**; used in Cadabra and Mathematica xAct) uses algorithm for *enumeration of double cosets in permutation groups*, which is $\mathcal{NP}$-complete

Under the hood: permutation groups algorithms

What algorithms are implemented to manage tensorial expressions:

• bases and strong generating sets
• setwise stabilizers, centralizers, normalizers
• coset representatives enumeration
• subgroups intersection
• and many more...

Example: symmetries, substitutions, simplifications

```
//Weyl tensor symmetries
addSymmetries 'W_abcd', [[0,2],[1,3]].p, -[[0,1]].p
//Define a substitution
subs = 'W_mnpq = (2*W_mnpq - W_mqnp + W_mpnq)/3'.t
expr = '''W^p_q^r_s*W_p^t_r^u*W_tv^qw*W_u^vs_w
    - W^p_q^r_s*W_p^qtu*W_rvtw*W^sv_u^w
    - W_mn^ab*W^n_pb^c*W^ms_cd*W_s^pd_a
    + W_mn^ab*W^ps_ba*W^m_p^c_d*W^n_s^d_c/4'''.t
println( (subs & Expand) >> expr )
```

▷ 0

Example: Dirac & SU(N) algebra

```
defineMatrices 'G_a', 'G5', Matrix1.matrix
expr = 'Tr[p^a*p^b*G_a*G_b*G_c*G_d*(1+G5)]'.t
println DiracTrace >> expr
```

▷ 4*p^b*p_b*g_cd

```
//take a trace in 4-d dimensions
tr = DiracTrace[[Dimension: '4-d', TraceOfOne: 4]]
expr = 'Tr[p^c*p^b*G_a*G_b*G_c*G^a]'.t
println tr >> expr
```

▷ 4*(4-d)*p^b*p_b*g_cd

Redberry: fearures & performance

Redberry is more than 10 times faster for tensor algebra than other CASs:

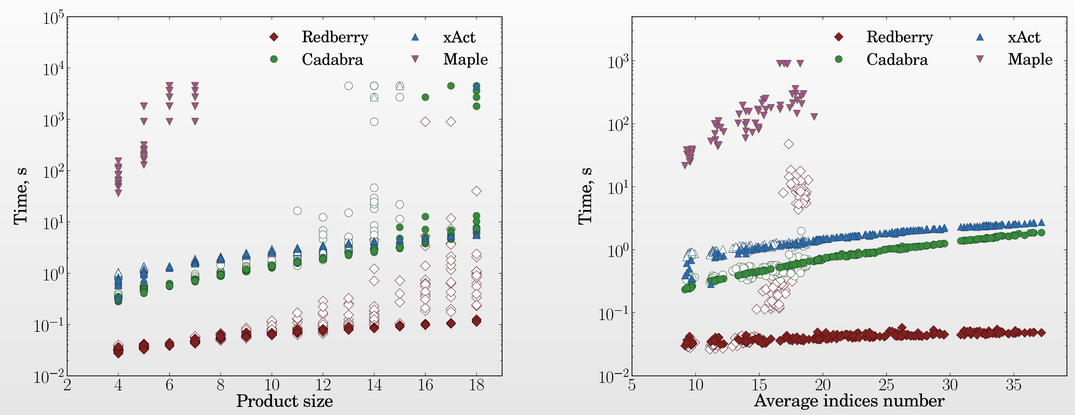

Figure 2: Comparison of Redberry performance with *Mathematica xAct*, *Cadabra* and *Maple 18*. For each point we measured time needed to simplify expression with ∼ 200 tensor monomials; filled markers correspond to the case with no symmetries specified, blank – with fully (anti) symmetric tensors.

• **Basic things:** algebraic manipulations, substitutions, functions and dozens of simplifications
• **Symmetries:** arbitrary permutational symmetries and antisymmetries of tensors
• **Feynman diagrams:** spinors, Dirac & SU(N) matrices and many related functionality
• **Programming language:** ability to write programms and implement custom functionality

Example: scalar-graviton scattering

```
// graviton propagator & scalar-graviton vertex
D = 'D_mnab[p_a]:= (g_ma*g_nb + g_mb*g_na - g_mn*g_ab)
  /(2*p_i*p^i)'.t
V = 'V_mn[p_a, k_a]:= p_m*k_n + p_n*k_m - g_mn*(p_a -
  k_a)*(p^a - k^a)/2'.t
// matrix element
M = 'V_ab[p_a,k_a]*D^abcd[p_a - k_a]*V_cd[q_a,r_a]'.t
ms = setMandelstam([p_a:'m',q_a:'m',k_a:'m',r_a:'m'])
M <<= ExpandAll[EliminateMetrics] & 'd^i_i = D'.t & ms
println Factor >> M
```

▷ -(1/8)*t**(-1)*(-8*D*m**2*t-4*s**2-4*t**2-16*m**4+16*u*m**2+t**2*D**2+16*s*m**2-4*u**2+2*D*t**2)

Technical details

• Programming language: **Java, Groovy** (for user interface)
• **132 914** lines and **~6 Mb** of source code covered with **1.5k** of tests
• Runs on **all operating systems** (Linux, OS X, Windows)
• Free and **open source**
• Many real-world problems had been solved using Redberry

See full documentation and examples at
http://redberry.cc

One loop counterterms in curved spacetime

Redberry implements algorithm **[4]** for one-loop counterterms calculation for general operators:

$$D^{(2)}{}_i{}^j = K^{(\mu\nu)}{}_i{}^j\nabla_\mu\nabla_\nu + S^\mu{}_i{}^j\nabla_\mu + W_i{}^j$$
$$D^{(4)}{}_i{}^j = K^{(\mu\nu\alpha\beta)}{}_i{}^j\nabla_\mu\nabla_\nu\nabla_\alpha\nabla_\beta + W^{(\mu\nu)}{}_i{}^j\nabla_\mu\nabla_\nu + M_i{}^j$$

E.g. for vector field operator defined by

$$S \sim \int d^4x\sqrt{-g}\, C_\alpha\left(\delta^\alpha_\beta\nabla_\mu\nabla^\mu - \lambda\nabla^\alpha\nabla_\beta + P^\alpha_\beta\right)C^\beta$$

```
addSymmetries 'R_abcd', [[0,2],[1,3]].p, -[[0,1]].p
setSymmetric 'R_ab', 'P_ab'
iK = 'iK_a^b = d_a^b + g*n_a*n^b'.t //propagator
K = 'K^mn_i^j = g^mn*d_i^j - 2*la*(g^mj*d_j^n + g^nj*
  d_i^m)'.t
W = 'W^j_i = P^j_i + la/2*R^j_i'.t
S = 'S^r^j_i = 0'.t
F = 'F_mnab = R_mnab'.t //curvature
div = oneloopdiv2(iK, K, S, W, F)
div <<= Collect['R', 'P', Factor]
println div
```

Gives ($\lambda = \gamma/(1+\gamma)$):

$$\Gamma^{(1)}_\infty = \frac{1}{16\pi(d-4)}\int d^4x\sqrt{-g}\left(\frac{1}{120}(-32+5\gamma^2+10\gamma)R_{\epsilon\mu}R^{\epsilon\mu} + \right.$$
$$+\frac{1}{240}R^2(28+5\gamma^2+20\gamma) + \frac{1}{24}(\gamma^2+12+6\gamma)P_{\beta\alpha}P^{\alpha\beta} +$$
$$\left.+\frac{1}{12}\gamma(4+\gamma)R_{\nu\epsilon}P^{\nu\epsilon} + \frac{1}{48}\gamma^2P^2 + \frac{1}{24}R(\gamma^2+4+2\gamma)P\right)$$

Equations with tensors

How to find propagator in a complicated theory? How to find tensor projector onto some subspace? Both problems require solving tensorial equations and Redberry automates such calculations. E.g., sum over polarisations for spin-2 is determined by:

$$J_{abcd}p^a = 0,\ J_{abc}{}^c = 0,\ J_{abcd}J^{abcd} = 5$$

```
addSymmetries 'J_abcd', [[0,1]].p, [[0,2], [1,3]].p
eq1 = 'J_abcd * p^a = 0'.t; eq2 = 'J_abc^c = 0'.t
eq3 = 'J_abcd * J^abcd = 5'.t
rules = 'd^n_n = 4'.t & 'p_a*p^a = m**2'.t
opts = [Transformations: rules,
        ExternalSolver : [Solver: 'Mathematica',
                          Path  : '/usr/bin']]
println Reduce([eq1,eq2,eq3], ['J_abcd'], opts)
```

Gives ($J_{ab} = -g_{ab} + p_ap_b/m^2$):

$$J_{abcd} = \pm\left((J_{ac}J_{bd} + J_{ad}J_{bc})/2 - J_{ab}J_{cd}/3\right)$$

References

[1] D.A. Bolotin, S.V. Poslavsky, *"Introduction to Redberry: a computer algebra system designed for tensor manipulation"*, arXiv:1302.1219

[2] B.D. McKay, *"Practical graph isomorphism"*, Proceedings of 10th. Manitoba Conference on Numerical Mathematics and Computing (Winnipeg,1980); Congressus Numerantium 30 (1981) 4587.

[3] A.Ya. Rodionov, A.Yu. Taranov, *"Combinatorial aspects of simplification of algebraic expressions"*, Eurocal'87 Lecture Notes in Computer Science 378 (1989) 192-201

[4] P.I.Pronin, K.V.Stepanyantz, *"One loop counterterms for the dimensional regularization of arbitrary Lagrangians"*, Nucl.Phys. B 485 (1997) 517544