xAct‘xTensor‘

This is the doc file xTensorDoc.nb of version 0.9.5 of `xTensor‘`. Last update on 13 May 2008.

Author

© 2002−2008, under the GNU General Public License (GPL)

José M. Martín−García

Instituto de Estructura de la Materia, CSIC, Spain

`jmm@iem.cfmac.csic.es`

`http://metric.iem.csic.es/Martin−Garcia/`

Helped by:

Alfonso García−Parrado (algar@mai.liu.se): design of vector bundles and complex structure from version 0.9.0.

Intro

`xTensor‘` extends *Mathematica* capabilities in abstract tensor calculus, specially in General Relativity. It works with tensors with arbitrary symmetries under permutations of indices, defined on several different manifolds and direct products of them. It computes covariant derivatives, Lie derivatives, parametric derivatives and variational derivatives. It allows the presence of a metric in each manifold and defines all the associated tensors (Riemann, Ricci, Einstein, Weyl, etc.)

`xTensor‘` does not perform component calculations. See the twin package `xCoba‘` or use another package like `GRTensorM‘` for that purpose.

`xTensor‘` needs the twin package `xPerm‘` in order to compute the canonical form of a list of indices under certain symmetry group.

`xTensor‘` has been designed with the following priorities in mind:

1.− Mathematical structure

2.− Efficiency

3.− Compliance with *Mathematica* style

4.− Simplicity of input/output

In particular, concerning the mathematical structure, `xTensor‘` imitates the usual cycle in mathematics ‘‘let **`X`** be a (**`Type`**) with properties **`props`**; then use it to do some **`computations`**´´ with the general form of declaration

```
DefType[X, props]

computations
```

Load the package

The latest version of the package can always be downloaded from

http://metric.iem.csic.es/Martin−Garcia/xAct/

This loads the package from the default directory, for example $Home/.Mathematica/Applications/xAct/ for a single−user installa− tion under Linux (see the installation notes for other possibilities),

```
In[1]:= MemoryInUse[]

Out[1]= 3059504

In[2]:= <<xAct‘xTensor‘

        ------------------------------------------------------------------------------
          --

        Package xAct‘xCore‘  version 0.5.0, {2008, 5, 16}

        CopyRight (C) 2007-2008, Jose M.
          Martin-Garcia, under the General Public License.

        ------------------------------------------------------------------------------
          --

        Package ExpressionManipulation‘

        CopyRight (C) 1999-2008, David J. M. Park and Ted Ersek

        ------------------------------------------------------------------------------
          --

        Package xAct‘xPerm‘  version 1.0.1, {2008, 5, 16}

        CopyRight (C) 2003-2008, Jose M.
          Martin-Garcia, under the General Public License.

        Connecting to external linux executable...

        Connection established.

        ------------------------------------------------------------------------------
          --

        Package xAct‘xTensor‘  version 0.9.5, {2008, 5, 16}

        CopyRight (C) 2002-2008, Jose M.
          Martin-Garcia, under the General Public License.

        ------------------------------------------------------------------------------
          --

        These packages come with ABSOLUTELY NO WARRANTY; for details type
          Disclaimer[]. This is free software, and you are welcome to redistribute
          it under certain conditions. See the General Public License for details.

        ------------------------------------------------------------------------------
          --
```

Memory, in bytes, used by both *Mathematica* and `xAct‘`, and then `xAct‘` alone:

```
In[3]:= MemoryInUse[]

Out[3]= 15787424

In[4]:= Out[3] - Out[1]

Out[4]= 12727920
```

Note the structure of the ContextPath. There are six contexts: `xAct‘xTensor‘`, `xAct‘xPerm‘`, `xAct‘xCore‘` and `xAct‘ExpressionManipulation‘` contain the respective reserved words. `System‘` contains *Mathematica*'s reserved words. The current context `Global‘` will contain your definitions and right now it is empty.

```
In[5]:=  $ContextPath

Out[5]=  {xAct‘xTensor‘, xAct‘xPerm‘, xAct‘xCore‘,
          xAct‘ExpressionManipulation‘, Global‘, System‘}

In[6]:=  Context[]

Out[6]=  Global‘

In[7]:=  ?Global‘*

         Information::nomatch : No symbol matching Global‘* found. More…
```

We turn off the annoying spell messages (since version 0.9.3 this is done by default when reading `xCore‘`):

```
In[8]:=  Off[General::spell]
         Off[General::spell1]
```

Additional technical notes:

There are several global variables that can be useful at loading time:

− If the variable `xAct‘xTensor‘$ReadingVerbose` is defined and has value `True` before loading the package, then several messages are printed while reading the package. This can be used to debug the input file in case there are errors at that point.

− The variables `xAct‘xCore‘$Version`, `xAct‘xPerm‘$Version` and `xAct‘xTensor‘$Version` contain the respective versions of the packages. The variable `xAct‘xTensor‘$xPermExpectedVersion` contains the oldest valid version of the package `xPerm‘` which is fully compatible with the current version of `xTensor‘`.

− The messages printed by `xAct‘xCore‘Disclaimer[]`, `xAct‘xPerm‘Disclaimer[]` and `xAct‘xTensor‘Disclaimer[]` are identical.

■ 0. Basics

0.1. Example session

This is a very simple example of the use of `xTensor‘`. We start by defining a manifold and some tensor fields living on its tangent bundle.

Define a 3d manifold M3:

```
In[10]:=  DefManifold[M3, 3, {a, b, c, d, e}]

            ** DefManifold: Defining manifold M3.

            ** DefVBundle: Defining vbundle TangentM3.
```

Define a contravariant vector v and a covariant antisymmetric tensor F:

```
In[11]:= DefTensor[v[a], M3]
         DefTensor[F[-a, -b], M3, Antisymmetric[{-a, -b}]]

           ** DefTensor: Defining tensor v[a].

           ** DefTensor: Defining tensor F[-a, -b].
```

Tensor product of both tensors, based on the abstract index notation:

```
In[13]:= F[-a, -b] v[b]
```

Out[13]= $F_{ab}\, v^{b}$

```
In[14]:= % v[a]
```

Out[14]= $F_{ab}\, v^{a}\, v^{b}$

There is no automatic simplification. It must be explicitly asked for:

```
In[15]:= Simplification[%]

Out[15]= 0
```

Example of dummy handling:

```
In[16]:= F[-a, -d] v[d] v[b] v[-b] + 3 v[b] v[c] v[-c] F[-b, d] F[-d, -a]
```

Out[16]= $3\, F_{b}{}^{d}\, F_{da}\, v^{b}\, v_{c}\, v^{c} + F_{ad}\, v_{b}\, v^{b}\, v^{d}$

```
In[17]:= % // Simplification
```

Out[17]= $(F_{ac} - 3\, F_{ad}\, F_{c}{}^{d})\, v_{b}\, v^{b}\, v^{c}$

There is not a metric and hence v cannot be covariant. Note the position of the head ERROR:

```
In[18]:= Validate[%]

      Validate::error : Invalid character of index in tensor F

      Validate::error : Invalid character of index in tensor v
```

Out[18]= $\mathrm{ERROR}[v_{b}]\, (F_{ac} - 3\, \mathrm{ERROR}[F_{c}{}^{d}]\, F_{ad})\, v^{b}\, v^{c}$

Internal structure:

```
In[19]:= InputForm[%%]

  Out[19]//InputForm=
     (F[-a, -c] - 3*F[-a, -d]*F[-c, d])*v[-b]*v[b]*v[c]
```

Now we define a covariant derivative operator and check the first Bianchi identity (valid for any symmetric connection). Note that `xTensor‘` uses unique (dollar−) variables as internal dummy indices to avoid index collisions. The function `ScreenDollarIndices` hides those indices replacing them by "normal" indices.

Covariant derivatives are defined by default with curvature but without torsion:

```
In[20]:=  Options[DefCovD]

Out[20]=  {Torsion → False, Curvature → True, FromMetric → Null, CurvatureRelations → True,
           ExtendedFrom → Null, OrthogonalTo → {}, ProjectedWith → {},
           WeightedWithBasis → Null, ProtectNewSymbol :→ $ProtectNewSymbols,
           Master → Null, Info → {covariant derivative, }}

In[21]:=  DefCovD[Cd[-a], {";", "∇"}]

            ** DefCovD: Defining covariant derivative Cd[-a].

            ** DefTensor: Defining vanishing torsion tensor TorsionCd[a, -b, -c].

            ** DefTensor: Defining symmetric Christoffel tensor ChristoffelCd[a, -b, -c].

            ** DefTensor: Defining Riemann tensor
             RiemannCd[-a, -b, -c, d]. Antisymmetric only in the first pair.

            ** DefTensor: Defining non-symmetric Ricci tensor RicciCd[-a, -b].

            ** DefCovD:  Contractions of Riemann automatically replaced by Ricci.

In[22]:=  Cd[-a][ F[-b, -c] v[c] ]
```

Out[22]= $v^c\,(\nabla_a F_{bc}) + F_{bc}\,(\nabla_a v^c)$

```
In[23]:=  % v[b] // Simplification
```

Out[23]= $F_{bc}\,v^b\,(\nabla_a v^c)$

The tensors have been defined with their expected symmetries:

```
In[24]:=  RiemannCd[-a, -b, -c, d] + RiemannCd[-b, -a, -c, d] // Simplification

Out[24]=  0
```

but it is possible to deduce them from more basic principles:

```
In[25]:=  RiemannCd[-a, -b, -c, d] + RiemannCd[-b, -a, -c, d] // RiemannToChristoffel //
           Simplification

Out[25]=  0

In[26]:=  RiemannCd[-a, -b, -c, d] + RiemannCd[-b, -c, -a, d] + RiemannCd[-c, -a, -b, d] //
            RiemannToChristoffel // Simplification

Out[26]=  0
```

Check of the second Bianchi identity:

```
In[27]:= Antisymmetrize[Cd[-e][RiemannCd[-c, -d, -b, a]], {-c, -d, -e}]
```

Out[27]=
$$\frac{1}{6}\left(\nabla_c R[\nabla]_{deb}{}^{a} - \nabla_c R[\nabla]_{edb}{}^{a} - \nabla_d R[\nabla]_{ceb}{}^{a} + \nabla_d R[\nabla]_{ecb}{}^{a} + \nabla_e R[\nabla]_{cdb}{}^{a} - \nabla_e R[\nabla]_{dcb}{}^{a}\right)$$

```
In[28]:= 3 % // Simplification
```

Out[28]=
$$\nabla_c R[\nabla]_{deb}{}^{a} - \nabla_d R[\nabla]_{ceb}{}^{a} + \nabla_e R[\nabla]_{cdb}{}^{a}$$

```
In[29]:= % // CovDToChristoffel
```

Out[29]=
$$\begin{aligned}
&\Gamma[\nabla]^{a}{}_{ee\$308} R[\nabla]_{cdb}{}^{e\$308} - \Gamma[\nabla]^{e\$309}{}_{eb} R[\nabla]_{cde\$309}{}^{a} - \\
&\Gamma[\nabla]^{a}{}_{de\$300} R[\nabla]_{ceb}{}^{e\$300} + \Gamma[\nabla]^{e\$301}{}_{db} R[\nabla]_{cee\$301}{}^{a} + \Gamma[\nabla]^{e\$303}{}_{de} R[\nabla]_{ce\$303b}{}^{a} - \\
&\Gamma[\nabla]^{e\$311}{}_{ed} R[\nabla]_{ce\$311b}{}^{a} + \Gamma[\nabla]^{a}{}_{ce\$292} R[\nabla]_{deb}{}^{e\$292} - \Gamma[\nabla]^{e\$293}{}_{cb} R[\nabla]_{dee\$293}{}^{a} - \\
&\Gamma[\nabla]^{e\$295}{}_{ce} R[\nabla]_{de\$295b}{}^{a} - \Gamma[\nabla]^{e\$294}{}_{cd} R[\nabla]_{e\$294eb}{}^{a} + \Gamma[\nabla]^{e\$302}{}_{dc} R[\nabla]_{e\$302eb}{}^{a} - \\
&\Gamma[\nabla]^{e\$310}{}_{ec} R[\nabla]_{e\$310db}{}^{a} + \partial_c R[\nabla]_{deb}{}^{a} - \partial_d R[\nabla]_{ceb}{}^{a} + \partial_e R[\nabla]_{cdb}{}^{a}
\end{aligned}$$

```
In[30]:= % // RiemannToChristoffel // ScreenDollarIndices
```

Out[30]=
$$\begin{aligned}
&-\Gamma[\nabla]^{e1}{}_{eb}\, \partial_c \Gamma[\nabla]^{a}{}_{de1} + \Gamma[\nabla]^{e1}{}_{db}\, \partial_c \Gamma[\nabla]^{a}{}_{ee1} + \Gamma[\nabla]^{a}{}_{ee1}\, \partial_c \Gamma[\nabla]^{e1}{}_{db} - \\
&\Gamma[\nabla]^{a}{}_{de1}\, \partial_c \Gamma[\nabla]^{e1}{}_{eb} - \partial_c \partial_d \Gamma[\nabla]^{a}{}_{eb} + \partial_c \partial_e \Gamma[\nabla]^{a}{}_{db} + \Gamma[\nabla]^{e1}{}_{eb}\, \partial_d \Gamma[\nabla]^{a}{}_{ce1} - \\
&\Gamma[\nabla]^{e1}{}_{eb} \left(\Gamma[\nabla]^{a}{}_{de2} \Gamma[\nabla]^{e2}{}_{ce1} - \Gamma[\nabla]^{a}{}_{ce2} \Gamma[\nabla]^{e2}{}_{de1} - \partial_c \Gamma[\nabla]^{a}{}_{de1} + \partial_d \Gamma[\nabla]^{a}{}_{ce1}\right) - \\
&\Gamma[\nabla]^{e1}{}_{cb}\, \partial_d \Gamma[\nabla]^{a}{}_{ee1} - \Gamma[\nabla]^{a}{}_{ee1}\, \partial_d \Gamma[\nabla]^{e1}{}_{cb} + \\
&\Gamma[\nabla]^{a}{}_{ee1} \left(\Gamma[\nabla]^{e1}{}_{de2} \Gamma[\nabla]^{e2}{}_{cb} - \Gamma[\nabla]^{e1}{}_{ce2} \Gamma[\nabla]^{e2}{}_{db} - \partial_c \Gamma[\nabla]^{e1}{}_{db} + \partial_d \Gamma[\nabla]^{e1}{}_{cb}\right) + \\
&\Gamma[\nabla]^{a}{}_{ce1}\, \partial_d \Gamma[\nabla]^{e1}{}_{eb} + \partial_d \partial_c \Gamma[\nabla]^{a}{}_{eb} - \partial_d \partial_e \Gamma[\nabla]^{a}{}_{cb} - \Gamma[\nabla]^{e1}{}_{db}\, \partial_e \Gamma[\nabla]^{a}{}_{ce1} + \\
&\Gamma[\nabla]^{e1}{}_{db} \left(\Gamma[\nabla]^{a}{}_{ee2} \Gamma[\nabla]^{e2}{}_{ce1} - \Gamma[\nabla]^{a}{}_{ce2} \Gamma[\nabla]^{e2}{}_{ee1} - \partial_c \Gamma[\nabla]^{a}{}_{ee1} + \partial_e \Gamma[\nabla]^{a}{}_{ce1}\right) + \\
&\Gamma[\nabla]^{e1}{}_{cb}\, \partial_e \Gamma[\nabla]^{a}{}_{de1} - \Gamma[\nabla]^{e1}{}_{cb} \\
&\quad \left(\Gamma[\nabla]^{a}{}_{ee2} \Gamma[\nabla]^{e2}{}_{de1} - \Gamma[\nabla]^{a}{}_{de2} \Gamma[\nabla]^{e2}{}_{ee1} - \partial_d \Gamma[\nabla]^{a}{}_{ee1} + \partial_e \Gamma[\nabla]^{a}{}_{de1}\right) + \Gamma[\nabla]^{a}{}_{de1}\, \partial_e \Gamma[\nabla]^{e1}{}_{cb} - \\
&\Gamma[\nabla]^{a}{}_{de1} \left(\Gamma[\nabla]^{e1}{}_{ee2} \Gamma[\nabla]^{e2}{}_{cb} - \Gamma[\nabla]^{e1}{}_{ce2} \Gamma[\nabla]^{e2}{}_{eb} - \partial_c \Gamma[\nabla]^{e1}{}_{eb} + \partial_e \Gamma[\nabla]^{e1}{}_{cb}\right) - \\
&\Gamma[\nabla]^{a}{}_{ce1}\, \partial_e \Gamma[\nabla]^{e1}{}_{db} + \\
&\Gamma[\nabla]^{a}{}_{ce1} \left(\Gamma[\nabla]^{e1}{}_{ee2} \Gamma[\nabla]^{e2}{}_{db} - \Gamma[\nabla]^{e1}{}_{de2} \Gamma[\nabla]^{e2}{}_{eb} - \partial_d \Gamma[\nabla]^{e1}{}_{eb} + \partial_e \Gamma[\nabla]^{e1}{}_{db}\right) - \partial_e \partial_c \Gamma[\nabla]^{a}{}_{db} + \\
&\partial_e \partial_d \Gamma[\nabla]^{a}{}_{cb} + \Gamma[\nabla]^{e1}{}_{de} \left(\Gamma[\nabla]^{a}{}_{e1e2} \Gamma[\nabla]^{e2}{}_{cb} - \Gamma[\nabla]^{a}{}_{ce2} \Gamma[\nabla]^{e2}{}_{e1b} - \partial_c \Gamma[\nabla]^{a}{}_{e1b} + \partial_{e1} \Gamma[\nabla]^{a}{}_{cb}\right) - \\
&\Gamma[\nabla]^{e1}{}_{ed} \left(\Gamma[\nabla]^{a}{}_{e1e2} \Gamma[\nabla]^{e2}{}_{cb} - \Gamma[\nabla]^{a}{}_{ce2} \Gamma[\nabla]^{e2}{}_{e1b} - \partial_c \Gamma[\nabla]^{a}{}_{e1b} + \partial_{e1} \Gamma[\nabla]^{a}{}_{cb}\right) - \\
&\Gamma[\nabla]^{e1}{}_{ec} \left(-\Gamma[\nabla]^{a}{}_{e1e2} \Gamma[\nabla]^{e2}{}_{db} + \Gamma[\nabla]^{a}{}_{de2} \Gamma[\nabla]^{e2}{}_{e1b} + \partial_d \Gamma[\nabla]^{a}{}_{e1b} - \partial_{e1} \Gamma[\nabla]^{a}{}_{db}\right) - \\
&\Gamma[\nabla]^{e1}{}_{ce} \left(\Gamma[\nabla]^{a}{}_{e1e2} \Gamma[\nabla]^{e2}{}_{db} - \Gamma[\nabla]^{a}{}_{de2} \Gamma[\nabla]^{e2}{}_{e1b} - \partial_d \Gamma[\nabla]^{a}{}_{e1b} + \partial_{e1} \Gamma[\nabla]^{a}{}_{db}\right) - \\
&\Gamma[\nabla]^{e1}{}_{cd} \left(-\Gamma[\nabla]^{a}{}_{e1e2} \Gamma[\nabla]^{e2}{}_{eb} + \Gamma[\nabla]^{a}{}_{ee2} \Gamma[\nabla]^{e2}{}_{e1b} + \partial_e \Gamma[\nabla]^{a}{}_{e1b} - \partial_{e1} \Gamma[\nabla]^{a}{}_{eb}\right) + \\
&\Gamma[\nabla]^{e1}{}_{dc} \left(-\Gamma[\nabla]^{a}{}_{e1e2} \Gamma[\nabla]^{e2}{}_{eb} + \Gamma[\nabla]^{a}{}_{ee2} \Gamma[\nabla]^{e2}{}_{e1b} + \partial_e \Gamma[\nabla]^{a}{}_{e1b} - \partial_{e1} \Gamma[\nabla]^{a}{}_{eb}\right)
\end{aligned}$$

```
In[31]:= Simplification[%] // AbsoluteTiming

Out[31]= {0.282918 Second, 0}
```

Commutation of covariant derivatives. It is possible to choose the indices (use `CommuteCovDs`) or just let `xTensor'` to bring the derivative indices into canonical order (use `SortCovDs`):

```
In[32]:= Cd[-a]@Cd[-b]@v[c]
```

Out[32]=
$$\nabla_a \nabla_b v^{c}$$

```
In[33]:= CommuteCovDs[%, Cd, {-b, -a}] // ScreenDollarIndices
```

Out[33]= $R[\nabla]_{bad}{}^{c}\, v^{d} + \nabla_{b}\, \nabla_{a}\, v^{c}$

```
In[34]:= SortCovDs[%%] // ScreenDollarIndices
```

Out[34]= $R[\nabla]_{bad}{}^{c}\, v^{d} + \nabla_{b}\, \nabla_{a}\, v^{c}$

Now we introduce a metric on the manifold. Its associated curvature tensors are defined: `RiemannCD`, `RiciCD`, `EinsteinCD`, etc:

```
In[35]:= DefMetric[-1, g[-a, -b], CD, {"|", "∇"}]

         ** DefTensor: Defining symmetric metric tensor g[-a, -b].
         ** DefTensor: Defining antisymmetric tensor epsilong[a, b, c].
         ** DefCovD: Defining covariant derivative CD[-a].
         ** DefTensor: Defining vanishing torsion tensor TorsionCD[a, -b, -c].
         ** DefTensor: Defining symmetric Christoffel tensor ChristoffelCD[a, -b, -c].
         ** DefTensor: Defining Riemann tensor RiemannCD[-a, -b, -c, -d].
         ** DefTensor: Defining symmetric Ricci tensor RicciCD[-a, -b].
         ** DefCovD:  Contractions of Riemann automatically replaced by Ricci.
         ** DefTensor: Defining Ricci scalar RicciScalarCD[].
         ** DefCovD:  Contractions of Ricci automatically replaced by RicciScalar.
         ** DefTensor: Defining symmetric Einstein tensor EinsteinCD[-a, -b].
         ** DefTensor: Defining vanishing Weyl tensor WeylCD[-a, -b, -c, -d].
         ** DefTensor: Defining symmetric TFRicci tensor TFRicciCD[-a, -b].
            Rules {1, 2} have been declared as DownValues for TFRicciCD.
         ** DefCovD:  Computing RiemannToWeylRules for dim 3
         ** DefCovD:  Computing RicciToTFRicci for dim 3
         ** DefCovD:  Computing RicciToEinsteinRules for dim 3

In[36]:= CD[-a][ g[-b, -c] ]

Out[36]= 0

In[37]:= CD[-a][ EinsteinCD[a, b] ]

Out[37]= 0

In[38]:= g[a, -a]

Out[38]= 3

In[39]:= g[a, b] g[-b, -c]
```

Out[39]= $\delta^{a}{}_{c}$

Let us check the antisymmetry of the second pair of indices in the Riemann tensor:

```
In[40]:= RiemannCD[-a, -b, -c, -d] + RiemannCD[-a, -b, -d, -c]
```

Out[40]= $R[\nabla]_{abcd} + R[\nabla]_{abdc}$

Because canonicalization is not automatic, the previous expression is not automatically zero. We expand it in derivatives of Christof−fels and canonicalize. The partial derivative does not commute with the metric, so that internally the code replaces temporarily those derivatives with Levi−Civita connections:

```
In[41]:= % // RiemannToChristoffel // Simplification // ScreenDollarIndices

        ToCanonical::cmods : Changing derivatives to canonicalize.

        ToCanonical::cmods : Changing derivatives to canonicalize.

        ToCanonical::cmods : Changing derivatives to canonicalize.

        General::stop : Further output of
           ToCanonical::cmods will be suppressed during this calculation. More…
```

Out[41]= $\Gamma[\nabla]_{db}{}^{e}\,\Gamma[\nabla]_{eac} + \Gamma[\nabla]_{cb}{}^{e}\,\Gamma[\nabla]_{ead} - \Gamma[\nabla]_{da}{}^{e}\,\Gamma[\nabla]_{ebc} - \Gamma[\nabla]_{ca}{}^{e}\,\Gamma[\nabla]_{ebd} + \Gamma[\nabla]^{e}{}_{bd}\,\Gamma[\nabla]^{e1}{}_{ae}\,g_{ce1} - \Gamma[\nabla]^{e}{}_{ad}\,\Gamma[\nabla]^{e1}{}_{be}\,g_{ce1} + \Gamma[\nabla]^{e}{}_{bc}\,\Gamma[\nabla]^{e1}{}_{ae}\,g_{de1} - \Gamma[\nabla]^{e}{}_{ac}\,\Gamma[\nabla]^{e1}{}_{be}\,g_{de1} - \partial_a\Gamma[\nabla]_{cbd} - \partial_a\Gamma[\nabla]_{dbc} + \partial_b\Gamma[\nabla]_{cad} + \partial_b\Gamma[\nabla]_{dac}$

Metric contraction is not automatic either. The resulting expression is not obviously zero:

```
In[42]:= % // ContractMetric // Simplification
```

Out[42]= $-\partial_a\Gamma[\nabla]_{cbd} - \partial_a\Gamma[\nabla]_{dbc} + \partial_b\Gamma[\nabla]_{cad} + \partial_b\Gamma[\nabla]_{dac}$

Actually it is only zero for connections deriving from a metric:

```
In[43]:= % // ChristoffelToGradMetric // Expand
```

Out[43]= $-\frac{1}{2}\,\partial_a\partial_b g_{cd} - \frac{1}{2}\,\partial_a\partial_b g_{dc} + \frac{1}{2}\,\partial_b\partial_a g_{cd} + \frac{1}{2}\,\partial_b\partial_a g_{dc}$

```
In[44]:= % // Simplification

Out[44]= 0
```

In this case a simple reordering of the partial derivatives (this is not automatic) would have been enough:

```
In[45]:= %% // SortCovDs

Out[45]= 0
```

Examples of Lie derivative and bracket:

```
In[46]:= LieD[3 v[a]][F[-a, -b] v[b]]
```

Out[46]= $3\,v^{b}\,(\mathcal{L}_v\,F_{ab}) + 3\,F_{ab}\,(\mathcal{L}_v\,v^{b})$

```
In[47]:= Bracket[a][v[a], F[a, -b] v[b]]
```

Out[47]= $[v^a, F^a{}_b\ v^b]^a$

Undefine all objects on the manifold and the manifold itself:

```
In[48]:= UndefMetric[g]

     ** UndefTensor: Undefined antisymmetric tensor epsilong

     ** UndefTensor: Undefined symmetric Christoffel tensor ChristoffelCD

     ** UndefTensor: Undefined symmetric Einstein tensor EinsteinCD

     ** UndefTensor: Undefined symmetric Ricci tensor RicciCD

     ** UndefTensor: Undefined Ricci scalar RicciScalarCD

     ** UndefTensor: Undefined Riemann tensor RiemannCD

     ** UndefTensor: Undefined symmetric TFRicci tensor TFRicciCD

     ** UndefTensor: Undefined vanishing torsion tensor TorsionCD

     ** UndefTensor: Undefined vanishing Weyl tensor WeylCD

     ** UndefCovD: Undefined covariant derivative CD

     ** UndefTensor: Undefined symmetric metric tensor g
```

```
In[49]:= UndefTensor /@ {v, F};
         UndefCovD[Cd]

     ** UndefTensor: Undefined tensor v

     ** UndefTensor: Undefined tensor F

     ** UndefTensor: Undefined symmetric Christoffel tensor ChristoffelCd

     ** UndefTensor: Undefined non-symmetric Ricci tensor RicciCd

     ** UndefTensor: Undefined Riemann tensor RiemannCd

     ** UndefTensor: Undefined vanishing torsion tensor TorsionCd

     ** UndefCovD: Undefined covariant derivative Cd
```

```
In[51]:= UndefManifold[M3]

     ** UndefVBundle: Undefined vbundle TangentM3

     ** UndefManifold: Undefined manifold M3
```

There are no symbols left in the `Global‘` context

```
In[52]:= ?Global‘*

   Information::nomatch : No symbol matching Global‘* found. More…
```

0.2. Summary

These are the objects that currently can be defined in `xTensor`‘.

− Manifolds

− Vector bundles

− Indices (of different types)

− Tensors (in particular, Metrics)

− Covariant derivative operators (including partial derivatives)

− Constant symbols

− Parameters

− Inert heads (wrappers for tensors)

− Scalar functions

− Charts (coordinate systems): use the twin package `xCoba`‘ (in collaboration with D. Yllanes)

− Bases (frames): use the twin package `xCoba`‘

We also have predefined mathematical operators:

− Lie derivative

− Lie bracket

− Parametric derivative

− Variational derivative

and useful mathematical operations:

− Relations and rules among indexed objects

− Indexed equations solving

− Canonicalization and simplification of expressions

− Automatic definition of curvature tensors, and relations among them

− Complex conjugation

There are also specialized algorithms, frequently used in General Relativity:

− Warped metric decomposition

− ADM−like decomposition

− Perturbation theory in General Relativity: use the twin package `xPert`‘ (in collaboration with D. Brizuela and G. Mena Marugán)

− Spherical tensor harmonics: use the twin package `Harmonics`‘ (in collaboration with D. Brizuela and G. Mena Marugán)

− Manipulation of invariants of the Riemann tensor: use the twin package `Invar`‘ (in collaboration with R. Portugal, L. Manssur and D. Yllanes)

0.3. Getting info

Every piece of information in *Mathematica* is associated to a symbol and can be obtained using the `?` and `??` symbols (see the command `Information` in the *Mathematica* help). `xTensor‘` strictly follows that rule. (In particular, we have explicitly avoided notations based on string input/output which would require a parser, much restricting the capabilities of the user.)

```
In[53]:=  ?DefManifold

          DefManifold[M, dim, {a, b, c,...}] defines M to be an n-dimensional
            differentiable manifold with dimension dim (a positive integer
            or a constant symbol) and tensor abstract indices a, b, c, ... .
            DefManifold[M, {M1, ..., Mm}, {a, b, c,...}] defines M to be the
            product manifold of previously defined manifolds M1 ... Mm. For
            backward compatibility dim can be a list of positive integers, whose
            length is interpreted as the dimension of the defined manifold.
```

```
In[54]:=  ?? ERROR

          ERROR[expr] is an expression where an error has
            been detected. ERROR is an inert head with a single argument.

          Attributes[ERROR] = {HoldFirst, Protected}

          InertHeadQ[ERROR] ^= True

          Info[ERROR] ^= {inert head, Generic head to wrap expressions with errors.}

          LinearQ[ERROR] ^= False

          PrintAs[ERROR] ^= ERROR
```

0.4. Too many brackets!

This is the main complaint from most users after their first contact with `xTensor‘`. My answer to that is the following:

1. That is how *Mathematica* internally works. For example, did you expect this simple derivative having internally six pairs of brackets, when it has just one in the "user−notation"?

```
In[55]:=  f'[2 x - y] // FullForm

  Out[55]//FullForm=
       Derivative[1][f][Plus[Times[2, x], Times[-1, y]]]
```

2. `xTensor‘` does not have different internal and external notations (at least not now). This helps understanding what is happening internally and allows the user to build pattern rules which always work. However, you are continually exposed to the internal notation.

3. The highest accummulation of brackets happens in derivatives. This simply follows the general idea in *Mathematica* (made clear in the example above) of treating a derivative as an operator on an expression, and not as a simple additional argument of the expression, as the traditional "comma−index" notation might suggest.

4. The *Mathematica* language is very similar to LISP, in which you have an awful lot of parenthesis.

5. If you are still not convinced by the previous three arguments, then I must say that it is possible to construct whatever other notation you like for a derivative, or for any other operator, and several examples of this are given in section 6.1 below.

0.5. Warnings

`xTensor‘` is a very powerful tool in the hands of an average *Mathematica* user, but could be difficult for an unexperi−enced *Mathematica* user. This is because I have not tried to hide *Mathematica* subtleties (like delayed assignments, unique symbols or upvalues), but rather I have tried to make as much use as possible of such powerful instruments. If you don’t understand those three listed concepts, please spend some (not really much) time with a *Mathematica* tutorial. It will pay off, even if you finally don’t use `xTensor‘`.

If you think you can do your tensor computation by hand in a few lines, then `xTensor‘` is probably too much. This system is designed to be efficient when developing big projects, not small computations: a general−purpose tensor computer algebra system always has hundreds of commands, and only big projects give you enough motivation to learn them.

■ 1. Manifolds, vector bundles and parameters

1.1. Define a manifold and its tangent bundle

The first basic action is defining a differentiable manifold. Just after loading, `xTensor‘` does not have any defined manifold.

`DefManifold`	Define a manifold or a product of manifolds
`UndefManifold`	Undefine a manifold
`ManifoldQ`	Check manifold
`$Manifolds`	List of defined manifolds
`$ProductManifolds`	List of defined product manifolds

Definition of a manifold

We define a 3−dimensional manifold `M3` with abstract indices {a, b, ..., h} on its tangent vector bundle:

```
In[56]:= DefManifold[M3, 3, {a, b, c, d, e, f, g, h}]

         ** DefManifold: Defining manifold M3.

         ** DefVBundle: Defining vbundle TangentM3.
```

We can ask *Mathematica* about this manifold using the question mark `?`. We see that the information is stored in a series of functions using the concept of "upvalue" (this is why we find the ^= sign below)

```
In[57]:= ?M3

         Global`M3

         DimOfManifold[M3] ^= 3

         Info[M3] ^= {manifold, }

         ManifoldQ[M3] ^= True

         ObjectsOf[M3] ^= {}

         PrintAs[M3] ^= M3

         ServantsOf[M3] ^= {TangentM3}

         SubmanifoldsOfManifold[M3] ^= {}

         TangentBundleOfManifold[M3] ^= TangentM3
```

Together with the manifold we have also defined its tangent vector bundle, which stores all the information related to indices and indexed objects. By default, the tangent bundle is named by joining the symbol `Tangent` with the name of the manifold, but that name can be freely choosen. Note that we shall often abbreviate "vector bundle" to "vbundle".

By default, tangent bundles are real and have no metric. The dimension of the tangent bundle is that of the vector space at each point, and coincides with the dimension of the base manifold.

```
In[58]:= ?TangentM3

         Global`TangentM3

         BaseOfVBundle[TangentM3] ^= M3

         Dagger[TangentM3] ^= TangentM3

         DimOfVBundle[TangentM3] ^= 3

         IndicesOfVBundle[TangentM3] ^= {{a, b, c, d, e, f, g, h}, {}}

         Info[TangentM3] ^= {vbundle, }

         MasterOf[TangentM3] ^= M3

         MetricsOfVBundle[TangentM3] ^= {}

         ObjectsOf[TangentM3] ^= {}

         PrintAs[TangentM3] ^= TangentM3

         SubvbundlesOfVBundle[TangentM3] ^= {}

         VBundleQ[TangentM3] ^= True
```

We define a second manifold `S2`. Note that we can use `C`, `D`, `K`, `N`, `O` as indices, even though *Mathematica* has reserved meanings for those symbols. The capitals `E` and `I` cannot be used as indices because they are always understood as the base of natural logarithms and the square root of −1, respectively. (Those seven are the only one−letter symbols used by *Mathematica*.) `xTensor'` issues warnings to remind you of this point:

```
In[59]:= DefManifold[S2, 2, {A, B, C, D, F, G, H}]

           ** DefManifold: Defining manifold S2.

           ** DefVBundle: Defining vbundle TangentS2.

      ValidateSymbol::capital : System name C is overloaded as an abstract index.

      ValidateSymbol::capital : System name D is overloaded as an abstract index.
```

The lists of manifolds and vbundles defined in the current session are given by the global variables `$Manifolds` and `$VBundles`, respectively:

```
In[60]:= $Manifolds

Out[60]= {M3, S2}

In[61]:= $VBundles

Out[61]= {TangentM3, TangentS2}
```

A manifold can be undefined (its properties are lost and the symbol is removed). In the process, the associated tangent bundle is also undefined:

```
In[62]:= UndefManifold[M3]

           ** UndefVBundle: Undefined vbundle TangentM3

           ** UndefManifold: Undefined manifold M3

In[63]:= ?M3

      Information::notfound : Symbol M3 not found. More…

In[64]:= $Manifolds

Out[64]= {S2}
```

And redefined:

```
In[65]:= DefManifold[M3, 3, {a, b, c, d, e, f, g, h}]

           ** DefManifold: Defining manifold M3.

           ** DefVBundle: Defining vbundle TangentM3.

In[66]:= $Manifolds

Out[66]= {S2, M3}
```

1.2. Product manifolds

Given two manifolds, it is always possible to define another manifold as the Cartesian product of those. The tangent bundle of the product will be then constructed as the direct sum of the tangent bundles of the submanifolds.

We define a product manifold as

```
In[67]:=  DefManifold[M5, {M3, S2}, {μ, ν, λ, σ, η, ρ}]

          ** DefManifold: Defining manifold M5.

          ** DefVBundle: Defining direct-sum vector bundle TangentM5.

In[68]:=  ?M5

          Global‘M5

          DimOfManifold[M5] ^= 5

          Info[M5] ^= {manifold, }

          ManifoldQ[M5] ^= True

          ObjectsOf[M5] ^= {}

          PrintAs[M5] ^= M5

          ServantsOf[M5] ^= {TangentM5}

          M5 /: SubmanifoldQ[M5, M3] = True

          M5 /: SubmanifoldQ[M5, S2] = True

          SubmanifoldsOfManifold[M5] ^= {M3, S2}

          TangentBundleOfManifold[M5] ^= TangentM5
```

whose tangent bundle is a direct sum of the corresponding subvbundles:

```
In[69]:=  ?TangentM5

             Global‘TangentM5

             BaseOfVBundle[TangentM5] ^= M5

             Dagger[TangentM5] ^= TangentM5

             DimOfVBundle[TangentM5] ^= 5

             IndicesOfVBundle[TangentM5] ^= {{μ, ν, λ, σ, η, ρ}, {}}

             Info[TangentM5] ^= {vbundle, }

             MasterOf[TangentM5] ^= M5

             MetricsOfVBundle[TangentM5] ^= {}

             ObjectsOf[TangentM5] ^= {}

             PrintAs[TangentM5] ^= TangentM5

             TangentM5 /: SubvbundleQ[TangentM5, TangentM3] = True

             TangentM5 /: SubvbundleQ[TangentM5, TangentS2] = True

             SubvbundlesOfVBundle[TangentM5] ^= {TangentM3, TangentS2}

             VBundleQ[TangentM5] ^= True
```

The list of product manifolds is given by another global variable:

```
In[70]:=  $ProductManifolds

Out[70]=  {M5}
```

Currently this is the only possible way of defining submanifolds of a manifold. That is, there is no code which allows you to define a submanifold of an already existing manifold. This is because that would pose the question of what is the complementary vector bundle.

1.3. Inner vector bundles

Starting in version 0.9, `xTensor‘` allows the definition of vector bundles other than tangent bundles. These ‘‘inner vbundles´´ can be real or complex, which forces the definition of the complex conjugated vbundle. Complex conjugation is controlled by the function `Dagger`.

`DefVBundle`	Define a vbundle or a sum of vbundles
`UndefVBundle`	Undefine a vbundle
`VBundleQ`	Check a vbundle
`$VBundles`	List of defined vbundles
`$SumVBundles`	List of defined sum vbundles

Definition of a vbundle.

Define a complex inner vbundle with dimension 4 (this is the fiber dimension) and base manifold `M3`:

```
In[71]:=  DefVBundle[InnerC, M3, 4, {𝔄, 𝔅, ℭ, 𝔇, 𝔈, 𝔉, 𝔊, ℌ}, Dagger → Complex]

          ** DefVBundle: Defining vbundle InnerC.

          ** DefVBundle: Defining conjugated vbundle InnerC†.
           Assuming fixed anti-isomorphism between InnerC and InnerC†

In[72]:=  ?InnerC

          Global`InnerC

          BaseOfVBundle[InnerC] ^= M3

          Dagger[InnerC] ^= InnerC†

          DimOfVBundle[InnerC] ^= 4

          IndicesOfVBundle[InnerC] ^= {{𝔄, 𝔅, ℭ, 𝔇, 𝔈, 𝔉, 𝔊, ℌ}, {}}

          Info[InnerC] ^= {vbundle, }

          MetricsOfVBundle[InnerC] ^= {}

          ObjectsOf[InnerC] ^= {}

          PrintAs[InnerC] ^= InnerC

          ServantsOf[InnerC] ^= {InnerC†}

          SubvbundlesOfVBundle[InnerC] ^= {}

          VBundleQ[InnerC] ^= True
```

In the process of definition, the conjugated vbundle `InnerC†`, with corresponding conjugated indices, has been created. It is a servant of `InnerC`, and hence can only be undefined through undefinition of the latter. By default, all conjugated symbols are constructed by appending the dagger character †. (This character is stored in the goblal variable `$DaggerCharacter`, which can be changed.)

```
In[73]:=  ?InnerC†

           Global‘InnerC†

           BaseOfVBundle[InnerC†] ^= M3

           Dagger[InnerC†] ^= InnerC

           DimOfVBundle[InnerC†] ^= 4

           IndicesOfVBundle[InnerC†] ^= {{𝔄†, 𝔅†, ℭ†, 𝔇†, 𝔈†, 𝔉†, 𝔊†, ℌ†}, {}}

           Info[InnerC†] ^= {conjugated vbundle,
             Assuming fixed anti-isomorphism between InnerC and InnerC†}

           MasterOf[InnerC†] ^= InnerC

           MetricsOfVBundle[InnerC†] ^= {}

           ObjectsOf[InnerC†] ^= {}

           PrintAs[InnerC†] ^= InnerC†

           SubvbundlesOfVBundle[InnerC†] ^= {}

           VBundleQ[InnerC†] ^= True
```

We have already defined five different vector bundles, only one of them being direct sum of others:

```
In[74]:=  $VBundles

Out[74]=  {TangentS2, TangentM3, TangentM5, InnerC, InnerC†}

In[75]:=  $SumVBundles

Out[75]=  {TangentM5}
```

1.4. Parameters

In `xTensor‘` the indexed objects can be fields on manifolds and/or functions of real parameters (for example proper time).

`DefParameter`	Define a real parameter
`UndefParameter`	Undefine a parameter
`ParameterQ`	Check a real parameter
`$Parameters`	List of defined parameters

Definition of a parameter

We define a real parameter that will be used below to compute parametric derivatives (see `ParamD` in subsection 6.4).

```
In[76]:= DefParameter[time]

          ** DefParameter: Defining parameter time.

In[77]:= ?time

         Global`time

          Info[time] ^= {parameter, }

          ObjectsOf[time] ^= {}

          ParameterQ[time] ^= True

          PrintAs[time] ^= time
```

Manifolds and parameters are the only possible "dependencies" of the objects in `xTensor`'. We shall later explain this in more detail.

An important question is the relation between fields on 1d manifolds and functions of a parameter. In decompositions of ADM−type it would be nice to have a way of going from a 4d structure to a 3+1 structure and then from here to a 3d structure plus a time parameter. I have no clue on how to program this idea. There are many things to change in a nontrivial way: tensor fields, covariant derivatives, metrics, etc. Help, please!

■ 2. Indices

2.0. Important comments

1) Indices are the key objects in `xTensor`', and the source of the generality of the system. They are carriers of contex−tual information, much like the decimal digits in numerical computations, or the letters in a text. An index by itself has little information (only an association to a vector space), but a group of indices together can encode a lot of information (rank, symmetries, metrics, bases, etc). In general, an index is anything which can be placed at an index−slot: that is, we shall later define four places in which we can put indices and anything which we can meaningfully put there will be called an index.

2) Starting version 0.9 of `xTensor`', indices are associated to vbundles, and not to manifolds, as was the case in previous versions. This allows us to work with vbundles other than tangent bundles to manifolds.

3) We need a "language" to talk about indices, and though we usually understand what a dummy index or a covariant index is, such a language is not rigorously defined anywhere. I have invented my own language for this. Please, suggest changes if you find inconsistencies, gaps or bad choices.

4) Taken from the Internet:

The plural of "index" for books, journals and newspapers is "indexes"; the plural of "index" for graphs and benchmarks for economics for example, is "indices"; but the plural of "appendix" is always "appendices".

I think the "index" of a tensor must follow the rule for the second group, and hence its plural will be "indices".

2.1. Properties of indices: type, character, state

1) There are five known types of index in `xTensor`` (other types could be added, but there is no user−defined way to do this):

− Abstract index (`AIndex`). See subsection 2.2.

− Basis index (`BIndex`), associated to some basis (coordinated or not) of vector fields. Described in `xCoba``.

− Component index (`CIndex`), also associated to some basis. Described in `xCoba``.

− Directional index (`DIndex`). See subsection 3.7.

− Label index (`LIndex`). See subsection 3.8.

Collectively we refer to all five classes as "generalized indices" (`GIndex`). Additionally, we can have patterns (`PIndex`) for g−indices at those positions where g−indices are expected, but patterns are not considered as g−indices. In the following "indices" will mean "g−indices" unless we specify the type.

2) All indices have a "character", which can be either covariant (`Down`−index) or contravariant (`Up`−index).

3) Some indices can be used to represent contractions, following the Einstein convention: two repeated indices in the same tensor or tensor product, but each having a different character. We call them "contractible" or "Einstein" indices (`EIndex`) and currently only abstract and basis indices are allowed to be contractible. (Note that having contractible indices with the same character is a syntactic error in `xTensor``.) E−indices have a "state": they can be `Free` or contracted (aka `Dummy`). For completeness, the state of components, directions and labels is said to be always "`Blocked`", what in particular means that they can be repeated (i.e. truly repeated, not simply staggered).

4) Basis and component indices are known by part of the routines of `xTensor`` (for example the formatting routines, the index selectors, the canonicalizer or the constructors of rules), but used in the twin package `xCoba``. From now on we shall not consider those two types of indices. See `xCobaDoc.nb` for further information.

5) There is a further property of an index, only used internally: its metric−state, saying whether an index can be raised or lowered using a given metric (not whether it has been or not actually raised already). This will be important in section 7.3.

2.2. Covariant and contravariant abstract indices

Abstract indices in `xTensor`` are valid atomic symbols without numeric value (excluding therefore `E` and `I`) and with no special output (excluding symbols like `Space`, `Tab`, etc.) One of the most important decisions when working with tensors is the notation for covariant and contravariant indices. Given the binary character of the problem there is only one natural simple choice: the use of signs `+` and `−` . We shall denote contravariant indices as `+a` or `a`, and covariant indices as `−a`. This choice is very simple, but not particularly convenient when doing pattern matching, due to the asymmetry of `a` being a symbol but `−a` being a composite expression.

`AIndexQ`	Validate an abstract index
`VB`Q`	Check that an abstract up−index belongs to vbundle `VB`
`VB`pmQ`	Check that an abstract (up− or down−) index belongs to vbundle `VB`

Validation of indices.

We can check whether a symbol has been registered as an abstract index using the function `AIndexQ`

```
In[78]:= AIndexQ[A]

Out[78]= True
```

```
In[79]:=  AIndexQ[-b]

Out[79]=  True

In[80]:=  AIndexQ[q]

Out[80]=  False
```

In particular, we can specify the vbundle it must belong to:

```
In[81]:=  AIndexQ[A, TangentS2]

Out[81]=  True

In[82]:=  AIndexQ[𝔄, TangentM3]

Out[82]=  False

In[83]:=  AIndexQ[𝔄, InnerC]

Out[83]=  True

In[84]:=  AIndexQ[𝔄†, InnerC]

Out[84]=  False

In[85]:=  AIndexQ[𝔄†, InnerC†]

Out[85]=  True
```

Each vbundle has two special functions that select abstract indices on that manifold. For example for the vbundle `TangentM3` they are `TangentM3‘Q` and `TangentM3‘pmQ`. The former only accepts contravariant indices; the latter accepts both characters. The use of context notation (the ‘) is purely historical : before using upvalues to store information I tried to use contexts, but it is less convenient.

```
In[86]:=  TangentM3‘Q[a]

Out[86]=  True

In[87]:=  TangentM3‘Q[-a]

Out[87]=  False

In[88]:=  TangentM3‘Q[A]

Out[88]=  False

In[89]:=  TangentM3‘pmQ[a]

Out[89]=  True

In[90]:=  TangentM3‘pmQ[-a]

Out[90]=  True
```

2.3. Get (new) indices

Sometimes we need to use many indices. There are two ways to do it. We can register more symbols using `AddIndices` or we can ask `xTensor`` to generate new indices for a vbundle using `NewIndexIn` (it uses a combination of the last registered symbol and integer numbers). User−defined indices and computer−defined indices are kept in two different lists.

`AddIndices`	Add user−defined indices to a vbundle
`RemoveIndices`	Remove user−defined indices from a vbundle
`NewIndexIn`	Generate a computer−defined index
`GetIndicesOfVBundle`	Give `n` indices

Functions that return or change the available indices.

Currently there are only user−defined indices. We add some more:

```
In[91]:= IndicesOfVBundle[TangentM3]

Out[91]= {{a, b, c, d, e, f, g, h}, {}}

In[92]:= AddIndices[TangentM3, {i, j, k, l}]

In[93]:= IndicesOfVBundle[TangentM3]

Out[93]= {{a, b, c, d, e, f, g, h, i, j, k, l}, {}}
```

We can also remove indices, but this is a very dangerous operation because some previous expressions could get corrupted if they contain removed indices. There is no built−in command to check whether this is happening or not.

```
In[94]:= RemoveIndices[TangentM3, {j, i, l, k}]

In[95]:= IndicesOfVBundle[TangentM3]

Out[95]= {{a, b, c, d, e, f, g, h}, {}}
```

`xTensor`` generates new indices using `NewIndexIn`

```
In[96]:= NewIndexIn[TangentM3]

Out[96]= h1

In[97]:= IndicesOfVBundle[TangentM3]

Out[97]= {{a, b, c, d, e, f, g, h}, {h1}}
```

The user−function to get any number of indices is `GetIndicesOfVBundle`. Note that the last argument is a list of symbols that we do not want to get:

```
In[98]:= GetIndicesOfVBundle[TangentM3, 14, {d, e, h3}]

Out[98]= {a, b, c, f, g, h, h1, h2, h4, h5, h6, h7, h8, h9}
```

```
In[99]:= IndicesOfVBundle[TangentM3]

Out[99]= {{a, b, c, d, e, f, g, h}, {h1, h2, h3, h4, h5, h6, h7, h8, h9}}
```

The indices of complex conjugated vbundles are paired, and therefore if we generate an index in one of them, automatically the complex conjugated index is generated in the other one:

```
In[100]:=
      NewIndexIn[InnerC]

Out[100]=
      ℋ1

In[101]:=
      IndicesOfVBundle[InnerC]

Out[101]=
      {{𝒜, ℬ, 𝒞, 𝒟, ℰ, ℱ, 𝒢, ℋ}, {ℋ1}}

In[102]:=
      IndicesOfVBundle[InnerC†]

Out[102]=
      {{𝒜†, ℬ†, 𝒞†, 𝒟†, ℰ†, ℱ†, 𝒢†, ℋ†}, {ℋ1†}}

In[103]:=
      GetIndicesOfVBundle[InnerC†, 10]

Out[103]=
      {𝒜†, ℬ†, 𝒞†, 𝒟†, ℰ†, ℱ†, 𝒢†, ℋ1†, ℋ†, ℋ2}

In[104]:=
      IndicesOfVBundle[InnerC]

Out[104]=
      {{𝒜, ℬ, 𝒞, 𝒟, ℰ, ℱ, 𝒢, ℋ}, {ℋ1, ℋ2}}
```

Note the information associated to an abstract index:

```
In[105]:=
      ?𝒜

         Global‘𝒜

          AbstractIndexQ[𝒜] ^= True

          DaggerIndex[𝒜] ^= 𝒜†

          xAct‘xTensor‘Private‘NoDollar[𝒜] ^= 𝒜

          PrintAs[𝒜] ^= 𝒜

          InnerC‘Q[𝒜] ^= True

          VBundleOfIndex[𝒜] ^= InnerC
```

3. Tensors and tensor slots

3.1. Define a tensor

Now we can define tensors. In `xTensor`` we always work with tensor fields on zero, one or several manifolds, having indices on zero, one or several of their vbundles.

`DefTensor`	Define a tensor
`UndefTensor`	Undefine a tensor
`xTensorQ`	Check a tensor
`$Tensors`	List of defined tensors

Definition of tensors.

Define a tensor field `T` on `M3` with two contravariant indices and one covariant index, and two other tensors. The actual indices used in the definition are irrelevant, and are only meant to specify the vbundle associated to each slot

```
In[106]:=
      DefTensor[T[a, b, -c], M3]

          ** DefTensor: Defining tensor T[a, b, -c].

In[107]:=
      DefTensor[S[a, b], M3]

          ** DefTensor: Defining tensor S[a, b].

In[108]:=
      DefTensor[v[-a], M3]

          ** DefTensor: Defining tensor v[-a].
```

Define a scalar field `r` (do not forget the empty pair of brackets!)

```
In[109]:=
      DefTensor[r[], M3]

          ** DefTensor: Defining tensor r[].
```

Define a tensor `U` on `M3` and `S2` with two contravariant indices and one covariant index, The tensor is also a function of the parameter time:

```
In[110]:=
      DefTensor[U[a, b, -C], {M3, S2, time}]

          ** DefTensor: Defining tensor U[a, b, -C].
```

The tensor is identified by a symbol and we shall associate to that symbol all definitions related to the tensor. This is internally done using `UpSet` (^=) rather than `Set` (=). The definitions can then be collected using `?`

```
In[111]:=
      ?U

         Global‘U

          Dagger[U] ^= U

          DependenciesOfTensor[U] ^= {time, M3, S2}

          Info[U] ^= {tensor, }

          PrintAs[U] ^= U

          SlotsOfTensor[U] ^= {TangentM3, TangentM3, -TangentS2}

          SymmetryGroupOfTensor[U] ^= StrongGenSet[{}, GenSet[]]

          TensorID[U] ^= {}

          xTensorQ[U] ^= True
```

A tensor can be undefined. Its properties are lost, and the symbol is removed.

```
In[112]:=
      UndefTensor[U]

          ** UndefTensor: Undefined tensor U

In[113]:=
      ?U

      Information::notfound : Symbol U not found. More…
```

An object with indices on a given vbundle is considered to be necessarily a field on the base manifold of that vbundle. `xTensor‘` checks that point. See the discussion below on Constant objects.

The tensor `U` with indices on vbundles `TangentM3` and `TangentS2` is defined to live only on `M3`. `xTensor‘` adds the manifold `S2` to the list `DependenciesOfTensor[U].`

```
In[114]:=
      DefTensor[U[a, A], M3]

          ** DefTensor: Defining tensor U[a, A].
```

```
In[115]:=
     ?U

        Global‘U

        Dagger[U] ^= U

        DependenciesOfTensor[U] ^= {M3, S2}

        Info[U] ^= {tensor, }

        PrintAs[U] ^= U

        SlotsOfTensor[U] ^= {TangentM3, TangentS2}

        SymmetryGroupOfTensor[U] ^= StrongGenSet[{}, GenSet[]]

        TensorID[U] ^= {}

        xTensorQ[U] ^= True

In[116]:=
     UndefTensor[U]

        ** UndefTensor: Undefined tensor U
```

3.2. Standard output

It is important to have nice−looking output expressions, mainly when we deal with very large expressions. `xTensor‘` has built−in rules to convert tensors and derivatives into 2−dimensional boxes for `StandardForm` output. Those boxes can be cut and pasted, using `InterpretationBox`. Additionally, translation to LATEX will be shown below.

The output of tensors in `StandardForm` has been defined with 2−dimensional boxes, but the internal structure is kept in 1−dimensional form (as usual in *Mathematica*):

```
In[117]:=
     {T[a, b, -c], S[a, b], v[-a], r[]}
```

Out[117]=

$\{T^{ab}{}_{c}, S^{ab}, v_{a}, r\}$

```
In[118]:=
     InputForm[%]

Out[118]//InputForm=
     {T[a, b, −c], S[a, b], v[−a], r[]}
```

No rules have been defined for `OutputForm` because the output is simple enough:

```
In[119]:=
     OutputForm[%%]

Out[119]//OutputForm=
     {T[a, b, −c], S[a, b], v[−a], r[]}
```

The ouput representation of the heads can be given at definition time, or changed later. Again, this is only possible in `StandardForm`. When using this option, it is very important to remember that the input and output names of the same tensor are different!

`PrintAs`	Option of `DefType` functions to define the output of a head

Output of defined heads.

The tensor `TT` will be output as τ. `PrintAs` must give a string or a function that applied on the defined symbol gives a string

```
In[120]:=
      DefTensor[TT[a, -b], M3, PrintAs → "τ"]

         ** DefTensor: Defining tensor TT[a, -b].

In[121]:=
      TT[a, -c]

Out[121]=
```
$\tau^{a}{}_{c}$

```
In[122]:=
      InputForm[%]

Out[122]//InputForm=
     TT[a, -c]
```

As an example of a printas function we can use:

```
In[123]:=
      FirstCharacter[symbol_] := First[Characters[ToString[symbol]]]

In[124]:=
      DefTensor[Force[a], M3, PrintAs → FirstCharacter]

         ** DefTensor: Defining tensor Force[a].

In[125]:=
      Force[a]

Out[125]=
```
F^{a}

The 2−dimensional boxes are based on a combination of three built−in's:

`SubsuperscriptBox`, which construct the sub/super−indices structures,

`StyleBox`, which arranges the indices with the proper interspaces, and

`InterpretationBox`, which stores both the box−form and the input−form of the expression. Note the option Editable−>−False, which prevents desynchronization.

```
In[126]:=
      ToBoxes[%]

Out[126]=
      InterpretationBox[
       StyleBox[SubsuperscriptBox[F,  , a], AutoSpacing → False], Fa, Editable → False]
```

We can cut−and−paste the output, but not edit it, nor construct a tensor directly in box−form:

```
In[127]:=
      F^a

In[127]:=
      InputForm[%]

Out[127]//InputForm=
     InterpretationBox[StyleBox[\(F\_\ \%a\), Rule[AutoSpacing, False]], Force[a],
     Rule[Editable, False]]
```

3.3. Expressions and validation

Once we use abstract indices, there is no need to have a special TensorTimes. The usual `Times` product is enough and has one important advantage: we can use many built−in rules in *Mathematica* to simplify `Times` expressions. As usual, contractions are given using Einstein rule of staggered up/down repeated indices.

The order of factors is irrelevant in `Times` expressions:

```
In[128]:=
      v[-a] T[a, b]
```

Out[128]=

$T^{ab}\ v_a$

```
In[129]:=
      v[-a] T[a, b] + T[a, b] v[-a]
```

Out[129]=

$2\ T^{ab}\ v_a$

The syntax of any expression can be checked using the function `Validate`. This function works recursively from the most external structures to the most internal ones, calling private functions `ValidateTensor`, `ValidateCovD`, etc. That gives you an idea of which problems will be detected first.

`Validate`	Validate an input expression

Functions that deal with validation.

If we get no errors, the input is syntactically correct:

```
In[130]:=
      Validate[T[a, b, -c] S[c, d] v[-d] + r[]^3 S[a, b]]
```

Out[130]=

$r^3\ S^{ab} + S^{cd}\ T^{ab}{}_c\ v_d$

```
In[131]:=
      Validate[T[a, b, c] S[-c, d] v[-d] + r[]^3 S[a, b]]

      Validate::error : Invalid character of index in tensor T

      Validate::error : Invalid character of index in tensor S

Out[131]=
```
$r^3\, S^{ab} + \mathrm{ERROR}[S_c{}^d]\, \mathrm{ERROR}[T^{abc}]\, v_d$

Some errors cannot be always localized in a single object. We do not return the expression:

```
In[132]:=
      Validate[T[a, b, -c] v[-c]]

      Validate::repeated : Found indices with the same name -c.

In[133]:=
      Validate[S[a, b] v[-b] + v[-a]]

      Validate::inhom : Found inhomogeneous indices: {{-a}, {a}}.
```

Because validation takes up some time, it is not automatic. `xTensor‘` has the command `Validate` to be used by the user as required. However, if validation is needed in all computations it is possible to switch on automatic validation, by acting on the `$Pre` variable:

Here, there is not automatic validation:

```
In[134]:=
      S[a, b] v[b]

Out[134]=
```
$S^{ab}\, v^b$

```
In[135]:=
      Validate[%]

      Validate::repeated : Found indices with the same name b.
```

This switches on automatic validation:

```
In[136]:=
      $Pre

Out[136]=
      $Pre

In[137]:=
      $Pre = Validate;

In[138]:=
      S[a, b] v[b]

      Validate::repeated : Found indices with the same name b.
```

This switches off automatic validation:

```
In[139]:=
      $Pre =.

In[140]:=
      S[a, b] v[b]

Out[140]=
      S^ab v^b

In[141]:=
      UndefTensor[S]

          ** UndefTensor: Undefined tensor S
```

3.4. Symmetries

There are two kinds of symmetries for a tensor. We can have monoterm (or permutation) symmetries, such that a tensor remains equal or changes sign under a permutation of its indices. Or we can have multiterm (for example cyclic) symmetries, such that a linear combination of several of those permuted tensors is equal to zero. In the former case the symmetry of a tensor of n indices can be described as a subgroup of the group {1,−1}x S_n. In the latter, the symmetry can be described as an algebra. Currently `xTensor‘` can deal with an arbitrary case (n not too large: say 200 indices at most) of monoterm symmetries. However, it does not implement any general algorithm for multiterm symmetries, because no efficient algorithm is known for that case.

The twin package `xPerm‘` is used to work with permutations. See `xPermDoc.nb` for further information.

`GenSet`	Generating set of a permutations group
`StrongGenSet`	Strong generating set of a permutations group
`Symmetric`	Completely symmetric permutation group
`Antisymmetric`	Completely antisymmetric permutation group
`RiemannSymmetric`	Group of symmetry of the Riemann tensor
`SymmetryOf`	Symmetry properties of an expression.
`ForceSymmetries`	Option of `DefTensor` to allow for symmetries among up/down indices

Functions related to symmetries

Any permutation group on a set of n points can be given using a set of generators (head `GenSet`) or, better, a strong set of generators (head `StrongGenSet`). The functions `Symmetric` and `Antisymmetric` are examples:

```
In[142]:=
      Symmetric[{a, b}]

Out[142]=
      StrongGenSet[{a}, GenSet[Cycles[{a, b}]]]

In[143]:=
      Antisymmetric[{a, b}]

Out[143]=
      StrongGenSet[{a}, GenSet[-Cycles[{a, b}]]]
```

Define a symmetric tensor S on M3

```
In[144]:=
     DefTensor[S[a, b], M3, Symmetric[{a, b}]]

         ** DefTensor: Defining tensor S[a, b].

In[145]:=
     ?S

        Global`S

         Dagger[S] ^= S

         DependenciesOfTensor[S] ^= {M3}

         Info[S] ^= {tensor, }

         PrintAs[S] ^= S

         SlotsOfTensor[S] ^= {TangentM3, TangentM3}

         SymmetryGroupOfTensor[S] ^= StrongGenSet[{1}, GenSet[Cycles[{1, 2}]]]

         TensorID[S] ^= {}

         xTensorQ[S] ^= True
```

Define a totally antisymmetric tensor U on M3. We can use the names of the indices or their positions in the tensor

```
In[146]:=
     DefTensor[U[-a, -b, -c], M3, Antisymmetric[{1, 2, 3}]]

         ** DefTensor: Defining tensor U[-a, -b, -c].

In[147]:=
     ?U

        Global`U

         Dagger[U] ^= U

         DependenciesOfTensor[U] ^= {M3}

         Info[U] ^= {tensor, }

         PrintAs[U] ^= U

         SlotsOfTensor[U] ^= {-TangentM3, -TangentM3, -TangentM3}

         SymmetryGroupOfTensor[U] ^=
          StrongGenSet[{1, 2}, GenSet[-Cycles[{1, 2}], -Cycles[{2, 3}]]]

         TensorID[U] ^= {}

         xTensorQ[U] ^= True
```

For example, for the tensor `U` the group is generated by the two signed permutations `-Cycles`[{1,2}] (representing a change of sign under the exchange of slots 1 and 2) and `-Cycles`[{2,3}] (representing a change of sign under the exchange of slots 2 and 3).

Note however that not every generating set is valid:

For example, in a group of three indices, antisymmetry in the first two indices is not consistent with symmetry in the second and third indices, unless the whole tensor is zero:

```
In[148]:=
      Catch@DefTensor[W[a, b, c], M3, GenSet[-Cycles[{1, 2}], Cycles[{2, 3}]]]

      SchreierOrbit::infty : Found Infinity as a point, with generating set GenSet[].

      DefTensor::invalid :
       GenSet[-Cycles[{1, 2}], Cycles[{2, 3}]] is not a valid symmetry identification.
```

The symmetries must be consistent with the character of the indices (irrespectively of whether there is a metric or not). This is checked.

We cannot define a symmetric tensor with two indices of different character:

```
In[149]:=
      Catch@DefTensor[Q[a, -b], M3, Symmetric[{1, 2}]]

      DefTensor::wrongsym : Symmetry properties are inconsistent with indices of tensor.
```

unless we force it (so that the check is not performed):

```
In[150]:=
      DefTensor[Q[a, -b], M3, Symmetric[{1, 2}], ForceSymmetries → True]

         ** DefTensor: Defining tensor Q[a, -b].
```

For symmetry groups of order larger than 10000 the check would take too long and is not performed:

```
In[151]:=
      DefTensor[supereta[a, b, c, d, e, -f, -g, -h, -h1], M3, Antisymmetric[Range[8]]]

           Order of group of symmetry: 40320 > 10000. ForceSymmetry check not performed.

         ** DefTensor: Defining tensor supereta[a, b, c, d, e, -f, -g, -h, -h1].

In[152]:=
      UndefTensor /@ {Q, supereta};

         ** UndefTensor: Undefined tensor Q

         ** UndefTensor: Undefined tensor supereta
```

3.5. delta and Gdelta

The identity tensor on vbundles is called `delta` and is the same for all vbundles. This has changed from previous versions of `xTensor‘`, in which each vbundle had its own identity tensor.

The `delta` tensor always has two indices with different character. However, their position is irrelevant and actually many authors (e.g. Penrose & Rindler) write its indices one on top of the other, rather than staggered. It is not natural to do that within the notation of `xTensor‘`. Instead, we shall define the `delta` tensor as symmetric, even though that really makes no sense because the indices never have the same character: a `delta` tensor with two indices of the same character is inmediately converted into the (first) metric of the corresponding vbundle.

`delta`	Identity tensor on every vbundle
`Gdelta`	Generalized `delta` tensor on every vbundle
`ExpandGdelta`	Expansion of `Gdelta` in products of `delta`’s

Identity tensors.

The `delta` tensor on the the vector bundle `TangentM3`:

```
In[153]:=
      delta[a, -b]

Out[153]=
```
$\delta^{a}{}_{b}$

and it is defined as symmetric:

```
In[154]:=
      SymmetryGroupOfTensor[delta]

Out[154]=
      StrongGenSet[{1}, GenSet[Cycles[{1, 2}]]]
```

There is automatic simplification with most of the expected rules for the `delta` tensors:

```
In[155]:=
      delta[a, -a]

Out[155]=
      3

In[156]:=
      delta[a, -b] delta[-c, b]

Out[156]=
```
$\delta_{c}{}^{a}$

```
In[157]:=
      delta[-b, a] v[-a]

Out[157]=
```
v_{b}

When both indices can be contracted, the second index of `delta` is contracted first:

```
In[158]:=
      delta[a, -b] S[-a, b]
```

Out[158]=

$S_a{}^a$

The indices of `delta` can never have the same character, unless automatic conversion to a metric is possible:

```
In[159]:=
      delta[a, b] // Catch

      MetricsOfVBundle::missing : There is no metric in TangentM3.
```

In version 0.9.0 `xTensor‘` has added a generalized delta tensor.

This is the generalized tensor with 2 indices:

```
In[160]:=
      Gdelta[a, b, -c, -d]
```

Out[160]=

$\delta^{ab}{}_{cd}$

It is antisymmetric in the first half of indices and separately antisymmetric in the second half of indices:

```
In[161]:=
      {Gdelta[b, a, -c, -d], Gdelta[a, b, -d, -c], Gdelta[-d, -c, b, a]} // ToCanonical
```

Out[161]=

$\{-\delta^{ab}{}_{cd}, -\delta^{ab}{}_{cd}, \delta^{ab}{}_{cd}\}$

Staggered indices are automatically contracted:

```
In[162]:=
      Gdelta[a, b, c, -a, -b, -c]

Out[162]=
      6

In[163]:=
      Gdelta[a, b, -a, c, -d, -e]

Out[163]=
      0

In[164]:=
      Gdelta[a, b, c, -d, -e, -c]
```

Out[164]=

$\delta^{ab}{}_{de}$

It is always possible to convert it into a determinant of normal delta tensors:

```
In[165]:=
      ExpandGdelta[%]
```

Out[165]=

$$-\delta^{a}{}_{e}\,\delta^{b}{}_{d}+\delta^{a}{}_{d}\,\delta^{b}{}_{e}$$

```
In[166]:=
      Gdelta[a, -b] // InputForm

Out[166]//InputForm=
     delta[a, -b]
```

3.6. Directional indices

From the abstract point of view, covariant (contravariant) indices of tensors represent slots where vectors (covectors) can be "contracted". It is interesting sometimes to write that contraction explicitly: `xTensor‘` uses the head `Dir` to denote that kind of "directional index". The introduction of a special head is a technical decision: we expect this feature to be seldom used and therefore it makes no sense to force `xTensor‘` to check continuously whether a given slot contains an abstract index or a vector. All definitions concerning directional−indices are associated to `Dir`, and hence they are not even considered unless the head `Dir` is found.

`Dir`	Head denoting directional slots in tensors

Directional indices.

This is the usual representation for a contraction:

```
In[167]:=
      v[-b] S[a, b]
```

Out[167]=

$$S^{ab}\,v_{b}$$

But it can also be given as (note the different results of the validation routine below)

```
In[168]:=
      S[a, Dir[v[-c]] ]
```

Out[168]=

$$S^{av}$$

```
In[169]:=
      Validate[%]
```

Out[169]=

$$S^{av}$$

Directional slots containing vectors are represented downstairs, and slots containing covectors are represented upstairs:

```
In[170]:=
      S[a, Dir[v[c]] ]
```

Out[170]=
$S^{a}{}_{v}$

```
In[171]:=
      Validate[%]

      Validate::error : Invalid character of index in tensor v

      Validate::error : Invalid character of index in tensor S
```

Out[171]=
ERROR[$S^{a}{}_{v}$]

```
In[172]:=
      U[a, b, Dir[v[-c]] ]
```

Out[172]=
U^{abv}

```
In[173]:=
      Validate[%]

      Validate::error : Invalid character of index in tensor U
```

Out[173]=
ERROR[U^{abv}]

It has the expected properties for a tensorial slot:

```
In[174]:=
      DefTensor[w[a], M3]

          ** DefTensor: Defining tensor w[a].

In[175]:=
      S[a, Dir[3 v[-c] + w[-c]] ]
```

Out[175]=
$3 S^{av} + S^{aw}$

```
In[176]:=
      UndefTensor[w]

          ** UndefTensor: Undefined tensor w
```

In particular, `xTensor‘` understands the whole expression as a vector, and not as a 2−tensor:

```
In[177]:=
      UpVectorQ[S[a, Dir[v[-c]]]]

Out[177]=
      True
```

It is important to realize that the free index `c` in `Dir[v[c]]` is completely irrelevant except for its sign and the tangent bundle it belongs to. It is there for consistency reasons: it identifies the argument of `Dir` as an upvector on `TangentM3`. I call those indices "ultraindices".

If you find annoying to write `Dir[v[c]]` many times, you can name this expression:

```
In[178]:=
      dirv = Dir[v[c]];
      S[a, dirv]
```

```
Out[179]=
```

$S^{a}{}_{v}$

Finally, there is a function to convert the Dir indices into normal contractions:

```
In[180]:=
      SeparateDir[%] // ScreenDollarIndices
```

```
Out[180]=
```

$S^{a}{}_{b}\, v^{b}$

3.7. Label indices

Sometimes we want to have indices with no tensorial meaning. For instance the labels l, m of the spherical harmonics can be defined as label indices. This is done using a new head `LI`, following similar ideas to those bringing us to the head `Dir`. There are no definitions associated to `LI`; the user can use `LI` for any purpose he likes.

`LI`	Head denoting label slots in tensors
`Labels`	A ficticious vbundle to which all label indices belong

Label indices.

We can define the spherical harmonics as

```
In[181]:=
      DefTensor[Y[LI@l, LI@m], S2]

          ** DefTensor: Defining tensor Y[LI[l], LI[m]].
```

```
In[182]:=
     ?Y

        Global`Y

        Dagger[Y] ^= Y

        DependenciesOfTensor[Y] ^= {S2}

        Info[Y] ^= {tensor, }

        PrintAs[Y] ^= Y

        SlotsOfTensor[Y] ^= {Labels, Labels}

        SymmetryGroupOfTensor[Y] ^= StrongGenSet[{}, GenSet[]]

        TensorID[Y] ^= {}

        xTensorQ[Y] ^= True
```

The up/down character of the labels only indicates the output position of the indices. Note that the signs inside `LI` are part of the label, and do not indicate the character.

```
In[183]:=
     Y[LI@L, LI@M]
```

```
Out[183]=
```
Y^{LM}

```
In[184]:=
     {Y[-LI[L], -LI[2]], Y[LI[-L], LI[-2]]}
```

```
Out[184]=
```
$\{Y_{L2}, Y^{-L-2}\}$

```
In[185]:=
     UndefTensor[Y]

        ** UndefTensor: Undefined tensor Y
```

3.8. Complex conjugation

Complex conjugation is performed by the function `Dagger`, except on indices, where we shall use the variant `Dagger-Index`, to avoid overloading `Dagger` too much.

`Dagger`	Complex conjugation of tensor expressions
`DaggerIndex`	Complex conjugation of indices
`DaggerQ`	Check for complex objects
`TransposeDagger`	Transposition of indices of complex conjugate vbundles
`$DaggerCharacter`	Character which denotes complex conjugates of other objects

Complex conjugation

A tensor with indices on a complex vbundle always has a dagger pair:

```
In[186]:=
      $Tensors

Out[186]=
      {T, v, r, TT, Force, S, U}

In[187]:=
      DefTensor[V[a, Ƀ], TangentM3, Dagger → Complex]

          ** DefTensor: Defining tensor V[a, Ƀ].

          ** DefTensor: Defining tensor V†[a, Ƀ†].

In[188]:=
      ?V

         Global`V

          Dagger[V] ^= V†

          DependenciesOfTensor[V] ^= {M3}

          Info[V] ^= {tensor, }

          PrintAs[V] ^= V

          ServantsOf[V] ^= {V†}

          SlotsOfTensor[V] ^= {TangentM3, InnerC}

          SymmetryGroupOfTensor[V] ^= StrongGenSet[{}, GenSet[]]

          TensorID[V] ^= {}

          xTensorQ[V] ^= True
```

```
In[189]:=
      ?V†

         Global`V†

          Dagger[V†] ^= V

          DependenciesOfTensor[V†] ^= {M3}

          Info[V†] ^= {tensor, }

          MasterOf[V†] ^= V

          PrintAs[V†] ^= V†

          SlotsOfTensor[V†] ^= {TangentM3, InnerC†}

          SymmetryGroupOfTensor[V†] ^= StrongGenSet[{}, GenSet[]]

          TensorID[V†] ^= {}

          xTensorQ[V†] ^= True
```

Complex conjugation acts on the name of the tensor and on its indices as well:

```
In[190]:=
      Dagger[ V[a, ℭ] ]

Out[190]=
      V†^{aℭ†}

In[191]:=
      UndefTensor[V]

          ** UndefTensor: Undefined tensor V†

          ** UndefTensor: Undefined tensor V
```

Conjugation of indices (this should be used only for programming):

```
In[192]:=
      DaggerIndex /@ {a, -b, 𝔄, -𝔅†}

Out[192]=
      {a, -b, 𝔄†, -𝔅}
```

Tensors can be defined with different properties against complex conjugation.

On real vbundles tensors are real by default:

```
In[193]:=
      Options[DefTensor]

Out[193]=
      {Dagger → Real, Master → Null, PrintAs → Identity,
       VanishingQ → False, ForceSymmetries → False, WeightOfTensor → 0,
       FrobeniusQ → False, OrthogonalTo → {}, ProjectedWith → {},
       ProtectNewSymbol :→ $ProtectNewSymbols, Info → {tensor, }, TensorID → {}}

In[194]:=
      DefTensor[V[a], M3]

          ** DefTensor: Defining tensor V[a].

In[195]:=
      Dagger[3 I V[a]]

Out[195]=
      -3 ⅈ V^a

In[196]:=
      UndefTensor[V]

          ** UndefTensor: Undefined tensor V
```

However, it is possible to "complexify" a real vbundle, in such a way that we duplicate its real dimension. Then vectors can have nontrivial complex conjugates:

```
In[197]:=
      DefTensor[V[a], M3, Dagger → Complex]

          ** DefTensor: Defining tensor V[a].

          ** DefTensor: Defining tensor V†[a].

In[198]:=
      Dagger[3 I V[a]]

Out[198]=
      -3 ⅈ V†^a

In[199]:=
      UndefTensor[V]

          ** UndefTensor: Undefined tensor V†

          ** UndefTensor: Undefined tensor V
```

If the vbundle is complex then all tensors on it must be complex:

```
In[200]:=
      Catch@DefTensor[V[𝔄], M3]

      DefTensor::invalid : Real is not a valid value for Dagger: complex indices.
```

```
In[201]:=
      DefTensor[V[𝔄], M3, Dagger → Complex]

         ** DefTensor: Defining tensor V[𝔄].

         ** DefTensor: Defining tensor V†[𝔄†].

In[202]:=
      Dagger[3 I V[𝔄]]

Out[202]=
```

$-3\, i\, V^{\dagger\,\mathfrak{A}\dagger}$

```
In[203]:=
      UndefTensor[V]

         ** UndefTensor: Undefined tensor V†

         ** UndefTensor: Undefined tensor V
```

On complex vbundles it is possible to define Hermitian tensors if they have equal number of indices on both conjugated vbundles (though there is no built−in operation of Hermitian conjugation):

```
In[204]:=
      DefTensor[V[𝔄, 𝔅, ℭ†, 𝔇†], M3, Dagger → Hermitian]

         ** DefTensor: Defining tensor V[𝔄, 𝔅, ℭ†, 𝔇†].

         ** DefTensor: Defining tensor V†[𝔄†, 𝔅†, ℭ, 𝔇].

In[205]:=
      Dagger[V[𝔄, 𝔅, ℭ†, 𝔇†]]

Out[205]=
```

$V^{\mathfrak{C}\mathfrak{D}\mathfrak{A}\dagger\mathfrak{B}\dagger}$

```
In[206]:=
      Dagger[V†[𝔄†, 𝔅†, ℭ, 𝔇]]

Out[206]=
```

$V^{\mathfrak{A}\mathfrak{B}\mathfrak{C}\dagger\mathfrak{D}\dagger}$

```
In[207]:=
      Catch@DefTensor[W[𝔄, 𝔅, ℭ†], M3, Dagger → Hermitian]

      TransposeDagger::error :
       Different number of indices of InnerC and its conjugate InnerC†.

In[208]:=
      UndefTensor[V];

         ** UndefTensor: Undefined tensor V†

         ** UndefTensor: Undefined tensor V
```

Note that the transposition involved for Hermitian objects does not require any particular ordering of indices of the tensors. However it follows the convention of exchanging the first slot of a given complex vbundle with the first slot of its complex conjugate:

```
In[209]:=
      DefTensor[V[𝔄, 𝔅, ℭ†, 𝔇†, 𝔈, 𝔉†], M3, Dagger → Hermitian]

          ** DefTensor: Defining tensor V[𝔄, 𝔅, ℭ†, 𝔇†, 𝔈, 𝔉†].

          ** DefTensor: Defining tensor V†[𝔄†, 𝔅†, ℭ, 𝔇, 𝔈†, 𝔉].
```

```
In[210]:=
      Dagger[V[𝔄, 𝔅, ℭ†, 𝔇†, 𝔈, 𝔉†]]
```

```
Out[210]=
```
$V^{\mathfrak{C}\mathfrak{D}\mathfrak{A}\dagger\mathfrak{B}\dagger\mathfrak{F}\mathfrak{E}\dagger}$

```
In[211]:=
      UndefTensor[V]

          ** UndefTensor: Undefined tensor V†

          ** UndefTensor: Undefined tensor V
```

The symbol for conjugation can be changed using the global variable `$DaggerCharacter`, but of course the command for complex conjugation will still be called `Dagger`.

```
In[212]:=
      $DaggerCharacter

Out[212]=
      †
```

3.9. Tensors with a variable number of slots

Sometimes it is desirable to work with tensors with a variable number of slots, and `xTensor‘` supports it. For example, this is important when dealing with tensor harmonics, and actually the special package `xAct‘Harmonics‘` makes use of it.

In general, once a tensor has been defined with a number of slots and the symmetry properties of those slots, it is possible to use the tensor with more slots, and the system will assume that there are no additional symmetry properties on the added slots. The problem, however, comes when we want all slots to participate in the symmetry of the tensor. The solution is very simple:

The symmetry of a tensor is given by the function `SymmetryGroupOfTensor`, which stores the symmetry as an upvalue for the name of the tensor. However, this function tries first to get the symmetry from the whole expression of the tensor (indices included), and only if this is not defined it will use the stored definition. Therefore we can give additional symmetry definitions for the tensor which will overwrite the initial symmetry assignment. From version 0.9.4 we include a special notation to indicate from the beginning the presence of a variable number of slots in vbundle.

Define a tensor with a variable number of indices on a given vbundle:

```
In[213]:=
      DefTensor[Z[AnyIndices@TangentM3], M3]

          ** DefTensor: No checks on indices for a variable-rank tensor.

          ** DefTensor: Defining tensor Z[AnyIndices[TangentM3]].
```

The tensor is given no symmetry:

```
In[214]:=
      Catch@SymmetryGroupOfTensor[Z]

      SymmetryGroupOfTensor::unknown : Unknown tensor Z.

In[215]:=
      Catch@SymmetryGroupOfTensor[Z[]]

      SymmetryGroupOfTensor::unknown : Unknown tensor Z.
```

Now make it symmetric for any number of indices:

```
In[216]:=
      SymmetryGroupOfTensor[Z[inds___]] ^:= Symmetric[Range[Length[{inds}]]]

In[217]:=
      SymmetryGroupOfTensor[Z[]]

Out[217]=
      StrongGenSet[{}, GenSet[]]

In[218]:=
      SymmetryGroupOfTensor[Z[a, b]]

Out[218]=
      StrongGenSet[{1}, GenSet[Cycles[{1, 2}]]]

In[219]:=
      SymmetryGroupOfTensor[Z[a, b, c, d]]

Out[219]=
      StrongGenSet[{1, 2, 3}, GenSet[Cycles[{1, 2}], Cycles[{2, 3}], Cycles[{3, 4}]]]
```

Note that any call to `SymmetryGroupOfTensor` must include explicitly the indices of the tensor. This is still undefined:

```
In[220]:=
      Catch@SymmetryGroupOfTensor[Z]

      SymmetryGroupOfTensor::unknown : Unknown tensor Z.
```

Clean up:

```
In[221]:=
      UndefTensor[Z]

          ** UndefTensor: Undefined tensor Z
```

In case you need symmetries other than total symmetry or total antisymmetry, you will need to use the general notation for permutation groups (see `xPermDoc.nb` for a description).

■ 4. Canonicalization

4.1. Basics

In any calculation it is essential to bring all tensors and products of tensors down to a canonical form, in order to be compared. This is done by the function `ToCanonical`, probably the most important ingredient of `xTensor‘`, and certainly the most complicated function.

Do not confuse the canonicalization proces, by which an expression is brought to a canonical form (not necessarily simple) from the simplification process, by which an expression is rewritten as an equivalent, but simpler form. The former is uniquely defined once the canonical form of every sintactycally correct expression is decided. The latter is largely subjective, and generally more difficult to work with. Fortunately *Mathematica* has a very good algorithm for simplification (`Simplify`). This is one of the main reasons to avoid introducing TensorTimes or TensorPlus, which would force us to build a new TensorSimplify, and another strong reason to use abstract index notation. In `xTensor‘` we have defined a command `Simplification`, which simply applies `ToCanonical` first, and then `Simplify`.

The hardest part of the process (the canonicalization of indices) is performed by the companion package `xPerm‘`. The canonicalization code is duplicated in that package: there is first a pure *Mathematica* version of the code, and then there is an external C−executable, which is much faster, but requires a *MathLink* connection, which does not work for some operating systems. Which code is used is controlled with several switches, as we will see later on.

`ToCanonical`	Canonicalization of indices of an expression
`Simplification`	Apply `ToCanonical` and then `Simplify`

Simplification functions

Canonical form of the totally antisymmetric tensor `U`: sort indices in alphabetical order:

```
In[222]:=
      ToCanonical[ U[-a, -d, -b] ]
```

Out[222]=
$-U_{abd}$

```
In[223]:=
      U[-a, -b, -c] + U[-c, -b, -a] // ToCanonical

Out[223]=
      0
```

Dummy indices are replaced so that the total number of different dummies is minimal:

```
In[224]:=
      T[a, b] v[-b] + T[a, c] v[-c] // ToCanonical
```

Out[224]=
$2\, T^{ab}\, v_b$

The process of canonicalization is far from trivial (see next section). We can monitor it using `Verbose` options.

`Verbose`	Gives a real−time report of the tensorial canonicalization process
`TimeVerbose`	Gives timings of several canonicalization steps
`xPermVerbose`	Gives a real−time report of the permutation canonicalization process

Verbosing options of `ToCanonical`.

These three cells show the output of the three `Verbose` options of `ToCanonical` in a simple case. The output of `xPermVerbose` can be extremely large even for small expressions. We do not use the external executable because there is not verbose information from it:

```
In[225]:=
      SetOptions[CanonicalPerm, MathLink → False]

Out[225]=
      {MathLink → False, TimeVerbose → False, xPermVerbose → False, OrderedBase → True}

In[226]:=
      ToCanonical[U[b, c, a] v[-b], TimeVerbose → True]

          Free algorithm applied in 0.004 secs.

          Dummy algorithm applied in 0.004 secs.

Out[226]=
```

$-U^{acb}\,v_b$

```
In[227]:=
     ToCanonical[U[b, c, a] v[-b], Verbose → True]

        ***********************************************************

        ToCanonical:: expr:
         xAct`xTensor`Private`TensorTimes[xAct`xTensor`Private`Object[
           v_b, {Tensor, 1, v}, {{-b}, {-b}, {}, {}, {{-TangentM3, Null}}}],
          xAct`xTensor`Private`Object[U^bca, {Tensor, 6, U}, {{b, c, a}, {a, b, c},
            {}, {}, {{TangentM3, Null}, {TangentM3, Null}, {TangentM3, Null}}}]]

        ToCanonical:: sym: Symmetry[4,
          xAct`xTensor`Private`TTimes[v^●1, U^●2●3●4], {●1 → -b, ●2 → b, ●3 → c, ●4 → a},
          StrongGenSet[{2, 3}, GenSet[-Cycles[{2, 3}], -Cycles[{3, 4}]]]]

        ToCanonicalOne:: Actual configuration: {-b, b, c, a}

        ToCanonicalOne:: Standard configuration: {a, c, b, -b}

        ToCanonicalOne:: Repeated indices: {}

        ToCanonicalOne:: Repeated indices: {}

        ToCanonicalOne:: Permutation to be canonicalized: Images[{4, 3, 2, 1}]

        ToCanonicalOne:: dummysets_tmp:
         {DummySet[TangentS2, {}, 0], DummySet[TangentM3, {b}, 0],
          DummySet[TangentM5, {}, 0], DummySet[InnerC, {}, 0], DummySet[InnerC†, {}, 0]}

        ToCanonicalOne:: dummysets:
         {DummySet[TangentS2, {}, 0], DummySet[TangentM3, {{3, 4}}, 0],
          DummySet[TangentM5, {}, 0], DummySet[InnerC, {}, 0], DummySet[InnerC†, {}, 0]}

        ToCanonicalOne:: Free indices: {1, 2}

        ToCanonicalOne:: calling: CanonicalPerm[Images[{4, 3, 2, 1}],
         4,StrongGenSet[{2, 3}, GenSet[-Cycles[{2, 3}], -Cycles[{3, 4}]]]
         ,{1, 2},{DummySet[TangentS2, {}, 0],
          DummySet[TangentM3, {{3, 4}}, 0], DummySet[TangentM5, {}, 0],
          DummySet[InnerC, {}, 0], DummySet[InnerC†, {}, 0]},Verbose → True]

        ToCanonicalOne:: Canonical permutation: -Images[{4, 1, 2, 3}]

        ToCanonical:: newindices: {-1, {-b, a, c, b}}

        ToCanonical:: needmetrics: False

        ToCanonical:: cQ: False

        ToCanonical:: result: -U^acb v_b

Out[227]=
     -U^acb v_b

In[228]:=
     ToCanonical[U[b, c, a] v[-b], xPermVerbose → True]

        Free indices at slots: {4, 3}

        RIGHT-COSET-REPRESENTATIVE ALGORITHM for Images[{4, 3, 2, 1}]

        which corresponds to the index list: Perm[{4, 3, 2, 1}]

        base: {1, 2, 3, 4}

        ****** Analysing element i=1 of base: slot 1 ******

        Symmetry orbit Delta of slots: {1}
```

```
Free slots: {4, 3}

Free slots that can go to that slot: {}

****** Analysing element i=2 of base: slot 2 ******

Symmetry orbit Delta of slots: {2, 3, 4}

Free slots: {4, 3}

Free slots that can go to that slot: {3, 4}

At those slots we respectively find indices Deltap: {2, 1}

The least index is 1, found at position pk: 2 of Deltap

That index is found in tensor at slot pp: 4

We can move slot 4 to slot 2 using permutation om: Images[{1, 4, 2, 3}] in S

New indices list: Perm[{4, 1, 3, 2}]

Computing stabilizer in S of slot 2

newbase before change: {1, 2, 3, 4}

newbase after change: {1, 3, 4}

****** Analysing element i=3 of base: slot 3 ******

Symmetry orbit Delta of slots: {3, 4}

Free slots: {2, 4}

Free slots that can go to that slot: {4}

At those slots we respectively find indices Deltap: {2}

The least index is 2, found at position pk: 1 of Deltap

That index is found in tensor at slot pp: 4

We can move slot 4 to slot 3 using permutation om: -Images[{1, 2, 4, 3}] in S

New indices list: -Perm[{4, 1, 2, 3}]

Computing stabilizer in S of slot 3

newbase before change: {1, 3, 4}

newbase after change: {1, 4}

Canonical Permutation after RightCosetRepresentative: -Images[{4, 1, 2, 3}]

DOUBLE-COSET-REPRESENTATIVE ALGORITHM for -Images[{4, 1, 2, 3}]

index-dummysets: {DummySet[TangentS2, {}, 0], DummySet[TangentM3, {{3, 4}}, 0],
  DummySet[TangentM5, {}, 0], DummySet[InnerC, {}, 0], DummySet[InnerC†, {}, 0]}

dummyindices: {3, 4}

dummyslots: {4, 1}

Extended {1, 4} to {1, 4}

Initial SGSS: StrongGenSet[{1, 4}, GenSet[]]

base for sorting: {3, 4}
```

```
Initial SGSD: StrongGenSet[{3}, GenSet[]]

****************** Loop i= 1 ********************

Analyzing slot 1 of tensor

nuS: Schreier[{1}, {2}, {3}, {4}, {0, 0, 0, 0}, {0, 0, 0, 0}] with first element 1

Under S, slot 1 can go to slots Deltab: {1}

Orbits of indices under D: DeltaD: {{1}, {2}, {3}, {4}}

With L={} we get sgd: -Images[{4, 1, 2, 3}]

which maps slots in Deltab to indices list: {4}

whose points belong to orbits {4}

Therefore at slot 1 we can have indices IMAGES: {4}

The least of them is p[[1]]: 4

Moved pairs {DummySet[TangentS2, {}, 0], DummySet[TangentM3, {{4, 3}}, 0],
  DummySet[TangentM5, {}, 0], DummySet[InnerC, {}, 0], DummySet[InnerC†, {}, 0]}
New SGS of D: StrongGenSet[{4}, GenSet[]]

with Schreier vector nuD: Schreier[{1}, {2}, {3}, {4}, {0, 0, 0, 0}, {0, 0, 0, 0}]

In particular, the orbit of index 4 is Deltap: {4}

Now looking for all permutations sgd that move index 4 to slot 1

Loop with l=1

L= {}

TAB[L] = {s, d} = {Images[{1, 2, 3, 4}], Images[{1, 2, 3, 4}]}

Calculating NEXT. We need the intersection of sets of slots {1} and {1}

Intermediate slots NEXT= {1}

From slot 1 to intermediate slot 1 use s1=Images[{1, 2, 3, 4}]

d1= Images[{1, 2, 3, 4}]

L1= {1}

This gives us the new index configuration: -Perm[{4, 1, 2, 3}]

New ALPHA: {{1}}

Checking consistency in set {-Images[{2, 3, 4, 1}]}

Removing permutations from SGS of S that move slot 1

New SGS of S: StrongGenSet[{4}, GenSet[]]

Removing permutations from SGS of D that move index 4

New SGS of D: StrongGenSet[{}, GenSet[]]

****************** Loop i= 2 ********************

Analyzing slot 4 of tensor

nuS: Schreier[{1}, {2}, {3}, {4}, {0, 0, 0, 0}, {0, 0, 0, 0}] with first element 4
```

```
Under S, slot 4 can go to slots Deltab: {4}

Orbits of indices under D: DeltaD: {{1}, {2}, {3}, {4}}

With L={1} we get sgd: -Images[{4, 1, 2, 3}]

which maps slots in Deltab to indices list: {3}

whose points belong to orbits {3}

Therefore at slot 4 we can have indices IMAGES: {3}

The least of them is p[[2]]: 3

Moved pairs {DummySet[TangentS2, {}, 0], DummySet[TangentM3, {}, 0],
  DummySet[TangentM5, {}, 0], DummySet[InnerC, {}, 0], DummySet[InnerC†, {}, 0]}

New SGS of D: StrongGenSet[{}, GenSet[]]

with Schreier vector nuD: Schreier[{1}, {2}, {3}, {4}, {0, 0, 0, 0}, {0, 0, 0, 0}]

In particular, the orbit of index 3 is Deltap: {3}

Now looking for all permutations sgd that move index 3 to slot 4

Loop with l=1

L= {1}

TAB[L] = {s, d} = {Images[{1, 2, 3, 4}], Images[{1, 2, 3, 4}]}

Calculating NEXT. We need the intersection of sets of slots {4} and {4}

Intermediate slots NEXT= {4}

From slot 4 to intermediate slot 4 use s1=Images[{1, 2, 3, 4}]

d1= Images[{1, 2, 3, 4}]

L1= {1, 4}

This gives us the new index configuration: -Perm[{4, 1, 2, 3}]

New ALPHA: {{1, 4}}

Checking consistency in set {-Images[{2, 3, 4, 1}]}

Removing permutations from SGS of S that move slot 4

New SGS of S: StrongGenSet[{}, GenSet[]]

Removing permutations from SGS of D that move index 3

New SGS of D: StrongGenSet[{}, GenSet[]]
```

Out[228]=

$-U^{acb}\, v_b$

The canonicalization of expressions involving a large number of indices is a slow process, mainly when most of those indices are dummies. In those cases the *Mathematica* code for `xPerm'`, which is interpreted, becomes too slow. A C executable called xperm is used to compute the hardest part of the calculations; `xTensor'` communicates with it using a *MathLink* connection. Currently this executable exists for Linux and Mac for any version of *Mathematica*, and for MS Windows starting *Mathematica* 6.0.

`MathLink`	Option of `CanonicalPerm` to use the external C executable `xperm`

MathLink connection.

We define an antisymmetric tensor `Anti`:

```
In[229]:=
      DefTensor[Anti[a, b], M3, Antisymmetric[{a, b}], PrintAs -> "A"]

         ** DefTensor: Defining tensor Anti[a, b].
```

Now we fake that there is a metric on `TangentM3`. We shall later explain how to define a metric properly.

```
In[230]:=
      MetricsOfVBundle[TangentM3] ^= {metricg};
      SymmetryGroupOfTensor[metricg] ^= Symmetric[{1, 2}];
```

Powers of an odd number of antisymmetric tensors are 0. These are timings with the interpreted *Mathematica* code:

```
In[232]:=
      AbsoluteTiming[ToCanonical[Anti[a, -a], TimeVerbose → True]]

         Free algorithm applied in 0. secs.

         Dummy algorithm applied in 0.004 secs.

Out[232]=
      {0.005253 Second, 0}

In[233]:=
      AbsoluteTiming[ToCanonical[Anti[a, b] Anti[-b, -a], TimeVerbose → True]]

         Free algorithm applied in 0. secs.

         Dummy algorithm applied in 0.024002 secs.
```

Out[233]=

{0.027611 Second, $-A_{ab} A^{ab}$}

```
In[234]:=
      AbsoluteTiming[ToCanonical[Anti[a, b] Anti[-b, -c] Anti[c, -a], TimeVerbose → True]]

         Free algorithm applied in 0.004 secs.

         Dummy algorithm applied in 0.044003 secs.

Out[234]=
      {0.052529 Second, 0}

In[235]:=
      AbsoluteTiming[
       ToCanonical[Anti[a, b] Anti[-b, -c] Anti[c, d] Anti[-d, -a], TimeVerbose → True]]

         Free algorithm applied in 0. secs.

         Dummy algorithm applied in 0.112007 secs.
```

Out[235]=

{0.120125 Second, $A_a{}^c A^{ab} A_b{}^d A_{cd}$}

```
In[236]:=
     AbsoluteTiming[ToCanonical[
       Anti[a, b] Anti[-b, -c] Anti[c, d] Anti[-d, -e] Anti[e, -a], TimeVerbose → True]]

        Free algorithm applied in 0.004001 secs.

        Dummy algorithm applied in 0.184011 secs.

Out[236]=
     {0.192207 Second, 0}
```

```
In[237]:=
     AbsoluteTiming[
      ToCanonical[Anti[a, b] Anti[-b, -c] Anti[c, d] Anti[-d, -e] Anti[e, f] Anti[-f, -a],
       TimeVerbose → True]]

        Free algorithm applied in 0. secs.

        Dummy algorithm applied in 0.352022 secs.

Out[237]=
```
$\{0.364298\ \text{Second},\ -A_a{}^c A^{ab} A_b{}^d A_c{}^e A_d{}^f A_{ef}\}$

```
In[238]:=
     AbsoluteTiming[ToCanonical[Anti[a, b] Anti[-b, -c] Anti[c, d]
        Anti[-d, -e] Anti[e, f] Anti[-f, -g] Anti[g, -a], TimeVerbose → True]]

        Free algorithm applied in 0. secs.

        Dummy algorithm applied in 0.516032 secs.

Out[238]=
     {0.534410 Second, 0}
```

In order to be able to connect to the external executable, the following global variable must return `True`. If you get `False`, contact JMM to see what is the problem, and whether it can be solved.

```
In[239]:=
     $xpermQ

Out[239]=
     True
```

These are the corresponding timings with the C executable:

```
In[240]:=
     SetOptions[CanonicalPerm, MathLink → True]

Out[240]=
     {MathLink → True, TimeVerbose → False, xPermVerbose → False, OrderedBase → True}

In[241]:=
     AbsoluteTiming[
      ToCanonical[Anti[a, b] Anti[-b, -c] Anti[c, d] Anti[-d, -e] Anti[e, -a]]]

Out[241]=
     {0.012787 Second, 0}
```

```
In[242]:=
     AbsoluteTiming[
      ToCanonical[Anti[a, b] Anti[-b, -c] Anti[c, d] Anti[-d, -e] Anti[e, f] Anti[-f, -a]]]
Out[242]=
```
{0.011747 Second, $-A_a{}^c A^{ab} A_b{}^d A_c{}^e A_d{}^f A_{ef}$}

```
In[243]:=
     AbsoluteTiming[ToCanonical[
       Anti[a, b] Anti[-b, -c] Anti[c, d] Anti[-d, -e] Anti[e, f] Anti[-f, -g] Anti[g, -a]]]
Out[243]=
     {0.015843 Second, 0}

In[244]:=
     AbsoluteTiming[ToCanonical[Anti[a, b] Anti[-b, -c]
        Anti[c, d] Anti[-d, -e] Anti[e, f] Anti[-f, -g] Anti[g, h] Anti[-h, -a]]]
Out[244]=
```
{0.016446 Second, $A_a{}^c A^{ab} A_b{}^d A_c{}^e A_d{}^f A_e{}^g A_f{}^h A_{gh}$}

```
In[245]:=
     AbsoluteTiming[ToCanonical[Anti[a, b] Anti[-b, -c] Anti[c, d]
        Anti[-d, -e] Anti[e, f] Anti[-f, -g] Anti[g, h] Anti[-h, -h1] Anti[h1, -a]]]
Out[245]=
     {0.018259 Second, 0}
```

4.2. Technical details

Canonicalization:

The canonicalization process has essentially six steps, of which only steps 4 and 5 are really hard. Assume we start from a generic syntactically correct input in `xTensor‘` on which we act with `ToCanonical`:

1) Expand the expression to convert it into a sum (`Plus`) of terms, each being a product (`Times`) of indexed objects, sharing dummy indices. This can be considered a polynomial in tensor variables, and as with any normal polynomial, it cannot be canonicalized without first expanding it. We use the *Mathematica* command `Expand`. It might happen that during the expansion some parts of the expression are evaluated and give new products to expand. That means that `Expand` must be used repeatedly until the expression does no longer change (we use the *Mathematica* command `FixedPoint` for that).

2) Minimize the number of different dummies in the whole expression. We use the `xTensor‘` command `SameDummies`.

3) `ToCanonical` is threaded on each term. Steps 4 and 5 cannot compare different terms in the expression. From now on we only deal with products of objects.

4) The objects in each term must be sorted according to certain priorities depending on the types and properties of the objects, but not on the actual indices they have. The term is then considered to be a single tensor, whose properties can be deduced from the properties of the elementary objects composing it. This idea is taken from R. Portugal (put journal ref. here).

5) We extract the list of indices of that composite tensor and canonicalize that list according to the symmetries of the composite object and comparing with the "ideal" ordering of those indices (given in `xTensor‘` by the function `IndexSort`). This "ideal" ordering is also decided by a number of priorities depending on the properties of the indices (type, character and state), as explained below. In general, the indices can change slot, but there is no creation of new indices, the exception being the treatment of derivatives which generate new terms. A very important issue at this step is the interaction between metrics and derivatives, discussed below. The problem will be even harder if frames are involved. The canonicalization algorithms for tensors are basically those of R. Portugal et al (put journal refs here).

6) By now each term is already canonicalized. The function `Plus` will add up equal terms and the result is returned, in canonical form. No simplification (e.g. factorization) is attempted by `ToCanonical`.

The priorities for sorting objects in step 4 are fixed, even though they could be easily changed if a user needs it, but the priorities for sorting indices in step 5 can be configured by the user. In this section we shall describe what those priori− ties are and how some of them affect the efficiency of the whole process.

Sorting objects:

Objects are sorted according to seven considerations, most based in lexicographic ordering (this is also what *Mathemat− ica* does). Objects are sorted with respect to the first of these criteria applying:

1) Type: use alphabetical order in the types:

Constant

ConstantSymbol

Parameter

Scalar (see subsection 8.7 for a description of this head)

Tensor

In subexpressions sometimes we need the type of a composite object: the type of a product is the product of types, and the type of a sum is the sum of types. Both Times and Plus sort their arguments lexicographically.

2) Order (in the sense of number of elements) of the group of symmetry of the objects. More symmetric (higher order) objects come last (to improve efficiency for tensors with lots of indices), though I have found that the opposite

choice is aesthetically better in general.

3) Names of objects. Every object is given a name which depends on the names and the structure of its components, but not on its indices.

4) Number of indices of objects. Sort first the objects with less indices. This is also more efficient.

5) Number of free indices (of the whole term) of objects. Sort first the objects with more free indices.

6) If the global variable `$CommuteFreeIndices` is `False` then sort first the object with the least free index according to `IndexSort`. If it is `True` (the default) then do not take into account the actual free indices and jump to step 6.

7) If several objects cannot be sorted according to the previous considerations, then they are considered equivalent (I call them "commuting") and left then marked as CommutingObjects[o1, o2, ...], what will be used by the permutations algorithms to add new symmetries among the indices of those equivalent tensors.

Sorting indices:

These are all possible things we can consider when sorting indices, in no special order.

− Relative position of free and dummy indices.

− Relative position of the members of the same dummy pair.

− Relative position of up and down indices.

− Alphabetic order of indices.

− The order that the indices have in `IndicesOfManifold` (the order at definition time).

The default is the following, chosen "experimentally" as the most efficient, in average (I don’t really have an explanation of why this seems to be more efficient than other choices):

1) free indices come before dummy indices.

2) sort the indices according to lexicographic order.

3) up−indices come before down−indices.

Those priorities can be changed. We need three priorities, that must be chosen from four pairs of strings, not choosing two of the same pair: `"free"` / `"dummy"`, `"up"` / `"down"`, `"lexicographic"` / `"antilexicographic"` and `"positional"` / `"antipositional"`.

`IndexSort`	Sort a list of indices
`SetIndexSortPriorities`	Define priorities for index sorting

Functions that sort indices.

```
In[246]:=
      ?IndexSort

          IndexSort[list] sorts the elements of list (assumed to be g-indices)
            according to three priorities set up by SetIndexSortPriorities.
            Currently: first: free; second: lexicographic; third: up.
```

Indices of manifold `TangentM3`:

```
In[247]:=
      IndicesOfVBundle[TangentM3]

Out[247]=
      {{a, b, c, d, e, f, g, h}, {h1, h2, h3, h4, h5, h6, h7, h8, h9}}
```

Study this case, taking into account the default priorities given above:

```
In[248]:=
      IndexSort[{a, b, h1, h2, -e, d, c, -c, -h2, e, f}]

Out[248]=
      {a, b, d, f, h1, c, -c, e, -e, h2, -h2}
```

Now we change priorities. The list is sorted differently

```
In[249]:=
      SetIndexSortPriorities["down", "free", "antilexicographic"]

In[250]:=
      ?IndexSort

         IndexSort[list] sorts the elements of list (assumed to be g-indices)
           according to three priorities set up by SetIndexSortPriorities.
           Currently: first: down; second: free; third: antilexicographic.

In[251]:=
      IndexSort[{a, b, h1, h2, -e, d, c, -c, -h2, e, f}]

Out[251]=
      {-h2, -e, -c, h1, f, d, b, a, h2, e, c}
```

Note that once `"positional"` or `"lexicographic"` is chosen (or their opposite), then the priority `"free"`/`"dummy"` becomes irrelevant:

```
In[252]:=
      SetIndexSortPriorities["positional", "down", "dummy"]

In[253]:=
      IndexSort[{a, b, h1, h2, -e, d, c, -c, -h2, e, f}]

Out[253]=
      {a, b, -c, c, d, -e, e, f, h1, -h2, h2}
```

The choice of priorities can make a huge difference in the efficiency of the dummy−canonicalization process. In princi− ple, it is not possible to predict which set of priorities will be better in a given situation, but experimentally we have found that trying to match pairs of dummies with symmetry pairs of S gives better results:

```
In[254]:=
      expr = Anti[a, b] Anti[-b, -c] Anti[c, d] Anti[-d, -e] Anti[e, f] Anti[-f, -a]

Out[254]=
```

$A^{ab} A_{bc} A^{cd} A_{de} A^{ef} A_{fa}$

Having one of "up"/"down" before "free"/"up" seems terrible in the Dummy algorithm:

```
In[255]:=
      SetIndexSortPriorities["up", "free", "lexicographic"]

In[256]:=
      AbsoluteTiming[ToCanonical[expr, TimeVerbose → True, MathLink → False]]

         Free algorithm applied in 0.004 secs.

         Dummy algorithm applied in 1.28408 secs.

Out[256]=
```
$\{1.318403\,\text{Second}, -A_{ac}\, A^{ab}\, A_{be}\, A^{cd}\, A_{df}\, A^{ef}\}$

This is the default choice, instead. Note that the final result is different. This is not a problem: what we want is a uniquely defined process of canonicalization, but which one we choose is "only" a matter of efficiency and aesthetics.

```
In[257]:=
      SetIndexSortPriorities["free", "lexicographic", "up"]

In[258]:=
      AbsoluteTiming[ToCanonical[expr, TimeVerbose → True, MathLink → False]]

         Free algorithm applied in 0. secs.

         Dummy algorithm applied in 0.344022 secs.

Out[258]=
```
$\{0.354651\,\text{Second}, -A_a{}^c\, A^{ab}\, A_b{}^d\, A_c{}^e\, A_d{}^f\, A_{ef}\}$

4.3. Benchmarking

This section contains a series of examples performed in a Linux box with a 1.7 GHz Dual core processor. At least 256 Mb RAM are required to repeat some of these examples.

We use the external executable *xperm*, compiled from the code xperm.c and linked from `xPerm'` through the *MathLink* protocol (xperm.tm template). The code xperm.c contains some C99 extensions, supported by GNU gcc, and therefore Linux and Mac can always use this executable. However, all other Windows C−compilers that *MathLink* can communi−cate to (Borland, MS Visual 2003, etc) are still C90 complier only. The gcc compiler under Windows (under the cygwin system) is compatible with *MathLink* only from version 6.0 of *Mathematica*, but not before and therefore pre−6.0 versions of *Mathematica* cannot use *xperm*.

This variable says whether the connection to the external executable has been performed correctly. If you get False, don't try to repeat these examples; they will take too long.

```
In[259]:=
      $xpermQ

Out[259]=
      True
```

Example 1:

Let us come back to the problem of products of antisymmetric tensors.

This simple code constructs the products we want to canonicalize:

```
In[260]:=
      productAnti[n_] := Times @@ Apply[Anti, Partition[RotateLeft@
           Flatten@Transpose[{-#, #} &@GetIndicesOfVBundle[TangentM3, n]], 2], {1}]
```

```
In[261]:=
      productAnti[1]
```

```
Out[261]=
```
$A^{a}{}_{a}$

```
In[262]:=
      productAnti[7]
```

```
Out[262]=
```
$A^{a}{}_{b}\, A^{b}{}_{c}\, A^{c}{}_{d}\, A^{d}{}_{e}\, A^{e}{}_{f}\, A^{f}{}_{g}\, A^{g}{}_{a}$

We shall need up to 59 different indices:

```
In[263]:=
      GetIndicesOfVBundle[TangentM3, 59, {}]
```

```
Out[263]=
      {a, b, c, d, e, f, g, h, h1, h2, h3, h4, h5, h6, h7, h8, h9, h10, h11, h12, h13, h14, h15,
       h16, h17, h18, h19, h20, h21, h22, h23, h24, h25, h26, h27, h28, h29, h30, h31, h32, h33,
       h34, h35, h36, h37, h38, h39, h40, h41, h42, h43, h44, h45, h46, h47, h48, h49, h50, h51}
```

And this canonicalizes it, giving the `AbsoluteTiming` spent. Note that we can get products of up to 25 tensors in less than half a second (in that case the dummy group D has more than 5 10^32 elements).

```
In[264]:=
      canAnti[n_] :=
       Flatten[{n, With[{product = productAnti[n]}, AbsoluteTiming[ToCanonical[product]]]}]
```

```
In[265]:=
      canAnti[6]
```

```
Out[265]=
```
{6, 0.011851 Second, $-A_{a}{}^{c}\, A^{ab}\, A_{b}{}^{d}\, A_{c}{}^{e}\, A_{d}{}^{f}\, A_{ef}$}

```
In[266]:=
      canAnti[7]
```

```
Out[266]=
      {7, 0.012763 Second, 0}
```

```
In[267]:=
      canAnti[25]
```

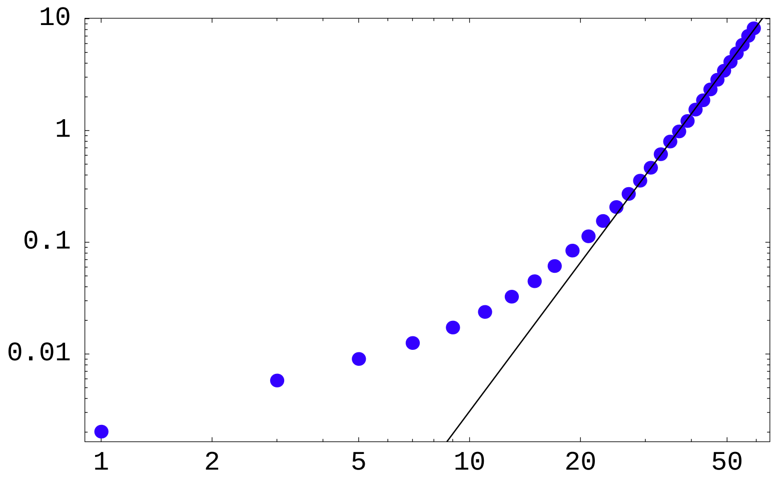
```
Out[267]=
      {25, 0.203955 Second, 0}
```

Let us see how the timings scale with the number of antisymmetric tensors. Evaluating this cell takes about a minute in a 1.8GHz Dual core processor. The executable xperm.linux requires here up to 30 Mbytes. Each computation is done only once, and that invariably introduces small deviations depending on which other tasks the computer is busy with.

```
In[268]:=
      data = canAnti /@ Range[1, 59, 2]

Out[268]=
      {{1, 0.002021 Second, 0}, {3, 0.005795 Second, 0}, {5, 0.009034 Second, 0},
       {7, 0.012553 Second, 0}, {9, 0.017264 Second, 0}, {11, 0.023804 Second, 0},
       {13, 0.032593 Second, 0}, {15, 0.044821 Second, 0}, {17, 0.061277 Second, 0},
       {19, 0.084243 Second, 0}, {21, 0.113119 Second, 0}, {23, 0.155098 Second, 0},
       {25, 0.206592 Second, 0}, {27, 0.270895 Second, 0}, {29, 0.355728 Second, 0},
       {31, 0.464568 Second, 0}, {33, 0.613694 Second, 0}, {35, 0.796980 Second, 0},
       {37, 0.981318 Second, 0}, {39, 1.216632 Second, 0}, {41, 1.539332 Second, 0},
       {43, 1.864288 Second, 0}, {45, 2.331849 Second, 0}, {47, 2.845383 Second, 0},
       {49, 3.429175 Second, 0}, {51, 4.115430 Second, 0}, {53, 4.908952 Second, 0},
       {55, 5.846119 Second, 0}, {57, 7.027919 Second, 0}, {59, 8.208648 Second, 0}}

In[269]:=
      TableForm[data, TableHeadings → {None, {"n", "time", "result"}}]

Out[269]//TableForm=
      n      time                result
      1      0.002021 Second     0
      3      0.005795 Second     0
      5      0.009034 Second     0
      7      0.012553 Second     0
      9      0.017264 Second     0
      11     0.023804 Second     0
      13     0.032593 Second     0
      15     0.044821 Second     0
      17     0.061277 Second     0
      19     0.084243 Second     0
      21     0.113119 Second     0
      23     0.155098 Second     0
      25     0.206592 Second     0
      27     0.270895 Second     0
      29     0.355728 Second     0
      31     0.464568 Second     0
      33     0.613694 Second     0
      35     0.796980 Second     0
      37     0.981318 Second     0
      39     1.216632 Second     0
      41     1.539332 Second     0
      43     1.864288 Second     0
      45     2.331849 Second     0
      47     2.845383 Second     0
      49     3.429175 Second     0
      51     4.115430 Second     0
      53     4.908952 Second     0
      55     5.846119 Second     0
      57     7.027919 Second     0
      59     8.208648 Second     0
```

This is a log−log plot. A straight line represents a power−law. The fit on the last half of points seems to favour a fourth power−law scaling.

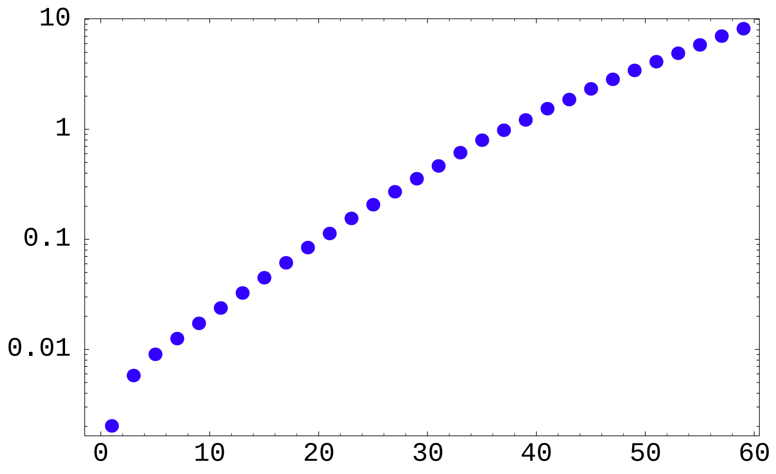
```
In[270]:=
      << Graphics/Graphics.m

In[271]:=
      fit[x_] = Fit[Log[10, data[[Range[15, 30], {1, 2}]] /. Second → 1], {1, x}, x]

Out[271]=
      -6.93 + 4.423 x

In[272]:=
      Show[LogLogListPlot[data[[All, {1, 2}]] /. Second → 1,
        PlotStyle → {PointSize[0.02], Hue[0.7]}, DisplayFunction → Identity],
       Plot[fit[x], {x, 0, 1.85}, DisplayFunction → Identity],
       DisplayFunction → $DisplayFunction, Frame → True, Axes → False]
```



```
Out[272]=
      - Graphics -
```

This is a linear−log plot. A straight line represents an exponential:

```
In[273]:=
      LogListPlot[data[[All, {1, 2}]] /. Second → 1,
       PlotStyle → {PointSize[0.02], Hue[0.7]}, Frame → True, Axes → False]
```



```
Out[273]=
      - Graphics -
```

It is difficult to decide from the plots whether the behaviour is polynomic or exponential, but seems polynomic. This is misleading. The Dummy algorithm is based on the Intersection algorithm, which is known to be exponential in the worst case, though highly efficient. The meaning of this will become clear in the second example below.

This linear−log plot (not produced in this notebook) compares the timings of several packages on the same problem. On the x−axis, the number of antisymmetric tensors to canonicalize (both when the result is zero and when it is not zero). On the y−axis, the timings in Seconds. The color code is the following:

Red: MathTensor

Yello: dhPark

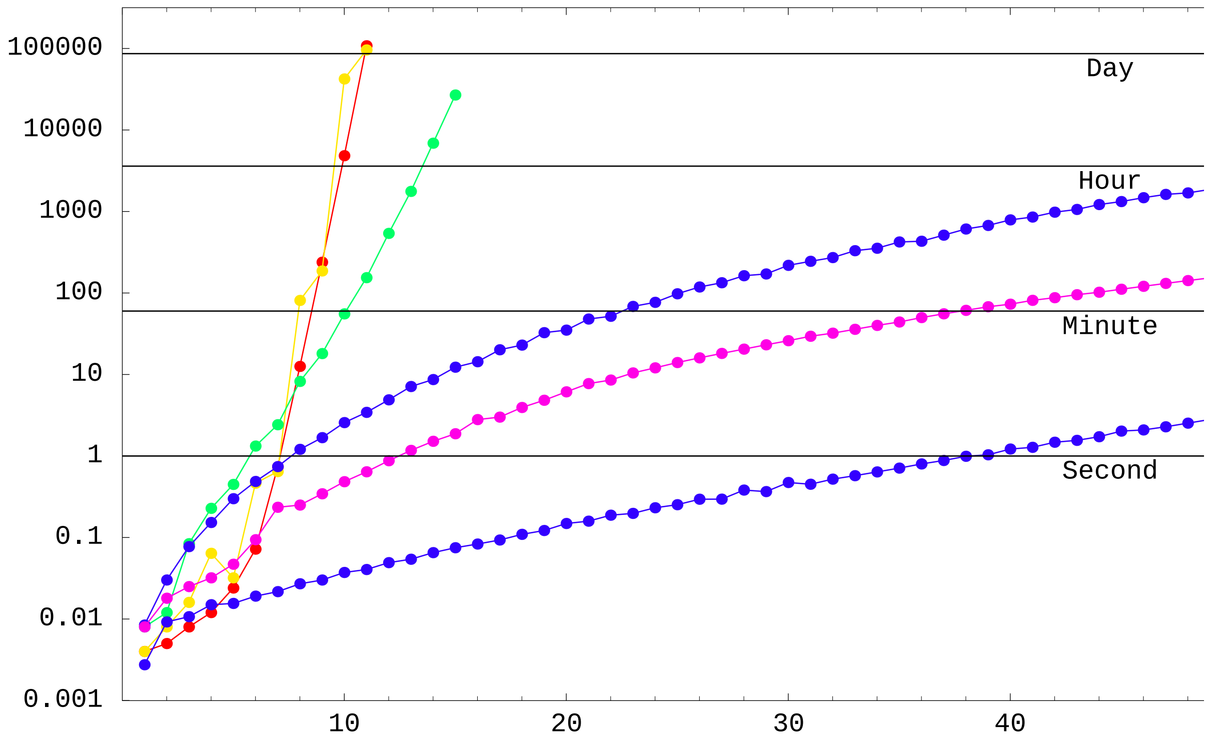
Green: Tools of Tensor Calculus

Blue (upper): xTensor with pure *Mathematica* code in xPerm

Magenta: Canon (R. Portugal’s own implementation in Maple of his canonicalization algorithms)

Blue (lower): xTensor with the external C−executable xperm

The efficiency of the permutation−based algorithms with respect to more classic algorithms is clear.



```
In[274]:=
      UndefTensor[Anti];
      Remove[productAnti, canAnti]

         ** UndefTensor: Undefined tensor Anti
```

Example 2:

As a second example, let us work with the canonicalization of products of Riemann tensors.

Define the Riemann tensor:

```
In[276]:=
      DefTensor[R[a, b, c, d], M3, RiemannSymmetric[{1, 2, 3, 4}]]

          ** DefTensor: Defining tensor R[a, b, c, d].
```

with the expected permutation symmetries:

```
In[277]:=
      R[a, -a, b, c] // ToCanonical

Out[277]=
      0

In[278]:=
      R[-b, -c, -d, -a] // ToCanonical
```

Out[278]=

$-R_{adbc}$

The function productR constructs random products of Riemann tensors. The second argument gives the list of free indices of the expression (always with even length). There is no defined conversion to Ricci or RicciScalar tensors.

```
In[279]:=
      canonindices[npairs_, frees_] := Join[frees,
         Flatten[{#, -#} & /@ GetIndicesOfVBundle[TangentM3, npairs, Join[frees, -frees]]]];
      productR[n_, frees_: {}] := Apply[Times, Apply[R,
           Partition[canonindices[2 n - Length[frees] / 2, frees][[First@RandomPerm[4 n]]],
            4], 1]] /; EvenQ[Length[frees]];

In[281]:=
      productR[1]
```

Out[281]=

$R^{a}{}_{a}{}^{b}{}_{b}$

```
In[282]:=
      productR[10]
```

Out[282]=

$R^{a}{}_{g}{}^{h13}{}_{h18}\, R_{b}{}^{hh12h10}\, R^{c}{}_{h1fc}\, R^{de}{}_{h16}{}^{h16}\, R_{e}{}^{b}{}_{h17}{}^{f}\, R_{h11h10h}{}^{h2}\, R^{h11}{}_{dh2a}\, R_{h13}{}^{h18h17}{}_{h12}\, R^{h14}{}_{h14}{}^{h15h1}\, R_{h15}{}^{g}{}_{h19}{}^{h19}$

```
In[283]:=
      productR[3, {a, b, c, -d}]
```

Out[283]=

$R^{acbg}\, R_{d}{}^{hef}\, R_{gehf}$

```
In[284]:=
      productR[3, {a, b, c}]

Out[284]=
      productR[3, {a, b, c}]
```

This second function canonicalizes the products of Riemanns, timing the process:

```
In[285]:=
      canR[n_, frees_: {}] := Flatten[
        {n, With[{product = productR[n, frees]}, AbsoluteTiming[ToCanonical[product]]]}]
```

Examples:

```
In[286]:=
      canR[1]
```

Out[286]=

$$\{1,\ 0.002172\ \mathrm{Second},\ -R^{ab}{}_{ab}\}$$

```
In[287]:=
      canR[2]
```

Out[287]=

$$\{2,\ 0.007247\ \mathrm{Second},\ R^{ab}{}_{a}{}^{c}\, R_{b}{}^{d}{}_{cd}\}$$

```
In[288]:=
      canR[3]
```

Out[288]=

$$\{3,\ 0.009427\ \mathrm{Second},\ 0\}$$

```
In[289]:=
      canR[4]
```

Out[289]=

$$\{4,\ 0.015283\ \mathrm{Second},\ R_{ac}{}^{ef}\, R^{abcd}\, R_{be}{}^{gh}\, R_{dgfh}\}$$

```
In[290]:=
      canR[10]
```

Out[290]=

$$\{10,\ 0.052343\ \mathrm{Second},\ R^{ab}{}_{a}{}^{c}\, R_{b}{}^{def}\, R_{c}{}^{g}{}_{e}{}^{h}\, R_{d}{}^{h1}{}_{h1}{}^{h10}\, R_{f}{}^{h11h12h13}\, R_{g}{}^{h14h15h16}\, R_{h}{}^{h17}{}_{h17}{}^{h18}\, R_{h10h14h15}{}^{h19}\, R_{h11h18h12}{}^{h2}\, R_{h13h16h19h2}\}$$

```
In[291]:=
      canR[20]
```

Out[291]=

$$\{20,\ 0.137078\ \mathrm{Second},\ 0\}$$

```
In[292]:=
      canR[25]
```

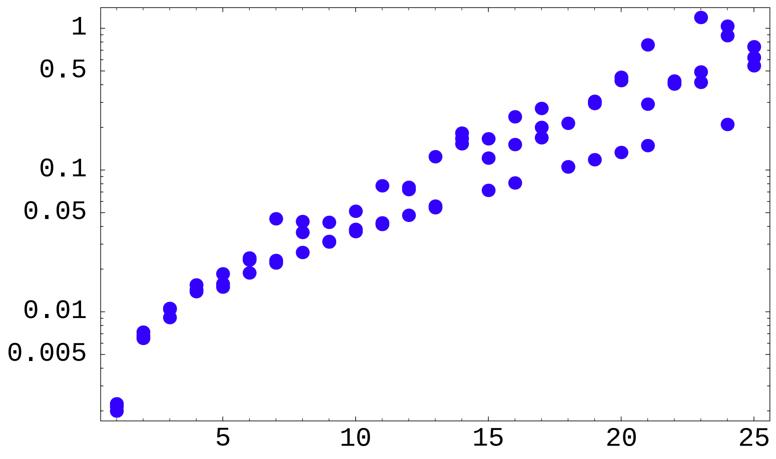
Out[292]=

$$\{25,\ 0.242999\ \mathrm{Second},\ 0\}$$

Scaling of timings. Evaluating next cell could take between half a minute and a minute:

```
In[293]:=
      data1 = canR /@ Range[1, 25];
      data2 = canR /@ Range[1, 25];
      data3 = canR /@ Range[1, 25];
```

In[296]:=
```
plot0 = LogListPlot[Join[data1, data2, data3][[All, {1, 2}]] /. Second → 1,
  PlotRange → All, Frame → True, PlotStyle → {PointSize[0.02], Hue[0.7]}]
```

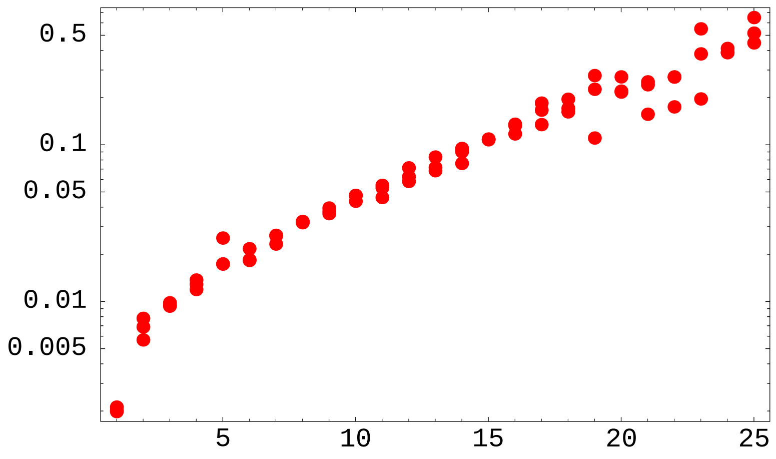


Out[296]=
```
- Graphics -
```

We repeat the computation, but now leaving two and four free indices in the expressions:

In[297]:=
```
data1 = canR[#, {a, b}] & /@ Range[1, 25];
data2 = canR[#, {a, b}] & /@ Range[1, 25];
data3 = canR[#, {a, b}] & /@ Range[1, 25];
```

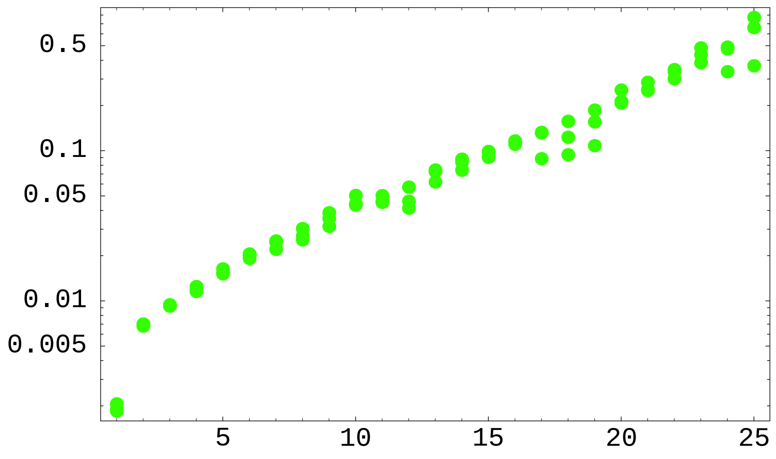
In[300]:=
```
plot1 = LogListPlot[Join[data1, data2, data3][[All, {1, 2}]] /. Second → 1,
  PlotRange → All, Frame → True, PlotStyle → {PointSize[0.02], Hue[0]}]
```



Out[300]=
```
- Graphics -
```

In[301]:=
```
data1 = canR[#, {a, b, c, d}] & /@ Range[1, 25];
data2 = canR[#, {a, b, c, d}] & /@ Range[1, 25];
data3 = canR[#, {a, b, c, d}] & /@ Range[1, 25];
```

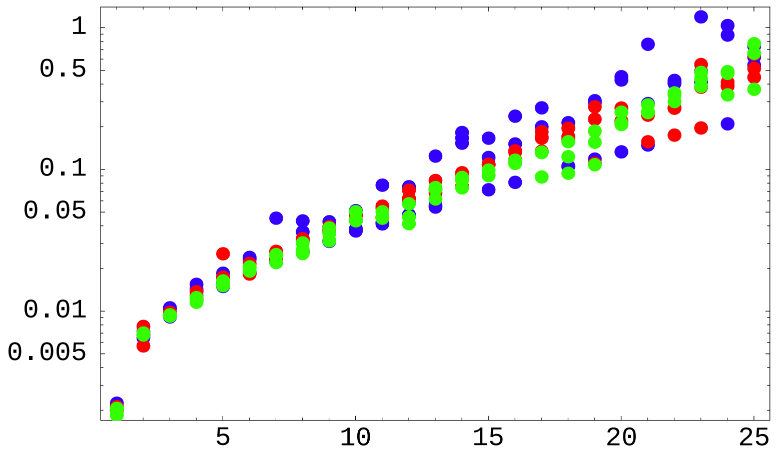
```
In[304]:=
        plot2 = LogListPlot[Join[data1, data2, data3][[All, {1, 2}]] /. Second → 1,
          PlotRange → All, Frame → True, PlotStyle → {PointSize[0.02], Hue[0.3]}]
```



```
Out[304]=
        - Graphics -
```

Combining the three plots, we do not see any important difference in the timings:

```
In[305]:=
        Show[plot0, plot1, plot2]
```

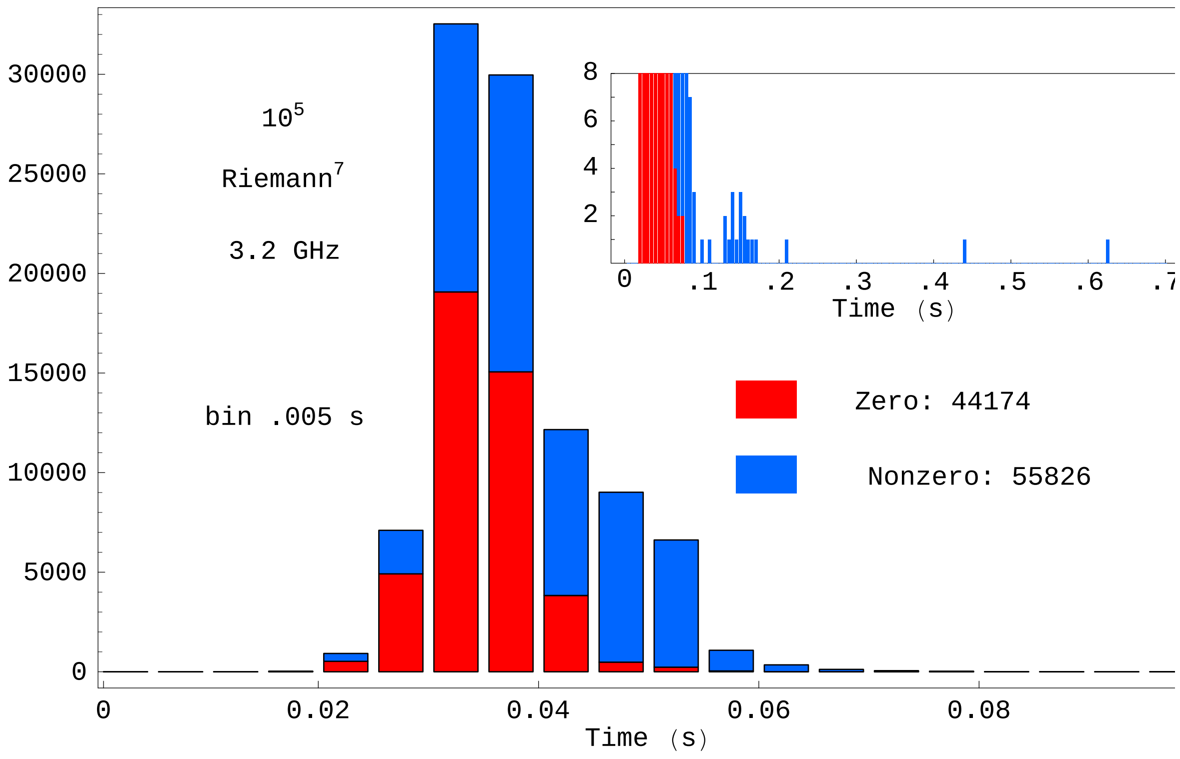


```
Out[305]=
        - Graphics -
```

Example 3:

Now let us concentrate on the products of seven Riemanns.

This figure (not constructed in this notebook) shows an histogram of the timings of canonicalization of a hundred thousand ramdom products of seven Riemanns. Note in the smaller plot that there are some cases which take much more time (and much more memory) to canonicalize. This is a consequence of the algorithm not being polynomial, but exponential in the worst case. Those are cases in which the symmetry is much higher than usual, for example because the product of Riemanns can be separated in products of decoupled monomials. There are cases in which, however, the reason is not at all clear, so that in general it is not possible to prepare the canonicalization algorithms to avoid all possible "worst cases".



Clean up

```
In[306]:=
        UndefTensor[R]
        Remove[productR, canR, data1, data2, data3, plot0, plot1, plot2]

            ** UndefTensor: Undefined tensor R
```

```
In[308]:=
        MetricsOfVBundle[TangentM3] ^= {};
        Clear[metricg];
```

Summarizing, the general canonicalization algorithms of `xTensor`' are highly efficient in most cases, much faster than the algorithms used by other tensor computer algebra systems. However, from time to time you will find special cases which take several orders of magnitude longer to canonicalize than other similar cases (for example a simple reordering of the indices keeping the same tensors), usually also swallowing up a large chunk of memory in your computer. This is normal behaviour, and it is due to the fact that the Group Intersection algorithm unavoidably behaves like that. Fortunately the special cases are always that: special; if you understand the meaning of "special" in your problem, you can prepare the system in advance to switch to "special" alternative algorithms.

In hindsight, this is the price we pay for using a general algorithm, but for the case of permutation symmetries it is worth paying it, because only in a very small fraction of cases the algorithm behaves exponentially, instead of polynomially. The case with multiterm symmetries is precisely the opposite: all known algorithms behave exponentially in most cases.

5. Rules and patterns

5.1. Unique dummies and screening

One of the most important issues in a tensor packages is how to deal with index contractions, i.e. with dummy indices. Until now this was not a real issue for us because all input expressions were simply canonicalized, a process which does not introduce new indices, but rather, eliminates indices. Now we face the opposite problem: expanding expressions into new expressions with more dummy indices, for example the conversion of a covariant derivative of a tensor in terms of a different covariant derivative. There are three problems with conversions that we must address.

This subsection explains in detail what those three problems are and how they will be solved. Though the explanations might seem a bit technical, it is important to understand them, because they are based on core concepts of *Mathematica*.

As an aside, there are two ways to define conversions of expressions into new expressions in *Mathematica*: assignments (`Set` and its variants `SetDelayed`, `TagSet`, `UpSet`, etc.) and rules (`Rule` and its variant `RuleDelayed`). Asign−ments can be considered as global or automatic rules. Rules can be considered as local assignments. See the *Mathemat−ica* Help.

First problem. Let us see a simple example using `Set`:

Define the vector `w` in terms of `T` and `v`. Introduce a simple dummy:

```
In[310]:=
      DefTensor[w[a], M3]

          ** DefTensor: Defining tensor w[a].

In[311]:=
      w[a_] = T[a, b] v[-b]

Out[311]=
```
$T^{ab}\, v_b$

[Note that we have used a Pattern on the right hand side. That is normal *Mathematica*, and a helpful feature which allows us to indicate the generality of the rule (which indices or tensors are accepted in the rule)].

Then these expressions would be totally wrong:

```
In[312]:=
      w[b]

Out[312]=
```
$T^{bb}\, v_b$

```
In[313]:=
      w[a] w[-a]

Out[313]=
```
$T_a{}^b\, T^{ab}\, v_b{}^2$

Mathematica recommends the following solution to this problem:

Use a `Module` construct, such that the dummy index b is replaced by a unique symbol denoted with a $ sign, and a unique integer number. Note that we must now use `SetDelayed` (:=), and not `Set` (=), to avoid inmediate evaluation of `Module`:

```
In[314]:=
      w[a_] := Module[{b}, T[a, b] v[-b]]

In[315]:=
      w[b]

Out[315]=
```
$T^{bb\$17552}\ v_{b\$17552}$

```
In[316]:=
      w[a] w[-a]

Out[316]=
```
$T_{a}{}^{b\$17554}\ T^{ab\$17553}\ v_{b\$17553}\ v_{b\$17554}$

This solves the first problem but generates two more, which are solved by "screening" the dollar−indices:

`ScreenDollarIndices`	Hide internal unique dummy indices

Screening of indices.

First, each time we use the assignment we get a different object, what complicates comparisons. The following is not automatically recognized as zero!

```
In[317]:=
      w[a] - w[a]

Out[317]=
```
$T^{ab\$17555}\ v_{b\$17555} - T^{ab\$17556}\ v_{b\$17556}$

```
In[318]:=
      % // InputForm

Out[318]//InputForm=
     T[a, b$17555]*v[-b$17555] - T[a, b$17556]*v[-b$17556]
```

The second problem is purely aesthetical: the dollar−indices are terribly ugly. `xTensor‘` has the function `ScreenDollarIndices` to convert the dollar−indices into normal−looking indices:

```
In[319]:=
      {w[a], w[b], w[c], w[a] w[-a], w[a] - w[a]}

Out[319]=
```
$\{T^{ab\$17557}\ v_{b\$17557},\ T^{bb\$17558}\ v_{b\$17558},\ T^{cb\$17559}\ v_{b\$17559},\ T_{a}{}^{b\$17561}\ T^{ab\$17560}\ v_{b\$17560}\ v_{b\$17561},\ T^{ab\$17562}\ v_{b\$17562} - T^{ab\$17563}\ v_{b\$17563}\}$

```
In[320]:=
      ScreenDollarIndices[%]

Out[320]=
```
$\{T^{ab}\ v_{b},\ T^{ba}\ v_{a},\ T^{ca}\ v_{a},\ T_{a}{}^{c}\ T^{ab}\ v_{b}\ v_{c},\ 0\}$

```
In[321]:=
      InputForm[%]

Out[321]//InputForm=
     {T[a, b]*v[-b], T[b, a]*v[-a], T[c, a]*v[-a], T[-a, c]*T[a, b]*v[-b]*v[-c], 0}
```

The `Module` construct is not the solution for all problems in tensor conversions. There are two more problems, as seen in the following examples. Second problem:

Objects and indices cannot share the same symbol names. Suppose there were a tensor also called `a`:

```
In[322]:=
      w[a_] := a[a]

In[323]:=
      w[b]

Out[323]=
      b[b]
```

In `xTensor‘`, this is solved by enforcing type declarations. Every object must be defined supplying a symbol, and that symbol is given a type. A symbol cannot have two different types. This can be seen as a restriction, but it is actually a useful safety feature.

The tensor `a[b]` cannot be defined:

```
In[324]:=
      Catch@DefTensor[a[b], M3]

      ValidateSymbol::used : Symbol a is already used as an abstract index.
```

The final (third) problem involves dummies again, and happens in the few cases in which the same symbol represents a dummy on both sides and there are additional dummies on the right. The only possible solution is changing the name of the conflicting dummies either on the left or on the right hand side.

Define the following relation with dummies both on the left and right hand sides. Note that the b dummy is present in both sides and is a pattern on the left hand side:

```
In[325]:=
      w[a_, b_, -b_] := Module[{b, c}, T[a, b, -b, c, -c]]

In[326]:=
      w[a, b, -b]

Out[326]=
```
$T^{ab\$17574}{}_{b\$17574}{}^{c\$17574}{}_{c\$17574}$

```
In[327]:=
      w[a, c, -c]

      Module::dups : Conflicting local variables c
         and c$ found in local variable specification {c, c$}. More…

Out[327]=
```
Module[{c, c$}, $T^{ac}{}_{c}{}^{c\$}{}_{c\$}$]

For example, we can change the name of the dummy index b on the right hand side:

```
In[328]:=
      w[a_, b_, -b_] := Module[{d, c}, T[a, d, -d, c, -c]]

In[329]:=
      w[a, b, -b]

Out[329]=
```
$T^{a d\$17576}{}_{d\$17576}{}^{c\$17576}{}_{c\$17576}$

```
In[330]:=
      w[a, c, -c]

Out[330]=
```
$T^{a d\$17577}{}_{d\$17577}{}^{c\$17577}{}_{c\$17577}$

As we said, the second problem is solved in xTensor‘ simply by enforcing type declarations. The first and third problems will be solved introducing a new collection of Index* assignment and rule functions which properly prepare the required Module constructs. They are IndexSet, IndexSetDelayed, IndexRule and IndexRuleDe−layed, imitating their respective *Mathematica* counterparts, and they are described in detail in the next subsections.

We finish this section coming back to the function ScreenDollarIndices. We recommend to automate its use by assigning one of the global variables $Post or $PrePrint to it. There is a small difference between these two global variables (mentioned below), and it is not clear that one of them is always better than the other for screening, but I generally prefer $PrePrint:

Assign $PrePrint:

```
In[331]:=
      $PrePrint = ScreenDollarIndices;
```

Now the screening is automatic:

```
In[332]:=
      w[a, c, -c]

Out[332]=
```
$T^{ac}{}_{c}{}^{b}{}_{b}$

The screening happened after the result was assigned to Out[...] and hence if we temporarily remove $PrePrint, we still see the dollar−indices. (That wouldn’t happen using $Post.)

```
In[333]:=
      $PrePrint =.

In[334]:=
      %%

Out[334]=
```
$T^{a d\$17578}{}_{d\$17578}{}^{c\$17578}{}_{c\$17578}$

Clean up

```
In[335]:=
      w[a_, b_, -b_] =.
```

From now on, in this notebook, we shall use automatic screening:

```
In[336]:=
      $PrePrint = ScreenDollarIndices;
```

5.2. IndexSet and IndexSetDelayed

The `xTensor‘` functions `IndexSet` and `IndexSetDelayed` imitate the behaviour of `Set` and `SetDelayed` (which delays the evaluation of the rhs to the moment when the lhs is converted into the rhs).

`IndexSet`	Set value of an indexed expression, at definition time
`IndexSetDelayed`	Set value of an indexed expression, at evaluation time

Set functions for indexed expressions.

Example of use of `IndexSet`:

```
In[337]:=
      IndexSet[w[a_], S[a, b] v[-b]]
```

Out[337]=

$S^{ab} v_b$

```
In[338]:=
      ?w

         Global‘w

          Dagger[w] ^= w

          DependenciesOfTensor[w] ^= {M3}

          Info[w] ^= {tensor, }

          PrintAs[w] ^= w

          SlotsOfTensor[w] ^= {TangentM3}

          SymmetryGroupOfTensor[w] ^= StrongGenSet[{}, GenSet[]]

          TensorID[w] ^= {}

          xTensorQ[w] ^= True
```

w^a := Module[{b}, $S^{ab} v_b$]

```
In[339]:=
      w[b]
```

Out[339]=

$S^{ba} v_a$

In this case the code finds the conflicting dummies, and then replaces the offending dummy on the right hand side:

```
In[340]:=
      IndexSet[S[a_, b_, -b_], T[a, b, -b, c, -c]]
```

```
Out[340]=
```
$T^{ab}{}_{b}{}^{c}{}_{c}$

```
In[341]:=
      ?S

         Global`S

         Dagger[S] ^= S

         DependenciesOfTensor[S] ^= {M3}

         Info[S] ^= {tensor, }

         PrintAs[S] ^= S

         SlotsOfTensor[S] ^= {TangentM3, TangentM3}

         SymmetryGroupOfTensor[S] ^= StrongGenSet[{1}, GenSet[Cycles[{1, 2}]]]

         TensorID[S] ^= {}

         xTensorQ[S] ^= True
```

$S^{\underline{a}\,\underline{b}}{}_{\underline{b}}$:= Module[{h$17587, c}, $T^{a\text{h\$17587}}{}_{\text{h\$17587}}{}^{c}{}_{c}$]

```
In[342]:=
      S[a, c, -c]
```

```
Out[342]=
```
$T^{ac}{}_{c}{}^{b}{}_{b}$

Example of use of `IndexSetDelayed`. Note that we only use a pattern in the first index of `U` and that `w` is still not evaluated:

```
In[343]:=
      IndexSetDelayed[U[-a_, -b, -c], w[-a] w[-b] w[-c]]
```

```
In[344]:=
      ?U

         Global`U

         Dagger[U] ^= U

         DependenciesOfTensor[U] ^= {M3}

         Info[U] ^= {tensor, }

         PrintAs[U] ^= U

         SlotsOfTensor[U] ^= {-TangentM3, -TangentM3, -TangentM3}

         SymmetryGroupOfTensor[U] ^=
          StrongGenSet[{1, 2}, GenSet[-Cycles[{1, 2}], -Cycles[{2, 3}]]]

         TensorID[U] ^= {}

         xTensorQ[U] ^= True
```

U_{abc} := Module[{}, $w_a\, w_b\, w_c$]

```
In[345]:=
      U[-a, -b, -c]

Out[345]=
```

$S_a{}^d\, S_b{}^e\, S_c{}^f\, v_d\, v_e\, v_f$

We can change the definition of `w` and the conversion from `U` to `w` will be still valid:

```
In[346]:=
      IndexSet[w[a_], S[a, b] S[-b, -c] v[c]]

Out[346]=
```

$S^{ab}\, S_{bc}\, v^c$

```
In[347]:=
      U[-a, -b, -c]

Out[347]=
```

$S_a{}^d\, S_b{}^e\, S_c{}^f\, S_{dg}\, S_{eh}\, S_{fh1}\, v^g\, v^h\, v^{h1}$

Clean up:

```
In[348]:=
      U[-a_, -b, -c] =.
```

5.3. Index patterns

In the previous definitions of the vector `w[a]` we always used the pattern `a_` which stands for any possible input, including up−indices and down−indices of any manifold, or even directional or label (or basis or component) indices. Sometimes this is what we want, but often it is not. In the following definitions there are no dummies or conflicts between indices and tensor names, so that we can safely use the usual `SetDelayed` (:=) definitions to simplify the examples.

The notation for tensors and indices in `xTensor‘` is very simple, but that complicates working with patterns. On the other hand, there are always several ways to represent the same pattern.

Define, for the sake of simplicity:

```
In[349]:=
      dirvup = Dir[v[d]];
      dirvdown = Dir[v[-d]];
```

This pattern stands for everything. Every `w` is converted into `T`:

```
In[351]:=
      w[a_] := T[a]

In[352]:=
      {w[a], w[-a], w[dirvdown], w[dirvup], w[A], w[-A]}

Out[352]=
```
$\{T^a, T_a, T^v, T_v, T^A, T_A\}$

```
In[353]:=
      w[a_] =.
```

Let us first work with the character of the indices: use the functions

`UpIndexQ`	Detect up−indices
`DownIndexQ`	Detect down−indices

This pattern matches only up−indices:

```
In[354]:=
      w[a_?UpIndexQ] := T[a]

In[355]:=
      {w[a], w[-a], w[dirvdown], w[dirvup], w[A], w[-A]}

Out[355]=
```
$\{T^a, w_a, T^v, w_v, T^A, w_A\}$

```
In[356]:=
      w[a_?UpIndexQ] =.
```

This pattern matches only down−indices:

```
In[357]:=
      w[a_?DownIndexQ] := T[a]
```

```
In[358]:=
      {w[a], w[-a], w[dirvdown], w[dirvup], w[A], w[-A]}
```

```
Out[358]=
      {w^a, T_a, w^v, T_v, w^A, T_A}
```

```
In[359]:=
      w[a_?DownIndexQ] =.
```

And now we work with the type of index: use the functions

`AIndexQ`	Detect abstract indices
`BIndexQ`	Detect basis indices
`CIndexQ`	Detect component indices
`DIndexQ`	Detect directional indices
`LIndexQ`	Detect label indices
`GIndexQ`	Detect all generalized indices, but not patterns
`ABIndexQ`	Detect contractible indices (abstract or basis indices)
`BCIndexQ`	Detect indices associated to a basis or chart (basis or component indices)
`CDIndexQ`	Detect indices representing a direction (component or directional indices)
`PIndexQ`	Detect pattern indices

Detect types of indices.

This pattern matches only abstract indices, from any manifold:

```
In[360]:=
      w[a_?AIndexQ] := T[a]
```

```
In[361]:=
      {w[a], w[-a], w[dirvup], w[dirvdown], w[A], w[-A]}
```

```
Out[361]=
      {T^a, T_a, w_v, w^v, T^A, T_A}
```

```
In[362]:=
      w[a_?AIndexQ] =.
```

There is a simple and efficient pattern for abstract up−indices or abstract down−indices:

```
In[363]:=
      w[a_Symbol] := T[a]
```

```
In[364]:=
      {w[a], w[-a], w[dirvup], w[dirvdown], w[A], w[-A]}
```

```
Out[364]=
      {T^a, w_a, w_v, w^v, T^A, w_A}
```

```
In[365]:=
      w[a_Symbol] =.
```

```
In[366]:=
      w[-a_Symbol] := T[-a]
```

```
In[367]:=
      {w[a], w[-a], w[dirvup], w[dirvdown], w[A], w[-A]}
```

```
Out[367]=
```
$\{w^a, T_a, w_v, w^v, w^A, T_A\}$

```
In[368]:=
      w[-a_Symbol] =.
```

This pattern matches only directional indices. Recall that directions do not have a sign in front. A message is sent complaining about the character of the vector `v`.

```
In[369]:=
      w[a_?DIndexQ] := T[a]
```

```
In[370]:=
      {w[a], w[-a], w[dirvup], w[dirvdown], w[A], w[-A]}

      Validate::error : Invalid character of index in tensor v
```

```
Out[370]=
```
$\{w^a, w_a, T_v, T^v, w^A, w_A\}$

```
In[371]:=
      w[a_?DIndexQ] =.
```

There is also a simpler version (which does not find the problem with the position of the ultraindex):

```
In[372]:=
      w[a_Dir] := T[a]
```

```
In[373]:=
      {w[a], w[-a], w[dirvup], w[dirvdown], w[A], w[-A]}
```

```
Out[373]=
```
$\{w^a, w_a, T_v, T^v, w^A, w_A\}$

```
In[374]:=
      w[a_Dir] =.
```

The functions `AIndexQ`, `BIndexQ`, `CIndexQ`, `DIndexQ` and `GIndexQ` admit a second argument restricting the manifold to which the indices must belong.

This pattern only matches abstract indices on `TangentM3`:

```
In[375]:=
      w[a_?(AIndexQ[#, TangentM3] &)] := T[a]
```

```
In[376]:=
      {w[a], w[-a], w[dirvup], w[dirvdown], w[A], w[-A]}
```

```
Out[376]=
```
$\{T^a, T_a, w_v, w^v, w^A, w_A\}$

```
In[377]:=
      w[a_?(AIndexQ[#, TangentM3] &)] =.
```

Again there are simpler and cleaner versions, also discriminating between upindices and downindices (`Q`−functions) or not (`pmQ`−functions). The use of `Symbol` is not needed now.

In[378]:=
```
w[a_?TangentM3‘Q] := T[a]
```

In[379]:=
```
{w[a], w[-a], w[dirvup], w[dirvdown], w[A], w[-A]}
```

Out[379]=
$\{T^{a}, w_{a}, w_{v}, w^{v}, w^{A}, w_{A}\}$

In[380]:=
```
w[a_?TangentM3‘Q] =.
```

In[381]:=
```
w[-a_?TangentM3‘Q] := T[-a]
```

In[382]:=
```
{w[a], w[-a], w[dirvup], w[dirvdown], w[A], w[-A]}
```

Out[382]=
$\{w^{a}, T_{a}, w_{v}, w^{v}, w^{A}, w_{A}\}$

In[383]:=
```
w[-a_?TangentM3‘Q] =.
```

In[384]:=
```
w[a_?TangentM3‘pmQ] := T[a]
```

In[385]:=
```
{w[a], w[-a], w[dirvup], w[dirvdown], w[A], w[-A]}
```

Out[385]=
$\{T^{a}, T_{a}, w_{v}, w^{v}, w^{A}, w_{A}\}$

In[386]:=
```
w[a_?TangentM3‘pmQ] =.
```

In case of doubt use the function `PatternIndex`, which constructs patterns of the required form.

These are examples of different patterns for abstract indices named `a`. Note the minus sign in front of `a` for down−indices. The pattern is for the symbol of the index, and not for the whole index.

In[387]:=
```
PatternIndex[a, AIndex]
```

Out[387]=
```
a_?AIndexQ
```

In[388]:=
```
PatternIndex[a, AIndex, Up]
```

Out[388]=
```
a_?AIndexQ
```

```
In[389]:=
      PatternIndex[a, AIndex, Down, TangentM3]

Out[389]=
      -a_Symbol?TangentM3‘Q

In[390]:=
      PatternIndex[a, AIndex, Null, TangentM3]

Out[390]=
      a_?TangentM3‘pmQ
```

These are patterns for directional indices named d:

```
In[391]:=
      PatternIndex[d, DIndex]

Out[391]=
      d_Dir

In[392]:=
      PatternIndex[d, DIndex, Up]

Out[392]=
      d_Dir

In[393]:=
      PatternIndex[d, DIndex, Down, TangentM3]

Out[393]=
      d_Dir?(DownIndexQ[#1] && DIndexQ[#1, TangentM3] &)

In[394]:=
      PatternIndex[d, DIndex, Null, TangentM3]

Out[394]=
      d_Dir?(DIndexQ[#1, TangentM3] &)
```

5.4. IndexRule and IndexRuleDelayed

The function `IndexRule` relates to `Rule` exactly in the same way that `IndexSet` relates to `Set`. It just introduces Module constructs. The symbol ↦ has been introduced. It is input as \[RightTeeArrow], but this input sequence doesn't work with the style used in this notebook (I don't know why).

`IndexRule`	Construct rule for an indexed expression, at definition time
`IndexRuleDelayed`	Construct rule for an indexed expression, at evaluation time

Rule functions for indexed expressions.

Here we have again the problem of repeated indices b:

```
In[395]:=
      {w[a], w[b]} /. w[a_] → T[a, b] v[-b]

      Validate::repeated : Found indices with the same name b.

      Throw::nocatch : Uncaught Throw[Null] returned to top level. More…

Out[395]=
      Hold[Throw[Null]]
```

Using IndexRule the problem disappears:

```
In[396]:=
      {w[a], w[b]} /. IndexRule[w[a_], T[a, b] v[-b]]
```

Out[396]=

$\{T^{ab} v_b, T^{ba} v_a\}$

```
In[397]:=
      {w[a], w[b]} /. w[a_] ↦ T[a, b] v[-b]
```

Out[397]=

$\{T^{ab} v_b, T^{ba} v_a\}$

5.5. MakeRule

Apart from the coding aspects solved by the `Index*` functions, there are other, more mathematical issues which must be adressed. For instance, given a rule LHS→RHS how can we apply the rule on expressions which are equivalent to LHS but indices are permuted due to symmetries or metric shifting? Can we apply the rule on all vbundles or just on some of them? These kind of options are controlled by `MakeRule`, the second most complicated function in `xTensor‘` after `ToCanonical`. This function uses the properties of the metric tensors, and hence we delay its description to subsection 8.1 below.

■ 6. Derivatives

6.1. Covariant derivatives

We understand a covariant derivative as a covariant derivative operator (see Wald). In particular, ordinary ("partial") derivative operators are obtained as covariant derivatives with the options `Curvature->False` and `Torsion->False`. Covariant derivatives are real operators acting on (possibly complex) vector bundles.

`DefCovD`	Define a covariant derivative operator
`UndefCovD`	Undefine a covariant derivative
`$CovDs`	List of defined covariant derivatives
`CovDQ`	Validate a covariant derivative symbol

Definition of a covariant derivative.

Default options for `DefCovD`:

```
In[398]:=
      Options[DefCovD]

Out[398]=
      {Torsion → False, Curvature → True, FromMetric → Null, CurvatureRelations → True,
       ExtendedFrom → Null, OrthogonalTo → {}, ProjectedWith → {}, WeightedWithBasis → Null,
       ProtectNewSymbol :→ $ProtectNewSymbols, Master → Null, Info → {covariant derivative, }}
```

Define a covariant derivative `Cd` on `TangentM3` (the default vbundle obtained from the index given in the derivative). Automatically the Christoffel symbols and the Riemann and Ricci tensors are defined. By default `xTensor'` does not assume that this derivative is associated to a metric and therefore the symmetries of the Riemann tensor are not as expected: there is just antisymmetry in the first pair of indices. The Ricci tensor has no symmetry at all. There is no Ricci scalar.

```
In[399]:=
      DefCovD[Cd[-a], {"|", "∇"}]

            ** DefCovD: Defining covariant derivative Cd[-a].

            ** DefTensor: Defining vanishing torsion tensor TorsionCd[a, -b, -c].

            ** DefTensor: Defining symmetric Christoffel tensor ChristoffelCd[a, -b, -c].

            ** DefTensor: Defining Riemann tensor
             RiemannCd[-a, -b, -c, d]. Antisymmetric only in the first pair.

            ** DefTensor: Defining non-symmetric Ricci tensor RicciCd[-a, -b].

            ** DefCovD:  Contractions of Riemann automatically replaced by Ricci.

In[400]:=
      SymmetryGroupOfTensor[RiemannCd]

Out[400]=
      StrongGenSet[{1}, GenSet[-Cycles[{1, 2}]]]

In[401]:=
      RiemannCd[-a, -b, -c, b]

Out[401]=
      R[∇]ac

In[402]:=
      SymmetryGroupOfTensor[RicciCd]

Out[402]=
      StrongGenSet[{}, GenSet[]]
```

The names of the associated tensors are determined by the function `GiveSymbol`. Their output symbols are determined by the function `GiveOutputString`. If you want to change any of those, modify the corresponding function before defining the object.

```
In[403]:=
      GiveSymbol[Riemann, Cd]

Out[403]=
      RiemannCd
```

```
In[404]:=
      GiveOutputString[Riemann, Cd]

Out[404]=
      R[▽]
```

Derivatives can be represented in two ways, encoded as the strings `"Prefix"` or `"Postfix"` in the global variable `$CovDFormat`. The default value is `"Prefix"`. Directional derivatives are always given in `"Prefix"` notation.

```
In[405]:=
      $CovDFormat

Out[405]=
      Prefix
```

```
In[406]:=
      Cd[-a][T[a, b, -c]]
```

Out[406]=

$\nabla_{a} T^{ab}{}_{c}$

```
In[407]:=
      $CovDFormat = "Postfix";
```

```
In[408]:=
      Cd[-a][T[a, b, -c]]
```

Out[408]=

$T^{ab}{}_{c|a}$

```
In[409]:=
      Cd[Dir[v[d]]][T[a, b, -c]]
```

Out[409]=

$T^{ab}{}_{c|v}$

Derivatives are linear operators and implement automatically the Leibnitz rule. The final ordering of terms is decided by *Mathematica.*

```
In[410]:=
      Cd[-a][7 r[] v[a] + S[a, b] v[-b]]
```

Out[410]=

$v_{b} S^{ab}{}_{|a} + 7 (v^{a} r_{|a} + r v^{a}{}_{|a}) + S^{ab} v_{b|a}$

Dummy indices are, by default, pushed to the right by the canonicalization process

```
In[411]:=
      Cd[-a][T[f, c, -b] U[-c, -e, -d]] // ToCanonical
```

Out[411]=

$-U_{dec} T^{fc}{}_{b|a} - T^{fc}{}_{b} U_{dec|a}$

`SortCovDs`	Commute covariant derivatives, adding Riemann tensors if needed
`SortCovDsStart`	Automatic commutation of covariant derivatives
`SortCovDsStop`	Remove automatic commutation of covariant derivatives
`$CommuteCovDsOnScalars`	Automatic commutation of torsionless derivatives on scalar expressions

Commutation of derivatives.

By default, a special ordinary derivative called PD is defined. It is one of the many covariant derivative operators without curvature and without torsion, but no further assumptions are made about it. It can be used as a generic "partial derivative" in most computa−tions, but its meaning is different. Note that they are not automatically sorted:

```
In[412]:=
      PD[-a][PD[-b][T[c, d, -e]]]
```

Out[412]=

$T^{cd}{}_{e,b,a}$

```
In[413]:=
      $CovDFormat = "Prefix";

In[414]:=
      PD[-a][PD[-b][T[c, d, -e]]]
```

Out[414]=

$\partial_a \partial_b T^{cd}{}_e$

```
In[415]:=
      PD[-a][PD[-b][T[c, d, -e]]] - PD[-b][PD[-a][T[c, d, -e]]]
```

Out[415]=

$\partial_a \partial_b T^{cd}{}_e - \partial_b \partial_a T^{cd}{}_e$

```
In[416]:=
      % // SortCovDs

Out[416]=
      0
```

We can automate the commutation of derivatives:

```
In[417]:=
      SortCovDsStart[PD]

In[418]:=
      PD[-a][PD[-b][T[c, d, -e]]] - PD[-b][PD[-a][T[c, d, -e]]]

Out[418]=
      0

In[419]:=
      SortCovDsStop[PD]

          Note that $CommuteCovDsOnScalars is still True.
```

All symmetric (torsionless) connections commute (with themselves) on scalar expressions. That can be avoided switching off the variable `$CommuteCovDsOnScalars`.

The input of high−order derivatives is certainly slow due to the sharp syntactic restrictions of `xTensor‘`. There are several options to avoid this problem. Of course, you cannot use `Validate` unless you modify it to recognize the new syntactic extensions.

The standard way to input a third order derivative is this:

```
In[420]:=
      PD[-a][PD[-b][PD[-c][T[c, d, -e]]]]
```

Out[420]=

$\partial_a \partial_b \partial_c T^{cd}{}_e$

We can use the prefix notation of *Mathematica*:

```
In[421]:=
      PD[-a]@PD[-b]@PD[-c]@T[c, d, -e]
```

Out[421]=

$\partial_a \partial_b \partial_c T^{cd}{}_e$

or we can define our own rules to input derivatives, breaking the standard syntax. The simplest possibility, based on the fact that the first bracket of PD always contains a single index, is:

```
In[422]:=
      Unprotect[PD];

In[423]:=
      PD[expr_, a__, b_] := PD[b][PD[expr, a]]
      PD[expr_, a_] := PD[a][expr]

In[425]:=
      PD[T[c, d, -e], -c, -b, -a]
```

Out[425]=

$\partial_a \partial_b \partial_c T^{cd}{}_e$

```
In[426]:=
      InputForm[%]

Out[426]//InputForm=
     PD[-a][PD[-b][PD[-c][T[c, d, -e]]]]

In[427]:=
      PD[expr_, a__, b_] =.
      PD[expr_, a_] =.
```

or an intermediate situation, based on the same fact, is:

```
In[429]:=
      PD[a_, b__][expr_] := PD[a][PD[b][expr]]

In[430]:=
      PD[-a, -b, -c][T[c, d, e]]
```

Out[430]=

$\partial_a \partial_b \partial_c T^{cde}$

```
In[431]:=
      InputForm[%]

Out[431]//InputForm=
     PD[-a][PD[-b][PD[-c][T[c, d, e]]]]
```

```
In[432]:=
      PD[a_, b__][expr_] =.
```

A totally different notation could be

```
In[433]:=
      PD /: tensor_Symbol?xTensorQ[indices___, PD[index_]] := PD[index][tensor[indices]]
```

```
In[434]:=
      T[c, d, -e, PD[-c], PD[-b], PD[-a]]
```

Out[434]=

$\partial_a \partial_b \partial_c T^{cd}{}_e$

```
In[435]:=
      InputForm[%]

Out[435]//InputForm=
     PD[-a][PD[-b][PD[-c][T[c, d, -e]]]]
```

```
In[436]:=
      PD /: tensor_Symbol?xTensorQ[indices___, PD[index_]] =.
```

```
In[437]:=
      Protect[PD];
```

6.2. Change of covariant derivative

In `xTensor‘` we can work simultaneously with any number of covariant derivatives.

The list of all covariant derivatives currently defined is

```
In[438]:=
      $CovDs

Out[438]=
      {PD, Cd}
```

We define another one

```
In[439]:=
      DefCovD[CD[-a], {";", "D"}]

          ** DefCovD: Defining covariant derivative CD[-a].

          ** DefTensor: Defining vanishing torsion tensor TorsionCD[a, -b, -c].

          ** DefTensor: Defining symmetric Christoffel tensor ChristoffelCD[a, -b, -c].

          ** DefTensor: Defining Riemann tensor
           RiemannCD[-a, -b, -c, d]. Antisymmetric only in the first pair.

          ** DefTensor: Defining non-symmetric Ricci tensor RicciCD[-a, -b].

          ** DefCovD:  Contractions of Riemann automatically replaced by Ricci.
```

The difference between two covariant derivatives of the same tensor can be expressed in terms of Christoffel tensors (the C tensors of Wald), and in `xTensor‘` we shall always use this point of view of Christoffels: they are always tensors, but associated to two covariant derivatives. The Christoffel tensor defined together with each derivative is the tensor associated to that derivative and the derivative `PD` associated to a generic chart.

`Christoffel` constructs the Christoffel tensor relating two derivatives. It is antisymmetric in its derivatives arguments.

```
In[440]:=
     Christoffel[CD, PD][a, -b, -c]

Out[440]=
     Γ[D]^a_bc

In[441]:=
     % // InputForm

Out[441]//InputForm=
     ChristoffelCD[a, -b, -c]

In[442]:=
     Christoffel[PD, CD][a, -b, -c]

Out[442]=
     -Γ[D]^a_bc

In[443]:=
     % // InputForm

Out[443]//InputForm=
     -ChristoffelCD[a, -b, -c]
```

The Christoffel tensor relating two non−`PD` derivatives is constructed automatically whenever needed, with (lexicographically) sorted derivatives, to avoid duplicity:

```
In[444]:=
     Christoffel[CD, Cd][a, -b, -c]

        ** DefTensor: Defining tensor ChristoffelCdCD[a, -b, -c].

Out[444]=
     -Γ[∇,D]^a_bc

In[445]:=
     % // InputForm

Out[445]//InputForm=
     -ChristoffelCdCD[a, -b, -c]
```

The origin of this curious conversion from `Christoffel[Cd,CD]` to `ChristoffelCdCD` is again the fact that we cannot associate information to the former.

Using this structure of Christoffel tensors we can relate any two derivatives of any tensor.

Suppose this derivative:

```
In[446]:=
     expr = CD[-d][U[-a, b, -c]]

Out[446]=
     D_d U_a^b_c
```

`ChangeCovD` (aka `CovDToChristoffel` in `xTensor‘` version 0.7) changes by default to `PD`:

In[447]:=
```
ChangeCovD[expr]
```

Out[447]=
$$-\Gamma[D]^{e}{}_{dc}\, U_{a}{}^{b}{}_{e} + \Gamma[D]^{b}{}_{de}\, U_{a}{}^{e}{}_{c} - \Gamma[D]^{e}{}_{da}\, U_{e}{}^{b}{}_{c} + \partial_{d} U_{a}{}^{b}{}_{c}$$

That is equivalent to

In[448]:=
```
ChangeCovD[expr, CD]
```

Out[448]=
$$-\Gamma[D]^{e}{}_{dc}\, U_{a}{}^{b}{}_{e} + \Gamma[D]^{b}{}_{de}\, U_{a}{}^{e}{}_{c} - \Gamma[D]^{e}{}_{da}\, U_{e}{}^{b}{}_{c} + \partial_{d} U_{a}{}^{b}{}_{c}$$

but we can also change to any desired derivative:

In[449]:=
```
ChangeCovD[expr, CD, Cd]
```

Out[449]=
$$\Gamma[\nabla,D]^{e}{}_{dc}\, U_{a}{}^{b}{}_{e} - \Gamma[\nabla,D]^{b}{}_{de}\, U_{a}{}^{e}{}_{c} + \Gamma[\nabla,D]^{e}{}_{da}\, U_{e}{}^{b}{}_{c} + \nabla_{d}\, U_{a}{}^{b}{}_{c}$$

In[450]:=
```
ChangeCovD[%, Cd, CD]
```

Out[450]=
$$D_{d}\, U_{a}{}^{b}{}_{c}$$

If you do not like the double−derivative Christoffels, you can always break them to the CovD−`PD` Christoffels:

In[451]:=
```
%% // BreakChristoffel
```

Out[451]=
$$(\Gamma[\nabla]^{e}{}_{dc} - \Gamma[D]^{e}{}_{dc})\, U_{a}{}^{b}{}_{e} - (\Gamma[\nabla]^{b}{}_{de} - \Gamma[D]^{b}{}_{de})\, U_{a}{}^{e}{}_{c} + (\Gamma[\nabla]^{e}{}_{da} - \Gamma[D]^{e}{}_{da})\, U_{e}{}^{b}{}_{c} + \nabla_{d}\, U_{a}{}^{b}{}_{c}$$

The expansion is recursive:

In[452]:=
```
ChangeCovD[Cd[-e][Cd[-a][T[f, c, -b]]], Cd]
```

Out[452]=
$$\begin{aligned}&-\Gamma[\nabla]^{h1}{}_{eb}\,(-\Gamma[\nabla]^{d}{}_{ah1}\, T^{fc}{}_{d} + \Gamma[\nabla]^{c}{}_{ag}\, T^{fg}{}_{h1} + \Gamma[\nabla]^{f}{}_{ah}\, T^{hc}{}_{h1} + \partial_{a} T^{fc}{}_{h1}) +\\ &\;\Gamma[\nabla]^{c}{}_{eh1}\,(\Gamma[\nabla]^{h1}{}_{ag}\, T^{fg}{}_{b} - \Gamma[\nabla]^{d}{}_{ab}\, T^{fh1}{}_{d} + \Gamma[\nabla]^{f}{}_{ah}\, T^{hh1}{}_{b} + \partial_{a} T^{fh1}{}_{b}) +\\ &\;\Gamma[\nabla]^{f}{}_{eh1}\,(\Gamma[\nabla]^{h1}{}_{ah}\, T^{hc}{}_{b} - \Gamma[\nabla]^{d}{}_{ab}\, T^{h1c}{}_{d} + \Gamma[\nabla]^{c}{}_{ag}\, T^{h1g}{}_{b} + \partial_{a} T^{h1c}{}_{b}) + T^{fd}{}_{b}\, \partial_{e}\Gamma[\nabla]^{c}{}_{ad} -\\ &\;T^{fc}{}_{d}\, \partial_{e}\Gamma[\nabla]^{d}{}_{ab} + T^{dc}{}_{b}\, \partial_{e}\Gamma[\nabla]^{f}{}_{ad} + \Gamma[\nabla]^{f}{}_{ad}\, \partial_{e} T^{dc}{}_{b} - \Gamma[\nabla]^{d}{}_{ab}\, \partial_{e} T^{fc}{}_{d} + \Gamma[\nabla]^{c}{}_{ad}\, \partial_{e} T^{fd}{}_{b} +\\ &\;\partial_{e}\partial_{a} T^{fc}{}_{b} - \Gamma[\nabla]^{h1}{}_{ea}\,(-\Gamma[\nabla]^{d}{}_{h1b}\, T^{fc}{}_{d} + \Gamma[\nabla]^{c}{}_{h1g}\, T^{fg}{}_{b} + \Gamma[\nabla]^{f}{}_{h1h}\, T^{hc}{}_{b} + \partial_{h1} T^{fc}{}_{b})\end{aligned}$$

`Christoffel`	Construct Christoffel tensor relating two derivatives
`BreakChristoffel`	Rewrite a Christoffel tensor as the difference of two other Christoffel tensors
`ChangeCovD`	Rewrite the covariant derivative of a tensor in terms of a second derivative and Christof− fels

Change of covariant derivative.

If the relation between two covariant derivatives is fully described by a Christoffel tensor, then the curvature and torsion tensors associated to them must be also related by those Christoffel tensors.

This is the curvature tensor of the derivative CD:

```
In[453]:=
      RiemannCD[-a, -b, -c, d]
```

Out[453]=

$$R[D]_{abc}{}^{d}$$

`ChangeCurvature` (aka `RiemannToChristoffel` in `xTensor'` version 0.7) changes any curvature tensor of a derivative to the curvature tensor of other derivative (by default `PD`, with zero curvature).

```
In[454]:=
      ChangeCurvature[%]
```

Out[454]=

$$\Gamma[D]^{d}{}_{be}\,\Gamma[D]^{e}{}_{ac} - \Gamma[D]^{d}{}_{ae}\,\Gamma[D]^{e}{}_{bc} - \partial_a\Gamma[D]^{d}{}_{bc} + \partial_b\Gamma[D]^{d}{}_{ac}$$

but we can convert to any derivative:

```
In[455]:=
      ChangeCurvature[%%, CD, Cd]
```

Out[455]=

$$\Gamma[\nabla,D]^{d}{}_{be}\,\Gamma[\nabla,D]^{e}{}_{ac} - \Gamma[\nabla,D]^{d}{}_{ae}\,\Gamma[\nabla,D]^{e}{}_{bc} + R[\nabla]_{abc}{}^{d} + \nabla_a\,\Gamma[\nabla,D]^{d}{}_{bc} - \nabla_b\,\Gamma[\nabla,D]^{d}{}_{ac}$$

```
In[456]:=
      ChangeCurvature[RiemannCD[-a, -b, -c, a], CD, Cd]
```

Out[456]=

$$\Gamma[\nabla,D]^{a}{}_{bd}\,\Gamma[\nabla,D]^{d}{}_{ac} - \Gamma[\nabla,D]^{a}{}_{ad}\,\Gamma[\nabla,D]^{d}{}_{bc} - R[\nabla]_{bc} + \nabla_a\,\Gamma[\nabla,D]^{a}{}_{bc} - \nabla_b\,\Gamma[\nabla,D]^{a}{}_{ac}$$

Define a covariant derivative with torsion, but not metric−compatible. Now the Christoffel tensor is non−symmetric:

In[457]:=
```
DefCovD[CDT[-a], {"#", "DT"}, Torsion → True]
```

```
** DefCovD: Defining covariant derivative CDT[-a].

** DefTensor: Defining torsion tensor TorsionCDT[a, -b, -c].

** DefTensor: Defining
 non-symmetric Christoffel tensor ChristoffelCDT[a, -b, -c].

** DefTensor: Defining Riemann tensor
 RiemannCDT[-a, -b, -c, d]. Antisymmetric only in the first pair.

** DefTensor: Defining non-symmetric Ricci tensor RicciCDT[-a, -b].

** DefCovD:  Contractions of Riemann automatically replaced by Ricci.
```

Now the formulas involving `CDT` will contain torsion terms:

In[458]:=
```
ChangeCurvature[RiemannCd[-a, -b, -c, a], Cd, CDT]
```

```
** DefTensor: Defining tensor ChristoffelCdCDT[a, -b, -c].
```

Out[458]=

$$\Gamma[\nabla,\mathrm{DT}]^{a}{}_{bd}\,\Gamma[\nabla,\mathrm{DT}]^{d}{}_{ac} - \Gamma[\nabla,\mathrm{DT}]^{a}{}_{ad}\,\Gamma[\nabla,\mathrm{DT}]^{d}{}_{bc} - R[\mathrm{DT}]_{bc} + \Gamma[\nabla,\mathrm{DT}]^{a}{}_{dc}\,T[\mathrm{DT}]^{d}{}_{ba} - \mathrm{DT}_{a}\,\Gamma[\nabla,\mathrm{DT}]^{a}{}_{bc} + \mathrm{DT}_{b}\,\Gamma[\nabla,\mathrm{DT}]^{a}{}_{ac}$$

In[459]:=
```
% // InputForm
```

Out[459]//InputForm=
```
-(ChristoffelCdCDT[h$17869, -h$17869, -h$17879]*ChristoffelCdCDT[h$17879, -b, -c])
+
 ChristoffelCdCDT[h$17869, -b, -h$17879]*ChristoffelCdCDT[h$17879, -h$17869, -c] -
RicciCDT[-b, -c] +
 ChristoffelCdCDT[h$17869, -h$17879, -c]*TorsionCDT[h$17879, -b, -h$17869] +
 CDT[-b][ChristoffelCdCDT[h$17869, -h$17869, -c]] -
CDT[-h$17869][ChristoffelCdCDT[h$17869, -b, -c]]
```

The difference between the torsion tensors of two derivatives is given by the antisymmetric part of the Christoffel relating them. In this case the torsion of `Cd` is zero. The change is performed by `ChangeTorsion` (aka `TorsionToChristoffel` in `xTensor‘` version 0.7):

In[460]:=
```
ChangeTorsion[TorsionCDT[a, -b, -c], CDT, Cd]
```

Out[460]=

$$-\Gamma[\nabla,\mathrm{DT}]^{a}{}_{bc} + \Gamma[\nabla,\mathrm{DT}]^{a}{}_{cb}$$

`ChangeCurvature`	Change in curvature when changing between two covariant derivatives
`ChangeTorsion`	Change in torsion when changing between two covariant derivatives

Induced changes in curvature and torsion tensors.

Undefine some derivatives:

```
In[461]:=
      UndefCovD /@ {CD, CDT};

          ** UndefTensor: Undefined symmetric Christoffel tensor ChristoffelCD

          ** UndefTensor: Undefined tensor ChristoffelCdCD

          ** UndefTensor: Undefined non-symmetric Ricci tensor RicciCD

          ** UndefTensor: Undefined Riemann tensor RiemannCD

          ** UndefTensor: Undefined vanishing torsion tensor TorsionCD

          ** UndefCovD: Undefined covariant derivative CD

          ** UndefTensor: Undefined tensor ChristoffelCdCDT

          ** UndefTensor: Undefined non-symmetric Christoffel tensor ChristoffelCDT

          ** UndefTensor: Undefined non-symmetric Ricci tensor RicciCDT

          ** UndefTensor: Undefined Riemann tensor RiemannCDT

          ** UndefTensor: Undefined torsion tensor TorsionCDT

          ** UndefCovD: Undefined covariant derivative CDT
```

Note that `xTensor‘` also allows metric−compatible connections with torsion. The symmetry properties of the associ− ated curvature tensors are not complete. We shall later illustrate this type of derivatives, after defining metric fields.

6.3. General Bianchi identities

Let us now check the Bianchi identities for general covariant derivatives with torsion. Note that these identities are not directly encoded in `xTensor‘`, but can be easily computed.

Define a covariant derivative with torsion. Note the special symmetry properties of the defined tensors:

```
In[462]:=
      DefCovD[CD[-a], {";", "∇"}, Torsion → True]

          ** DefCovD: Defining covariant derivative CD[-a].

          ** DefTensor: Defining torsion tensor TorsionCD[a, -b, -c].

          ** DefTensor: Defining
           non-symmetric Christoffel tensor ChristoffelCD[a, -b, -c].

          ** DefTensor: Defining Riemann tensor
           RiemannCD[-a, -b, -c, d]. Antisymmetric only in the first pair.

          ** DefTensor: Defining non-symmetric Ricci tensor RicciCD[-a, -b].

          ** DefCovD:  Contractions of Riemann automatically replaced by Ricci.
```

This is the general form of the first Bianchi identity:

```
In[463]:=
      RiemannCD[-a, -b, -c, d] + CD[-a][ TorsionCD[d, -b, -c] ] -
       TorsionCD[e, -a, -b] TorsionCD[d, -c, -e]
```

Out[463]=

$$R[\triangledown]_{abc}{}^{d} - T[\triangledown]^{d}{}_{ce}\, T[\triangledown]^{e}{}_{ab} + \triangledown_{a} T[\triangledown]^{d}{}_{bc}$$

```
In[464]:=
      6 Antisymmetrize[%, {-a, -b, -c}]
```

Out[464]=

$$\begin{aligned}&R[\triangledown]_{abc}{}^{d} - R[\triangledown]_{acb}{}^{d} - R[\triangledown]_{bac}{}^{d} + R[\triangledown]_{bca}{}^{d} + R[\triangledown]_{cab}{}^{d} - R[\triangledown]_{cba}{}^{d} - T[\triangledown]^{d}{}_{ce}\, T[\triangledown]^{e}{}_{ab} +\\
&T[\triangledown]^{d}{}_{be}\, T[\triangledown]^{e}{}_{ac} + T[\triangledown]^{d}{}_{ce}\, T[\triangledown]^{e}{}_{ba} - T[\triangledown]^{d}{}_{ae}\, T[\triangledown]^{e}{}_{bc} - T[\triangledown]^{d}{}_{be}\, T[\triangledown]^{e}{}_{ca} +\\
&T[\triangledown]^{d}{}_{ae}\, T[\triangledown]^{e}{}_{cb} + \triangledown_{a} T[\triangledown]^{d}{}_{bc} - \triangledown_{a} T[\triangledown]^{d}{}_{cb} - \triangledown_{b} T[\triangledown]^{d}{}_{ac} + \triangledown_{b} T[\triangledown]^{d}{}_{ca} + \triangledown_{c} T[\triangledown]^{d}{}_{ab} - \triangledown_{c} T[\triangledown]^{d}{}_{ba}\end{aligned}$$

The computation can be performed by transforming the derivative CD and its associated tensors into PD and Christoffel tensors:

```
In[465]:=
      % // ChangeCovD
```

Out[465]=

$$\begin{aligned}&R[\triangledown]_{abc}{}^{d} - R[\triangledown]_{acb}{}^{d} - R[\triangledown]_{bac}{}^{d} + R[\triangledown]_{bca}{}^{d} + R[\triangledown]_{cab}{}^{d} - R[\triangledown]_{cba}{}^{d} + \Gamma[\triangledown]^{e}{}_{bc}\, T[\triangledown]^{d}{}_{ae} -\\
&\Gamma[\triangledown]^{e}{}_{cb}\, T[\triangledown]^{d}{}_{ae} - \Gamma[\triangledown]^{e}{}_{ac}\, T[\triangledown]^{d}{}_{be} + \Gamma[\triangledown]^{e}{}_{ca}\, T[\triangledown]^{d}{}_{be} + \Gamma[\triangledown]^{e}{}_{ab}\, T[\triangledown]^{d}{}_{ce} -\\
&\Gamma[\triangledown]^{e}{}_{ba}\, T[\triangledown]^{d}{}_{ce} - \Gamma[\triangledown]^{e}{}_{bc}\, T[\triangledown]^{d}{}_{ea} + \Gamma[\triangledown]^{e}{}_{cb}\, T[\triangledown]^{d}{}_{ea} + \Gamma[\triangledown]^{e}{}_{ac}\, T[\triangledown]^{d}{}_{eb} - \Gamma[\triangledown]^{e}{}_{ca}\, T[\triangledown]^{d}{}_{eb} -\\
&\Gamma[\triangledown]^{e}{}_{ab}\, T[\triangledown]^{d}{}_{ec} + \Gamma[\triangledown]^{e}{}_{ba}\, T[\triangledown]^{d}{}_{ec} + \Gamma[\triangledown]^{d}{}_{ce}\, T[\triangledown]^{e}{}_{ab} - T[\triangledown]^{d}{}_{ce}\, T[\triangledown]^{e}{}_{ab} -\\
&\Gamma[\triangledown]^{d}{}_{be}\, T[\triangledown]^{e}{}_{ac} + T[\triangledown]^{d}{}_{be}\, T[\triangledown]^{e}{}_{ac} - \Gamma[\triangledown]^{d}{}_{ce}\, T[\triangledown]^{e}{}_{ba} + T[\triangledown]^{d}{}_{ce}\, T[\triangledown]^{e}{}_{ba} +\\
&\Gamma[\triangledown]^{d}{}_{ae}\, T[\triangledown]^{e}{}_{bc} - T[\triangledown]^{d}{}_{ae}\, T[\triangledown]^{e}{}_{bc} + \Gamma[\triangledown]^{d}{}_{be}\, T[\triangledown]^{e}{}_{ca} - T[\triangledown]^{d}{}_{be}\, T[\triangledown]^{e}{}_{ca} - \Gamma[\triangledown]^{d}{}_{ae}\, T[\triangledown]^{e}{}_{cb} +\\
&T[\triangledown]^{d}{}_{ae}\, T[\triangledown]^{e}{}_{cb} + \partial_{a} T[\triangledown]^{d}{}_{bc} - \partial_{a} T[\triangledown]^{d}{}_{cb} - \partial_{b} T[\triangledown]^{d}{}_{ac} + \partial_{b} T[\triangledown]^{d}{}_{ca} + \partial_{c} T[\triangledown]^{d}{}_{ab} - \partial_{c} T[\triangledown]^{d}{}_{ba}\end{aligned}$$

```
In[466]:=
      % // RiemannToChristoffel
```

Out[466]=

$$\begin{aligned}&-2\, \Gamma[\triangledown]^{d}{}_{ce}\, \Gamma[\triangledown]^{e}{}_{ab} + 2\, \Gamma[\triangledown]^{d}{}_{be}\, \Gamma[\triangledown]^{e}{}_{ac} + 2\, \Gamma[\triangledown]^{d}{}_{ce}\, \Gamma[\triangledown]^{e}{}_{ba} - 2\, \Gamma[\triangledown]^{d}{}_{ae}\, \Gamma[\triangledown]^{e}{}_{bc} -\\
&2\, \Gamma[\triangledown]^{d}{}_{be}\, \Gamma[\triangledown]^{e}{}_{ca} + 2\, \Gamma[\triangledown]^{d}{}_{ae}\, \Gamma[\triangledown]^{e}{}_{cb} + \Gamma[\triangledown]^{e}{}_{bc}\, T[\triangledown]^{d}{}_{ae} - \Gamma[\triangledown]^{e}{}_{cb}\, T[\triangledown]^{d}{}_{ae} - \Gamma[\triangledown]^{e}{}_{ac}\, T[\triangledown]^{d}{}_{be} +\\
&\Gamma[\triangledown]^{e}{}_{ca}\, T[\triangledown]^{d}{}_{be} + \Gamma[\triangledown]^{e}{}_{ab}\, T[\triangledown]^{d}{}_{ce} - \Gamma[\triangledown]^{e}{}_{ba}\, T[\triangledown]^{d}{}_{ce} - \Gamma[\triangledown]^{e}{}_{bc}\, T[\triangledown]^{d}{}_{ea} + \Gamma[\triangledown]^{e}{}_{cb}\, T[\triangledown]^{d}{}_{ea} +\\
&\Gamma[\triangledown]^{e}{}_{ac}\, T[\triangledown]^{d}{}_{eb} - \Gamma[\triangledown]^{e}{}_{ca}\, T[\triangledown]^{d}{}_{eb} - \Gamma[\triangledown]^{e}{}_{ab}\, T[\triangledown]^{d}{}_{ec} + \Gamma[\triangledown]^{e}{}_{ba}\, T[\triangledown]^{d}{}_{ec} + \Gamma[\triangledown]^{d}{}_{ce}\, T[\triangledown]^{e}{}_{ab} -\\
&T[\triangledown]^{d}{}_{ce}\, T[\triangledown]^{e}{}_{ab} - \Gamma[\triangledown]^{d}{}_{be}\, T[\triangledown]^{e}{}_{ac} + T[\triangledown]^{d}{}_{be}\, T[\triangledown]^{e}{}_{ac} - \Gamma[\triangledown]^{d}{}_{ce}\, T[\triangledown]^{e}{}_{ba} + T[\triangledown]^{d}{}_{ce}\, T[\triangledown]^{e}{}_{ba} +\\
&\Gamma[\triangledown]^{d}{}_{ae}\, T[\triangledown]^{e}{}_{bc} - T[\triangledown]^{d}{}_{ae}\, T[\triangledown]^{e}{}_{bc} + \Gamma[\triangledown]^{d}{}_{be}\, T[\triangledown]^{e}{}_{ca} - T[\triangledown]^{d}{}_{be}\, T[\triangledown]^{e}{}_{ca} - \Gamma[\triangledown]^{d}{}_{ae}\, T[\triangledown]^{e}{}_{cb} +\\
&T[\triangledown]^{d}{}_{ae}\, T[\triangledown]^{e}{}_{cb} - 2\, \partial_{a} \Gamma[\triangledown]^{d}{}_{bc} + 2\, \partial_{a} \Gamma[\triangledown]^{d}{}_{cb} + \partial_{a} T[\triangledown]^{d}{}_{bc} - \partial_{a} T[\triangledown]^{d}{}_{cb} + 2\, \partial_{b} \Gamma[\triangledown]^{d}{}_{ac} -\\
&2\, \partial_{b} \Gamma[\triangledown]^{d}{}_{ca} - \partial_{b} T[\triangledown]^{d}{}_{ac} + \partial_{b} T[\triangledown]^{d}{}_{ca} - 2\, \partial_{c} \Gamma[\triangledown]^{d}{}_{ab} + 2\, \partial_{c} \Gamma[\triangledown]^{d}{}_{ba} + \partial_{c} T[\triangledown]^{d}{}_{ab} - \partial_{c} T[\triangledown]^{d}{}_{ba}\end{aligned}$$

In[467]:=
```
% // TorsionToChristoffel
```

Out[467]=
$$\begin{aligned}
&-2\,\Gamma[\nabla]^{d}{}_{ce}\,\Gamma[\nabla]^{e}{}_{ab} + (\Gamma[\nabla]^{d}{}_{ce} - \Gamma[\nabla]^{d}{}_{ec})\,\Gamma[\nabla]^{e}{}_{ab} - (-\Gamma[\nabla]^{d}{}_{ce} + \Gamma[\nabla]^{d}{}_{ec})\,\Gamma[\nabla]^{e}{}_{ab} + \\
&2\,\Gamma[\nabla]^{d}{}_{be}\,\Gamma[\nabla]^{e}{}_{ac} - (\Gamma[\nabla]^{d}{}_{be} - \Gamma[\nabla]^{d}{}_{eb})\,\Gamma[\nabla]^{e}{}_{ac} + (-\Gamma[\nabla]^{d}{}_{be} + \Gamma[\nabla]^{d}{}_{eb})\,\Gamma[\nabla]^{e}{}_{ac} + \\
&\Gamma[\nabla]^{d}{}_{ce}\,(\Gamma[\nabla]^{e}{}_{ab} - \Gamma[\nabla]^{e}{}_{ba}) - (\Gamma[\nabla]^{d}{}_{ce} - \Gamma[\nabla]^{d}{}_{ec})\,(\Gamma[\nabla]^{e}{}_{ab} - \Gamma[\nabla]^{e}{}_{ba}) + \\
&2\,\Gamma[\nabla]^{d}{}_{ce}\,\Gamma[\nabla]^{e}{}_{ba} - (\Gamma[\nabla]^{d}{}_{ce} - \Gamma[\nabla]^{d}{}_{ec})\,\Gamma[\nabla]^{e}{}_{ba} + (-\Gamma[\nabla]^{d}{}_{ce} + \Gamma[\nabla]^{d}{}_{ec})\,\Gamma[\nabla]^{e}{}_{ba} - \\
&\Gamma[\nabla]^{d}{}_{ce}\,(-\Gamma[\nabla]^{e}{}_{ab} + \Gamma[\nabla]^{e}{}_{ba}) + (\Gamma[\nabla]^{d}{}_{ce} - \Gamma[\nabla]^{d}{}_{ec})\,(-\Gamma[\nabla]^{e}{}_{ab} + \Gamma[\nabla]^{e}{}_{ba}) - \\
&2\,\Gamma[\nabla]^{d}{}_{ae}\,\Gamma[\nabla]^{e}{}_{bc} + (\Gamma[\nabla]^{d}{}_{ae} - \Gamma[\nabla]^{d}{}_{ea})\,\Gamma[\nabla]^{e}{}_{bc} - (-\Gamma[\nabla]^{d}{}_{ae} + \Gamma[\nabla]^{d}{}_{ea})\,\Gamma[\nabla]^{e}{}_{bc} - \\
&\Gamma[\nabla]^{d}{}_{be}\,(\Gamma[\nabla]^{e}{}_{ac} - \Gamma[\nabla]^{e}{}_{ca}) + (\Gamma[\nabla]^{d}{}_{be} - \Gamma[\nabla]^{d}{}_{eb})\,(\Gamma[\nabla]^{e}{}_{ac} - \Gamma[\nabla]^{e}{}_{ca}) - \\
&2\,\Gamma[\nabla]^{d}{}_{be}\,\Gamma[\nabla]^{e}{}_{ca} + (\Gamma[\nabla]^{d}{}_{be} - \Gamma[\nabla]^{d}{}_{eb})\,\Gamma[\nabla]^{e}{}_{ca} - (-\Gamma[\nabla]^{d}{}_{be} + \Gamma[\nabla]^{d}{}_{eb})\,\Gamma[\nabla]^{e}{}_{ca} + \\
&\Gamma[\nabla]^{d}{}_{be}\,(-\Gamma[\nabla]^{e}{}_{ac} + \Gamma[\nabla]^{e}{}_{ca}) - (\Gamma[\nabla]^{d}{}_{be} - \Gamma[\nabla]^{d}{}_{eb})\,(-\Gamma[\nabla]^{e}{}_{ac} + \Gamma[\nabla]^{e}{}_{ca}) + \\
&\Gamma[\nabla]^{d}{}_{ae}\,(\Gamma[\nabla]^{e}{}_{bc} - \Gamma[\nabla]^{e}{}_{cb}) - (\Gamma[\nabla]^{d}{}_{ae} - \Gamma[\nabla]^{d}{}_{ea})\,(\Gamma[\nabla]^{e}{}_{bc} - \Gamma[\nabla]^{e}{}_{cb}) + \\
&2\,\Gamma[\nabla]^{d}{}_{ae}\,\Gamma[\nabla]^{e}{}_{cb} - (\Gamma[\nabla]^{d}{}_{ae} - \Gamma[\nabla]^{d}{}_{ea})\,\Gamma[\nabla]^{e}{}_{cb} + (-\Gamma[\nabla]^{d}{}_{ae} + \Gamma[\nabla]^{d}{}_{ea})\,\Gamma[\nabla]^{e}{}_{cb} - \\
&\Gamma[\nabla]^{d}{}_{ae}\,(-\Gamma[\nabla]^{e}{}_{bc} + \Gamma[\nabla]^{e}{}_{cb}) + (\Gamma[\nabla]^{d}{}_{ae} - \Gamma[\nabla]^{d}{}_{ea})\,(-\Gamma[\nabla]^{e}{}_{bc} + \Gamma[\nabla]^{e}{}_{cb})
\end{aligned}$$

In[468]:=
```
% // Expand
```

Out[468]=
```
0
```

This is the general form of the second Bianchi identity:

In[469]:=
```
CD[-a][ RiemannCD[-b, -c, -d, e] ] - TorsionCD[f, -a, -b] RiemannCD[-c, -f, -d, e]
```

Out[469]=
$$-R[\nabla]_{cfd}{}^{e}\,T[\nabla]^{f}{}_{ab} + \nabla_a R[\nabla]_{bcd}{}^{e}$$

In[470]:=
```
6 Antisymmetrize[%, {-a, -b, -c}]
```

Out[470]=
$$\begin{aligned}
&-R[\nabla]_{cfd}{}^{e}\,T[\nabla]^{f}{}_{ab} + R[\nabla]_{bfd}{}^{e}\,T[\nabla]^{f}{}_{ac} + R[\nabla]_{cfd}{}^{e}\,T[\nabla]^{f}{}_{ba} - R[\nabla]_{afd}{}^{e}\,T[\nabla]^{f}{}_{bc} - R[\nabla]_{bfd}{}^{e}\,T[\nabla]^{f}{}_{ca} + \\
&R[\nabla]_{afd}{}^{e}\,T[\nabla]^{f}{}_{cb} + \nabla_a R[\nabla]_{bcd}{}^{e} - \nabla_a R[\nabla]_{cbd}{}^{e} - \nabla_b R[\nabla]_{acd}{}^{e} + \nabla_b R[\nabla]_{cad}{}^{e} + \nabla_c R[\nabla]_{abd}{}^{e} - \nabla_c R[\nabla]_{bad}{}^{e}
\end{aligned}$$

The computation proceeds along the same lines:

In[471]:=
```
% // ChangeCovD
```

Out[471]=
$$\begin{aligned}
&\Gamma[\nabla]^{e}{}_{cf}\,R[\nabla]_{abd}{}^{f} - \Gamma[\nabla]^{f}{}_{cd}\,R[\nabla]_{abf}{}^{e} - \Gamma[\nabla]^{e}{}_{bf}\,R[\nabla]_{acd}{}^{f} + \Gamma[\nabla]^{f}{}_{bd}\,R[\nabla]_{acf}{}^{e} + \\
&\Gamma[\nabla]^{f}{}_{bc}\,R[\nabla]_{afd}{}^{e} - \Gamma[\nabla]^{f}{}_{cb}\,R[\nabla]_{afd}{}^{e} - \Gamma[\nabla]^{e}{}_{cf}\,R[\nabla]_{bad}{}^{f} + \Gamma[\nabla]^{f}{}_{cd}\,R[\nabla]_{baf}{}^{e} + \Gamma[\nabla]^{e}{}_{af}\,R[\nabla]_{bcd}{}^{f} - \\
&\Gamma[\nabla]^{f}{}_{ad}\,R[\nabla]_{bcf}{}^{e} - \Gamma[\nabla]^{f}{}_{ac}\,R[\nabla]_{bfd}{}^{e} + \Gamma[\nabla]^{f}{}_{ca}\,R[\nabla]_{bfd}{}^{e} + \Gamma[\nabla]^{e}{}_{bf}\,R[\nabla]_{cad}{}^{f} - \Gamma[\nabla]^{f}{}_{bd}\,R[\nabla]_{caf}{}^{e} - \\
&\Gamma[\nabla]^{e}{}_{af}\,R[\nabla]_{cbd}{}^{f} + \Gamma[\nabla]^{f}{}_{ad}\,R[\nabla]_{cbf}{}^{e} + \Gamma[\nabla]^{f}{}_{ab}\,R[\nabla]_{cfd}{}^{e} - \Gamma[\nabla]^{f}{}_{ba}\,R[\nabla]_{cfd}{}^{e} - \Gamma[\nabla]^{f}{}_{bc}\,R[\nabla]_{fad}{}^{e} + \\
&\Gamma[\nabla]^{f}{}_{cb}\,R[\nabla]_{fad}{}^{e} + \Gamma[\nabla]^{f}{}_{ac}\,R[\nabla]_{fbd}{}^{e} - \Gamma[\nabla]^{f}{}_{ca}\,R[\nabla]_{fbd}{}^{e} - \Gamma[\nabla]^{f}{}_{ab}\,R[\nabla]_{fcd}{}^{e} + \Gamma[\nabla]^{f}{}_{ba}\,R[\nabla]_{fcd}{}^{e} - \\
&R[\nabla]_{cfd}{}^{e}\,T[\nabla]^{f}{}_{ab} + R[\nabla]_{bfd}{}^{e}\,T[\nabla]^{f}{}_{ac} + R[\nabla]_{cfd}{}^{e}\,T[\nabla]^{f}{}_{ba} - R[\nabla]_{afd}{}^{e}\,T[\nabla]^{f}{}_{bc} - R[\nabla]_{bfd}{}^{e}\,T[\nabla]^{f}{}_{ca} + \\
&R[\nabla]_{afd}{}^{e}\,T[\nabla]^{f}{}_{cb} + \partial_a R[\nabla]_{bcd}{}^{e} - \partial_a R[\nabla]_{cbd}{}^{e} - \partial_b R[\nabla]_{acd}{}^{e} + \partial_b R[\nabla]_{cad}{}^{e} + \partial_c R[\nabla]_{abd}{}^{e} - \partial_c R[\nabla]_{bad}{}^{e}
\end{aligned}$$

```
In[472]:=
      % // RiemannToChristoffel
```

```
Out[472]=
```

$$
\begin{aligned}
&-2\,\Gamma[\nabla]^{f}{}_{cd}\,\partial_a\Gamma[\nabla]^{e}{}_{bf}+2\,\Gamma[\nabla]^{f}{}_{bd}\,\partial_a\Gamma[\nabla]^{e}{}_{cf}+\\
&2\,\Gamma[\nabla]^{e}{}_{cf}\,\partial_a\Gamma[\nabla]^{f}{}_{bd}-2\,\Gamma[\nabla]^{e}{}_{bf}\,\partial_a\Gamma[\nabla]^{f}{}_{cd}-2\,\partial_a\partial_b\Gamma[\nabla]^{e}{}_{cd}+2\,\partial_a\partial_c\Gamma[\nabla]^{e}{}_{bd}+\\
&\Gamma[\nabla]^{f}{}_{cd}\,(-\Gamma[\nabla]^{e}{}_{bg}\,\Gamma[\nabla]^{g}{}_{af}+\Gamma[\nabla]^{e}{}_{ag}\,\Gamma[\nabla]^{g}{}_{bf}+\partial_a\Gamma[\nabla]^{e}{}_{bf}-\partial_b\Gamma[\nabla]^{e}{}_{af})+\\
&2\,\Gamma[\nabla]^{f}{}_{cd}\,\partial_b\Gamma[\nabla]^{e}{}_{af}-\Gamma[\nabla]^{f}{}_{cd}\,(\Gamma[\nabla]^{e}{}_{bg}\,\Gamma[\nabla]^{g}{}_{af}-\Gamma[\nabla]^{e}{}_{ag}\,\Gamma[\nabla]^{g}{}_{bf}-\partial_a\Gamma[\nabla]^{e}{}_{bf}+\partial_b\Gamma[\nabla]^{e}{}_{af})-\\
&2\,\Gamma[\nabla]^{f}{}_{ad}\,\partial_b\Gamma[\nabla]^{e}{}_{cf}-\Gamma[\nabla]^{e}{}_{cf}\,(-\Gamma[\nabla]^{f}{}_{bg}\,\Gamma[\nabla]^{g}{}_{ad}+\Gamma[\nabla]^{f}{}_{ag}\,\Gamma[\nabla]^{g}{}_{bd}+\partial_a\Gamma[\nabla]^{f}{}_{bd}-\partial_b\Gamma[\nabla]^{f}{}_{ad})-\\
&2\,\Gamma[\nabla]^{e}{}_{cf}\,\partial_b\Gamma[\nabla]^{f}{}_{ad}+\Gamma[\nabla]^{e}{}_{cf}\,(\Gamma[\nabla]^{f}{}_{bg}\,\Gamma[\nabla]^{g}{}_{ad}-\Gamma[\nabla]^{f}{}_{ag}\,\Gamma[\nabla]^{g}{}_{bd}-\partial_a\Gamma[\nabla]^{f}{}_{bd}+\partial_b\Gamma[\nabla]^{f}{}_{ad})+\\
&2\,\Gamma[\nabla]^{e}{}_{af}\,\partial_b\Gamma[\nabla]^{f}{}_{cd}+2\,\partial_b\partial_a\Gamma[\nabla]^{e}{}_{cd}-2\,\partial_b\partial_c\Gamma[\nabla]^{e}{}_{ad}-\\
&\Gamma[\nabla]^{f}{}_{bd}\,(-\Gamma[\nabla]^{e}{}_{cg}\,\Gamma[\nabla]^{g}{}_{af}+\Gamma[\nabla]^{e}{}_{ag}\,\Gamma[\nabla]^{g}{}_{cf}+\partial_a\Gamma[\nabla]^{e}{}_{cf}-\partial_c\Gamma[\nabla]^{e}{}_{af})-\\
&2\,\Gamma[\nabla]^{f}{}_{bd}\,\partial_c\Gamma[\nabla]^{e}{}_{af}+\Gamma[\nabla]^{f}{}_{bd}\,(\Gamma[\nabla]^{e}{}_{cg}\,\Gamma[\nabla]^{g}{}_{af}-\Gamma[\nabla]^{e}{}_{ag}\,\Gamma[\nabla]^{g}{}_{cf}-\partial_a\Gamma[\nabla]^{e}{}_{cf}+\partial_c\Gamma[\nabla]^{e}{}_{af})+\\
&\Gamma[\nabla]^{f}{}_{ad}\,(-\Gamma[\nabla]^{e}{}_{cg}\,\Gamma[\nabla]^{g}{}_{bf}+\Gamma[\nabla]^{e}{}_{bg}\,\Gamma[\nabla]^{g}{}_{cf}+\partial_b\Gamma[\nabla]^{e}{}_{cf}-\partial_c\Gamma[\nabla]^{e}{}_{bf})+\\
&2\,\Gamma[\nabla]^{f}{}_{ad}\,\partial_c\Gamma[\nabla]^{e}{}_{bf}-\Gamma[\nabla]^{f}{}_{ad}\,(\Gamma[\nabla]^{e}{}_{cg}\,\Gamma[\nabla]^{g}{}_{bf}-\Gamma[\nabla]^{e}{}_{bg}\,\Gamma[\nabla]^{g}{}_{cf}-\partial_b\Gamma[\nabla]^{e}{}_{cf}+\partial_c\Gamma[\nabla]^{e}{}_{bf})+\\
&\Gamma[\nabla]^{e}{}_{bf}\,(-\Gamma[\nabla]^{f}{}_{cg}\,\Gamma[\nabla]^{g}{}_{ad}+\Gamma[\nabla]^{f}{}_{ag}\,\Gamma[\nabla]^{g}{}_{cd}+\partial_a\Gamma[\nabla]^{f}{}_{cd}-\partial_c\Gamma[\nabla]^{f}{}_{ad})+\\
&2\,\Gamma[\nabla]^{e}{}_{bf}\,\partial_c\Gamma[\nabla]^{f}{}_{ad}-\Gamma[\nabla]^{e}{}_{bf}\,(\Gamma[\nabla]^{f}{}_{cg}\,\Gamma[\nabla]^{g}{}_{ad}-\Gamma[\nabla]^{f}{}_{ag}\,\Gamma[\nabla]^{g}{}_{cd}-\partial_a\Gamma[\nabla]^{f}{}_{cd}+\partial_c\Gamma[\nabla]^{f}{}_{ad})-\\
&\Gamma[\nabla]^{e}{}_{af}\,(-\Gamma[\nabla]^{f}{}_{cg}\,\Gamma[\nabla]^{g}{}_{bd}+\Gamma[\nabla]^{f}{}_{bg}\,\Gamma[\nabla]^{g}{}_{cd}+\partial_b\Gamma[\nabla]^{f}{}_{cd}-\partial_c\Gamma[\nabla]^{f}{}_{bd})-2\,\Gamma[\nabla]^{e}{}_{af}\,\partial_c\Gamma[\nabla]^{f}{}_{bd}+\\
&\Gamma[\nabla]^{e}{}_{af}\,(\Gamma[\nabla]^{f}{}_{cg}\,\Gamma[\nabla]^{g}{}_{bd}-\Gamma[\nabla]^{f}{}_{bg}\,\Gamma[\nabla]^{g}{}_{cd}-\partial_b\Gamma[\nabla]^{f}{}_{cd}+\partial_c\Gamma[\nabla]^{f}{}_{bd})-2\,\partial_c\partial_a\Gamma[\nabla]^{e}{}_{bd}+\\
&2\,\partial_c\partial_b\Gamma[\nabla]^{e}{}_{ad}-\Gamma[\nabla]^{f}{}_{bc}\,(-\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{ad}+\Gamma[\nabla]^{e}{}_{ag}\,\Gamma[\nabla]^{g}{}_{fd}+\partial_a\Gamma[\nabla]^{e}{}_{fd}-\partial_f\Gamma[\nabla]^{e}{}_{ad})+\\
&\Gamma[\nabla]^{f}{}_{cb}\,(-\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{ad}+\Gamma[\nabla]^{e}{}_{ag}\,\Gamma[\nabla]^{g}{}_{fd}+\partial_a\Gamma[\nabla]^{e}{}_{fd}-\partial_f\Gamma[\nabla]^{e}{}_{ad})+\\
&\Gamma[\nabla]^{f}{}_{bc}\,(\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{ad}-\Gamma[\nabla]^{e}{}_{ag}\,\Gamma[\nabla]^{g}{}_{fd}-\partial_a\Gamma[\nabla]^{e}{}_{fd}+\partial_f\Gamma[\nabla]^{e}{}_{ad})-\\
&\Gamma[\nabla]^{f}{}_{cb}\,(\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{ad}-\Gamma[\nabla]^{e}{}_{ag}\,\Gamma[\nabla]^{g}{}_{fd}-\partial_a\Gamma[\nabla]^{e}{}_{fd}+\partial_f\Gamma[\nabla]^{e}{}_{ad})-\\
&T[\nabla]^{f}{}_{bc}\,(\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{ad}-\Gamma[\nabla]^{e}{}_{ag}\,\Gamma[\nabla]^{g}{}_{fd}-\partial_a\Gamma[\nabla]^{e}{}_{fd}+\partial_f\Gamma[\nabla]^{e}{}_{ad})+\\
&T[\nabla]^{f}{}_{cb}\,(\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{ad}-\Gamma[\nabla]^{e}{}_{ag}\,\Gamma[\nabla]^{g}{}_{fd}-\partial_a\Gamma[\nabla]^{e}{}_{fd}+\partial_f\Gamma[\nabla]^{e}{}_{ad})+\\
&\Gamma[\nabla]^{f}{}_{ac}\,(-\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{bd}+\Gamma[\nabla]^{e}{}_{bg}\,\Gamma[\nabla]^{g}{}_{fd}+\partial_b\Gamma[\nabla]^{e}{}_{fd}-\partial_f\Gamma[\nabla]^{e}{}_{bd})-\\
&\Gamma[\nabla]^{f}{}_{ca}\,(-\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{bd}+\Gamma[\nabla]^{e}{}_{bg}\,\Gamma[\nabla]^{g}{}_{fd}+\partial_b\Gamma[\nabla]^{e}{}_{fd}-\partial_f\Gamma[\nabla]^{e}{}_{bd})-\\
&\Gamma[\nabla]^{f}{}_{ac}\,(\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{bd}-\Gamma[\nabla]^{e}{}_{bg}\,\Gamma[\nabla]^{g}{}_{fd}-\partial_b\Gamma[\nabla]^{e}{}_{fd}+\partial_f\Gamma[\nabla]^{e}{}_{bd})+\\
&\Gamma[\nabla]^{f}{}_{ca}\,(\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{bd}-\Gamma[\nabla]^{e}{}_{bg}\,\Gamma[\nabla]^{g}{}_{fd}-\partial_b\Gamma[\nabla]^{e}{}_{fd}+\partial_f\Gamma[\nabla]^{e}{}_{bd})+\\
&T[\nabla]^{f}{}_{ac}\,(\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{bd}-\Gamma[\nabla]^{e}{}_{bg}\,\Gamma[\nabla]^{g}{}_{fd}-\partial_b\Gamma[\nabla]^{e}{}_{fd}+\partial_f\Gamma[\nabla]^{e}{}_{bd})-\\
&T[\nabla]^{f}{}_{ca}\,(\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{bd}-\Gamma[\nabla]^{e}{}_{bg}\,\Gamma[\nabla]^{g}{}_{fd}-\partial_b\Gamma[\nabla]^{e}{}_{fd}+\partial_f\Gamma[\nabla]^{e}{}_{bd})-\\
&\Gamma[\nabla]^{f}{}_{ab}\,(-\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{cd}+\Gamma[\nabla]^{e}{}_{cg}\,\Gamma[\nabla]^{g}{}_{fd}+\partial_c\Gamma[\nabla]^{e}{}_{fd}-\partial_f\Gamma[\nabla]^{e}{}_{cd})+\\
&\Gamma[\nabla]^{f}{}_{ba}\,(-\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{cd}+\Gamma[\nabla]^{e}{}_{cg}\,\Gamma[\nabla]^{g}{}_{fd}+\partial_c\Gamma[\nabla]^{e}{}_{fd}-\partial_f\Gamma[\nabla]^{e}{}_{cd})+\\
&\Gamma[\nabla]^{f}{}_{ab}\,(\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{cd}-\Gamma[\nabla]^{e}{}_{cg}\,\Gamma[\nabla]^{g}{}_{fd}-\partial_c\Gamma[\nabla]^{e}{}_{fd}+\partial_f\Gamma[\nabla]^{e}{}_{cd})-\\
&\Gamma[\nabla]^{f}{}_{ba}\,(\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{cd}-\Gamma[\nabla]^{e}{}_{cg}\,\Gamma[\nabla]^{g}{}_{fd}-\partial_c\Gamma[\nabla]^{e}{}_{fd}+\partial_f\Gamma[\nabla]^{e}{}_{cd})-\\
&T[\nabla]^{f}{}_{ab}\,(\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{cd}-\Gamma[\nabla]^{e}{}_{cg}\,\Gamma[\nabla]^{g}{}_{fd}-\partial_c\Gamma[\nabla]^{e}{}_{fd}+\partial_f\Gamma[\nabla]^{e}{}_{cd})+\\
&T[\nabla]^{f}{}_{ba}\,(\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{cd}-\Gamma[\nabla]^{e}{}_{cg}\,\Gamma[\nabla]^{g}{}_{fd}-\partial_c\Gamma[\nabla]^{e}{}_{fd}+\partial_f\Gamma[\nabla]^{e}{}_{cd})
\end{aligned}
$$

In[473]:=

```
% // TorsionToChristoffel
```

Out[473]=

$$
\begin{aligned}
&-2\,\Gamma[\nabla]^{f}{}_{cd}\,\partial_{a}\Gamma[\nabla]^{e}{}_{bf}+2\,\Gamma[\nabla]^{f}{}_{bd}\,\partial_{a}\Gamma[\nabla]^{e}{}_{cf}+\\
&2\,\Gamma[\nabla]^{e}{}_{cf}\,\partial_{a}\Gamma[\nabla]^{f}{}_{bd}-2\,\Gamma[\nabla]^{e}{}_{bf}\,\partial_{a}\Gamma[\nabla]^{f}{}_{cd}-2\,\partial_{a}\partial_{b}\Gamma[\nabla]^{e}{}_{cd}+2\,\partial_{a}\partial_{c}\Gamma[\nabla]^{e}{}_{bd}+\\
&\Gamma[\nabla]^{f}{}_{cd}\,(-\Gamma[\nabla]^{e}{}_{bg}\,\Gamma[\nabla]^{g}{}_{af}+\Gamma[\nabla]^{e}{}_{ag}\,\Gamma[\nabla]^{g}{}_{bf}+\partial_{a}\Gamma[\nabla]^{e}{}_{bf}-\partial_{b}\Gamma[\nabla]^{e}{}_{af})+\\
&2\,\Gamma[\nabla]^{f}{}_{cd}\,\partial_{b}\Gamma[\nabla]^{e}{}_{af}-\Gamma[\nabla]^{f}{}_{cd}\,(\Gamma[\nabla]^{e}{}_{bg}\,\Gamma[\nabla]^{g}{}_{af}-\Gamma[\nabla]^{e}{}_{ag}\,\Gamma[\nabla]^{g}{}_{bf}-\partial_{a}\Gamma[\nabla]^{e}{}_{bf}+\partial_{b}\Gamma[\nabla]^{e}{}_{af})-\\
&2\,\Gamma[\nabla]^{f}{}_{ad}\,\partial_{b}\Gamma[\nabla]^{e}{}_{cf}-\Gamma[\nabla]^{e}{}_{cf}\,(-\Gamma[\nabla]^{f}{}_{bg}\,\Gamma[\nabla]^{g}{}_{ad}+\Gamma[\nabla]^{f}{}_{ag}\,\Gamma[\nabla]^{g}{}_{bd}+\partial_{a}\Gamma[\nabla]^{f}{}_{bd}-\partial_{b}\Gamma[\nabla]^{f}{}_{ad})-\\
&2\,\Gamma[\nabla]^{e}{}_{cf}\,\partial_{b}\Gamma[\nabla]^{f}{}_{ad}+\Gamma[\nabla]^{e}{}_{cf}\,(\Gamma[\nabla]^{f}{}_{bg}\,\Gamma[\nabla]^{g}{}_{ad}-\Gamma[\nabla]^{f}{}_{ag}\,\Gamma[\nabla]^{g}{}_{bd}-\partial_{a}\Gamma[\nabla]^{f}{}_{bd}+\partial_{b}\Gamma[\nabla]^{f}{}_{ad})+\\
&2\,\Gamma[\nabla]^{e}{}_{af}\,\partial_{b}\Gamma[\nabla]^{f}{}_{cd}+2\,\partial_{b}\partial_{a}\Gamma[\nabla]^{e}{}_{cd}-2\,\partial_{b}\partial_{c}\Gamma[\nabla]^{e}{}_{ad}-\\
&\Gamma[\nabla]^{f}{}_{bd}\,(-\Gamma[\nabla]^{e}{}_{cg}\,\Gamma[\nabla]^{g}{}_{af}+\Gamma[\nabla]^{e}{}_{ag}\,\Gamma[\nabla]^{g}{}_{cf}+\partial_{a}\Gamma[\nabla]^{e}{}_{cf}-\partial_{c}\Gamma[\nabla]^{e}{}_{af})-\\
&2\,\Gamma[\nabla]^{f}{}_{bd}\,\partial_{c}\Gamma[\nabla]^{e}{}_{af}+\Gamma[\nabla]^{f}{}_{bd}\,(\Gamma[\nabla]^{e}{}_{cg}\,\Gamma[\nabla]^{g}{}_{af}-\Gamma[\nabla]^{e}{}_{ag}\,\Gamma[\nabla]^{g}{}_{cf}-\partial_{a}\Gamma[\nabla]^{e}{}_{cf}+\partial_{c}\Gamma[\nabla]^{e}{}_{af})+\\
&\Gamma[\nabla]^{f}{}_{ad}\,(-\Gamma[\nabla]^{e}{}_{cg}\,\Gamma[\nabla]^{g}{}_{bf}+\Gamma[\nabla]^{e}{}_{bg}\,\Gamma[\nabla]^{g}{}_{cf}+\partial_{b}\Gamma[\nabla]^{e}{}_{cf}-\partial_{c}\Gamma[\nabla]^{e}{}_{bf})+\\
&2\,\Gamma[\nabla]^{f}{}_{ad}\,\partial_{c}\Gamma[\nabla]^{e}{}_{bf}-\Gamma[\nabla]^{f}{}_{ad}\,(\Gamma[\nabla]^{e}{}_{cg}\,\Gamma[\nabla]^{g}{}_{bf}-\Gamma[\nabla]^{e}{}_{bg}\,\Gamma[\nabla]^{g}{}_{cf}-\partial_{b}\Gamma[\nabla]^{e}{}_{cf}+\partial_{c}\Gamma[\nabla]^{e}{}_{bf})+\\
&\Gamma[\nabla]^{e}{}_{bf}\,(-\Gamma[\nabla]^{f}{}_{cg}\,\Gamma[\nabla]^{g}{}_{ad}+\Gamma[\nabla]^{f}{}_{ag}\,\Gamma[\nabla]^{g}{}_{cd}+\partial_{a}\Gamma[\nabla]^{f}{}_{cd}-\partial_{c}\Gamma[\nabla]^{f}{}_{ad})+\\
&2\,\Gamma[\nabla]^{e}{}_{bf}\,\partial_{c}\Gamma[\nabla]^{f}{}_{ad}-\Gamma[\nabla]^{e}{}_{bf}\,(\Gamma[\nabla]^{f}{}_{cg}\,\Gamma[\nabla]^{g}{}_{ad}-\Gamma[\nabla]^{f}{}_{ag}\,\Gamma[\nabla]^{g}{}_{cd}-\partial_{a}\Gamma[\nabla]^{f}{}_{cd}+\partial_{c}\Gamma[\nabla]^{f}{}_{ad})-\\
&\Gamma[\nabla]^{e}{}_{af}\,(-\Gamma[\nabla]^{f}{}_{cg}\,\Gamma[\nabla]^{g}{}_{bd}+\Gamma[\nabla]^{f}{}_{bg}\,\Gamma[\nabla]^{g}{}_{cd}+\partial_{b}\Gamma[\nabla]^{f}{}_{cd}-\partial_{c}\Gamma[\nabla]^{f}{}_{bd})-2\,\Gamma[\nabla]^{e}{}_{af}\,\partial_{c}\Gamma[\nabla]^{f}{}_{bd}+\\
&\Gamma[\nabla]^{e}{}_{af}\,(\Gamma[\nabla]^{f}{}_{cg}\,\Gamma[\nabla]^{g}{}_{bd}-\Gamma[\nabla]^{f}{}_{bg}\,\Gamma[\nabla]^{g}{}_{cd}-\partial_{b}\Gamma[\nabla]^{f}{}_{cd}+\partial_{c}\Gamma[\nabla]^{f}{}_{bd})-2\,\partial_{c}\partial_{a}\Gamma[\nabla]^{e}{}_{bd}+\\
&2\,\partial_{c}\partial_{b}\Gamma[\nabla]^{e}{}_{ad}-\Gamma[\nabla]^{f}{}_{bc}\,(-\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{ad}+\Gamma[\nabla]^{e}{}_{ag}\,\Gamma[\nabla]^{g}{}_{fd}+\partial_{a}\Gamma[\nabla]^{e}{}_{fd}-\partial_{f}\Gamma[\nabla]^{e}{}_{ad})+\\
&\Gamma[\nabla]^{f}{}_{cb}\,(-\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{ad}+\Gamma[\nabla]^{e}{}_{ag}\,\Gamma[\nabla]^{g}{}_{fd}+\partial_{a}\Gamma[\nabla]^{e}{}_{fd}-\partial_{f}\Gamma[\nabla]^{e}{}_{ad})+\\
&\Gamma[\nabla]^{f}{}_{bc}\,(\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{ad}-\Gamma[\nabla]^{e}{}_{ag}\,\Gamma[\nabla]^{g}{}_{fd}-\partial_{a}\Gamma[\nabla]^{e}{}_{fd}+\partial_{f}\Gamma[\nabla]^{e}{}_{ad})-\\
&(\Gamma[\nabla]^{f}{}_{bc}-\Gamma[\nabla]^{f}{}_{cb})\,(\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{ad}-\Gamma[\nabla]^{e}{}_{ag}\,\Gamma[\nabla]^{g}{}_{fd}-\partial_{a}\Gamma[\nabla]^{e}{}_{fd}+\partial_{f}\Gamma[\nabla]^{e}{}_{ad})-\\
&\Gamma[\nabla]^{f}{}_{cb}\,(\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{ad}-\Gamma[\nabla]^{e}{}_{ag}\,\Gamma[\nabla]^{g}{}_{fd}-\partial_{a}\Gamma[\nabla]^{e}{}_{fd}+\partial_{f}\Gamma[\nabla]^{e}{}_{ad})+\\
&(-\Gamma[\nabla]^{f}{}_{bc}+\Gamma[\nabla]^{f}{}_{cb})\,(\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{ad}-\Gamma[\nabla]^{e}{}_{ag}\,\Gamma[\nabla]^{g}{}_{fd}-\partial_{a}\Gamma[\nabla]^{e}{}_{fd}+\partial_{f}\Gamma[\nabla]^{e}{}_{ad})+\\
&\Gamma[\nabla]^{f}{}_{ac}\,(-\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{bd}+\Gamma[\nabla]^{e}{}_{bg}\,\Gamma[\nabla]^{g}{}_{fd}+\partial_{b}\Gamma[\nabla]^{e}{}_{fd}-\partial_{f}\Gamma[\nabla]^{e}{}_{bd})-\\
&\Gamma[\nabla]^{f}{}_{ca}\,(-\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{bd}+\Gamma[\nabla]^{e}{}_{bg}\,\Gamma[\nabla]^{g}{}_{fd}+\partial_{b}\Gamma[\nabla]^{e}{}_{fd}-\partial_{f}\Gamma[\nabla]^{e}{}_{bd})-\\
&\Gamma[\nabla]^{f}{}_{ac}\,(\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{bd}-\Gamma[\nabla]^{e}{}_{bg}\,\Gamma[\nabla]^{g}{}_{fd}-\partial_{b}\Gamma[\nabla]^{e}{}_{fd}+\partial_{f}\Gamma[\nabla]^{e}{}_{bd})+\\
&(\Gamma[\nabla]^{f}{}_{ac}-\Gamma[\nabla]^{f}{}_{ca})\,(\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{bd}-\Gamma[\nabla]^{e}{}_{bg}\,\Gamma[\nabla]^{g}{}_{fd}-\partial_{b}\Gamma[\nabla]^{e}{}_{fd}+\partial_{f}\Gamma[\nabla]^{e}{}_{bd})+\\
&\Gamma[\nabla]^{f}{}_{ca}\,(\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{bd}-\Gamma[\nabla]^{e}{}_{bg}\,\Gamma[\nabla]^{g}{}_{fd}-\partial_{b}\Gamma[\nabla]^{e}{}_{fd}+\partial_{f}\Gamma[\nabla]^{e}{}_{bd})-\\
&(-\Gamma[\nabla]^{f}{}_{ac}+\Gamma[\nabla]^{f}{}_{ca})\,(\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{bd}-\Gamma[\nabla]^{e}{}_{bg}\,\Gamma[\nabla]^{g}{}_{fd}-\partial_{b}\Gamma[\nabla]^{e}{}_{fd}+\partial_{f}\Gamma[\nabla]^{e}{}_{bd})-\\
&\Gamma[\nabla]^{f}{}_{ab}\,(-\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{cd}+\Gamma[\nabla]^{e}{}_{cg}\,\Gamma[\nabla]^{g}{}_{fd}+\partial_{c}\Gamma[\nabla]^{e}{}_{fd}-\partial_{f}\Gamma[\nabla]^{e}{}_{cd})+\\
&\Gamma[\nabla]^{f}{}_{ba}\,(-\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{cd}+\Gamma[\nabla]^{e}{}_{cg}\,\Gamma[\nabla]^{g}{}_{fd}+\partial_{c}\Gamma[\nabla]^{e}{}_{fd}-\partial_{f}\Gamma[\nabla]^{e}{}_{cd})+\\
&\Gamma[\nabla]^{f}{}_{ab}\,(\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{cd}-\Gamma[\nabla]^{e}{}_{cg}\,\Gamma[\nabla]^{g}{}_{fd}-\partial_{c}\Gamma[\nabla]^{e}{}_{fd}+\partial_{f}\Gamma[\nabla]^{e}{}_{cd})-\\
&(\Gamma[\nabla]^{f}{}_{ab}-\Gamma[\nabla]^{f}{}_{ba})\,(\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{cd}-\Gamma[\nabla]^{e}{}_{cg}\,\Gamma[\nabla]^{g}{}_{fd}-\partial_{c}\Gamma[\nabla]^{e}{}_{fd}+\partial_{f}\Gamma[\nabla]^{e}{}_{cd})-\\
&\Gamma[\nabla]^{f}{}_{ba}\,(\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{cd}-\Gamma[\nabla]^{e}{}_{cg}\,\Gamma[\nabla]^{g}{}_{fd}-\partial_{c}\Gamma[\nabla]^{e}{}_{fd}+\partial_{f}\Gamma[\nabla]^{e}{}_{cd})+\\
&(-\Gamma[\nabla]^{f}{}_{ab}+\Gamma[\nabla]^{f}{}_{ba})\,(\Gamma[\nabla]^{e}{}_{fg}\,\Gamma[\nabla]^{g}{}_{cd}-\Gamma[\nabla]^{e}{}_{cg}\,\Gamma[\nabla]^{g}{}_{fd}-\partial_{c}\Gamma[\nabla]^{e}{}_{fd}+\partial_{f}\Gamma[\nabla]^{e}{}_{cd})
\end{aligned}
$$

```
In[474]:=
      % // Expand
```

Out[474]=

$$2\,\Gamma[\nabla]^{e}{}_{cf}\,\Gamma[\nabla]^{f}{}_{bg}\,\Gamma[\nabla]^{g}{}_{ad} - 2\,\Gamma[\nabla]^{e}{}_{bf}\,\Gamma[\nabla]^{f}{}_{cg}\,\Gamma[\nabla]^{g}{}_{ad} + 2\,\Gamma[\nabla]^{e}{}_{cg}\,\Gamma[\nabla]^{f}{}_{bd}\,\Gamma[\nabla]^{g}{}_{af} -$$
$$2\,\Gamma[\nabla]^{e}{}_{bg}\,\Gamma[\nabla]^{f}{}_{cd}\,\Gamma[\nabla]^{g}{}_{af} - 2\,\Gamma[\nabla]^{e}{}_{cf}\,\Gamma[\nabla]^{f}{}_{ag}\,\Gamma[\nabla]^{g}{}_{bd} + 2\,\Gamma[\nabla]^{e}{}_{af}\,\Gamma[\nabla]^{f}{}_{cg}\,\Gamma[\nabla]^{g}{}_{bd} -$$
$$2\,\Gamma[\nabla]^{e}{}_{cg}\,\Gamma[\nabla]^{f}{}_{ad}\,\Gamma[\nabla]^{g}{}_{bf} + 2\,\Gamma[\nabla]^{e}{}_{ag}\,\Gamma[\nabla]^{f}{}_{cd}\,\Gamma[\nabla]^{g}{}_{bf} + 2\,\Gamma[\nabla]^{e}{}_{bf}\,\Gamma[\nabla]^{f}{}_{ag}\,\Gamma[\nabla]^{g}{}_{cd} -$$
$$2\,\Gamma[\nabla]^{e}{}_{af}\,\Gamma[\nabla]^{f}{}_{bg}\,\Gamma[\nabla]^{g}{}_{cd} + 2\,\Gamma[\nabla]^{e}{}_{bg}\,\Gamma[\nabla]^{f}{}_{ad}\,\Gamma[\nabla]^{g}{}_{cf} - 2\,\Gamma[\nabla]^{e}{}_{ag}\,\Gamma[\nabla]^{f}{}_{bd}\,\Gamma[\nabla]^{g}{}_{cf} -$$
$$2\,\partial_a\partial_b\Gamma[\nabla]^{e}{}_{cd} + 2\,\partial_a\partial_c\Gamma[\nabla]^{e}{}_{bd} + 2\,\partial_b\partial_a\Gamma[\nabla]^{e}{}_{cd} - 2\,\partial_b\partial_c\Gamma[\nabla]^{e}{}_{ad} - 2\,\partial_c\partial_a\Gamma[\nabla]^{e}{}_{bd} + 2\,\partial_c\partial_b\Gamma[\nabla]^{e}{}_{ad}$$

```
In[475]:=
      % // ToCanonical

Out[475]=
      0
```

Clean up:

```
In[476]:=
      UndefCovD[CD]

        ** UndefTensor: Undefined non-symmetric Christoffel tensor ChristoffelCD

        ** UndefTensor: Undefined non-symmetric Ricci tensor RicciCD

        ** UndefTensor: Undefined Riemann tensor RiemannCD

        ** UndefTensor: Undefined torsion tensor TorsionCD

        ** UndefCovD: Undefined covariant derivative CD
```

6.4. Constant symbols

It is important to discuss the concept of a constant in `xTensor`‘. A scalar field `c[]` is a constant if it is defined on no manifold. However a nonscalar tensor field cannot be constant on the base manifold of its indices (actually `xTensor`‘ automatically defines the tensor to be a field on the base manifolds on its indices). On the other hand we have to use symbols denoting constants (as Newton's constant). We shall use the command `DefConstantSymbol` to define those constants.

`DefConstantSymbol`	Define a constant
`UndefConstantSymbol`	Undefine a constant
`$ConstantSymbols`	List of defined constants
`ConstantSymbolQ`	Check a constant symbol

Functions that operate with constants.

This defines a constant scalar field:

```
In[477]:=
      DefTensor[const[], {}]

        ** DefTensor: Defining tensor const[].
```

```
In[478]:=
      ?const

         Global`const

         Dagger[const] ^= const

         DependenciesOfTensor[const] ^= {}

         Info[const] ^= {tensor, }

         PrintAs[const] ^= const

         SlotsOfTensor[const] ^= {}

         SymmetryGroupOfTensor[const] ^= StrongGenSet[{}, GenSet[]]

         TensorID[const] ^= {}

         xTensorQ[const] ^= True
```

By default, it is not automatic to check that a derivative on an object living on a different manifold is zero because that is a slow process that could be internally happening too often.

```
In[479]:=
      Cd[-a][const[]]

Out[479]=
      ∇_a const

In[480]:=
      CheckZeroDerivative[%]

Out[480]=
      0
```

We can automate it for a particular derivative (or for all):

```
In[481]:=
      CheckZeroDerivativeStart[PD]

In[482]:=
      PD[-a][const[]]

Out[482]=
      0

In[483]:=
      CheckZeroDerivativeStop[PD]
```

An arbitrary symbol is not understood by default as a constant. We must define it as a constant:

```
In[484]:=
      Cd[-a][GNewton]

Out[484]=
      ∇_a GNewton
```

```
In[485]:=
     DefConstantSymbol[GNewton]

        ** DefConstantSymbol: Defining constant symbol GNewton.

In[486]:=
     Cd[-a][GNewton]

Out[486]=
     0

In[487]:=
     UndefConstantSymbol[GNewton]

        ** UndefConstantSymbol: Undefined constant symbol GNewton

In[488]:=
     PD[-a][GNewton]

Out[488]=
```
∂_aGNewton

6.5. Parameter derivatives

Given a parameter defined with `DefParameter` we can take parametric derivatives of expressions which depend on that parameter, using `ParamD`. For historical reasons `xTensor‘` has a generic parametric derivative called `OverDot` such that every field is considered to depend on the corresponding (undefined) parameter.

`ParamD`	Parametric derivative with respect to a defined parameter
`OverDot`	Generic parametric derivative with respect to an unspecified parameter

Parametric derivatives.

The generic parametric derivative `OverDot` (Dot is used by *Mathematica* as inner scalar product) is represented by a (very) small over dot:

```
In[489]:=
     OverDot[r[]^2]

Out[489]=
```
$2\,\dot{r}\,r$

`OverDot` has been defined to commute with partial derivatives, but not with other covariant derivatives or with Lie derivatives:

```
In[490]:=
     OverDot[PD[-a][r[]]] // InputForm

Out[490]//InputForm=
     PD[-a][OverDot[r[]]]

In[491]:=
     OverDot[Cd[-a][r[]]] // InputForm

Out[491]//InputForm=
     OverDot[Cd[-a][r[]]]
```

All fields currently defined do not depend on external parameters, so that we define new objects:

```
In[492]:=
      ParamD[time][r[]]

Out[492]=
      0

In[493]:=
      DefTensor[Q[-a], {M3, time}]

          ** DefTensor: Defining tensor Q[-a].

In[494]:=
      ?Q

         Global`Q

          Dagger[Q] ^= Q

          DependenciesOfTensor[Q] ^= {time, M3}

          Info[Q] ^= {tensor, }

          PrintAs[Q] ^= Q

          SlotsOfTensor[Q] ^= {-TangentM3}

          SymmetryGroupOfTensor[Q] ^= StrongGenSet[{}, GenSet[]]

          TensorID[Q] ^= {}

          xTensorQ[Q] ^= True
```

Currently I have not found a good representation for the parametric derivatives, so that they remain as they are:

```
In[495]:=
      ParamD[time][3 Q[-a] v[a]]
```

Out[495]=

$3\, v^a$ `ParamD[time]`$[Q_a]$

```
In[496]:=
      ParamD[time, time][3 Q[-a] v[a]]
```

Out[496]=

$3\, v^a$ `ParamD[time, time]`$[Q_a]$

6.6. Lie derivatives

`xTensor`` does not define independent Lie derivatives, but a single command `LieD` which depends both on a contravariant vector field and on the expression that we want to derive.

`LieD`	Lie derivative
`LieDToCovD`	Expansion of Lie derivative in terms of covariant derivatives

Lie derivatives.

Lie derivatives of expressions with respect to a contravariant vector field are not automatically expanded in terms of derivatives.

```
In[497]:=
      LieD[v[a]][T[-b, c]]
```

Out[497]=

$\mathcal{L}_v T_b{}^c$

However, we can expand them using any derivative (`PD` is the default).

```
In[498]:=
      LieDToCovD[LieD[w[a]][T[-b, c]], PD]
```

Out[498]=

$w^a \partial_a T_b{}^c - T_b{}^a \partial_a w^c + T_a{}^c \partial_b w^a$

```
In[499]:=
      LieDToCovD[LieD[w[a]][T[-b, c]], Cd]
```

Out[499]=

$w^a (\nabla_a T_b{}^c) - T_b{}^a (\nabla_a w^c) + T_a{}^c (\nabla_b w^a)$

The derivative can also be expanded directly:

```
In[500]:=
      LieD[w[a], Cd][T[-b, c]]
```

Out[500]=

$w^a (\nabla_a T_b{}^c) - T_b{}^a (\nabla_a w^c) + T_a{}^c (\nabla_b w^a)$

`LieD` knows the Leibniz rule:

```
In[501]:=
      LieD[w[a]][T[a] T[-a] ]
```

Out[501]=

$T^a (\mathcal{L}_w T_a) + T_a (\mathcal{L}_w T^a)$

We can have any expression in the "directional argument". Recall however that it is not a tensorial slot (it hence cannot be described with an abstract index):

```
In[502]:=
      LieD[7 r[] w[a] + v[a]][T[a]]
```

Out[502]=

$\mathcal{L}_v T^a + 7 (\mathcal{L}_{r\, w^a} T^a)$

6.7. Directional derivatives

From the mathematical point of view, the simplest concept of a derivative is that of a "derivation", a mapping from scalar fields to scalar fields. In fact, the concept of tanget space at a point is constructed as the space of derivations at that point. We can work with operators acting as derivations (or "directional derivatives") using the `Dir` head.

The object `PD[Dir[v[a]]]` can be interpreted as the directional derivative along the vector `v[a]`:

```
In[503]:=
      PD[Dir[w[a]]][r[]]

Out[503]=
      ∂_w r
```

and it has all the expected properties:

```
In[504]:=
      PD[Dir[ 3 S[a, Dir[v[-b]]] + w[a] ]][ 1 / r[]^2 ]

Out[504]=
```

$$-\frac{2\,(3\,\partial_{\#}r+\partial_{w}r)}{r^{3}}$$

Actually, it is possible to work directly with the derivations acting on scalars, saving 6 brackets (!) in the notation. For example we define the derivation `wder` associated to the vector `w[a]`:

```
In[505]:=
      wder := PD[Dir[w[a]]]

In[506]:=
      wder[ r[] ]

Out[506]=
      ∂_w r
```

Or even better, in prefix notation:

```
In[507]:=
      wder@r[]

Out[507]=
      ∂_w r

In[508]:=
      ScalarQ[%]

Out[508]=
      True
```

6.8. Lie brackets

The Lie bracket of two vector fields is a third vector field.

Using abstract indices a bracket can be expressed as

```
In[509]:=
      DefTensor[x[a], M3]

          ** DefTensor: Defining tensor x[a].
```

```
In[510]:=
      Bracket[a][w[b], x[b]]
```

Out[510]=

$[w^b, x^b]^a$

Note that the indices of the vector fields are ultraindices. The index of the resulting vector field is represented outside the bracket.

It has almost all expected properties: bilinearity, antisymmetry, derivation, ..., but not yet the Jacobi rule.

```
In[511]:=
      Bracket[a][x[b] + w[b], 3 w[b] + r[] x[b]]
```

Out[511]=

$-3\,[w^b, x^b]^a + r\,[w^b, x^b]^a + x^a\,\partial_w r + x^a\,\partial_x r$

```
In[512]:=
      UndefTensor /@ {x, w};

          ** UndefTensor: Undefined tensor x

          ** UndefTensor: Undefined tensor w
```

6.9. Derivatives on inner vbundles

It is possible to define a derivative which acts on an inner vector bundle (as used in Yang−Mills gauge theories).

Define a connection on a complex inner vbundle:

```
In[513]:=
      DefCovD[ICD[-a], InnerC, {"*", "D"}]

          ** DefCovD: Defining covariant derivative ICD[-a].

          ** DefTensor: Defining vanishing torsion tensor TorsionICD[a, -b, -c].

          ** DefTensor: Defining symmetric Christoffel tensor ChristoffelICD[a, -b, -c].

          ** DefTensor: Defining Riemann tensor
           RiemannICD[-a, -b, -c, d]. Antisymmetric only in the first pair.

          ** DefTensor: Defining non-symmetric Ricci tensor RicciICD[-a, -b].

          ** DefCovD:  Contractions of Riemann automatically replaced by Ricci.

          ** DefTensor: Defining
           nonsymmetric AChristoffel tensor  AChristoffelICD[𝔄, -b, -ℭ].

          ** DefTensor: Defining
           nonsymmetric AChristoffel tensor  AChristoffelICD†[𝔄†, -b, -ℭ†].

          ** DefTensor: Defining FRiemann tensor
           FRiemannICD[-a, -b, -ℭ, 𝔇]. Antisymmetric only in the first pair.

          ** DefTensor: Defining FRiemann tensor
           FRiemannICD†[-a, -b, -ℭ†, 𝔇†]. Antisymmetric only in the first pair.
```

We see that a single torsion tensor has been defined, but two Christoffel and two Riemann tensors. The inner Christoffel is called `AChristoffelICD` and the inner Riemann tensor is called `FRiemannICD`. They are both complex tensors and therefore have their complex conjugates. Note the different vbundles of their indices.

```
In[514]:=
        ?AChristoffelICD

          Global‘AChristoffelICD

           Dagger[AChristoffelICD] ^= AChristoffelICD†

           DependenciesOfTensor[AChristoffelICD] ^= {M3}

           Info[AChristoffelICD] ^= {nonsymmetric AChristoffel tensor , }

           MasterOf[AChristoffelICD] ^= ICD

           PrintAs[AChristoffelICD] ^= A[D]

           ServantsOf[AChristoffelICD] ^= {AChristoffelICD†}

           SlotsOfTensor[AChristoffelICD] ^= {InnerC, -TangentM3, -InnerC}

           SymmetryGroupOfTensor[AChristoffelICD] ^= StrongGenSet[{}, GenSet[]]

           TensorID[AChristoffelICD] ^= {AChristoffel, ICD, PD}

           xTensorQ[AChristoffelICD] ^= True

In[515]:=
        ?FRiemannICD

          Global‘FRiemannICD

           Dagger[FRiemannICD] ^= FRiemannICD†

           DependenciesOfTensor[FRiemannICD] ^= {M3}

           Info[FRiemannICD] ^= {FRiemann tensor, Antisymmetric only in the first pair.}

           MasterOf[FRiemannICD] ^= ICD

           PrintAs[FRiemannICD] ^= FICD

           ServantsOf[FRiemannICD] ^= {FRiemannICD†}

           SlotsOfTensor[FRiemannICD] ^= {-TangentM3, -TangentM3, -InnerC, InnerC}

           SymmetryGroupOfTensor[FRiemannICD] ^= StrongGenSet[{1}, GenSet[-Cycles[{1, 2}]]]

           TensorID[FRiemannICD] ^= {FRiemann, ICD}

           xTensorQ[FRiemannICD] ^= True
```

```
In[516]:=
      ? AChristoffelICD†

         Global‘AChristoffelICD†

          Dagger[AChristoffelICD†] ^= AChristoffelICD

          DependenciesOfTensor[AChristoffelICD†] ^= {M3}

          Info[AChristoffelICD†] ^= {nonsymmetric AChristoffel tensor , }

          MasterOf[AChristoffelICD†] ^= AChristoffelICD

          PrintAs[AChristoffelICD†] ^= A[D]†

          SlotsOfTensor[AChristoffelICD†] ^= {InnerC†, -TangentM3, -InnerC†}

          SymmetryGroupOfTensor[AChristoffelICD†] ^= StrongGenSet[{}, GenSet[]]

          TensorID[AChristoffelICD†] ^= {AChristoffel, ICD, PD}

          xTensorQ[AChristoffelICD†] ^= True
In[517]:=
      ? FRiemannICD†

         Global‘FRiemannICD†

          Dagger[FRiemannICD†] ^= FRiemannICD

          DependenciesOfTensor[FRiemannICD†] ^= {M3}

          Info[FRiemannICD†] ^= {FRiemann tensor, Antisymmetric only in the first pair.}

          MasterOf[FRiemannICD†] ^= FRiemannICD

          PrintAs[FRiemannICD†] ^= FICD†

          SlotsOfTensor[FRiemannICD†] ^= {-TangentM3, -TangentM3, -InnerC†, InnerC†}

          SymmetryGroupOfTensor[FRiemannICD†] ^=
           StrongGenSet[{1}, GenSet[-Cycles[{1, 2}]]]

          TensorID[FRiemannICD†] ^= {FRiemann, ICD}

          xTensorQ[FRiemannICD†] ^= True
```

The derivative `ICD` has curvature in both the tangent bundle `TangentM3` and in the inner bundle `InnerC`. That becomes most apparent when changing to a different derivative:

```
In[518]:=
      DefTensor[X[a, Ƀ], M3, Dagger → Complex]

          ** DefTensor: Defining tensor X[a, Ƀ].

          ** DefTensor: Defining tensor X†[a, Ƀ†].
```

In[519]:=
```
ICD[-c][X[a, 𝔅]]
```
Out[519]=
$$D_c\, X^{a\mathfrak{B}}$$

In[520]:=
```
ChangeCovD[%, ICD]
```
Out[520]=
$$A[D]^{\mathfrak{B}}{}_{c\mathfrak{A}}\, X^{a\mathfrak{A}} + \Gamma[D]^{a}{}_{cb}\, X^{b\mathfrak{B}} + \partial_c X^{a\mathfrak{B}}$$

In[521]:=
```
ChangeCovD[%, PD, ICD]
```
Out[521]=
$$D_c\, X^{a\mathfrak{B}}$$

or when commuting two derivatives:

In[522]:=
```
ICD[-c]@ICD[-d]@X[a, 𝔅]
```
Out[522]=
$$D_c\, D_d\, X^{a\mathfrak{B}}$$

In[523]:=
```
SortCovDs[%]
```
Out[523]=
$$\mathrm{FICD}_{dc\mathfrak{A}}{}^{\mathfrak{B}}\, X^{a\mathfrak{A}} + R[D]_{dcb}{}^{a}\, X^{b\mathfrak{B}} + D_d\, D_c\, X^{a\mathfrak{B}}$$

For their complex conjugates:

In[524]:=
```
ICD[-c][X[a, 𝔅]] // Dagger
```
Out[524]=
$$D_c\, X\dagger^{a\mathfrak{B}\dagger}$$

In[525]:=
```
ChangeCovD[%, ICD]
```
Out[525]=
$$A[D]\,\dagger^{\mathfrak{B}\dagger}{}_{c\mathfrak{A}\dagger}\, X\dagger^{a\mathfrak{A}\dagger} + \Gamma[D]^{a}{}_{cb}\, X\dagger^{b\mathfrak{B}\dagger} + \partial_c X\dagger^{a\mathfrak{B}\dagger}$$

In[526]:=
```
ICD[-c]@ICD[-d]@X[a, 𝔅] // Dagger
```
Out[526]=
$$D_c\, D_d\, X\dagger^{a\mathfrak{B}\dagger}$$

In[527]:=
```
SortCovDs[%]
```
Out[527]=
$$\mathrm{FICD}\dagger_{dc\mathfrak{A}\dagger}{}^{\mathfrak{B}\dagger}\, X\dagger^{a\mathfrak{A}\dagger} + R[D]_{dcb}{}^{a}\, X\dagger^{b\mathfrak{B}\dagger} + D_d\, D_c\, X\dagger^{a\mathfrak{B}\dagger}$$

```
In[528]:=
      UndefTensor[X]

          ** UndefTensor: Undefined tensor X†

          ** UndefTensor: Undefined tensor X
```

An important concept, introduced in version 0.9.1 of `xTensor'`, is that of extension of a derivative. Given a derivative `Cd` it is possible to define a second one `CD` in such a way that the associated tensors of `CD` on its tangent bundle will be those of `Cd`. We say then that `CD` extends `Cd`, or that `CD` has been constructed by extension of `Cd`.

Let us take the `Cd` derivative as base covd:

```
In[529]:=
      $CovDs

Out[529]=
      {PD, Cd, ICD}
```

We now extend that derivative to `CD`:

```
In[530]:=
      DefCovD[CD[-a], InnerC, {"#", "D"}, ExtendedFrom → Cd]

          ** DefCovD: Defining covariant derivative CD[-a].

          ** DefTensor: Defining
           nonsymmetric AChristoffel tensor  AChristoffelCD[𝔄, -b, -C].

          ** DefTensor: Defining
           nonsymmetric AChristoffel tensor  AChristoffelCD†[𝔄†, -b, -C†].

          ** DefTensor: Defining FRiemann tensor
           FRiemannCD[-a, -b, -C, D]. Antisymmetric only in the first pair.

          ** DefTensor: Defining FRiemann tensor
           FRiemannCD†[-a, -b, -C†, D†]. Antisymmetric only in the first pair.
```

Now the tensors associated to the extended derivative will be those of the base derivative:

```
In[531]:=
      {ChristoffelCD[a, -b, -c], RiemannCD[-a, -b, -c, d], TorsionCD[a, -b, -c]}

Out[531]=
```
$\{\Gamma[\nabla]^{a}{}_{bc},\ R[\nabla]_{abc}{}^{d},\ 0\}$

And then we have the inner curvature of the extended derivative:

```
In[532]:=
      {AChristoffelCD[𝔄, -b, -C], FRiemannCD[-a, -b, -C, D]}

Out[532]=
```
$\{A[D]^{\mathfrak{A}}{}_{bC},\ FCD_{abC}{}^{D}\}$

We can undefine the extended derivative, without removing any of the base derivative objects:

```
In[533]:=
     UndefCovD[CD]

        ** UndefTensor: Undefined nonsymmetric AChristoffel tensor  AChristoffelCD†

        ** UndefTensor: Undefined nonsymmetric AChristoffel tensor  AChristoffelCD

        ** UndefTensor: Undefined FRiemann tensor FRiemannCD†

        ** UndefTensor: Undefined FRiemann tensor FRiemannCD

        ** UndefCovD: Undefined covariant derivative CD

In[534]:=
     {ChristoffelCd[a, -b, -c], TorsionCd[a, -b, -c],
      RiemannCd[-a, -b, -c, d], RicciCd[-a, -b]}
```

Out[534]=

$\{\Gamma[\nabla]^{a}{}_{bc},\ 0,\ R[\nabla]_{abc}{}^{d},\ R[\nabla]_{ab}\}$

```
In[535]:=
     {ChristoffelCD[a, -b, -c], TorsionCD[a, -b, -c],
      RiemannCD[-a, -b, -c, d], RicciCD[-a, -b]}

Out[535]=
     {ChristoffelCD[a, -b, -c], TorsionCD[a, -b, -c],
      RiemannCD[-a, -b, -c, d], RicciCD[-a, -b]}
```

■ 7. Metrics

7.1. Definition

Now we introduce metrics on the vbundles.

DefMetric	Define a metric on a single vbundle
DefProductMetric	Define a metric on a product vbundle
UndefMetric	Undefine a metric
$Metrics	List of defined metrics
$ProductMetrics	List of defined product metrics
MetricQ	Validate name of metric

Definition of a metric.

We define a metric `metricg` with negative determinant. The associated covariant derivative will be called `CD` and will be represented with a semicolon under `IndexForm`. All the needed associated tensors are defined (note that epsilon has a suffix `metricg` and the other tensors have a suffix `CD`). Because we use indices of `M3` it is understood that `metricg` will be a metric on `M3`.

```
In[536]:=
     DefMetric[-1, metricg[-a, -b], CD, {";", "∇"}, PrintAs -> "g"]

        ** DefTensor: Defining symmetric metric tensor metricg[-a, -b].

        ** DefTensor: Defining antisymmetric tensor epsilonmetricg[a, b, c].

        ** DefCovD: Defining covariant derivative CD[-a].

        ** DefTensor: Defining vanishing torsion tensor TorsionCD[a, -b, -c].

        ** DefTensor: Defining symmetric Christoffel tensor ChristoffelCD[a, -b, -c].

        ** DefTensor: Defining Riemann tensor RiemannCD[-a, -b, -c, -d].

        ** DefTensor: Defining symmetric Ricci tensor RicciCD[-a, -b].

        ** DefCovD:  Contractions of Riemann automatically replaced by Ricci.

        ** DefTensor: Defining Ricci scalar RicciScalarCD[].

        ** DefCovD:  Contractions of Ricci automatically replaced by RicciScalar.

        ** DefTensor: Defining symmetric Einstein tensor EinsteinCD[-a, -b].

        ** DefTensor: Defining vanishing Weyl tensor WeylCD[-a, -b, -c, -d].

        ** DefTensor: Defining symmetric TFRicci tensor TFRicciCD[-a, -b].

           Rules {1, 2} have been declared as DownValues for TFRicciCD.

        ** DefCovD:  Computing RiemannToWeylRules for dim 3

        ** DefCovD:  Computing RicciToTFRicci for dim 3

        ** DefCovD:  Computing RicciToEinsteinRules for dim 3

In[537]:=
     RiemannCD[-a, b, -b, -c]

Out[537]=
```
$-R[\nabla]_{ac}$

```
In[538]:=
     % // InputForm

Out[538]//InputForm=
     -RicciCD[-a, -c]

In[539]:=
     RiemannCD[-a, -b, b, a]

Out[539]=
     -R[∇]

In[540]:=
     % // InputForm

Out[540]//InputForm=
     -RicciScalarCD[]
```

```
In[541]:=
      CD[-c][metricg[-a, -b]]

Out[541]=
      0

In[542]:=
      Cd[-c][metricg[-a, -b]]

Out[542]=
      ▽c gab

In[543]:=
      CD[-c][epsilonmetricg[a, b, c]]

Out[543]=
      0

In[544]:=
      CD[-a][EinsteinCD[b, c]]

Out[544]=
      ▽a G[▽]bc

In[545]:=
      CD[-a][EinsteinCD[a, b]]

Out[545]=
      0
```

Its Weyl tensor is zero because the manifold is 3d:

```
In[546]:=
      WeylCD[-a, -b, -c, -d]

Out[546]=
      0
```

There is also the traceless part of the Ricci tensor:

```
In[547]:=
      TFRicciCD[-a, -b]

Out[547]=
      S[▽]ab

In[548]:=
      TFRicciCD[a, -a]

Out[548]=
      0
```

Now xTensor` accepts index structures that do not correspond to the original definition, as long as they only involve defined metrics:

```
In[549]:=
      SlotsOfTensor[T]

Out[549]=
      {TangentM3, TangentM3, -TangentM3}

In[550]:=
      Validate[T[-a, -b, -c]]

Out[550]=
```

T_{abc}

```
In[551]:=
      DefTensor[X[a, b, C], {M3, S2}]

          ** DefTensor: Defining tensor X[a, b, C].

In[552]:=
      SlotsOfTensor[X]

Out[552]=
      {TangentM3, TangentM3, TangentS2}

In[553]:=
      Validate[X[a, b, -C]]

      Validate::error : Invalid character of index in tensor X

Out[553]=
```

ERROR$[X^{ab}{}_{C}]$

```
In[554]:=
      UndefTensor[X]

          ** UndefTensor: Undefined tensor X
```

Expression of the Christoffel symbols (with respect to PD) in terms of the metric:

```
In[555]:=
      CD[-a][v[-b]] // ChangeCovD

Out[555]=
```

$-\Gamma[\nabla]^{c}{}_{ab}\, v_c + \partial_a v_b$

```
In[556]:=
      % // ChristoffelToGradMetric

Out[556]=
```

$\partial_a v_b - \frac{1}{2} g^{cd}\, v_c\, (\partial_a g_{bd} + \partial_b g_{ad} - \partial_d g_{ab})$

7.2. Contraction

Metrics are not contracted by default because in general that obscures the manipulation of indices and the canonicalization process becomes slower. However, `xTensor‘` has a number of commands that perform metric contraction. When the expression is a pure tensor contraction is a simple process, but when there are derivatives which are not associated to the (first) metric of the vbundle then problematic situations could develop. There are two situations we need to worry about: one is the process of contraction itself, shown in this subsection; the other is the influence of the metric in the process of canonicalization, studied in the next subsection.

`ContractMetric`	Perform metric contractions without differentiating metrics
`OverDerivatives`	Option to perform metric contractions with metric differentiation if needed
`AllowUpperDerivatives`	Option stating if it is possible to contract an inverse metric with a derivative operator

Metric contraction.

Metric factors are automatically contracted with themselve:

```
In[557]:=
      metricg[a, b] metricg[-b, -c]
```

Out[557]=

$\delta^{a}{}_{c}$

```
In[558]:=
      metricg[a, -a]

Out[558]=
      3
```

However they are only contracted with other objects if required. This is to have more control on expressions formed by metric factors, and mostly to avoid unexpected index−shiftings.

```
In[559]:=
      metricg[a, b] T[-a, c]
```

Out[559]=

$g^{ab}\, T_{a}{}^{c}$

```
In[560]:=
      ContractMetric[%, metricg]
```

Out[560]=

T^{bc}

Derivatives of a metric are automatically converted so that the indices of the metric are covariant, introducing new dummies. Note that derivatives of mixed−indices metrics are zero. That means that all non−zero derivatives of metrics will have covariant indices:

```
In[561]:=
      PD[-a][metricg[b, c]]
```

Out[561]=

$-g^{bd}\, g^{ce}\, \partial_{a} g_{de}$

```
In[562]:=
      PD[-a][metricg[b, -c]]
```

```
Out[562]=
      0
```

Contraction of derivative indices is not allowed by default, but can be forced using the option `AllowUpperDerivatives`:

```
In[563]:=
      ContractMetric[metricg[a, b] PD[-b][T[c, d]], metricg]
```

Out[563]=

$g^{ab}\ \partial_b T^{cd}$

```
In[564]:=
      ContractMetric[metricg[a, b] PD[-b][T[c, d]], metricg, AllowUpperDerivatives → True]
```

```
Out[564]=
      ∂\(\(\_ \)\%a\) T^cd
```

With `ContractMetric` a metric can be contracted through its Levi−Civita derivative, but not through other derivatives:

```
In[565]:=
      ContractMetric[metricg[a, b] CD[-d][T[-a, c]], metricg]
```

Out[565]=

$\nabla_d T^{bc}$

```
In[566]:=
      ContractMetric[metricg[a, b] PD[-d][T[-a, c]], metricg]
```

Out[566]=

$g^{ab}\ \partial_d T_a{}^c$

In those cases we need to use a different function which forces contraction over derivatives, but at the price of producing new terms with derivatives of the metric:

```
In[567]:=
      ContractMetric[metricg[a, b] PD[-d][T[-a, c]], metricg, OverDerivatives → True]
```

Out[567]=

$g^{be}\ T^{ac}\ \partial_d g_{ae} + \partial_d T^{bc}$

7.3. Canonicalization with a metric

In the presence of a metric new symmetries can be added to the process of canonicalization. The command `ToCanoni−cal` has an option `UseMetricOnVBundle` that specifies on which vbundles the metric (if there is one) must be used during canonicalization. Possible values are `All` (default), `None` or a list of (maybe one) vbundles.

Tensors S and T have no symmetries and therefore their canonicalization without metric is trivial:

```
In[568]:=
      ToCanonical[S[c, a] T[-a, -b, b], UseMetricOnVBundle → None]
```

Out[568]=

$S^{ca}\ T_{ab}{}^b$

However with metric the canonical form is different because pairs of dummies are sorted such that the up−index comes first:

```
In[569]:=
      ToCanonical[S[c, a] T[-a, -b, b]]
```

Out[569]=

$S^{c}{}_{a}\, T^{ab}{}_{b}$

When derivatives are present canonicalization with a metric is harder, unless the only derivative is its Levi−Civita associated operator. In other cases, `ToCanonical` shows a warning message and changes to a different algorithm rather than using the default algorithm which just moves indices around and that could get easily confused. The new algorithm is based on a temporary internal change of the offending derivatives to the Levi−Civita connection of the metric of the corresponding vbundle. This new algorithm is much slower; compare the following timings:

We use the same tensors S and T:

```
In[570]:=
      ToCanonical[S[c, a] CD[-d][T[-a, -b, b]], UseMetricOnVBundle → None] // AbsoluteTiming
```

Out[570]=

{0.004705 Second, $S^{ca}\, (\nabla_{d}\, T_{ab}{}^{b})$}

```
In[571]:=
      ToCanonical[S[c, a] CD[-d][T[-a, -b, b]]] // AbsoluteTiming
```

Out[571]=

{0.005225 Second, $S^{c}{}_{a}\, (\nabla_{d}\, T^{ab}{}_{b})$}

But now if we use PD instead:

```
In[572]:=
      ToCanonical[S[c, a] PD[-d][T[-a, -b, b]], UseMetricOnVBundle → None] // AbsoluteTiming
```

Out[572]=

{0.005743 Second, $S^{ca}\, \partial_{d} T_{ab}{}^{b}$}

```
In[573]:=
      ToCanonical[S[c, a] PD[-d][T[-a, -b, b]]] // AbsoluteTiming

      ToCanonical::cmods : Changing derivatives to canonicalize.
```

Out[573]=

{0.033768 Second, $\Gamma[\nabla]_{ad}{}^{e}\, S^{c}{}_{e}\, T^{ab}{}_{b} + \Gamma[\nabla]^{e}{}_{da}\, S^{c}{}_{e}\, T^{ab}{}_{b} + S^{c}{}_{a}\, \partial_{d} T^{ab}{}_{b}$}

This form of canonicalization leaves lots of Christoffel hanging around. You can get rid of them in three steps: first use `ChristoffelToGradMetric`; then use `ContractMetric`,

```
In[574]:=
      Last[%] // ChristoffelToGradMetric // ContractMetric
```

Out[574]=

$$\frac{1}{2} S^{ce}\, T^{ab}{}_{b}\, \partial_{d} g_{ae} + \frac{1}{2} S^{ce}\, T^{ab}{}_{b}\, \partial_{d} g_{ea} + S^{c}{}_{a}\, \partial_{d} T^{ab}{}_{b}$$

Finally, canonicalize again, but this time without moving indices, to avoid bringing back the Christoffels:

```
In[575]:=
      ToCanonical[%, UseMetricOnVBundle → None]
```

Out[575]=

$$S^{ce}\, T^{ab}{}_{b}\, \partial_d g_{ae} + S^{c}{}_{a}\, \partial_d T^{ab}{}_{b}$$

The whole process is automatic with the option `ExpandChristoffel`:

```
In[576]:=
      ToCanonical[S[c, a] PD[-d][T[-a, -b, b]], ExpandChristoffel → True] // AbsoluteTiming

      ToCanonical::cmods : Changing derivatives to canonicalize.
```

Out[576]=

$$\{0.071727\,\text{Second},\ S^{ce}\, T^{ab}{}_{b}\, \partial_d g_{ae} + S^{c}{}_{a}\, \partial_d T^{ab}{}_{b}\}$$

This is another, somehow harder, example. Partial derivatives do not commute when having upper indices:

```
In[577]:=
      PD[-b]@PD[a]@v[-a] - PD[a]@PD[-b]@v[-a] // Simplification

      ToCanonical::cmods : Changing derivatives to canonicalize.

      ToCanonical::cmods : Changing derivatives to canonicalize.

      ToCanonical::cmods : Changing derivatives to canonicalize.

      General::stop : Further output of
         ToCanonical::cmods will be suppressed during this calculation. More…
```

Out[577]=

$$-\Gamma[\nabla]_{c}{}^{d}{}_{d}\, \Gamma[\nabla]^{c}{}_{ba}\, v^{a} - \Gamma[\nabla]^{c}{}_{ba}\, \Gamma[\nabla]^{d}{}_{cd}\, v^{a} - \Gamma[\nabla]_{ab}{}^{c}\, (\Gamma[\nabla]_{c}{}^{d}{}_{d} + \Gamma[\nabla]^{d}{}_{cd})\, v^{a} - \partial_a \partial_b v^{a} + v^{a}\, \partial_b \Gamma[\nabla]_{a}{}^{c}{}_{c} + v^{a}\, \partial_b \Gamma[\nabla]^{c}{}_{ac} + \partial_b \partial_a v^{a} - v^{a}\, \partial_c \Gamma[\nabla]_{ab}{}^{c} - v^{a}\, \partial_c \Gamma[\nabla]^{c}{}_{ba} - \Gamma[\nabla]_{abc}\, \partial\texttt{\textbackslash(\textbackslash(\textbackslash\_ \textbackslash)\textbackslash\%c\textbackslash)}\, v^{a} - \Gamma[\nabla]_{cba}\, \partial\texttt{\textbackslash(\textbackslash(\textbackslash\_ \textbackslash)\textbackslash\%c\textbackslash)}\, v^{a}$$

Now we can drop the two second derivatives, and remove the Christoffels:

```
In[578]:=
      % // SortCovDs
```

Out[578]=

$$-\Gamma[\nabla]_{c}{}^{d}{}_{d}\, \Gamma[\nabla]^{c}{}_{ba}\, v^{a} - \Gamma[\nabla]^{c}{}_{ba}\, \Gamma[\nabla]^{d}{}_{cd}\, v^{a} - \Gamma[\nabla]_{ab}{}^{c}\, (\Gamma[\nabla]_{c}{}^{d}{}_{d} + \Gamma[\nabla]^{d}{}_{cd})\, v^{a} + v^{a}\, \partial_b \Gamma[\nabla]_{a}{}^{c}{}_{c} + v^{a}\, \partial_b \Gamma[\nabla]^{c}{}_{ac} - v^{a}\, \partial_c \Gamma[\nabla]_{ab}{}^{c} - v^{a}\, \partial_c \Gamma[\nabla]^{c}{}_{ba} - \Gamma[\nabla]_{abc}\, \partial\texttt{\textbackslash(\textbackslash(\textbackslash\_ \textbackslash)\textbackslash\%c\textbackslash)}\, v^{a} - \Gamma[\nabla]_{cba}\, \partial\texttt{\textbackslash(\textbackslash(\textbackslash\_ \textbackslash)\textbackslash\%c\textbackslash)}\, v^{a}$$

```
In[579]:=
      Simplification[% // ChristoffelToGradMetric // ContractMetric,
        UseMetricOnVBundle → None] // AbsoluteTiming
```

Out[579]=

$$\{0.645404\,\text{Second},\ -\partial_b g_{ac}\, \partial\texttt{\textbackslash(\textbackslash(\textbackslash\_ \textbackslash)\textbackslash\%c\textbackslash)}\, v^{a} - g^{cd}\, g^{ef}\, v^{a}\, \partial_b g_{ce}\, \partial_f g_{ad}\}$$

General recommendations:

1. The simplest situation is having just one tangent bundle, with one metric and its Levi−Civita connection. In that case the metric can be always contracted, and we recommend to do it, just to decrease the number of indices.

2. If there are several tangent bundles, each one having one metric with its Levi−Civita connection, then the metrics can be freely contracted, and the only additional problem is that you decide whether to use CheckZeroDerivative at some intermediate steps of the calculation, or else automate that point.

3. If there is one vbundle, one metric with its Levi−Civita connection and an additional derivative (ordinary or not, that is irrelevant) then the canonicalizer must change to the slower algorithm. I recommend not to change the character of the slots of the tensors. To do that set UseMetricOnVBundle−>False in ToCanonical and never use Contract−Metric. The metric factors will be always explicit, and the number of indices in the expressions will be larger, but overall the canonicalizations will be faster.

4. If there are several vbundles with several derivatives, combine comments 2 and 3.

5. If furthermore there are several metrics on the vbundles, each with its own Levi−Civita connection, then before start crying, see next subsection.

The canonicalizer also takes into account whether indices can be moved up and down or not. This is done internally using the concept of metric−state for indices. Study next example carefully:

This expression is clearly antisymmetric under the exchange of derivatives:

```
In[580]:=
      U[a, b, c] PD[-a][v[d]] PD[-b][v[e]] metricg[-d, -e]
```

Out[580]=

$$g_{de}\, U^{abc}\, \partial_a v^d\, \partial_b v^e$$

and it is thus zero:

```
In[581]:=
      ToCanonical[%]

Out[581]=
      0
```

We contract the metric through a derivative. The expression however is now different and not zero:

```
In[582]:=
      U[a, b, c] PD[-a][v[-d]] PD[-b][v[e]] metricg[d, -e]
```

Out[582]=

$$U^{abc}\, \partial_a v_d\, \partial_b v^d$$

but the system detects it and change to the alternative algorithm:

```
In[583]:=
      ToCanonical[%]

      ToCanonical::cmods : Changing derivatives to canonicalize.
```

Out[583]=

$$-\Gamma[\nabla]_{ab}{}^{e}\, U^{c}{}_{de}\, v^{a}\, \partial\backslash(\backslash(\backslash\_\ \backslash)\backslash\%d\backslash)\, v^{b} - \Gamma[\nabla]_{ba}{}^{e}\, U^{c}{}_{de}\, v^{a}\, \partial\backslash(\backslash(\backslash\_\ \backslash)\backslash\%d\backslash)\, v^{b}$$

Another example:

```
In[584]:=
      ToCanonical[T[a, -b, c] PD[-d][v[-a]] PD[-e][v[b]] metricg[d, e]]

      ToCanonical::cmods : Changing derivatives to canonicalize.

      ToCanonical::cmods : Changing derivatives to canonicalize.

      ToCanonical::cmods : Changing derivatives to canonicalize.

      General::stop : Further output of
         ToCanonical::cmods will be suppressed during this calculation. More…

Out[584]=
```

$$-\Gamma[\nabla]_{bd}{}^{f}\, g_{ef}\, T^{abc}\, v^{d}\, \partial\backslash(\backslash(\backslash\_\ \backslash)\backslash\%e\backslash)\, v_{a} - \Gamma[\nabla]_{db}{}^{f}\, g_{ef}\, T^{abc}\, v^{d}\, \partial\backslash(\backslash(\backslash\_\ \backslash)\backslash\%e\backslash)\, v_{a} + g_{de}\, T^{abc}\, \partial\backslash(\backslash(\backslash\_\ \backslash)\backslash\%d\backslash)\, v_{a}\, \partial\backslash(\backslash(\backslash\_\ \backslash)\backslash\%e\backslash)\, v_{b}$$

```
In[585]:=
      ToCanonical[T[a, -b, c] PD[-d][v[-a]] PD[-e][v[b]] metricg[d, e],
        ExpandChristoffel → True]

      ToCanonical::cmods : Changing derivatives to canonicalize.

      ToCanonical::cmods : Changing derivatives to canonicalize.

      ToCanonical::cmods : Changing derivatives to canonicalize.

      General::stop : Further output of
         ToCanonical::cmods will be suppressed during this calculation. More…

Out[585]=
```

$$T^{abc}\, \partial_{d} v_{b}\, \partial\backslash(\backslash(\backslash\_\ \backslash)\backslash\%d\backslash)\, v_{a} - T^{abc}\, v^{d}\, \partial_{e} g_{bd}\, \partial\backslash(\backslash(\backslash\_\ \backslash)\backslash\%e\backslash)\, v_{a}$$

7.4. Several metrics

It is possible to work with several metrics on the same vbundle. Of course, not all of them can raise and lower the indices of other tensors because we would inmediately loose track of the metric that was used to move a given index. The situation in `xTensor‘` is as follows. The list of metrics on a vbundle is stored as an upvalue of the vbundle for the function `MetricsOfVBundle`. The metrics are placed in that list in order of definition. The first metric is considered special and it is the only one which can move indices. All other metrics (called "frozen" metrics) are simply symmetric two−tensors with a Levi−Civita associated connection and a number of associated tensors. For a frozen metric, many of the expected rules do not work.

It is possible to remove the first metric, suddenly converting the second metric into the first of the list. I cannot think of any situation where that could be valid or just safe.

I have never seriously tested this part of the code. Please use it with maximum care!

Currently there is only one metric on `TangentM3`:

```
In[586]:=
      MetricsOfVBundle[TangentM3]

Out[586]=
      {metricg}
```

We define a second (frozen) metric:

```
In[587]:=
      DefMetric[1, frozen[-a, -b], CD2, {"|", "D"}, PrintAs → "f"]

      DefMetric::old : There are already metrics
         {metricg} in vbundle TangentM3. Defined metric is frozen.

         ** DefTensor: Defining symmetric metric tensor frozen[-a, -b].

         ** DefTensor: Defining inverse metric tensor Invfrozen[a, b]. Metric is frozen!

         ** DefMetric: Don't know yet how to define epsilon for a frozen metric.

         ** DefCovD: Defining covariant derivative CD2[-a].

         ** DefTensor: Defining vanishing torsion tensor TorsionCD2[a, -b, -c].

         ** DefTensor: Defining symmetric Christoffel tensor ChristoffelCD2[a, -b, -c].

         ** DefTensor: Defining Riemann tensor RiemannCD2[-a, -b, -c, -d].

         ** DefTensor: Defining symmetric Ricci tensor RicciCD2[-a, -b].

         ** DefCovD:  Contractions of Riemann automatically replaced by Ricci.

         ** DefTensor: Defining Ricci scalar RicciScalarCD2[].

         ** DefTensor: Defining symmetric Einstein tensor EinsteinCD2[-a, -b].

         ** DefTensor: Defining vanishing Weyl tensor WeylCD2[-a, -b, -c, -d].

         ** DefTensor: Defining symmetric TFRicci tensor TFRicciCD2[-a, -b].

         ** DefCovD:  Computing RicciToEinsteinRules for dim 3

In[588]:=
      MetricsOfVBundle[TangentM3]

Out[588]=
      {metricg, frozen}
```

Compare these two results:

```
In[589]:=
      {metricg[a, -b], frozen[a, -b]}

Out[589]=
```
$\{\delta^{a}{}_{b}, f^{a}{}_{b}\}$

That is because these three expressions behave very differently:

```
In[590]:=
      {frozen[a, c] frozen[-c, -b],
       metricg[a, c] frozen[-c, -b], metricg[a, c] metricg[-c, -b]}

Out[590]=
```
$\{f^{ac} f_{cb}, f_{cb} g^{ac}, \delta^{a}{}_{b}\}$

```
In[591]:=
      ContractMetric[%]
```

```
Out[591]=
```
$\{f^{ac}\, f_{cb},\ f^{a}{}_{b},\ \delta^{a}{}_{b}\}$

In particular the inverse of `frozen[−a,−b]` is not `frozen[a,b]`, but `Invfrozen[a,b]`:

```
In[592]:=
      Invfrozen[a, b] frozen[-b, -c]
```

```
Out[592]=
```
$\delta_{c}{}^{a}$

It is problematic to generalize typical expressions. For example this is not zero now:

```
In[593]:=
      CD2[-a][EinsteinCD2[a, b]]
```

```
Out[593]=
```
$D_a\, G[D]^{ab}$

but this is zero (though it is not encoded):

```
In[594]:=
      Invfrozen[a, b] CD2[-a][EinsteinCD2[-b, -c]]
```

```
Out[594]=
```
$f^{ab}\,(D_a\, G[D]_{bc})$

```
In[595]:=
      UndefMetric[frozen]

          ** UndefTensor: Undefined inverse metric tensor Invfrozen

          ** UndefTensor: Undefined symmetric Christoffel tensor ChristoffelCD2

          ** UndefTensor: Undefined symmetric Einstein tensor EinsteinCD2

          ** UndefTensor: Undefined symmetric Ricci tensor RicciCD2

          ** UndefTensor: Undefined Ricci scalar RicciScalarCD2

          ** UndefTensor: Undefined Riemann tensor RiemannCD2

          ** UndefTensor: Undefined symmetric TFRicci tensor TFRicciCD2

          ** UndefTensor: Undefined vanishing torsion tensor TorsionCD2

          ** UndefTensor: Undefined vanishing Weyl tensor WeylCD2

          ** UndefCovD: Undefined covariant derivative CD2

          ** UndefTensor: Undefined symmetric metric tensor frozen
```

7.5. Curvature tensors of the metric

When we define a metric, a number of curvature tensors are defined. Currently they are: `Riemann`, `Ricci`, `RicciScalar`, `Einstein`, `TFRici`, `Weyl`. Note that they are all associated to the Levi−Civita connection of the metric, and not directly to the metric. There are a number of functions to change among them.

First we have some automatic contractions. These can be prevented with the option `CurvatureRelations` of `DefCovD`. By default, contractions of the Riemann tensor are automatically replaced by the Ricci tensor:

```
In[596]:=
      RiemannCD[-a, -b, -c, a]
```

Out[596]=

$$-R[\nabla]_{bc}$$

```
In[597]:=
      % // InputForm

Out[597]//InputForm=
     -RicciCD[-b, -c]
```

And contractions of the Ricci tensor are replaced by the Ricci scalar:

```
In[598]:=
      metricg[c, d] RicciCD[-d, -c]
```

Out[598]=

$$R[\nabla]$$

```
In[599]:=
      % // InputForm

Out[599]//InputForm=
     RicciScalarCD[]

In[600]:=
      RicciCD[a, -a]
```

Out[600]=

$$R[\nabla]$$

Then we can change among the tensors:

```
In[601]:=
      EinsteinCD[-a, -b] // EinsteinToRicci
```

Out[601]=

$$R[\nabla]_{ab} - \frac{1}{2} g_{ba} R[\nabla]$$

```
In[602]:=
      % // RicciToEinstein
```

Out[602]=

$$G[\nabla]_{ab}$$

In a 3d manifold the Weyl tensor is zero:

```
In[603]:=
      WeylCD[-a, -b, -c, -d]

Out[603]=
      0

In[604]:=
      RiemannCD[-a, -b, -c, -d] // RiemannToWeyl

Out[604]=
```

$$g_{bd}\,R[\triangledown]_{ac} - g_{bc}\,R[\triangledown]_{ad} - g_{ad}\,R[\triangledown]_{bc} + g_{ac}\,R[\triangledown]_{bd} + \frac{1}{2}\,g_{ad}\,g_{bc}\,R[\triangledown] - \frac{1}{2}\,g_{ac}\,g_{bd}\,R[\triangledown]$$

We can also change between Ricci and TFRicci:

```
In[605]:=
      RicciCD[-a, -b] // RicciToTFRicci

Out[605]=
```

$$\frac{1}{3}\,g_{ab}\,R[\triangledown] + S[\triangledown]_{ab}$$

```
In[606]:=
      % // TFRicciToRicci

Out[606]=
```

$$R[\triangledown]_{ab} + \frac{1}{3}\,g_{ab}\,R[\triangledown] - \frac{1}{3}\,g_{ba}\,R[\triangledown]$$

RiemannToWeyl	Expand Riemann tensors into Weyl, Ricci and RicciScalar tensors
WeylToRiemann	Expand Weyl tensors into Riemann, Ricci and RicciScalar tensors
RicciToEinstein	Expand Ricci tensors into Einstein and RicciScalar tensors
EinsteinToRicci	Expand Einstein tensors into Ricci and RicciScalar tensors
RicciToTFRicci	Expand Ricci tensors into TFRicci and RicciScalar tensors
TFRicciToRicci	Expand TFRicci tensors into Ricci and RicciScalar tensors

Relations among curvature objects.

As we said, we can work with connections which are compatible with a given metric field but have torsion too.

We define a new connection associated to our metric field, but this time with torsion. The Riemann tensor is antisymmetric in both pairs, but now those pairs cannot be exchanged. Hence the Ricci tensor is not symmetric. Currently Weyl is not even defined:

```
In[607]:=
      DefCovD[CDT[-a], {"#", "D"}, Torsion → True, FromMetric → metricg]

          ** DefCovD: Defining covariant derivative CDT[-a].

          ** DefTensor: Defining torsion tensor TorsionCDT[a, -b, -c].

          ** DefTensor: Defining
           non-symmetric Christoffel tensor ChristoffelCDT[a, -b, -c].

          ** DefTensor: Defining Riemann tensor
           RiemannCDT[-a, -b, -c, -d]. Antisymmetric pairs cannot be exchanged.

          ** DefTensor: Defining non-symmetric Ricci tensor RicciCDT[-a, -b].

          ** DefCovD:  Contractions of Riemann automatically replaced by Ricci.

          ** DefTensor: Defining Ricci scalar RicciScalarCDT[].

          ** DefCovD:  Contractions of Ricci automatically replaced by RicciScalar.

          ** DefTensor: Defining non-symmetric Einstein tensor EinsteinCDT[-a, -b].

          ** DefTensor: Defining non-symmetric TFRicci tensor TFRicciCDT[-a, -b].

             Rules {1, 2} have been declared as DownValues for TFRicciCDT.

          ** DefCovD:  Computing RicciToTFRicci for dim 3

          ** DefCovD:  Computing RicciToEinsteinRules for dim 3
```

The difference of covariant derivatives

```
In[608]:=
      CDT[-a][v[b]] - CD[-a][v[b]]
```

Out[608]=

$$-(\nabla_a v^b) + D_a v^b$$

is given by a Christoffel tensor

```
In[609]:=
      ChangeCovD[%, CDT, CD]

          ** DefTensor: Defining tensor ChristoffelCDCDT[a, -b, -c].
```

Out[609]=

$$-\Gamma[\nabla,D]^b{}_{ac}\, v^c$$

which can be expressed uniquely in terms of torsion and metric fields:

```
In[610]:=
      % // ChristoffelToGradMetric
```

Out[610]=

$$-\frac{1}{2} g^{bd} \left(T[D]_{acd} + T[D]_{cad} - T[D]_{dac}\right) v^c$$

```
In[611]:=
      UndefCovD[CDT]

          ** UndefTensor: Undefined tensor ChristoffelCDCDT

          ** UndefTensor: Undefined non-symmetric Christoffel tensor ChristoffelCDT

          ** UndefTensor: Undefined non-symmetric Einstein tensor EinsteinCDT

          ** UndefTensor: Undefined non-symmetric Ricci tensor RicciCDT

          ** UndefTensor: Undefined Ricci scalar RicciScalarCDT

          ** UndefTensor: Undefined Riemann tensor RiemannCDT

          ** UndefTensor: Undefined non-symmetric TFRicci tensor TFRicciCDT

          ** UndefTensor: Undefined torsion tensor TorsionCDT

          ** UndefCovD: Undefined covariant derivative CDT
```

7.6. Product metrics

Spacetimes with a high degree of symmetry can be sometimes locally decomposed as products of reduced manifolds and the orbits of symmetry. The metric of the whole manifold can then be given in terms of metrics on simpler vbundles. Currently there is a stupid limitation to two subvbundles. It will be dropped soon.

Define a product metric. We already have a scalar `r[]` on `M3`. Now we define another scalar `w[]` on `S2`:

```
In[612]:=
      DefTensor[w[], S2]

          ** DefTensor: Defining tensor w[].

In[613]:=
      Catch@
       DefProductMetric[g5[-μ, -ν], {{TangentM3, w[]}, {TangentS2, r[]}}, Cd5, {"#", "D"}]

      DefProductMetric::nometric : VBundle TangentS2 does not have a metric.
```

```
In[614]:=
     DefMetric[1, gamma[-A, -B], CdS, {":", "D"}, PrintAs → "γ"]

        ** DefTensor: Defining symmetric metric tensor gamma[-A, -B].

        ** DefTensor: Defining antisymmetric tensor epsilongamma[A, B].

        ** DefCovD: Defining covariant derivative CdS[-A].

        ** DefTensor: Defining vanishing torsion tensor TorsionCdS[A, -B, -C].

        ** DefTensor: Defining symmetric Christoffel tensor ChristoffelCdS[A, -B, -C].

        ** DefTensor: Defining Riemann tensor RiemannCdS[-A, -B, -C, -D].

        ** DefTensor: Defining symmetric Ricci tensor RicciCdS[-A, -B].

        ** DefCovD:  Contractions of Riemann automatically replaced by Ricci.

        ** DefTensor: Defining Ricci scalar RicciScalarCdS[].

        ** DefCovD:  Contractions of Ricci automatically replaced by RicciScalar.

        ** DefTensor: Defining vanishing Einstein tensor EinsteinCdS[-A, -B].

        ** DefTensor: Defining vanishing Weyl tensor WeylCdS[-A, -B, -C, -D].

        ** DefTensor: Defining vanishing TFRicci tensor TFRicciCdS[-A, -B].

        ** DefCovD:  Computing RiemannToWeylRules for dim 2

        ** DefCovD:  Computing RicciToTFRicci for dim 2

        ** DefCovD:  Computing RicciToEinsteinRules for dim 2
```

We define the warped metric `w[]^2 MetricOfVBundle[TangentM3]+r[]^2 MetricOfVBundle[TangentS2]` :

```
In[615]:=
     DefProductMetric[g5[-μ, -ν], {{TangentM3, w[]}, {TangentS2, r[]}}, Cd5, {"#", "D"}]

        ** DefTensor: Defining symmetric metric tensor g5[-μ, -ν].

        ** DefTensor: Defining antisymmetric tensor epsilong5[η, λ, μ, ν, ρ].

        ** DefCovD: Defining covariant derivative Cd5[-μ].

        ** DefTensor: Defining vanishing torsion tensor TorsionCd5[η, -λ, -μ].

        ** DefTensor: Defining symmetric Christoffel tensor ChristoffelCd5[η, -λ, -μ].

        ** DefTensor: Defining Riemann tensor RiemannCd5[-η, -λ, -μ, -ν].

        ** DefTensor: Defining symmetric Ricci tensor RicciCd5[-η, -λ].

        ** DefTensor: Defining Ricci scalar RicciScalarCd5[].

        ** DefTensor: Defining symmetric Einstein tensor EinsteinCd5[-η, -λ].

        ** DefTensor: Defining Weyl tensor WeylCd5[-η, -λ, -μ, -ν].

           Rules {1, 2, 3, 4, 5, 6, 7, 8} have been declared as DownValues for WeylCd5.

        ** DefTensor: Defining symmetric TFRicci tensor TFRicciCd5[-η, -λ].

           Rules {1, 2} have been declared as DownValues for TFRicciCd5.

        ** DefCovD:  Computing RiemannToWeylRules for dim 5

        ** DefCovD:  Computing RicciToTFRicci for dim 5

        ** DefCovD:  Computing RicciToEinsteinRules for dim 5
```

`ExpandProducMetric`	Expansion of the metric of a product manifold into objects of its submanifolds

Computations with product metrics.

The delta tensors are automatically expanded:

```
In[616]:=
     {g5[a, -b], g5[A, -B]}
```

Out[616]=

$\{\delta^{a}{}_{b}, \delta^{A}{}_{B}\}$

```
In[617]:=
     InputForm /@ %

Out[617]=
     {delta[a, -b], delta[A, -B]}
```

Now we can compute any object on M5 in terms of objects of M3 and S2:

```
In[618]:=
     ExpandProductMetric[{g5[-a, -b], g5[-a, -B], g5[-A, -B]}, g5]
```

Out[618]=

$$\{g_{ab}\, w^2,\ 0,\ \gamma_{AB}\, r^2\}$$

```
In[619]:=
     ExpandProductMetric[{ChristoffelCd5[A, -B, -C],
        ChristoffelCd5[a, -B, -C], ChristoffelCd5[A, b, -C], ChristoffelCd5[a, -b, -C],
        ChristoffelCd5[A, -b, -c], ChristoffelCd5[a, -b, -c]}, g5] // ContractMetric
```

Out[619]=

$$\left\{\Gamma[D]^{A}{}_{BC},\ -\frac{\gamma_{BC}\, g^{ab}\, r\,(\nabla_b r)}{w^2},\ \frac{\delta^A{}_C\, g^{ba}\,(\nabla_a r)}{r\, w^2},\ \frac{\delta^a{}_b\,(D_C w)}{w},\ -\frac{\gamma^{AB}\, g_{bc}\, w\,(D_B w)}{r^2},\ \Gamma[\nabla]^{a}{}_{bc}\right\}$$

```
In[620]:=
     RiemannCd5[-a, -b, -c, -d] // ExpandProductMetric // ContractMetric // Simplify
```

Out[620]=

$$\frac{w^2\,(r^2\, R[\nabla]_{abcd} + \gamma^{AB}\,(g_{ad}\, g_{bc} - g_{ac}\, g_{bd})\,(D_A w)\,(D_B w))}{r^2}$$

```
In[621]:=
     RicciScalarCd5[] // ExpandProductMetric // ContractMetric // Simplification
```

Out[621]=

$$\frac{r^2\, R[\nabla] + R[D]\, w^2 - 2\, g_{ab}\,((\nabla^a r)\,(\nabla^b r) + 2\, r\,(\nabla^b \nabla^a r)) - 6\,\gamma_{AB}\,((D^A w)\,(D^B w) + w\,(D^B D^A w))}{r^2\, w^2}$$

By default derivative indices cannot be contracted with metric tensors. This behaviour can be changed using the option AllowUpperDerivatives:

```
In[622]:=
     ContractMetric[%, AllowUpperDerivatives → True] // Simplification
```

Out[622]=

$$\frac{r^2\, R[\nabla] + R[D]\, w^2 - 4\, r\,(\nabla_a \nabla^a r) - 2\,(\nabla_a r)\,(\nabla^a r) - 6\, w\,(D_A D^A w) - 6\,(D_A w)\,(D^A w)}{r^2\, w^2}$$

Clean up

```
In[623]:=
     UndefMetric[g5]

        ** UndefTensor: Undefined antisymmetric tensor epsilong5

        ** UndefTensor: Undefined symmetric Christoffel tensor ChristoffelCd5

        ** UndefTensor: Undefined symmetric Einstein tensor EinsteinCd5

        ** UndefTensor: Undefined symmetric Ricci tensor RicciCd5

        ** UndefTensor: Undefined Ricci scalar RicciScalarCd5

        ** UndefTensor: Undefined Riemann tensor RiemannCd5

        ** UndefTensor: Undefined symmetric TFRicci tensor TFRicciCd5

        ** UndefTensor: Undefined vanishing torsion tensor TorsionCd5

        ** UndefTensor: Undefined Weyl tensor WeylCd5

        ** UndefCovD: Undefined covariant derivative Cd5

        ** UndefTensor: Undefined symmetric metric tensor g5
```

```
In[624]:=
     UndefMetric[metricg]

        ** UndefTensor: Undefined antisymmetric tensor epsilonmetricg

        ** UndefTensor: Undefined symmetric Christoffel tensor ChristoffelCD

        ** UndefTensor: Undefined symmetric Einstein tensor EinsteinCD

        ** UndefTensor: Undefined symmetric Ricci tensor RicciCD

        ** UndefTensor: Undefined Ricci scalar RicciScalarCD

        ** UndefTensor: Undefined Riemann tensor RiemannCD

        ** UndefTensor: Undefined symmetric TFRicci tensor TFRicciCD

        ** UndefTensor: Undefined vanishing torsion tensor TorsionCD

        ** UndefTensor: Undefined vanishing Weyl tensor WeylCD

        ** UndefCovD: Undefined covariant derivative CD

        ** UndefTensor: Undefined symmetric metric tensor metricg
```

```
In[625]:=
      UndefMetric[gamma]

          ** UndefTensor: Undefined antisymmetric tensor epsilongamma

          ** UndefTensor: Undefined symmetric Christoffel tensor ChristoffelCdS

          ** UndefTensor: Undefined vanishing Einstein tensor EinsteinCdS

          ** UndefTensor: Undefined symmetric Ricci tensor RicciCdS

          ** UndefTensor: Undefined Ricci scalar RicciScalarCdS

          ** UndefTensor: Undefined Riemann tensor RiemannCdS

          ** UndefTensor: Undefined vanishing TFRicci tensor TFRicciCdS

          ** UndefTensor: Undefined vanishing torsion tensor TorsionCdS

          ** UndefTensor: Undefined vanishing Weyl tensor WeylCdS

          ** UndefCovD: Undefined covariant derivative CdS

          ** UndefTensor: Undefined symmetric metric tensor gamma

In[626]:=
      UndefManifold[M5]

          ** UndefVBundle: Undefined vbundle TangentM5

          ** UndefManifold: Undefined manifold M5
```

7.7. Flat and Cartesian metrics

A flat metric is one without curvature. We define its Christoffel and epsilon objects, and then zero curvature tensors.

We can define a flat metric using the option `FlatMetric`:

```
In[627]:=
      DefMetric[-1, gflat[-a, -b], Cdflat, {"%", "∂"}, FlatMetric → True]

          ** DefTensor: Defining symmetric metric tensor gflat[-a, -b].

          ** DefTensor: Defining antisymmetric tensor epsilonflat[a, b, c].

          ** DefCovD: Defining covariant derivative Cdflat[-a].

          ** DefTensor: Defining vanishing torsion tensor TorsionCdflat[a, -b, -c].

          ** DefTensor: Defining
        symmetric Christoffel tensor ChristoffelCdflat[a, -b, -c].

          ** DefTensor: Defining vanishing Riemann tensor RiemannCdflat[-a, -b, -c, -d].

          ** DefTensor: Defining vanishing Ricci tensor RicciCdflat[-a, -b].

          ** DefTensor: Defining vanishing Ricci scalar RicciScalarCdflat[].

          ** DefTensor: Defining vanishing Einstein tensor EinsteinCdflat[-a, -b].

          ** DefTensor: Defining vanishing Weyl tensor WeylCdflat[-a, -b, -c, -d].

          ** DefTensor: Defining vanishing TFRicci tensor TFRicciCdflat[-a, -b].
```

```
In[628]:=
      UndefMetric[gflat]

          ** UndefTensor: Undefined antisymmetric tensor epsilongflat

          ** UndefTensor: Undefined symmetric Christoffel tensor ChristoffelCdflat

          ** UndefTensor: Undefined vanishing Einstein tensor EinsteinCdflat

          ** UndefTensor: Undefined vanishing Ricci tensor RicciCdflat

          ** UndefTensor: Undefined vanishing Ricci scalar RicciScalarCdflat

          ** UndefTensor: Undefined vanishing Riemann tensor RiemannCdflat

          ** UndefTensor: Undefined vanishing TFRicci tensor TFRicciCdflat

          ** UndefTensor: Undefined vanishing torsion tensor TorsionCdflat

          ** UndefTensor: Undefined vanishing Weyl tensor WeylCdflat

          ** UndefCovD: Undefined covariant derivative Cdflat

          ** UndefTensor: Undefined symmetric metric tensor gflat
```

A delicate issue is that of a Cartesian metric. The concept of a flat metric is covariantly defined and therefore perfectly fits in the current structure of `xTensor‘`. However we say that a metric in a given basis of vectors is Cartesian if all metric components in that basis are constant (not necessarily of unit modulus). For a flat metric coordinate systems of that kind always exist. We introduce the possibility of associating `PD` as the covariant derivative of a flat metric. That means that `PD` is the partial derivative of one of those coordinate systems with respect to that metric and hence `PD` is not general anymore. This is an ugly trick for a problem which is only properly solved in the twin package `xCoba‘`.

We redefine the flat metric. In this case not even the Christoffel symbol is defined. By definition it is zero.

```
In[629]:=
      DefMetric[-1, gflat[-a, -b], PD, {",", "∂"}, FlatMetric → True]

          ** DefTensor: Defining symmetric metric tensor gflat[-a, -b].

          ** DefTensor: Defining antisymmetric tensor epsilongflat[a, b, c].

In[630]:=
      PD[-a][gflat[-b, -c]]

Out[630]=
      0

In[631]:=
      UndefMetric[gflat]

          ** UndefTensor: Undefined antisymmetric tensor epsilongflat

          ** UndefTensor: Undefined symmetric metric tensor gflat
```

7.8. Induced metrics

`xTensor‘` has a simple way to define the induced metric obtained from projection of another metric along hypersurface−orthogonal vector fields, as it is typically done in the ADM 3+1 decomposition of the spacetime metric:

We first define the global ambient metric:

```
In[632]:=
      DefMetric[-1, metricg[-a, -b], CD, {";", "∇"}, PrintAs -> "g"];

          ** DefTensor: Defining symmetric metric tensor metricg[-a, -b].

          ** DefTensor: Defining antisymmetric tensor epsilonmetricg[a, b, c].

          ** DefCovD: Defining covariant derivative CD[-a].

          ** DefTensor: Defining vanishing torsion tensor TorsionCD[a, -b, -c].

          ** DefTensor: Defining symmetric Christoffel tensor ChristoffelCD[a, -b, -c].

          ** DefTensor: Defining Riemann tensor RiemannCD[-a, -b, -c, -d].

          ** DefTensor: Defining symmetric Ricci tensor RicciCD[-a, -b].

          ** DefCovD:  Contractions of Riemann automatically replaced by Ricci.

          ** DefTensor: Defining Ricci scalar RicciScalarCD[].

          ** DefCovD:  Contractions of Ricci automatically replaced by RicciScalar.

          ** DefTensor: Defining symmetric Einstein tensor EinsteinCD[-a, -b].

          ** DefTensor: Defining vanishing Weyl tensor WeylCD[-a, -b, -c, -d].

          ** DefTensor: Defining symmetric TFRicci tensor TFRicciCD[-a, -b].

             Rules {1, 2} have been declared as DownValues for TFRicciCD.

          ** DefCovD:  Computing RiemannToWeylRules for dim 3

          ** DefCovD:  Computing RicciToTFRicci for dim 3

          ** DefCovD:  Computing RicciToEinsteinRules for dim 3
```

Define the timelike normal to the slices. The use of `MakeRule` and `AutomaticRules` will be explained later.

```
In[633]:=
      DefTensor[n[a], M3]

          ** DefTensor: Defining tensor n[a].

In[634]:=
      AutomaticRules[n, MakeRule[{n[a] n[-a], -1}]]

             Rules {1} have been declared as UpValues for n.

In[635]:=
      AutomaticRules[n, MakeRule[{metricg[-a, -b] n[a] n[b], -1}]]

             Rules {1, 2} have been declared as UpValues for n.
```

We choose the conventions as in Choptuik:

```
In[636]:=
      $ExtrinsicKSign = -1;
      $AccelerationSign = -1;
```

Now we can define the induced metric associated to the pair {ambient metric, orthogonal vector field}. Note that even though there is another metric on the tangent bundle, an induced metric is not considered to be a frozen metric.

```
In[638]:=
      DefMetric[1, metrich[-a, -b], cd,
       {"|", "D"}, InducedFrom → {metricg, n}, PrintAs -> "h"]

      DefMetric::old : There are already metrics
         {metricg} in vbundle TangentM3. Defined metric is frozen.

         ** DefTensor: Defining symmetric metric tensor metrich[-a, -b].

         ** DefTensor: Defining antisymmetric tensor epsilonmetrich[a, b].

         ** DefCovD: Defining covariant derivative cd[-a].

         ** DefTensor: Defining vanishing torsion tensor Torsioncd[a, -b, -c].

         ** DefTensor: Defining symmetric Christoffel tensor Christoffelcd[a, -b, -c].

         ** DefTensor: Defining Riemann tensor Riemanncd[-a, -b, -c, -d].

         ** DefTensor: Defining symmetric Ricci tensor Riccicd[-a, -b].

         ** DefCovD:  Contractions of Riemann automatically replaced by Ricci.

         ** DefTensor: Defining Ricci scalar RicciScalarcd[].

         ** DefCovD:  Contractions of Ricci automatically replaced by RicciScalar.

         ** DefTensor: Defining symmetric Einstein tensor Einsteincd[-a, -b].

         ** DefTensor: Defining vanishing Weyl tensor Weylcd[-a, -b, -c, -d].

         ** DefTensor: Defining symmetric TFRicci tensor TFRiccicd[-a, -b].

            Rules {1, 2} have been declared as DownValues for TFRiccicd.

         ** DefCovD:  Computing RiemannToWeylRules for dim 3

         ** DefCovD:  Computing RicciToTFRicci for dim 3

         ** DefCovD:  Computing RicciToEinsteinRules for dim 3

         ** DefTensor: Defining extrinsic curvature tensor
          ExtrinsicKmetrich[a, b]. Associated to vector n

         ** DefTensor: Defining acceleration vector
          Accelerationn[a]. Associated to vector n

         ** DefInertHead: Defining projector inert-head Projectormetrich.
```

For a hypersurface orthogonal vector we can express the acceleration vector in terms of a lapse function:

```
In[639]:=
      DefTensor[lapse[], M3, PrintAs -> "α"]

         ** DefTensor: Defining tensor lapse[].

In[640]:=
      LapseRule = Accelerationn[a_] → cd[a][lapse[]] / lapse[]

Out[640]=
```

$$An^a \to \frac{D^a \alpha}{\alpha}$$

We define a generic vector field and a spatial vector field:

```
In[641]:=
      DefTensor[V[-a], M3]

         ** DefTensor: Defining tensor V[-a].
```

```
In[642]:=
      DefTensor[W[-a], M3, OrthogonalTo → {n[a]}, ProjectedWith → {metrich[a, -b]}]

         ** DefTensor: Defining tensor W[-a].
```

The first operation we need is the decomposition of any expression along the normal vector. This is given by the command `InducedDecomposition`, which requires as second argument the pair {projector, normal}. The decomposition process does not change the input; it simply rewrites it in a different way:

Any expression can be decomposed in projected and orthogonal parts with respect to the vector field. Note the special output of the head `Projectormetrich`:

```
In[643]:=
      InducedDecomposition[W[-a], {metrich, n}]
```

Out[643]=

$$W_a$$

```
In[644]:=
      InducedDecomposition[V[-a], {metrich, n}]
```

Out[644]=

$$\underset{h}{P}[V_a] - n_a\,\mathrm{Scalar}[n^a\,V_a]$$

```
In[645]:=
      InducedDecomposition[7 V[-a] V[-b] + W[-a] W[-b], {metrich, n}]
```

Out[645]=

$$7\left(\underset{h}{P}[V_a] - n_a\,\mathrm{Scalar}[n^a\,V_a]\right)\left(\underset{h}{P}[V_b] - n_b\,\mathrm{Scalar}[n^a\,V_a]\right) + W_a\,W_b$$

```
In[646]:=
      InducedDecomposition[RiemannCD[-a, -b, -c, -d], {metrich, n}]
```

Out[646]=

$$\begin{aligned}
&\underset{h}{P}[R[\nabla]_{abcd}] - n_d\,\underset{h}{P}[n^e\;R[\nabla]_{abce}] - n_c\,\underset{h}{P}[n^e\;R[\nabla]_{abed}] + \\
&\quad n_c\,n_d\,\underset{h}{P}[n^e\;n^f\;R[\nabla]_{abef}] - n_b\,\underset{h}{P}[n^e\;R[\nabla]_{aecd}] + n_b\,n_d\,\underset{h}{P}[n^e\;n^f\;R[\nabla]_{aecf}] + \\
&\quad n_b\,n_c\,\underset{h}{P}[n^e\;n^f\;R[\nabla]_{aefd}] - n_b\,n_c\,n_d\,\underset{h}{P}[n^e\;n^f\;n^g\;R[\nabla]_{aefg}] - n_a\,\underset{h}{P}[n^e\;R[\nabla]_{ebcd}] + \\
&\quad n_a\,n_d\,\underset{h}{P}[n^e\;n^f\;R[\nabla]_{ebcf}] + n_a\,n_c\,\underset{h}{P}[n^e\;n^f\;R[\nabla]_{ebfd}] - n_a\,n_c\,n_d\,\underset{h}{P}[n^e\;n^f\;n^g\;R[\nabla]_{ebfg}] + \\
&\quad n_a\,n_b\,\underset{h}{P}[n^e\;n^f\;R[\nabla]_{efcd}] - n_a\,n_b\,n_d\,\underset{h}{P}[n^e\;n^f\;n^g\;R[\nabla]_{efcg}] - \\
&\quad n_a\,n_b\,n_c\,\underset{h}{P}[n^e\;n^f\;n^g\;R[\nabla]_{efgd}] + n_a\,n_b\,n_c\,n_d\,\mathrm{Scalar}[n^a\,n^b\,n^c\,n^d\,R[\nabla]_{abcd}]
\end{aligned}$$

```
In[647]:=
      ToCanonical[%]
```

Out[647]=

$$\underset{h}{P}[R[\nabla]_{abcd}] - n_d \underset{h}{P}[n^e\ R[\nabla]_{abce}] + n_c \underset{h}{P}[n^e\ R[\nabla]_{abde}] - n_b \underset{h}{P}[n^e\ R[\nabla]_{aecd}] + n_b\, n_d \underset{h}{P}[n^e\ n^f\ R[\nabla]_{aecf}] - n_b\, n_c \underset{h}{P}[n^e\ n^f\ R[\nabla]_{aedf}] + n_a \underset{h}{P}[n^e\ R[\nabla]_{becd}] - n_a\, n_d \underset{h}{P}[n^e\ n^f\ R[\nabla]_{becf}] + n_a\, n_c \underset{h}{P}[n^e\ n^f\ R[\nabla]_{bedf}]$$

Projection along the normal vector is automatic:

```
In[648]:=
      n[d] % // Expand
```

Out[648]=

$$\underset{h}{P}[n^d\ R[\nabla]_{abcd}] - n_b \underset{h}{P}[n^d\ n^e\ R[\nabla]_{adce}] + n_a \underset{h}{P}[n^d\ n^e\ R[\nabla]_{bdce}]$$

```
In[649]:=
      n[c] % // Expand

Out[649]=
      0

In[650]:=
      n[b] %% // Expand
```

Out[650]=

$$\underset{h}{P}[n^b\ n^d\ R[\nabla]_{abcd}]$$

The head `Projectormetrich` is the formal projector, but we now need a command which actually performs the projection. This is done with `ProjectWith[metrich]`. Do not confuse them. They have been constructed as a matched pair, and it is very useful to go from the former to the latter.

As long as we need to work only with the projected part of `RicciCD`, we can use this object:

```
In[651]:=
      Projectormetrich[RicciCD[-a, -b]]
```

Out[651]=

$$\underset{h}{P}[R[\nabla]_{ab}]$$

However, if we need to perform the projection, we can do this:

```
In[652]:=
      % /. Projectormetrich → ProjectWith[metrich]
```

Out[652]=

$$h_a{}^c\ h_b{}^d\ R[\nabla]_{cd}$$

Finally, we can expand the projectors as

```
In[653]:=
      % // ProjectorToMetric
```

Out[653]=

$(\delta_a{}^c + n_a n^c)\,(\delta_b{}^d + n_b n^d)\,R[\nabla]_{cd}$

```
In[654]:=
      % // Expand
```

Out[654]=

$R[\nabla]_{ab} + n_b n^c R[\nabla]_{ac} + n_a n^c R[\nabla]_{cb} + n_a n_b n^c n^d R[\nabla]_{cd}$

The projection process can be easily undone using `InducedDecomposition`:

```
In[655]:=
      InducedDecomposition[%, {metrich, n}] // ToCanonical
```

Out[655]=

$\underset{h}{P}[R[\nabla]_{ab}]$

`InducedDecomposition`	Decompose an expression along the normal vector
`Projector`	Formal projector head
`ProjectWith`	Actual projection operator, using the projected metric
`ProjectorToMetric`	Convert projected metric into sum of metric and projector onto the normal
`MetricToProjector`	Convert metric into sum of projected metric and projector onto the normal

Projection operators.

The second issue is the manipulation of derivatives. The projected metric has its own Levi−Civita connection, which is a proper covariant derivative only when acting on objects on the projected manifold. The relation between the original and the new covariant derivatives is given by a kind of "Christoffel" encoded in two objects: the extrinsic curvature `ExtrinsicK` tensor and the `Acceleration` vector:

This is the derivative of the normal (note the sign convention):

```
In[656]:=
      CD[-a][n[b]]
```

Out[656]=

$\nabla_a n^b$

```
In[657]:=
      % // GradNormalToExtrinsicK
```

Out[657]=

$-Kh_a{}^b - An^b n_a$

It is useful to remove derivatives in favour of these two tensors whenever possible. For example for "spatial" vectors:

```
In[658]:=
      n[-a] CD[-b][W[a]]
```

Out[658]=

$n_a\ (\nabla_b W^a)$

```
In[659]:=
      % // GradNormalToExtrinsicK // ContractMetric
```

Out[659]=

$Kh_b{}^a W_a + An^a n_b W_a$

The opposite command also exists:

```
In[660]:=
      ExtrinsicKmetrich[-a, -b]
```

Out[660]=

Kh_{ab}

```
In[661]:=
      % // ExtrinsicKToGradNormal // ContractMetric
```

Out[661]=

$-An_b n_a - \nabla_a n_b$

Let us now project the derivative on a "spatial" vector:

```
In[662]:=
      CD[-c][W[-b]]
```

Out[662]=

$\nabla_c W_b$

```
In[663]:=
      % // ProjectWith[metrich]
```

Out[663]=

$h_b{}^a h_c{}^d\ (\nabla_d W_a)$

```
In[664]:=
      % // ProjectorToMetric
```

Out[664]=

$(\delta_b{}^a + n^a n_b)\ (\delta_c{}^d + n_c n^d)\ (\nabla_d W_a)$

```
In[665]:=
      % // Expand
```

Out[665]=

$n^a n_c\ (\nabla_a W_b) + n^a n_b\ (\nabla_c W_a) + \nabla_c W_b + n^a n_b n_c n^d\ (\nabla_d W_a)$

```
In[666]:=
      % // GradNormalToExtrinsicK
```

Out[666]=

$$-n_b\ (-Kh_c{}^a - An^a\ n_c)\ W_a - n_b\ n_c\ (-Kh_d{}^a - An^a\ n_d)\ n^d\ W_a + n^a\ n_c\ (\nabla_a W_b) + \nabla_c W_b$$

```
In[667]:=
      % // Simplification
```

Out[667]=

$$Kh_{ca}\ n_b\ W^a + n^a\ n_c\ (\nabla_a W_b) + \nabla_c W_b$$

A different way to do it is this:

```
In[668]:=
      cd[-c][W[-b]]
```

Out[668]=

$$D_c W_b$$

```
In[669]:=
      ProjectDerivative[%]
```

Out[669]=

$$\underset{h}{P}[\nabla_c W_b]$$

```
In[670]:=
      % /. Projectormetrich → ProjectWith[metrich]
```

Out[670]=

$$h_b{}^a\ h_c{}^d\ (\nabla_d W_a)$$

`ExtrinsicK`	Extrinsic curvature tensor
`Acceleration`	Acceleration vector
`GradNormalToExtrinsicK`	Convert from derivatives of the normal to the extrinsic curvature tensor
`ExtrinsicKToGradNormal`	Convert form the extrinsic curvature tensor to derivatives of the normal
`ProjectDerivative`	Convert derivatives on the projected vbundle into projected derivatives

Induced derivatives.

A very important detail is that of the Leibnitz rule for induced derivatives. It is not valid on expressions not on the projected vbundle!

The vector `W` lives on the projected vbundle and therefore the Leibnitz rule is valid:

```
In[671]:=
      cd[-a][W[b] W[c]]
```

Out[671]=

$$W^c\ (D_a W^b) + W^b\ (D_a W^c)$$

If there are free non−spatial indices we throw an error message:

```
In[672]:=
      Catch@cd[-a][V[b] V[c]]

      Validate::error : Induced derivative acting on non-projected expression.
```

With dummy indices we can expand both spatial indices and non−spatial indices:

```
In[673]:=
      cd[-a][W[a] W[b]]
```

Out[673]=
$W^b (D_a W^a) + W^a (D_a W^b)$

```
In[674]:=
      cd[-a][V[b] V[-b]]
```

Out[674]=
$\underset{h}{P}[V^b] \left(D_a \underset{h}{P}[V_b]\right) + \underset{h}{P}[V_b] \left(D_a \underset{h}{P}[V^b]\right) -$
$\text{Scalar}[n_a V^a] (D_a \text{Scalar}[n^a V_a]) - \text{Scalar}[n^a V_a] (D_a \text{Scalar}[n_a V^a])$

In other cases, we prefer to show a warning and live the expression untouched:

```
In[675]:=
      Catch@cd[-a][V[a] V[b]]

      Validate::error : Induced derivative acting on non-projected expression.
```

The most important application of this in GR is the ADM−type decomposition of a spacetime. In the rest of this section we reproduce some equations from the well−organized notes on ADM by M. Choptuik.

1) Foliations and normals. We use his same notations for the normal vector and the acceleration vector:

```
In[676]:=
      n[a] CD[-a][n[b]] // GradNormalToExtrinsicK // Expand
```

Out[676]=
An^b

2) The projection tensor and the spatial metric. We do not follow the sign switch of York. The orthogonal projector is our pair Projectormetrich / ProjectWith[metrich] as explained above.

3) The spatial derivative operator and curvature tensor.

```
In[677]:=
      cd[-a][metrich[-b, -c]]

Out[677]=
      0

In[678]:=
      (cd[-a]@cd[-b][#] - cd[-b]@cd[-a][#]) &@W[-c]
```

Out[678]=
$D_a D_b W_c - D_b D_a W_c$

```
In[679]:=
      % // SortCovDs // ToCanonical
```

Out[679]=

$R[D]_{abcd} W^{d}$

```
In[680]:=
      Riemanncd[-a, -b, -c, d] {n[a], n[b], n[c], n[-d]}

Out[680]=
      {0, 0, 0, 0}

In[681]:=
      Riemanncd[-a, -c, -b, c]
```

Out[681]=

$R[D]_{ab}$

The contraction of Ricci into RicciScalar does not work:

```
In[682]:=
      {Riccicd[-a, a], metrich[a, b] Riccicd[-a, -b], metricg[a, b] Riccicd[-a, -b]}
```

Out[682]=

$\{R[D], R[D], g^{ab} R[D]_{ab}\}$

4) The extrinsic curvature tensor

```
In[683]:=
      CD[-a][n[-b]] // GradNormalToExtrinsicK // ContractMetric
```

Out[683]=

$-Kh_{ab} - An_{b} n_{a}$

```
In[684]:=
      Antisymmetrize[ProjectWith[metrich][%], {-a, -b}] // ContractMetric // ToCanonical

Out[684]=
      0

In[685]:=
      ProjectWith[metrich][-1 / 2 LieD[n[c], CD][metricg[-a, -b]] //
          GradNormalToExtrinsicK] // ContractMetric // ToCanonical
```

Out[685]=

Kh_{ab}

```
In[686]:=
      -1 / 2 LieD[n[c], CD][metrich[-a, -b]] // ProjectorToMetric // GradNormalToExtrinsicK //
        ContractMetric // ToCanonical
```

Out[686]=

Kh_{ab}

5) The Gauss−Codazzi equations. Single derivative:

```
In[687]:=
      UndefTensor[v]

         ** UndefTensor: Undefined tensor v

In[688]:=
      DefTensor[v[a], M3, OrthogonalTo → {n[-a]}, ProjectedWith → {metrich[-a, b]}]

         ** DefTensor: Defining tensor v[a].

In[689]:=
      CD[-c][v[-b]]

Out[689]=
```

$\nabla_c v_b$

```
In[690]:=
      % // ProjectWith[metrich]

Out[690]=
```

$h_b{}^a h_c{}^d (\nabla_d v_a)$

```
In[691]:=
      % // ProjectorToMetric

Out[691]=
```

$(\delta_b{}^a + n^a n_b) (\delta_c{}^d + n_c n^d) (\nabla_d v_a)$

```
In[692]:=
      % // Expand

Out[692]=
```

$n^a n_c (\nabla_a v_b) + n^a n_b (\nabla_c v_a) + \nabla_c v_b + n^a n_b n_c n^d (\nabla_d v_a)$

```
In[693]:=
      % // GradNormalToExtrinsicK

Out[693]=
```

$-n_b (-Kh_c{}^a - An^a n_c) v_a - n_b n_c (-Kh_d{}^a - An^a n_d) n^d v_a + n^a n_c (\nabla_a v_b) + \nabla_c v_b$

```
In[694]:=
      % // Simplification

Out[694]=
```

$Kh_{ca} n_b v^a + n^a n_c (\nabla_a v_b) + \nabla_c v_b$

We compare with expression (47) of Choptuik's notes:

```
In[695]:=
      -% + CD[-c][v[-b]] - n[-b] v[-f] CD[-c][n[f]] +
       n[-c] n[e] CD[-e][v[-b]] - n[-c] n[-b] v[-f] Accelerationn[f]

Out[695]=
```

$-Kh_{ca} n_b v^a - An^f n_b n_c v_f - n^a n_c (\nabla_a v_b) - n_b v_f (\nabla_c n^f) + n_c n^e (\nabla_e v_b)$

In[696]:=
```
% // GradNormalToExtrinsicK
```
Out[696]=
$$-Kh_{ca}\, n_b\, v^a - An^f\, n_b\, n_c\, v_f - n_b\, (-Kh_c{}^f - An^f\, n_c)\, v_f - n^a\, n_c\, (\nabla_a v_b) + n_c\, n^e\, (\nabla_e v_b)$$

In[697]:=
```
% // Simplification
```
Out[697]=
```
0
```

Second derivatives. We correct Choptuik's equations (49) and (50), which are missing one term:

In[698]:=
```
ProjectWith[metrich][CD[-d][CD[-c][v[-b]]]] - cd[-d][cd[-c][v[-b]]]
```
Out[698]=
$$-(D_d\, D_c\, v_b) + h_b{}^a\, h_c{}^e\, h_d{}^f\, (\nabla_f \nabla_e v_a)$$

In[699]:=
```
% // ProjectDerivative // ProjectDerivative
```
Out[699]=
$$-\underset{h}{P}\left[\nabla_d\, \underset{h}{P}[\nabla_c v_b]\right] + h_b{}^a\, h_c{}^e\, h_d{}^f\, (\nabla_f \nabla_e v_a)$$

In[700]:=
```
% /. Projectormetrich → ProjectWith[metrich]
```
Out[700]=
$$-h_b{}^f\, h_c{}^e\, h_d{}^g\, (\nabla_e v_a)\, (\nabla_g h_f{}^a) - h_b{}^a\, h_c{}^f\, h_d{}^g\, (\nabla_e v_a)\, (\nabla_g h_f{}^e)$$

In[701]:=
```
% // ProjectorToMetric // Expand
```
Out[701]=
$$-n^a\, (\nabla_c v_a)\, (\nabla_d n_b) - n^a\, (\nabla_a v_b)\, (\nabla_d n_c) - n^a\, n_c\, n^e\, (\nabla_a v_b)\, (\nabla_d n_e) -$$
$$n^a\, n_b\, n^e\, (\nabla_c v_a)\, (\nabla_d n_e) - n^a\, n_d\, n^e\, (\nabla_c v_a)\, (\nabla_e n_b) - n^a\, n_d\, n^e\, (\nabla_a v_b)\, (\nabla_e n_c) -$$
$$n^a\, n_c\, n^e\, (\nabla_d n_b)\, (\nabla_e v_a) - n^a\, n_b\, n^e\, (\nabla_d n_c)\, (\nabla_e v_a) - 2\, n^a\, n_b\, n_c\, n^e\, n^f\, (\nabla_d n_f)\, (\nabla_e v_a) -$$
$$n^a\, n_c\, n_d\, n^e\, n^f\, (\nabla_e v_a)\, (\nabla_f n_b) - n^a\, n_b\, n_d\, n^e\, n^f\, (\nabla_e v_a)\, (\nabla_f n_c) - n^a\, n_c\, n_d\, n^e\, n^f\, (\nabla_a v_b)\, (\nabla_f n_e) -$$
$$n^a\, n_b\, n_d\, n^e\, n^f\, (\nabla_c v_a)\, (\nabla_f n_e) - 2\, n^a\, n_b\, n_c\, n_d\, n^e\, n^f\, n^g\, (\nabla_e v_a)\, (\nabla_g n_f)$$

In[702]:=
```
% // GradNormalToExtrinsicK // Expand
```
Out[702]=
$$Kh_c{}^e\, Kh_d{}^a\, g_{ab}\, v_e - An^e\, Kh_d{}^a\, g_{ac}\, n_b\, v_e + Kh_c{}^e\, Kh_d{}^a\, g_{af}\, n_b\, n^f\, v_e -$$
$$An^e\, Kh_d{}^a\, g_{af}\, n_b\, n_c\, n^f\, v_e + Kh_d{}^a\, g_{ac}\, n^e\, (\nabla_e v_b) + Kh_d{}^a\, g_{af}\, n_c\, n^e\, n^f\, (\nabla_e v_b)$$

In[703]:=
```
% // ContractMetric
```
Out[703]=
$$Kh_c{}^a\, Kh_{db}\, v_a - An^a\, Kh_{dc}\, n_b\, v_a + Kh_{dc}\, n^a\, (\nabla_a v_b)$$

In[704]:=
```
% // Simplification
```

Out[704]=

$$-An^{a}\, Kh_{cd}\, n_{b}\, v_{a} + Kh_{bd}\, Kh_{ca}\, v^{a} + Kh_{cd}\, n^{a}\, (\nabla_{a} v_{b})$$

First Gauss−Codazzi equation:

In[705]:=
```
expr1 = Projectormetrich[CD[-d][CD[-c][v[-b]]] - CD[-c][CD[-d][v[-b]]]]
```

Out[705]=

$$-\underset{h}{P}[\nabla_{c} \nabla_{d} v_{b}] + \underset{h}{P}[\nabla_{d} \nabla_{c} v_{b}]$$

In[706]:=
```
expr1A = expr1 // SortCovDs
```

Out[706]=

$$\underset{h}{P}[R[\nabla]_{dcb}{}^{a}\; v_{a}]$$

In[707]:=
```
expr1 /. Projectormetrich → ProjectWith[metrich] // ProjectorToMetric // Expand
```

Out[707]=

$$n^{a} n_{d} (\nabla_{a} \nabla_{c} v_{b}) - n^{a} n_{c} (\nabla_{a} \nabla_{d} v_{b}) - n^{a} n_{c} n_{d} n^{e} (\nabla_{a} \nabla_{e} v_{b}) - n^{a} n_{d} (\nabla_{c} \nabla_{a} v_{b}) - n^{a} n_{b} (\nabla_{c} \nabla_{d} v_{a}) - \nabla_{c} \nabla_{d} v_{b} - n^{a} n_{b} n_{d} n^{e} (\nabla_{c} \nabla_{e} v_{a}) + n^{a} n_{c} (\nabla_{d} \nabla_{a} v_{b}) + n^{a} n_{b} (\nabla_{d} \nabla_{c} v_{a}) + \nabla_{d} \nabla_{c} v_{b} + n^{a} n_{b} n_{c} n^{e} (\nabla_{d} \nabla_{e} v_{a}) + n^{a} n_{c} n_{d} n^{e} (\nabla_{e} \nabla_{a} v_{b}) + n^{a} n_{b} n_{d} n^{e} (\nabla_{e} \nabla_{c} v_{a}) - n^{a} n_{b} n_{c} n^{e} (\nabla_{e} \nabla_{d} v_{a}) - n^{a} n_{b} n_{c} n_{d} n^{e} n^{f} (\nabla_{e} \nabla_{f} v_{a}) + n^{a} n_{b} n_{c} n_{d} n^{e} n^{f} (\nabla_{f} \nabla_{e} v_{a})$$

In[708]:=
```
expr1B = % // GradNormalToExtrinsicK // Simplification
```

Out[708]=

$$n^{a} n_{d} (\nabla_{a} \nabla_{c} v_{b}) - n^{a} n_{c} (\nabla_{a} \nabla_{d} v_{b}) - n^{a} n_{d} (\nabla_{c} \nabla_{a} v_{b}) - \nabla_{c} \nabla_{d} v_{b} + n^{a} n_{c} (\nabla_{d} \nabla_{a} v_{b}) + \nabla_{d} \nabla_{c} v_{b} + n^{a} n_{b} (-(\nabla_{c} \nabla_{d} v_{a}) + \nabla_{d} \nabla_{c} v_{a} + n_{c} n^{e} (\nabla_{d} \nabla_{e} v_{a}) + n_{d} n^{e} (-(\nabla_{c} \nabla_{e} v_{a}) + \nabla_{e} \nabla_{c} v_{a}) - n_{c} n^{e} (\nabla_{e} \nabla_{d} v_{a}))$$

In[709]:=
```
expr2 = cd[-d][cd[-c][v[-b]]] - cd[-c][cd[-d][v[-b]]]
```

Out[709]=

$$-(D_{c} D_{d} v_{b}) + D_{d} D_{c} v_{b}$$

In[710]:=
```
expr2A = expr2 // SortCovDs
```

Out[710]=

$$R[D]_{dcb}{}^{a}\, v_{a}$$

In[711]:=
```
expr2 // ProjectDerivative // ProjectDerivative
```

Out[711]=

$$-\underset{h}{P}\left[\nabla_{c} \underset{h}{P}[\nabla_{d} v_{b}]\right] + \underset{h}{P}\left[\nabla_{d} \underset{h}{P}[\nabla_{c} v_{b}]\right]$$

In[712]:=
```
% /. Projectormetrich → ProjectWith[metrich] // ProjectorToMetric // Expand
```

Out[712]=

$$
\begin{aligned}
& n^a\, n_d\, (\triangledown_a \triangledown_c v_b) - n^a\, n_c\, (\triangledown_a \triangledown_d v_b) - n^a\, (\triangledown_a v_b)\, (\triangledown_c n_d) - n^a\, n_d\, n^e\, (\triangledown_a v_b)\, (\triangledown_c n_e) - \\
& n^a\, n_d\, (\triangledown_c \triangledown_a v_b) - n^a\, n_b\, (\triangledown_c \triangledown_d v_a) - \triangledown_c \triangledown_d v_b - n^a\, n_b\, n_d\, n^e\, (\triangledown_c \triangledown_e v_a) + n^a\, (\triangledown_c v_a)\, (\triangledown_d n_b) + \\
& n^a\, (\triangledown_a v_b)\, (\triangledown_d n_c) + n^a\, n_c\, n^e\, (\triangledown_a v_b)\, (\triangledown_d n_e) + n^a\, n_b\, n^e\, (\triangledown_c v_a)\, (\triangledown_d n_e) - n^a\, (\triangledown_c n_b)\, (\triangledown_d v_a) - \\
& n^a\, n_b\, n^e\, (\triangledown_c n_e)\, (\triangledown_d v_a) + n^a\, n_c\, (\triangledown_d \triangledown_a v_b) + n^a\, n_b\, (\triangledown_d \triangledown_c v_a) + \triangledown_d \triangledown_c v_b + n^a\, n_b\, n_c\, n^e\, (\triangledown_d \triangledown_e v_a) + \\
& n^a\, n_d\, n^e\, (\triangledown_c v_a)\, (\triangledown_e n_b) - n^a\, n_c\, n^e\, (\triangledown_d v_a)\, (\triangledown_e n_b) + n^a\, n_d\, n^e\, (\triangledown_a v_b)\, (\triangledown_e n_c) - \\
& n^a\, n_c\, n^e\, (\triangledown_a v_b)\, (\triangledown_e n_d) - n^a\, n_c\, n_d\, n^e\, n^f\, (\triangledown_a v_b)\, (\triangledown_e n_f) - n^a\, n_d\, n^e\, (\triangledown_c n_b)\, (\triangledown_e v_a) - \\
& n^a\, n_b\, n^e\, (\triangledown_c n_d)\, (\triangledown_e v_a) - 2\, n^a\, n_b\, n_d\, n^e\, n^f\, (\triangledown_c n_f)\, (\triangledown_e v_a) + n^a\, n_c\, n^e\, (\triangledown_d n_b)\, (\triangledown_e v_a) + \\
& n^a\, n_b\, n^e\, (\triangledown_d n_c)\, (\triangledown_e v_a) + 2\, n^a\, n_b\, n_c\, n^e\, n^f\, (\triangledown_d n_f)\, (\triangledown_e v_a) + n^a\, n_b\, n_d\, n^e\, (\triangledown_e \triangledown_c v_a) - \\
& n^a\, n_b\, n_c\, n^e\, (\triangledown_e \triangledown_d v_a) + n^a\, n_b\, n_d\, n^e\, n^f\, (\triangledown_e v_a)\, (\triangledown_f n_c) - n^a\, n_b\, n_c\, n^e\, n^f\, (\triangledown_e v_a)\, (\triangledown_f n_d) + \\
& n^a\, n_c\, n_d\, n^e\, n^f\, (\triangledown_a v_b)\, (\triangledown_f n_e) + n^a\, n_b\, n_d\, n^e\, n^f\, (\triangledown_c v_a)\, (\triangledown_f n_e) - n^a\, n_b\, n_c\, n^e\, n^f\, (\triangledown_d v_a)\, (\triangledown_f n_e) - \\
& n^a\, n_b\, n_c\, n_d\, n^e\, n^f\, n^g\, (\triangledown_e v_a)\, (\triangledown_f n_g) + n^a\, n_b\, n_c\, n_d\, n^e\, n^f\, n^g\, (\triangledown_e v_a)\, (\triangledown_g n_f)
\end{aligned}
$$

In[713]:=
```
% // GradNormalToExtrinsicK // ContractMetric
```

Out[713]=

$$
\begin{aligned}
& Kh_{cb}\, Kh_d{}^a\, v_a - Kh_c{}^a\, Kh_{db}\, v_a - An^a\, Kh_{cd}\, n_b\, v_a + An^a\, Kh_{dc}\, n_b\, v_a + Kh_{cd}\, n^a\, (\triangledown_a v_b) - \\
& Kh_{dc}\, n^a\, (\triangledown_a v_b) + n^a\, n_d\, (\triangledown_a \triangledown_c v_b) - n^a\, n_c\, (\triangledown_a \triangledown_d v_b) - n^a\, n_d\, (\triangledown_c \triangledown_a v_b) - \\
& n^a\, n_b\, (\triangledown_c \triangledown_d v_a) - \triangledown_c \triangledown_d v_b - n^a\, n_b\, n_d\, n^e\, (\triangledown_c \triangledown_e v_a) + n^a\, n_c\, (\triangledown_d \triangledown_a v_b) + n^a\, n_b\, (\triangledown_d \triangledown_c v_a) + \\
& \triangledown_d \triangledown_c v_b + n^a\, n_b\, n_c\, n^e\, (\triangledown_d \triangledown_e v_a) + n^a\, n_b\, n_d\, n^e\, (\triangledown_e \triangledown_c v_a) - n^a\, n_b\, n_c\, n^e\, (\triangledown_e \triangledown_d v_a)
\end{aligned}
$$

In[714]:=
```
expr2B = % // Simplification
```

Out[714]=

$$
\begin{aligned}
& -Kh_{bd}\, Kh_{ca}\, v^a + Kh_{bc}\, Kh_{da}\, v^a + n^a\, n_d\, (\triangledown_a \triangledown_c v_b) - n^a\, n_c\, (\triangledown_a \triangledown_d v_b) - n^a\, n_d\, (\triangledown_c \triangledown_a v_b) - \\
& n^a\, n_b\, (\triangledown_c \triangledown_d v_a) - \triangledown_c \triangledown_d v_b - n^a\, n_b\, n_d\, n^e\, (\triangledown_c \triangledown_e v_a) + n^a\, n_c\, (\triangledown_d \triangledown_a v_b) + n^a\, n_b\, (\triangledown_d \triangledown_c v_a) + \\
& \triangledown_d \triangledown_c v_b + n^a\, n_b\, n_c\, n^e\, (\triangledown_d \triangledown_e v_a) + n^a\, n_b\, n_d\, n^e\, (\triangledown_e \triangledown_c v_a) - n^a\, n_b\, n_c\, n^e\, (\triangledown_e \triangledown_d v_a)
\end{aligned}
$$

In[715]:=
```
(expr1A - expr1B) - (expr2A - expr2B) // Simplification
```

Out[715]=

$$
-\underset{h}{P}[R[\triangledown]_{bacd}\ v^a] + (-Kh_{bd}\, Kh_{ca} + Kh_{bc}\, Kh_{da} + R[D]_{bacd})\, v^a
$$

In[716]:=
```
IndexSet[GCRiemann1[-a_, -b_, -c_, -d_],
 Simplification[(% /. v[_] → 1 // ScreenDollarIndices)]]
```

Out[716]=

$$
Kh_{ad}\, Kh_{bc} - Kh_{ac}\, Kh_{bd} + \underset{h}{P}[R[\triangledown]_{abcd}] - R[D]_{abcd}
$$

If we decompose the Riemann tensor along the vector n

In[717]:=
```
inducedRiemann[-a_, -b_, -c_, -d_] =
 InducedDecomposition[RiemannCD[-a, -b, -c, -d], {metrich, n}] // ToCanonical
```

Out[717]=

$$
\begin{aligned}
& \underset{h}{P}[R[\triangledown]_{abcd}] - n_d\, \underset{h}{P}[n^e\ R[\triangledown]_{abce}] + n_c\, \underset{h}{P}[n^e\ R[\triangledown]_{abde}] - \\
& n_b\, \underset{h}{P}[n^e\ R[\triangledown]_{aecd}] + n_b\, n_d\, \underset{h}{P}[n^e\ n^f\ R[\triangledown]_{aecf}] - n_b\, n_c\, \underset{h}{P}[n^e\ n^f\ R[\triangledown]_{aedf}] + \\
& n_a\, \underset{h}{P}[n^e\ R[\triangledown]_{becd}] - n_a\, n_d\, \underset{h}{P}[n^e\ n^f\ R[\triangledown]_{becf}] + n_a\, n_c\, \underset{h}{P}[n^e\ n^f\ R[\triangledown]_{bedf}]
\end{aligned}
$$

then we can express it as

```
In[718]:=
      inducedRiemann[-a_, -b_, -c_, -d_] =
       inducedRiemann[-a, -b, -c, -d] - GCRiemann1[-a, -b, -c, -d] // ToCanonical
```

Out[718]=

$$-Kh_{ad}\,Kh_{bc}+Kh_{ac}\,Kh_{bd}-n_d\,\underset{h}{P}[n^e\;R[\nabla]_{abce}]+n_c\,\underset{h}{P}[n^e\;R[\nabla]_{abde}]-n_b\,\underset{h}{P}[n^e\;R[\nabla]_{aecd}]+n_b\,n_d\,\underset{h}{P}[n^e\;n^f\;R[\nabla]_{aecf}]-n_b\,n_c\,\underset{h}{P}[n^e\;n^f\;R[\nabla]_{aedf}]+n_a\,\underset{h}{P}[n^e\;R[\nabla]_{becd}]-n_a\,n_d\,\underset{h}{P}[n^e\;n^f\;R[\nabla]_{becf}]+n_a\,n_c\,\underset{h}{P}[n^e\;n^f\;R[\nabla]_{bedf}]+R[D]_{abcd}$$

Second Gauss−Codazzi equation:

```
In[719]:=
      expr1 = Projectormetrich[CD[-d][CD[-c][n[-b]]] - CD[-c][CD[-d][n[-b]]]]
```

Out[719]=

$$-\underset{h}{P}[\nabla_c\,\nabla_d\,n_b]+\underset{h}{P}[\nabla_d\,\nabla_c\,n_b]$$

```
In[720]:=
      expr1A = expr1 // SortCovDs
```

Out[720]=

$$\underset{h}{P}[n_a\;R[\nabla]_{dcb}{}^{a}]$$

```
In[721]:=
      expr1 /. Projectormetrich → ProjectWith[metrich] // ProjectorToMetric // Expand
```

Out[721]=

$$n^a\,n_d\,(\nabla_a\,\nabla_c\,n_b)-n^a\,n_c\,(\nabla_a\,\nabla_d\,n_b)-n^a\,n_c\,n_d\,n^e\,(\nabla_a\,\nabla_e\,n_b)-n^a\,n_d\,(\nabla_c\,\nabla_a\,n_b)-n^a\,n_b\,(\nabla_c\,\nabla_d\,n_a)-\nabla_c\,\nabla_d\,n_b-n^a\,n_b\,n_d\,n^e\,(\nabla_c\,\nabla_e\,n_a)+n^a\,n_c\,(\nabla_d\,\nabla_a\,n_b)+n^a\,n_b\,(\nabla_d\,\nabla_c\,n_a)+\nabla_d\,\nabla_c\,n_b+n^a\,n_b\,n_c\,n^e\,(\nabla_d\,\nabla_e\,n_a)+n^a\,n_c\,n_d\,n^e\,(\nabla_e\,\nabla_a\,n_b)+n^a\,n_b\,n_d\,n^e\,(\nabla_e\,\nabla_c\,n_a)-n^a\,n_b\,n_c\,n^e\,(\nabla_e\,\nabla_d\,n_a)-n^a\,n_b\,n_c\,n_d\,n^e\,n^f\,(\nabla_e\,\nabla_f\,n_a)+n^a\,n_b\,n_c\,n_d\,n^e\,n^f\,(\nabla_f\,\nabla_e\,n_a)$$

```
In[722]:=
      expr1B = % // GradNormalToExtrinsicK // ContractMetric // GradNormalToExtrinsicK //
        Simplification
```

Out[722]=

$$Kh_b{}^{a}\,(-Kh_{da}\,n_c+Kh_{ca}\,n_d)-n^a\,n_d\,(\nabla_a\,Kh_{bc})+n^a\,n_c\,(\nabla_a\,Kh_{bd})+\nabla_c\,Kh_{bd}-\nabla_d\,Kh_{bc}$$

```
In[723]:=
      expr2A = cd[-c][ExtrinsicKmetrich[-b, -d]] - cd[-d][ExtrinsicKmetrich[-b, -c]]
```

Out[723]=

$$D_c\,Kh_{bd}-D_d\,Kh_{bc}$$

```
In[724]:=
      expr2A // ProjectDerivative
```

Out[724]=

$$\underset{h}{P}[\nabla_c\,Kh_{bd}]-\underset{h}{P}[\nabla_d\,Kh_{bc}]$$

In[725]:=
```
% /. Projectormetrich → ProjectWith[metrich] // ProjectorToMetric // Expand
```
Out[725]=

$$-n^a n_d (\nabla_a Kh_{bc}) + n^a n_c (\nabla_a Kh_{bd}) + n^a n_c n_d n^e (\nabla_a Kh_{be}) + n^a n_b (\nabla_c Kh_{ad}) + n^a n_b n_d n^e (\nabla_c Kh_{ae}) + n^a n_d (\nabla_c Kh_{ba}) + \nabla_c Kh_{bd} - n^a n_b (\nabla_d Kh_{ac}) - n^a n_b n_c n^e (\nabla_d Kh_{ae}) - n^a n_c (\nabla_d Kh_{ba}) - \nabla_d Kh_{bc} - n^a n_b n_d n^e (\nabla_e Kh_{ac}) + n^a n_b n_c n^e (\nabla_e Kh_{ad}) + n^a n_b n_c n_d n^e n^f (\nabla_e Kh_{af}) - n^a n_c n_d n^e (\nabla_e Kh_{ba}) - n^a n_b n_c n_d n^e n^f (\nabla_f Kh_{ae})$$

In[726]:=
```
expr2B = % // GradNormalToExtrinsicK // ContractMetric // Simplification
```
Out[726]=

$$Kh_b{}^a (-Kh_{da} n_c + Kh_{ca} n_d) - n^a n_d (\nabla_a Kh_{bc}) + n^a n_c (\nabla_a Kh_{bd}) + \nabla_c Kh_{bd} - \nabla_d Kh_{bc}$$

In[727]:=
```
IndexSet[GCRiemann2[-b_, -c_, -d_],
 (expr1A - expr1B) - (expr2A - expr2B) // Simplification]
```
Out[727]=

$$-\underset{h}{P}[n^a \; R[\nabla]_{bacd}] - D_c Kh_{bd} + D_d Kh_{bc}$$

We can further decompose the Riemann tensor now:

In[728]:=
```
inducedRiemann[-a_, -b_, -c_, -d_] =
 inducedRiemann[-a, -b, -c, -d] - n[-d] GCRiemann2[-c, -a, -b] +
   n[-c] GCRiemann2[-d, -a, -b] - n[-b] GCRiemann2[-a, -c, -d] +
   n[-a] GCRiemann2[-b, -c, -d] // ToCanonical
```
Out[728]=

$$-Kh_{ad} Kh_{bc} + Kh_{ac} Kh_{bd} + n_b n_d \underset{h}{P}[n^e \; n^f \; R[\nabla]_{aecf}] - n_b n_c \underset{h}{P}[n^e \; n^f \; R[\nabla]_{aedf}] - n_a n_d \underset{h}{P}[n^e \; n^f \; R[\nabla]_{becf}] + n_a n_c \underset{h}{P}[n^e \; n^f \; R[\nabla]_{bedf}] + R[D]_{abcd} + n_d (D_a Kh_{bc}) - n_c (D_a Kh_{bd}) - n_d (D_b Kh_{ac}) + n_c (D_b Kh_{ad}) + n_b (D_c Kh_{ad}) - n_a (D_c Kh_{bd}) - n_b (D_d Kh_{ac}) + n_a (D_d Kh_{bc})$$

"Third Gauss−Codazzi equation":

In[729]:=
```
expr1 = Projectormetrich[n[d] (CD[-d][CD[-c][n[-b]]] - CD[-c][CD[-d][n[-b]]])]
```
Out[729]=

$$\underset{h}{P}[n^d \; (-(\nabla_c \nabla_d n_b) + \nabla_d \nabla_c n_b)]$$

In[730]:=
```
expr1A = expr1 // SortCovDs
```
Out[730]=

$$\underset{h}{P}[n_a \; n^d \; R[\nabla]_{dcb}{}^a]$$

In[731]:=
```
expr1 /. Projectormetrich → ProjectWith[metrich] // ProjectorToMetric // Expand
```
Out[731]=

$$-n^a n_c n^d (\nabla_a \nabla_d n_b) - n^a n_b n^d (\nabla_c \nabla_d n_a) - n^d (\nabla_c \nabla_d n_b) + n^a n_c n^d (\nabla_d \nabla_a n_b) + n^a n_b n^d (\nabla_d \nabla_c n_a) + n^d (\nabla_d \nabla_c n_b) + n^a n_b n_c n^d n^e (\nabla_d \nabla_e n_a) - n^a n_b n_c n^d n^e (\nabla_e \nabla_d n_a)$$

```
In[732]:=
      expr1B = % // GradNormalToExtrinsicK // ContractMetric // GradNormalToExtrinsicK //
        Simplification
```

Out[732]=

$-An_b\, An_c + Kh_b{}^d\, Kh_{cd} + An^d\, Kh_{bd}\, n_c - \nabla_c An_b - n_c\, n^d\, (\nabla_d An_b) - n^d\, (\nabla_d Kh_{bc})$

```
In[733]:=
      expr2A = -cd[-c][Accelerationn[-b]] -
        Projectormetrich[n[d] CD[-d][ExtrinsicKmetrich[-b, -c]]]
```

Out[733]=

$-\underset{h}{P}[n^d\, (\nabla_d Kh_{bc})] - D_c An_b$

```
In[734]:=
      expr2A // ProjectDerivative
```

Out[734]=

$-\underset{h}{P}[\nabla_c An_b] - \underset{h}{P}[n^d\, (\nabla_d Kh_{bc})]$

```
In[735]:=
      % /. Projectormetrich → ProjectWith[metrich] // ProjectorToMetric // Expand
```

Out[735]=

$-n^a\, n_c\, (\nabla_a An_b) - n^a\, n_b\, (\nabla_c An_a) - \nabla_c An_b - n^a\, n_b\, n_c\, n^d\, (\nabla_d An_a) - n^a\, n_b\, n^d\, (\nabla_d Kh_{ac}) - n^a\, n_b\, n_c\, n^d\, n^e\, (\nabla_d Kh_{ae}) - n^a\, n_c\, n^d\, (\nabla_d Kh_{ba}) - n^d\, (\nabla_d Kh_{bc})$

```
In[736]:=
      expr2B = % // GradNormalToExtrinsicK // ContractMetric // Simplification
```

Out[736]=

$An^d\, Kh_{bd}\, n_c - \nabla_c An_b - n_c\, n^d\, (\nabla_d An_b) - n^d\, (\nabla_d Kh_{bc})$

Now we have (note that this pair of equations is equivalent to (105) of Choptuik):

```
In[737]:=
      IndexSet[GCRiemann3[-b_, -c_], (expr1A - expr1B) - (expr2A - expr2B) // Simplification]
```

Out[737]=

$An_b\, An_c - Kh_b{}^d\, Kh_{cd} - \underset{h}{P}[n^a\, n^d\, R[\nabla]_{bdca}] + \underset{h}{P}[n^d\, (\nabla_d Kh_{bc})] + D_c An_b$

```
In[738]:=
      ProjectWith[metrich][LieD[n[a], CD][ExtrinsicKmetrich[-b, -c]] -
           n[a] CD[-a][ExtrinsicKmetrich[-b, -c]] //
          GradNormalToExtrinsicK] // Simplification // ContractMetric
```

Out[738]=

$-2\, Kh_b{}^a\, Kh_{ca}$

Finally we can combine the three Gauss−Codazzi equations into a single expression:

```
In[739]:=
      inducedRiemann[-a_, -b_, -c_, -d_] =
       inducedRiemann[-a, -b, -c, -d] + n[-b] n[-d] GCRiemann3[-a, -c] -
         n[-b] n[-c] GCRiemann3[-a, -d] - n[-a] n[-d] GCRiemann3[-b, -c] +
         n[-a] n[-c] GCRiemann3[-b, -d] // ToCanonical
```

Out[739]=

$$-Kh_{ad}\,Kh_{bc} + Kh_{ac}\,Kh_{bd} + An_b\,An_d\,n_a\,n_c - Kh_b{}^e\,Kh_{de}\,n_a\,n_c - An_a\,An_d\,n_b\,n_c + Kh_a{}^e\,Kh_{de}\,n_b\,n_c - An_b\,An_c\,n_a\,n_d + Kh_b{}^e\,Kh_{ce}\,n_a\,n_d + An_a\,An_c\,n_b\,n_d - Kh_a{}^e\,Kh_{ce}\,n_b\,n_d + n_b\,n_d\,\underset{h}{P}[n^e\ (\nabla_e Kh_{ac})] - n_b\,n_c\,\underset{h}{P}[n^e\ (\nabla_e Kh_{ad})] - n_a\,n_d\,\underset{h}{P}[n^e\ (\nabla_e Kh_{bc})] + n_a\,n_c\,\underset{h}{P}[n^e\ (\nabla_e Kh_{bd})] + R[D]_{abcd} + n_d\,(D_a Kh_{bc}) - n_c\,(D_a Kh_{bd}) - n_d\,(D_b Kh_{ac}) + n_c\,(D_b Kh_{ad}) + n_b\,n_d\,(D_c An_a) - n_a\,n_d\,(D_c An_b) + n_b\,(D_c Kh_{ad}) - n_a\,(D_c Kh_{bd}) - n_b\,n_c\,(D_d An_a) + n_a\,n_c\,(D_d An_b) - n_b\,(D_d Kh_{ac}) + n_a\,(D_d Kh_{bc})$$

```
In[740]:=
      GausCodazziRiemannRule =
       IndexRule[RiemannCD[-a, -b, -c, -d], inducedRiemann[-a, -b, -c, -d]]
```

Out[740]=

$$\text{HoldPattern}[R[\nabla]_{abcd}] \mapsto \text{Module}\Big[\{e\},\ -Kh_{ad}\,Kh_{bc} + Kh_{ac}\,Kh_{bd} + An_b\,An_d\,n_a\,n_c - Kh_b{}^e\,Kh_{de}\,n_a\,n_c - An_a\,An_d\,n_b\,n_c + Kh_a{}^e\,Kh_{de}\,n_b\,n_c - An_b\,An_c\,n_a\,n_d + Kh_b{}^e\,Kh_{ce}\,n_a\,n_d + An_a\,An_c\,n_b\,n_d - Kh_a{}^e\,Kh_{ce}\,n_b\,n_d + n_b\,n_d\,\underset{h}{P}[n^e\ (\nabla_e Kh_{ac})] - n_b\,n_c\,\underset{h}{P}[n^e\ (\nabla_e Kh_{ad})] - n_a\,n_d\,\underset{h}{P}[n^e\ (\nabla_e Kh_{bc})] + n_a\,n_c\,\underset{h}{P}[n^e\ (\nabla_e Kh_{bd})] + R[D]_{abcd} + n_d\,(D_a Kh_{bc}) - n_c\,(D_a Kh_{bd}) - n_d\,(D_b Kh_{ac}) + n_c\,(D_b Kh_{ad}) + n_b\,n_d\,(D_c An_a) - n_a\,n_d\,(D_c An_b) + n_b\,(D_c Kh_{ad}) - n_a\,(D_c Kh_{bd}) - n_b\,n_c\,(D_d An_a) + n_a\,n_c\,(D_d An_b) - n_b\,(D_d Kh_{ac}) + n_a\,(D_d Kh_{bc})\Big]$$

Most of the nontrivial computations in this area can now be easily performed using the Gauss−Codazzi formula. For example, let us compute the projections of the Ricci tensor and the Ricci scalar:

Decompose:

```
In[741]:=
      inducedRicci[-a_, -b_] =
       InducedDecomposition[RicciCD[-a, -b], {metrich, n}] // NoScalar
```

Out[741]=

$$\underset{h}{P}[R[\nabla]_{ab}] - n_b\,\underset{h}{P}[n^c\ R[\nabla]_{ac}] - n_a\,\underset{h}{P}[n^c\ R[\nabla]_{cb}] + n_a\,n_b\,n^c\,n^d\,R[\nabla]_{cd}$$

First formula:

```
In[742]:=
      zero =
       RicciCD[-a, -b] - inducedRiemann[-a, -c, -b, -d] metricg[c, d] // ContractMetric //
        NoScalar
```

Out[742]=

$$An_a\,An_b - Kh_a{}^c\,Kh_{bc} + Kh_a{}^c\,Kh_{cb} - Kh_{ab}\,Kh_c{}^c - An_c\,An^c\,n_a\,n_b + Kh_c{}^d\,Kh^c{}_d\,n_a\,n_b + \underset{h}{P}[n^c\ (\nabla_c Kh_{ab})] - R[D]_{ab} + R[\nabla]_{ab} + n_b\,(D_a Kh_c{}^c) + D_b An_a + n_a\,(D_b Kh_c{}^c) - n_b\,(D_c Kh_a{}^c) - n_a\,n_b\,(D_d An^d) - n_a\,(D_d Kh_b{}^d) - h^{cd}\,n_a\,n_b\,n^e\,(\nabla_e Kh_{cd})$$

```
In[743]:=
      zeroA = ProjectWith[metrich][%] // ContractMetric // ToCanonical
```

Out[743]=

$An_a\, An_b - Kh_{ab}\, Kh^c{}_c + \underset{h}{P}[n^c\ (\nabla_c Kh_{ab})] - R[D]_{ab} + h_a{}^c\, h_b{}^d\, R[\nabla]_{cd} + D_b\, An_a$

```
In[744]:=
      zeroB = Projectormetrich[RicciCD[-a, -b]] - ProjectWith[metrich][RicciCD[-a, -b]]
```

Out[744]=

$\underset{h}{P}[R[\nabla]_{ab}] - h_a{}^c\, h_b{}^d\, R[\nabla]_{cd}$

```
In[745]:=
      IndexSet[GCRicci1[-a_, -b_], zeroA + zeroB // ToCanonical]
```

Out[745]=

$An_a\, An_b - Kh_{ab}\, Kh^c{}_c + \underset{h}{P}[R[\nabla]_{ab}] + \underset{h}{P}[n^c\ (\nabla_c Kh_{ab})] - R[D]_{ab} + D_b\, An_a$

```
In[746]:=
      inducedRicci[-a_, -b_] = inducedRicci[-a, -b] - GCRicci1[-a, -b] // ToCanonical
```

Out[746]=

$-An_a\, An_b + Kh_{ab}\, Kh^c{}_c - n_b\, \underset{h}{P}[n^c\ R[\nabla]_{ac}] - n_a\, \underset{h}{P}[n^c\ R[\nabla]_{bc}] - \underset{h}{P}[n^c\ (\nabla_c Kh_{ab})] + R[D]_{ab} + n_a\, n_b\, n^c\, n^d\, R[\nabla]_{cd} - D_b\, An_a$

Second formula:

```
In[747]:=
      zero = (RicciCD[-a, -c] - inducedRiemann[-a, -b, -c, -d] metricg[b, d]) n[c] //
         ContractMetric // NoScalar
```

Out[747]=

$An_b\, An^b\, n_a - Kh_b{}^c\, Kh^b{}_c\, n_a + n^c\, R[\nabla]_{ac} - D_a\, Kh_b{}^b + D_b\, Kh_a{}^b + n_a\, (D_d\, An^d) + h^{bc}\, n_a\, n^d\, (\nabla_d Kh_{bc})$

```
In[748]:=
      zeroA = ProjectWith[metrich][%] // MetricToProjector // ToCanonical
```

Out[748]=

$h_a{}^c\, n^b\, R[\nabla]_{bc} - D_a\, Kh^b{}_b + D_b\, Kh_a{}^b$

```
In[749]:=
      zeroB =
       Projectormetrich[RicciCD[-a, -b] n[b]] - ProjectWith[metrich][RicciCD[-a, -b] n[b]]
```

Out[749]=

$\underset{h}{P}[n^b\ R[\nabla]_{ab}] - h_a{}^c\, n^b\, R[\nabla]_{cb}$

```
In[750]:=
      IndexSet[GCRicci2[-a_], zeroA + zeroB // ToCanonical]
```

Out[750]=

$\underset{h}{P}[n^b\ R[\nabla]_{ab}] - D_a\, Kh^b{}_b + D_b\, Kh_a{}^b$

In[751]:=
```
inducedRicci[-a_, -b_] =
 inducedRicci[-a, -b] + GCRicci2[-a] n[-b] + GCRicci2[-b] n[-a] // ToCanonical
```

Out[751]=
$$-An_a\, An_b + Kh_{ab}\, Kh^c{}_c - \underset{h}{P}[n^c\ (\nabla_c\, Kh_{ab})] + R[D]_{ab} + n_a\, n_b\, n^c\, n^d\, R[\nabla]_{cd} - n_b\, (D_a\, Kh^c{}_c) - D_b\, An_a - n_a\, (D_b\, Kh^c{}_c) + n_b\, (D_c\, Kh_a{}^c) + n_a\, (D_c\, Kh_b{}^c)$$

Third formula:

In[752]:=
```
zero =
 (RicciCD[-a, -c] - inducedRiemann[-a, -b, -c, -d] metricg[b, d]) n[a] n[c] // Expand //
  ReplaceDummies
```

Out[752]=
$$-An_a\, An_b\, g^{ab} + Kh_a{}^c\, Kh_{bc}\, g^{ab} - h^{ab}\, \underset{h}{P}[n^c\ (\nabla_c\, Kh_{ab})] + n^a\, n^b\, R[\nabla]_{ab} - h^{ab}\, (D_b\, An_a)$$

In[753]:=
```
inducedRicci[-a_, -b_] = inducedRicci[-a, -b] - n[-a] n[-b] zero // Expand // SameDummies
```

Out[753]=
$$-An_a\, An_b + Kh_{ab}\, Kh^c{}_c + An_c\, An_d\, g^{cd}\, n_a\, n_b - Kh_c{}^e\, Kh_{de}\, g^{cd}\, n_a\, n_b - \underset{h}{P}[n^c\ (\nabla_c\, Kh_{ab})] + h^{cd}\, n_a\, n_b\, \underset{h}{P}[n^e\ (\nabla_e\, Kh_{cd})] + R[D]_{ab} - n_b\, (D_a\, Kh^c{}_c) - D_b\, An_a - n_a\, (D_b\, Kh^c{}_c) + n_b\, (D_c\, Kh_a{}^c) + n_a\, (D_c\, Kh_b{}^c) + h^{cd}\, n_a\, n_b\, (D_d\, An_c)$$

Finally, the Ricci scalar:

In[754]:=
```
inducedRiemann[-a, -b, -c, -d] metricg[b, d] metricg[a, c] // ContractMetric // NoScalar
```

Out[754]=
$$-An_a\, An^a - An_b\, An^b + Kh_a{}^b\, Kh^a{}_b - Kh_a{}^b\, Kh_b{}^a + Kh_a{}^a\, Kh_b{}^b + Kh_b{}^a\, Kh^b{}_a + R[D] - D_c\, An^c - D_d\, An^d - 2\, h^{ab}\, n^c\, (\nabla_c\, Kh_{ab})$$

In[755]:=
```
% /. Projectormetrich → ProjectWith[metrich] // ContractMetric // ToCanonical
```

Out[755]=
$$-2\, An_a\, An^a + Kh_{ab}\, Kh^{ab} + Kh^a{}_a\, Kh^b{}_b + R[D] - 2\, (D_a\, An^a) - 2\, h^{bc}\, n^a\, (\nabla_a\, Kh_{bc})$$

In[756]:=
```
% // ProjectorToMetric // ContractMetric
```

Out[756]=
$$-2\, An_a\, An^a + Kh_{ab}\, Kh^{ab} + Kh^a{}_a\, Kh^b{}_b + R[D] - 2\, (D_a\, An^a) - 2\, n^a\, (\nabla_a\, Kh_b{}^b) - 2\, n^a\, n^b\, n^c\, (\nabla_a\, Kh_{bc})$$

In[757]:=
```
% // GradNormalToExtrinsicK
```

Out[757]=
$$-2\, An_a\, An^a + Kh_{ab}\, Kh^{ab} + Kh^a{}_a\, Kh^b{}_b + R[D] - 2\, (D_a\, An^a) - 2\, n^a\, (\nabla_a\, Kh_b{}^b)$$

With a lapse:

```
In[758]:=
      inducedRicciScalar[] = % /. LapseRule // Expand
```

Out[758]=

$$Kh_{ab}\, Kh^{ab} + Kh^{a}{}_{a}\, Kh^{b}{}_{b} + R[D] - \frac{2\,(D_a D^a \alpha)}{\alpha} - 2\, n^a\, (\nabla_a Kh_b{}^b)$$

```
In[759]:=
      UndefMetric[metrich]

          ** UndefTensor: Undefined antisymmetric tensor epsilonmetrich

          ** UndefTensor: Undefined symmetric Christoffel tensor Christoffelcd

          ** UndefTensor: Undefined symmetric Einstein tensor Einsteincd

          ** UndefTensor: Undefined symmetric Ricci tensor Riccicd

          ** UndefTensor: Undefined Ricci scalar RicciScalarcd

          ** UndefTensor: Undefined Riemann tensor Riemanncd

          ** UndefTensor: Undefined symmetric TFRicci tensor TFRiccicd

          ** UndefTensor: Undefined vanishing torsion tensor Torsioncd

          ** UndefTensor: Undefined vanishing Weyl tensor Weylcd

          ** UndefCovD: Undefined covariant derivative cd

          ** UndefTensor: Undefined extrinsic curvature tensor ExtrinsicKmetrich

          ** UndefTensor: Undefined acceleration vector Accelerationn

          ** UndefInertHead: Undefined projector inert-head Projectormetrich

          ** UndefTensor: Undefined symmetric metric tensor metrich

In[760]:=
      UndefTensor /@ {V, W};

          ** UndefTensor: Undefined tensor V

          ** UndefTensor: Undefined tensor W
```

7.9. Products of epsilon tensors

In many applications we need to manipulate products of two epsilon tensors...

7.10. Variational derivatives

There is not yet a concept of integration in `xTensor‘`. Instead of working with a variational derivative of a functional, we shall assume that such a functional is the integral of some integrand (or "Lagrangian") and that the integral is computed with respect to a volume form which vanishes under the action of some derivative (this is the derivative which will be "integrated by parts").

Let us fake the presence of a scalar density:

```
In[761]:=
      DefTensor[s[], M3]

          ** DefTensor: Defining tensor s[].
```

We need only these two properties:

```
In[762]:=
      s /: VarD[metricg[a_, b_], PD][s[], rest_] := -rest / 2 metricg[-a, -b] s[]

In[763]:=
      s /: PD[a_][s[]] := 1 / 2 Module[{c, d}, s[] metricg[c, d] PD[a][metricg[-c, -d]]]
```

For instance, let us compute the RicciScalar field of our metric:

```
In[764]:=
      rs = RicciScalarCD[] // RiemannToChristoffel // ChristoffelToGradMetric // Expand

Out[764]=
```

$$
\begin{aligned}
&-\frac{1}{2} g^{ab} g^{cd} \partial_a\partial_b g_{cd} - \frac{1}{2} g^{ab} g^{cd} \partial_a\partial_c g_{bd} + \frac{1}{2} g^{ab} g^{cd} \partial_a\partial_d g_{cb} + \\
&\frac{1}{2} g^{ab} g^{ce} g^{df} \partial_a g_{ef} \partial_b g_{cd} - \frac{1}{4} g^{ab} g^{ce} g^{df} \partial_a g_{de} \partial_b g_{cf} + \frac{1}{2} g^{ab} g^{ce} g^{df} \partial_a g_{ef} \partial_c g_{bd} - \\
&\frac{1}{4} g^{ab} g^{ce} g^{df} \partial_a g_{de} \partial_c g_{bf} + \frac{1}{4} g^{ab} g^{ce} g^{df} \partial_a g_{bf} \partial_c g_{de} + \frac{1}{4} g^{ab} g^{ce} g^{df} \partial_b g_{af} \partial_c g_{de} - \\
&\frac{1}{2} g^{ab} g^{ce} g^{df} \partial_a g_{bd} \partial_c g_{ef} - \frac{1}{2} g^{ab} g^{ce} g^{df} \partial_b g_{ad} \partial_c g_{ef} + \frac{1}{2} g^{ab} g^{cd} \partial_c\partial_a g_{bd} + \\
&\frac{1}{2} g^{ab} g^{cd} \partial_c\partial_b g_{ad} - \frac{1}{2} g^{ab} g^{cd} \partial_c\partial_d g_{ab} + \frac{1}{2} g^{ab} g^{ce} g^{df} \partial_c g_{ef} \partial_d g_{ab} - \\
&\frac{1}{4} g^{ab} g^{ce} g^{df} \partial_b g_{cf} \partial_d g_{ae} - \frac{1}{4} g^{ab} g^{ce} g^{df} \partial_c g_{bf} \partial_d g_{ae} - \frac{1}{2} g^{ab} g^{ce} g^{df} \partial_a g_{ef} \partial_d g_{cb} + \\
&\frac{1}{4} g^{ab} g^{ce} g^{df} \partial_a g_{bf} \partial_d g_{ce} + \frac{1}{4} g^{ab} g^{ce} g^{df} \partial_b g_{af} \partial_d g_{ce} + \frac{1}{4} g^{ab} g^{ce} g^{df} \partial_b g_{cf} \partial_e g_{ad} + \\
&\frac{1}{4} g^{ab} g^{ce} g^{df} \partial_c g_{bf} \partial_e g_{ad} - \frac{1}{4} g^{ab} g^{ce} g^{df} \partial_a g_{bf} \partial_e g_{cd} - \frac{1}{4} g^{ab} g^{ce} g^{df} \partial_b g_{af} \partial_e g_{cd} - \\
&\frac{1}{4} g^{ab} g^{ce} g^{df} \partial_c g_{de} \partial_f g_{ab} - \frac{1}{4} g^{ab} g^{ce} g^{df} \partial_d g_{ce} \partial_f g_{ab} + \frac{1}{4} g^{ab} g^{ce} g^{df} \partial_e g_{cd} \partial_f g_{ab} + \\
&\frac{1}{4} g^{ab} g^{ce} g^{df} \partial_a g_{de} \partial_f g_{cb} + \frac{1}{4} g^{ab} g^{ce} g^{df} \partial_d g_{ae} \partial_f g_{cb} - \frac{1}{4} g^{ab} g^{ce} g^{df} \partial_e g_{ad} \partial_f g_{cb}
\end{aligned}
$$

Working intensively with partial derivatives, it is interesting to set

```
In[765]:=
      SetOptions[ToCanonical, UseMetricOnVBundle → None]

Out[765]=
      {Verbose → False, Notation → Images,
       UseMetricOnVBundle → None, ExpandChristoffel → False, GivePerm → False}
```

Now we perform a direct variational derivative of the Einstein−Hilbert Lagrangian with respect to the inverse metric. It is a long computation, producing 501 terms:

```
In[766]:=
      Length[result = Expand@VarD[metricg[a, b], PD][s[] rs]]

Out[766]=
      501
```

The expected result is the Einstein tensor:

```
In[767]:=
      EinsteinCD[-a, -b] // EinsteinToRicci // RiemannToChristoffel //
        ChristoffelToGradMetric // ToCanonical

Out[767]=
```

$$\begin{aligned}
&\frac{1}{4} g^{cd} g^{ef} \partial_a g_{ce} \partial_b g_{df} - \frac{1}{2} g^{cd} \partial_b \partial_a g_{cd} + \frac{1}{4} g^{cd} g^{ef} \partial_a g_{bc} \partial_d g_{ef} + \\
&\frac{1}{4} g^{cd} g^{ef} \partial_b g_{ac} \partial_d g_{ef} - \frac{1}{4} g^{cd} g^{ef} \partial_c g_{ab} \partial_d g_{ef} + \frac{1}{2} g^{cd} \partial_d \partial_a g_{bc} + \frac{1}{2} g^{cd} \partial_d \partial_b g_{ac} - \\
&\frac{1}{2} g^{cd} \partial_d \partial_c g_{ab} - \frac{1}{2} g^{cd} g^{ef} \partial_d g_{bf} \partial_e g_{ac} + \frac{1}{2} g^{cd} g^{ef} \partial_e g_{ac} \partial_f g_{bd} - \frac{1}{2} g^{cd} g^{ef} \partial_a g_{bc} \partial_f g_{de} - \\
&\frac{1}{2} g^{cd} g^{ef} \partial_b g_{ac} \partial_f g_{de} + \frac{1}{2} g^{cd} g^{ef} \partial_c g_{ab} \partial_f g_{de} + \frac{1}{8} g_{ab} g^{cd} g^{ef} g^{gh} \partial_e g_{cd} \partial_f g_{gh} - \\
&\frac{1}{2} g_{ab} g^{cd} g^{ef} \partial_f \partial_d g_{ce} + \frac{1}{2} g_{ab} g^{cd} g^{ef} \partial_f \partial_e g_{cd} + \frac{1}{4} g_{ab} g^{cd} g^{ef} g^{gh} \partial_f g_{dh} \partial_g g_{ce} - \\
&\frac{3}{8} g_{ab} g^{cd} g^{ef} g^{gh} \partial_g g_{ce} \partial_h g_{df} + \frac{1}{2} g_{ab} g^{cd} g^{ef} g^{gh} \partial_d g_{ce} \partial_h g_{fg} - \frac{1}{2} g_{ab} g^{cd} g^{ef} g^{gh} \partial_e g_{cd} \partial_h g_{fg}
\end{aligned}$$

```
In[768]:=
      result - s[] % // ToCanonical

Out[768]=
      0
```

Let us now define the electromagnetic fields:

```
In[769]:=
      DefTensor[MaxwellA[-a], M3, PrintAs -> "A"]

          ** DefTensor: Defining tensor MaxwellA[-a].

In[770]:=
      DefTensor[MaxwellF[-a, -b], M3, Antisymmetric[{1, 2}], PrintAs -> "F"]

          ** DefTensor: Defining tensor MaxwellF[-a, -b].

In[771]:=
      MaxwellF[a_, b_] := PD[a][MaxwellA[b]] - PD[b][MaxwellA[a]]
```

This is the electromagnetic Lagrangian:

```
In[772]:=
      metricg[a, b] metricg[c, d] MaxwellF[-a, -c] MaxwellF[-b, -d] / 4

Out[772]=
```

$$\frac{1}{4} g^{ab} g^{cd} (\partial_a A_c - \partial_c A_a)(\partial_b A_d - \partial_d A_b)$$

and these are the Maxwell equations on a curved background:

```
In[773]:=
      result = VarD[MaxwellA[a], PD][s[] %]
```

Out[773]=

$$\frac{1}{2} g^{bc} g^{de} s \,\partial_a A_b \,\partial_c g_{de} - \frac{1}{2} g^{bc} g^{de} s \,\partial_b A_a \,\partial_c g_{de} + g^{bc} s \,\partial_c \partial_a A_b - g^{bc} s \,\partial_c \partial_b A_a - g^{bd} g^{ce} s \,\partial_c A_b \,\partial_d g_{ae} + g^{bd} g^{ce} s \,\partial_c A_b \,\partial_e g_{ad} - g^{bc} g^{de} s \,\partial_a A_b \,\partial_e g_{cd} + g^{bc} g^{de} s \,\partial_b A_a \,\partial_e g_{cd}$$

```
In[774]:=
      metricg[b, c] CD[-b][MaxwellF[-a, -c]] // ChangeCovD // ChristoffelToGradMetric //
       ToCanonical
```

Out[774]=

$$\frac{1}{2} g^{bc} g^{de} \,\partial_a A_b \,\partial_c g_{de} - \frac{1}{2} g^{bc} g^{de} \,\partial_b A_a \,\partial_c g_{de} + g^{bc} \,\partial_c \partial_a A_b - g^{bc} \,\partial_c \partial_b A_a - g^{bd} g^{ce} \,\partial_c A_b \,\partial_d g_{ae} + g^{bd} g^{ce} \,\partial_c A_b \,\partial_e g_{ad} - g^{bc} g^{de} \,\partial_a A_b \,\partial_e g_{cd} + g^{bc} g^{de} \,\partial_b A_a \,\partial_e g_{cd}$$

```
In[775]:=
      result - s[] % // ToCanonical

Out[775]=
      0

In[776]:=
      UndefTensor /@ {s, MaxwellA, MaxwellF};

          ** UndefTensor: Undefined tensor s

          ** UndefTensor: Undefined tensor MaxwellA

          ** UndefTensor: Undefined tensor MaxwellF
```

Restore standard options:

```
In[777]:=
      SetOptions[ToCanonical, UseMetricOnVBundle → All]

Out[777]=
      {Verbose → False, Notation → Images,
       UseMetricOnVBundle → All, ExpandChristoffel → False, GivePerm → False}
```

8. More on rules

8.1. MakeRule

Defining rules is one of the most important components of a tensor package. The function `IndexRule` takes care of dummy generation, but there are other points to consider when defining a rule:

1. Checking consistency between left and right hand sides.
2. Defining rules that work only for a given vbundle.
3. Extending the rule to pattern expressions with equivalent index−structure under symmetries.
4. Extendig the rule to patterns with a different up/down character if metrics are present.

We define the function `MakeRule`, which takes care of all these things. Note that there is not a parallel function for `Set`, but below we present a way to automatize a collection of rules.

`MakeRule`	Construct rules between `xTensor‘` expressions
`AutomaticRules`	Automate rules

Construction of rules.

```
In[778]:=
      Options[MakeRule]

Out[778]=
      {PatternIndices → All, TestIndices → True, MetricOn → None,
       UseSymmetries → True, Verbose → False, ContractMetrics → False}
```

The simplest way to define a rule would easily give corrupted answers:

```
In[779]:=
      $Tensors

Out[779]=
      {T, r, TT, Force, S, U, TorsionCd, ChristoffelCd, RiemannCd, RicciCd, const, Q,
       TorsionICD, ChristoffelICD, RiemannICD, RicciICD, AChristoffelICD, AChristoffelICD†,
       FRiemannICD, FRiemannICD†, w, metricg, epsilonmetricg, TorsionCD, ChristoffelCD,
       RiemannCD, RicciCD, RicciScalarCD, EinsteinCD, WeylCD, TFRicciCD, n, lapse, v}

In[780]:=
      rule = v[a_] → T[a, -b] v[b]

Out[780]=
```

$v^a \to T^a{}_b\, v^b$

```
In[781]:=
      v[b] /. rule

      Validate::repeated : Found indices with the same name b.

      Throw::nocatch : Uncaught Throw[Null] returned to top level. More…

Out[781]=
      Hold[Throw[Null]]
```

We have seen that changing to `IndexRule` we can solve those problems (underlined indices denote patterns):

```
In[782]:=
      rule = v[a_] ↦ T[a, b] v[-b]
```

```
Out[782]=
      HoldPattern[v^a] :→ Module[{b}, T^ab v_b]
```

```
In[783]:=
      v[b] /. rule
```

```
Out[783]=
```
$T^{ba} v_a$

```
In[784]:=
      % /. rule
```

```
Out[784]=
```
$T_a{}^c T^{ba} v_c$

We can also define the same rule using `MakeRule`. Note that there is no pattern on the lhs; by default all indices on the LHS are converted into patterns:

```
In[785]:=
      rule = MakeRule[{v[a], T[a, -b] v[b]}, MetricOn → None]
```

```
Out[785]=
      {HoldPattern[v^a] :→ Module[{b}, T^a_b v^b]}
```

```
In[786]:=
      T[-a, -b] v[a] v[b] /. rule
```

```
Out[786]=
```
$T_{ab} T^a{}_c T^b{}_d v^c v^d$

```
In[787]:=
      InputForm[rule]
```

```
Out[787]//InputForm=
      {HoldPattern[v[(a_Symbol)?TangentM3‘Q]] :> Module[{h$41203}, T[a,
      -h$41203]*v[h$41203]]}
```

Due to the option `TestIndices→True`, membership of the indices of the LHS is checked. The function `M3‘Q` checks that an up−index belongs to `M3`, but it does not accept down−indices (this is due to the option `MetricOn−>None`):

```
In[788]:=
      v[a] /. rule
```

```
Out[788]=
```
$T^a{}_b v^b$

```
In[789]:=
      v[-a] /. rule
```

```
Out[789]=
```
v_a

In[790]:=

```
v[A] /. rule
```

Out[790]=

v^A

In order to allow both up/down−indices of a given vbundle we use `MetricOn`. We need to have a metric first. (If the vbundle of the index did not have a metric `MakeRule` would refuse to change the up/down character):

In[791]:=

```
rule = MakeRule[{v[a], T[a, -b] v[b]}, MetricOn → {a}, ContractMetrics → True]
```

Out[791]=

$\{\text{HoldPattern}[v^{\underline{a}}] :\to \text{Module}[\{b\}, T^a{}_b\, v^b]\}$

In[792]:=

```
InputForm[rule]
```

Out[792]//InputForm=

```
{HoldPattern[v[(a_)?TangentM3‘pmQ]] :> Module[{h$41215}, T[a,
-h$41215]*v[h$41215]]}
```

In[793]:=

```
v[a] /. rule
```

Out[793]=

$T^a{}_b\, v^b$

In[794]:=

```
v[-a] /. rule
```

Out[794]=

$T_{ab}\, v^b$

If we have symmetries the situation gets even more complicated (here we need to use 6 rules):

In[795]:=

```
(rule = MakeRule[{U[a, b, c] v[-c], v[a] T[b] - v[b] T[a]},
   MetricOn → All, ContractMetrics → True])
```

Out[795]=

$$\{\text{HoldPattern}[U^{\underline{a}\,\underline{b}\,\underline{c}}\, v_{\underline{c}}] :\to \text{Module}[\{\}, T^b v^a - T^a v^b],$$
$$\text{HoldPattern}[U^{\underline{a}\,\underline{b}}{}_{\underline{c}}\, v^{\underline{c}}] :\to \text{Module}[\{\}, T^b v^a - T^a v^b],$$
$$\text{HoldPattern}[U^{\underline{a}\,\underline{c}\,\underline{b}}\, v_{\underline{c}}] :\to \text{Module}[\{\}, -(T^b v^a - T^a v^b)],$$
$$\text{HoldPattern}[U^{\underline{a}}{}_{\underline{c}}{}^{\underline{b}}\, v^{\underline{c}}] :\to \text{Module}[\{\}, -(T^b v^a - T^a v^b)],$$
$$\text{HoldPattern}[U^{\underline{c}\,\underline{a}\,\underline{b}}\, v_{\underline{c}}] :\to \text{Module}[\{\}, T^b v^a - T^a v^b],$$
$$\text{HoldPattern}[U_{\underline{c}}{}^{\underline{a}\,\underline{b}}\, v^{\underline{c}}] :\to \text{Module}[\{\}, T^b v^a - T^a v^b]\}$$

In[796]:=

```
U[-a, b, c] v[-b] /. rule
```

Out[796]=

$-T^c\, v_a + T_a\, v^c$

```
In[797]:=
      U[-a, b, c] v[a] /. rule
```

Out[797]=

$T^c v^b - T^b v^c$

```
In[798]:=
      U[-a, c, -c] v[a] /. rule // ToCanonical
```

Out[798]=

0

It is possible to automate rules, declaring them as definitions for a symbol, in this case the tensor U:

```
In[799]:=
      AutomaticRules[U, rule]

            Rules {1, 2, 3, 4, 5, 6} have been declared as UpValues for U.
```

```
In[800]:=
      U[-a, b, c] v[-b]
```

Out[800]=

$-T^c v_a + T_a v^c$

```
In[801]:=
      U[-a, b, c] v[a]
```

Out[801]=

$T^c v^b - T^b v^c$

```
In[802]:=
     ?U

        Global`U

        Dagger[U] ^:= U

        DependenciesOfTensor[U] ^:= {M3}

        Info[U] ^:= {tensor, }

        PrintAs[U] ^:= U

        SlotsOfTensor[U] ^:= {-TangentM3, -TangentM3, -TangentM3}

        SymmetryGroupOfTensor[U] ^:=
         StrongGenSet[{1, 2}, GenSet[-Cycles[{1, 2}], -Cycles[{2, 3}]]]

        TensorID[U] ^:= {}

        xTensorQ[U] ^:= True
```

$U \,/\!: U^{\underline{a}\,\underline{b}\,\underline{c}}\, v_{\underline{c}} := \mathrm{Module}[\{\}, T^b v^a - T^a v^b]$

$U \,/\!: U^{\underline{a}\,\underline{b}}{}_{\underline{c}}\, v^{\underline{c}} := \mathrm{Module}[\{\}, T^b v^a - T^a v^b]$

$U \,/\!: U^{\underline{a}\,\underline{c}\,\underline{b}}\, v_{\underline{c}} := \mathrm{Module}[\{\}, -(T^b v^a - T^a v^b)]$

$U \,/\!: U^{\underline{a}}{}_{\underline{c}}{}^{\underline{b}}\, v^{\underline{c}} := \mathrm{Module}[\{\}, -(T^b v^a - T^a v^b)]$

$U \,/\!: U^{\underline{c}\,\underline{a}\,\underline{b}}\, v_{\underline{c}} := \mathrm{Module}[\{\}, T^b v^a - T^a v^b]$

$U \,/\!: U_{\underline{c}}{}^{\underline{a}\,\underline{b}}\, v^{\underline{c}} := \mathrm{Module}[\{\}, T^b v^a - T^a v^b]$

The simplest way to remove those definitions for U is removing the tensor and defining it again:

```
In[803]:=
     UndefTensor[U]

        ** UndefTensor: Undefined tensor U

In[804]:=
     DefTensor[U[-a, -b, -c], M3, Antisymmetric[{1, 2, 3}]]

        ** DefTensor: Defining tensor U[-a, -b, -c].
```

8.2. Riemann expressions

As we said, there is no algorithm at present in `xTensor‘` to canonicalize expressions with multiterm symmetries. That is an obstacle in GR when one wants to work with the Riemann tensor, which has a cyclic symmetry. A simple, brute−force, solution is to prepare a number of rules which are only valid for the Riemann tensor. We follow MathTensor’s discussion of this point (see MathTensor book).

There are 48 rules in total, numbered from 1 to 40, some of them having subcases a, b, c, ...

14 of those rules are simple consequences of the monoterm symmetries of the Riemann tensor, and hence can be consid−ered already implemented in `ToCanonical`:

RiemannRule1:

```
In[805]:=
      RiemannCD[-a, a, -c, -d] // ToCanonical

Out[805]=
      0

In[806]:=
      RiemannCD[-a, -b, -c, c] // ToCanonical

Out[806]=
      0
```

RiemannRule2:

```
In[807]:=
      RiemannCD[-a, -b, a, -d] // ToCanonical

Out[807]=
```
$R[\nabla]_{bd}$

RiemannRule3:

```
In[808]:=
      RiemannCD[-a, -b, -c, a] // ToCanonical

Out[808]=
```
$-R[\nabla]_{bc}$

RiemannRule4:

```
In[809]:=
      RiemannCD[-a, -b, b, -d] // ToCanonical

Out[809]=
```
$-R[\nabla]_{ad}$

RiemannRule5:

```
In[810]:=
      RiemannCD[-a, -b, -c, b] // ToCanonical

Out[810]=
      R[▽]ac
```

RiemannRule6:

```
In[811]:=
      RiemannCD[-a, -b, -c, -d] RicciCD[a, b] // ToCanonical

Out[811]=
      0

In[812]:=
      RiemannCD[-a, -b, -c, -d] RicciCD[c, d] // ToCanonical

Out[812]=
      0
```

RiemannRule31:

```
In[813]:=
      CD[-e]@RiemannCD[-a, -b, -c, -d] RicciCD[a, b] // ToCanonical

Out[813]=
      0

In[814]:=
      CD[-e]@RiemannCD[-a, -b, -c, -d] RicciCD[c, d] // ToCanonical

Out[814]=
      0
```

RiemannRule32:

```
In[815]:=
      CD[-e]@RiemannCD[-a, -b, -c, -d] CD[-f]@RicciCD[a, b] // ToCanonical

Out[815]=
      0

In[816]:=
      CD[-e]@RiemannCD[-a, -b, -c, -d] CD[-f]@RicciCD[c, d] // ToCanonical

Out[816]=
      0
```

RiemannRule33:

```
In[817]:=
     CD[-c]@RicciCD[-a, -b] RiemannCD[a, b, d, e] // ToCanonical

Out[817]=
     0

In[818]:=
     CD[-c]@RicciCD[-a, -b] RiemannCD[d, e, a, b] // ToCanonical

Out[818]=
     0
```

There are 2 more rules which can be obtained from the fact that the Einstein tensor is divergence free:

RiemannRule16:

```
In[819]:=
     CD[b][RicciCD[-a, -b]] // RicciToEinstein // ContractMetric

Out[819]=
```

$$\frac{1}{2}\,(\triangledown_a R[\triangledown])$$

RiemannRule17:

```
In[820]:=
     CD[a][RicciCD[-a, -b]] // RicciToEinstein // ContractMetric

Out[820]=
```

$$\frac{1}{2}\,(\triangledown_b R[\triangledown])$$

The other 32 rules are actual consequences of the cyclic symmetry of the Riemann tensor, and therefore must be pro−grammed independently. Using `MakeRule` we can avoid typing equivalent rules (up to permutations of indices), reducing the number to 17 independent rules.

We first study 13 rules having derivatives of a single curvature tensor on the LHS. They are in fact only 6 independent rules:

```
RuleDivRiemann

RuleDivGradRicci

RuleBoxRiemann

RuleDivGradRiemann

RuleD4RicciScalar

RuleD4Ricci
```

The four rules 20−−23 are all the same:

```
In[821]:=
     RuleDivRiemann = MakeRule[{CD[a]@RiemannCD[-a, -b, -c, -d], -CD[-d]@RicciCD[-b, -c] +
          CD[-c]@RicciCD[-b, -d]}, UseSymmetries → True, MetricOn → All];
```

RiemannRule20:

```
In[822]:=
      CD[a]@RiemannCD[-a, -b, -c, -d] /. RuleDivRiemann
```

Out[822]=

$$\nabla_c R[\nabla]_{bd} - \nabla_d R[\nabla]_{bc}$$

RiemannRule21:

```
In[823]:=
      CD[b]@RiemannCD[-a, -b, -c, -d] /. RuleDivRiemann
```

Out[823]=

$$-(\nabla_c R[\nabla]_{ad}) + \nabla_d R[\nabla]_{ac}$$

RiemannRule22:

```
In[824]:=
      CD[c]@RiemannCD[-a, -b, -c, -d] /. RuleDivRiemann
```

Out[824]=

$$\nabla_a R[\nabla]_{db} - \nabla_b R[\nabla]_{da}$$

RiemannRule23:

```
In[825]:=
      CD[d]@RiemannCD[-a, -b, -c, -d] /. RuleDivRiemann
```

Out[825]=

$$-(\nabla_a R[\nabla]_{cb}) + \nabla_b R[\nabla]_{ca}$$

Two rules with second derivatives of Ricci can be combined in a single rule:

```
In[826]:=
      RuleDivGradRicci =
        MakeRule[{CD[b]@CD[-c]@RicciCD[-a, -b], 1/2 CD[-c]@CD[-a]@RicciScalarCD[] +
           RicciCD[-h1, -a] RicciCD[-c, h1] - RicciCD[-h1, -h2] RiemannCD[-a, h2, -c, h1]},
         UseSymmetries → True, MetricOn → All];
```

RiemannRule18:

```
In[827]:=
      CD[b]@CD[-c]@RicciCD[-a, -b] /. RuleDivGradRicci
```

Out[827]=

$$R[\nabla]_{ba} R[\nabla]_c{}^b - R[\nabla]_{bd} R[\nabla]_a{}^d{}_c{}^b + \frac{1}{2} (\nabla_c \nabla_a R[\nabla])$$

RiemannRule19:

```
In[828]:=
     CD[a]@CD[-c]@RicciCD[-a, -b] /. RuleDivGradRicci
```

Out[828]=

$$R[\triangledown]_{ab}\, R[\triangledown]_c{}^a - R[\triangledown]_{ad}\, R[\triangledown]_b{}^d{}_c{}^a + \frac{1}{2}\, (\triangledown_c \triangledown_b R[\triangledown])$$

Rules 26 to 30 are now two rules in `xTensor‘`:

```
In[829]:=
     RuleBoxRiemann = MakeRule[{CD[e]@CD[-e]@RiemannCD[-a, -b, -c, -d],
         CD[-b]@CD[-d]@RicciCD[-a, -c] - CD[-b]@CD[-c]@RicciCD[-a, -d] -
          CD[-a]@CD[-d]@RicciCD[-b, -c] + CD[-a]@CD[-c]@RicciCD[-b, -d] +
          RicciCD[-h1, -b] RiemannCD[-a, h1, -c, -d] -
          RicciCD[-h1, -a] RiemannCD[-b, h1, -c, -d] -
          RiemannCD[-h1, -h2, -c, -d] RiemannCD[-a, h1, -b, h2] -
          RiemannCD[-h1, -b, -h2, -d] RiemannCD[-a, h1, -c, h2] +
          RiemannCD[-h1, -b, -h2, -c] RiemannCD[-a, h1, -d, h2] +
          RiemannCD[-h1, -a, -h2, -d] RiemannCD[-b, h1, -c, h2] -
          RiemannCD[-h1, -a, -h2, -c] RiemannCD[-b, h1, -d, h2] -
          RiemannCD[-h1, -a, -h2, -b] RiemannCD[-c, -d, h1, h2]},
        UseSymmetries → True, MetricOn → All];
```

```
In[830]:=
     RuleDivGradRiemann = MakeRule[
        {CD[a]@CD[-e]@RiemannCD[-a, -b, -c, -d], -CD[-e]@CD[-d]@RicciCD[-b, -c] +
          CD[-e]@CD[-c]@RicciCD[-b, -d] - RicciCD[-h1, -e] RiemannCD[-b, h1, -c, -d] +
          RiemannCD[-h1, -h2, -c, -d] RiemannCD[-b, h1, -e, h2] -
          RiemannCD[-h1, -b, -h2, -d] RiemannCD[-c, h2, -e, h1] +
          RiemannCD[-h1, -b, -h2, -c] RiemannCD[-d, h2, -e, h1]},
        UseSymmetries → True, MetricOn → All];
```

RiemannRule26:

```
In[831]:=
     CD[e]@CD[-e]@RiemannCD[-a, -b, -c, -d] /. RuleBoxRiemann
```

Out[831]=

$$R[\triangledown]_{eb}\, R[\triangledown]_a{}^e{}_{cd} - R[\triangledown]_{ea}\, R[\triangledown]_b{}^e{}_{cd} - R[\triangledown]_{cd}{}^{ef}\, R[\triangledown]_{eafb} - R[\triangledown]_b{}^e{}_d{}^f\, R[\triangledown]_{eafc} + R[\triangledown]_b{}^e{}_c{}^f\, R[\triangledown]_{eafd} + R[\triangledown]_a{}^e{}_d{}^f\, R[\triangledown]_{ebfc} - R[\triangledown]_a{}^e{}_c{}^f\, R[\triangledown]_{ebfd} - R[\triangledown]_a{}^e{}_b{}^f\, R[\triangledown]_{efcd} + \triangledown_a \triangledown_c R[\triangledown]_{bd} - \triangledown_a \triangledown_d R[\triangledown]_{bc} - \triangledown_b \triangledown_c R[\triangledown]_{ad} + \triangledown_b \triangledown_d R[\triangledown]_{ac}$$

RiemannRule27:

```
In[832]:=
     CD[a]@CD[-e]@RiemannCD[-a, -b, -c, -d] /. RuleDivGradRiemann
```

Out[832]=

$$-R[\triangledown]_{ae}\, R[\triangledown]_b{}^a{}_{cd} + R[\triangledown]_{afcd}\, R[\triangledown]_b{}^a{}_e{}^f - R[\triangledown]_{abfd}\, R[\triangledown]_c{}^f{}_e{}^a + R[\triangledown]_{abfc}\, R[\triangledown]_d{}^f{}_e{}^a + \triangledown_e \triangledown_c R[\triangledown]_{bd} - \triangledown_e \triangledown_d R[\triangledown]_{bc}$$

RiemannRule28:

```
In[833]:=
     CD[b]@CD[-e]@RiemannCD[-a, -b, -c, -d] /. RuleDivGradRiemann
```

Out[833]=

$$R[\nabla]_{be}\,R[\nabla]_{a}{}^{b}{}_{cd}-R[\nabla]_{a}{}^{b}{}_{e}{}^{f}\,R[\nabla]_{bfcd}+R[\nabla]_{bafd}\,R[\nabla]_{c}{}^{f}{}_{e}{}^{b}-R[\nabla]_{bafc}\,R[\nabla]_{d}{}^{f}{}_{e}{}^{b}-\nabla_e\nabla_c R[\nabla]_{ad}+\nabla_e\nabla_d R[\nabla]_{ac}$$

RiemannRule29:

```
In[834]:=
     CD[c]@CD[-e]@RiemannCD[-a, -b, -c, -d] /. RuleDivGradRiemann
```

Out[834]=

$$R[\nabla]_{b}{}^{f}{}_{e}{}^{c}\,R[\nabla]_{cdfa}-R[\nabla]_{a}{}^{f}{}_{e}{}^{c}\,R[\nabla]_{cdfb}-R[\nabla]_{ce}\,R[\nabla]_{d}{}^{c}{}_{ab}+R[\nabla]_{cfab}\,R[\nabla]_{d}{}^{c}{}_{e}{}^{f}+\nabla_e\nabla_a R[\nabla]_{db}-\nabla_e\nabla_b R[\nabla]_{da}$$

RiemannRule30:

```
In[835]:=
     CD[d]@CD[-e]@RiemannCD[-a, -b, -c, -d] /. RuleDivGradRiemann
```

Out[835]=

$$R[\nabla]_{de}\,R[\nabla]_{c}{}^{d}{}_{ab}-R[\nabla]_{b}{}^{f}{}_{e}{}^{d}\,R[\nabla]_{dcfa}+R[\nabla]_{a}{}^{f}{}_{e}{}^{d}\,R[\nabla]_{dcfb}-R[\nabla]_{c}{}^{d}{}_{e}{}^{f}\,R[\nabla]_{dfab}-\nabla_e\nabla_a R[\nabla]_{cb}+\nabla_e\nabla_b R[\nabla]_{ca}$$

Fourth derivative of the Ricci scalar: conversion to Box^2:

```
In[836]:=
     RuleD4RicciScalar = MakeRule[{CD[b]@CD[a]@CD[-b]@CD[-a]@RicciScalarCD[],
         1 / 2 CD[-h1]@RicciScalarCD[] CD[h1]@RicciScalarCD[] +
          CD[h2]@CD[-h2]@CD[h1]@CD[-h1]@RicciScalarCD[] +
          CD[-h2]@CD[-h1]@RicciScalarCD[] RicciCD[h1, h2]},
        UseSymmetries → True, MetricOn → All];
```

RiemannRule24:

```
In[837]:=
     CD[b]@CD[a]@CD[-b]@CD[-a]@RicciScalarCD[] /. RuleD4RicciScalar
```

Out[837]=

$$\frac{1}{2}\left(\nabla_a R[\nabla]\right)\left(\nabla^a R[\nabla]\right)+R[\nabla]^{ab}\left(\nabla_b\nabla_a R[\nabla]\right)+\nabla^b\nabla_b\nabla^a\nabla_a R[\nabla]$$

Fourth derivative of the Ricci tensor:

```
In[838]:=
     RuleD4Ricci = MakeRule[{CD[b]@CD[a]@CD[c]@CD[-c]@RicciCD[-a, -b],
         1 / 2 CD[-h1]@RicciScalarCD[] CD[h1]@RicciScalarCD[] -
          3 CD[-h3]@RicciCD[-h1, -h2] CD[h3]@RicciCD[h1, h2] + 4 CD[-h3]@RicciCD[-h1, -h2]
           CD[h2]@RicciCD[h1, h3] + 1 / 2 CD[h2]@CD[-h2]@CD[h1]@CD[-h1]@RicciScalarCD[] +
          2 CD[-h2]@CD[-h1]@RicciScalarCD[] RicciCD[h1, h2] -
          CD[h3]@CD[-h3]@RicciCD[-h1, -h2] RicciCD[h1, h2] +
          2 RicciCD[-h1, -h2] RicciCD[-h3, h1] RicciCD[h2, h3] -
          2 CD[-h4]@CD[-h3]@RicciCD[-h1, -h2] RiemannCD[h1, h3, h2, h4] -
          2 RicciCD[-h1, -h2] RicciCD[-h3, -h4] RiemannCD[h1, h3, h2, h4]},
        UseSymmetries → True, MetricOn → All];
```

RiemannRule25:

```
In[839]:=
     CD[b]@CD[a]@CD[c]@CD[-c]@RicciCD[-a, -b] /. RuleD4Ricci
```

Out[839]=

$$2 R[\nabla]_{ab} R[\nabla]^{bc} R[\nabla]_{c}{}^{a} - 2 R[\nabla]_{ab} R[\nabla]_{cd} R[\nabla]^{acbd} + \frac{1}{2} (\nabla_a R[\nabla]) (\nabla^a R[\nabla]) + 2 R[\nabla]^{ab} (\nabla_b \nabla_a R[\nabla]) + \frac{1}{2} (\nabla^b \nabla_b \nabla^a \nabla_a R[\nabla]) + 4 (\nabla^b R[\nabla]^{ac}) (\nabla_c R[\nabla]_{ab}) - 3 (\nabla_c R[\nabla]_{ab}) (\nabla^c R[\nabla]^{ab}) - R[\nabla]^{ab} (\nabla^c \nabla_c R[\nabla]_{ab}) - 2 R[\nabla]^{acbd} (\nabla_d \nabla_c R[\nabla]_{ab})$$

Now we study six rules with products of two Riemann tensors on the LHS. There are only two different rules, depending on whether there are two or three contracted indices:

```
RuleTwoRiemann3

RuleTwoRiemann2
```

These are the independent rules:

```
In[840]:=
     RuleTwoRiemann3 = MakeRule[{RiemannCD[-a, -b, -c, -d] RiemannCD[-e, a, b, c],
         -1 / 2 RiemannCD[-f, -d, -g, -h] RiemannCD[-e, f, g, h]},
        UseSymmetries → True, MetricOn → All];

In[841]:=
     RuleTwoRiemann2 = MakeRule[{RiemannCD[-a, -b, -c, -d] RiemannCD[-e, a, -g, b],
         1 / 2 RiemannCD[-f, -h, -c, -d] RiemannCD[-e, -g, f, h]},
        UseSymmetries → True, MetricOn → All];
```

RiemannRule7:

```
In[842]:=
     RiemannCD[-a, -b, -c, -d] RiemannCD[-e, a, b, c] /. RuleTwoRiemann3
```

Out[842]=

$$-\frac{1}{2} R[\nabla]_{adbc} R[\nabla]_{e}{}^{abc}$$

RiemannRule8:

```
In[843]:=
      RiemannCD[-a, -b, -c, -d] RiemannCD[a, c, b, h] /. RuleTwoRiemann3
```

Out[843]=

$$-\frac{1}{2}\, g^{he}\, R[\nabla]_{adbc}\, R[\nabla]_{e}{}^{abc}$$

RiemannRule9a:

```
In[844]:=
      RiemannCD[-a, -b, -c, -d] RiemannCD[-e, a, -g, b] /. RuleTwoRiemann2
```

Out[844]=

$$\frac{1}{2}\, R[\nabla]_{abcd}\, R[\nabla]_{eg}{}^{ab}$$

RiemannRule9b:

```
In[845]:=
      RiemannCD[-a, -b, -c, -d] RiemannCD[e, a, b, h] /. RuleTwoRiemann2
```

Out[845]=

$$-\frac{1}{2}\, g^{ef}\, g^{hg}\, R[\nabla]_{abcd}\, R[\nabla]_{fg}{}^{ab}$$

RiemannRule9c:

```
In[846]:=
      RiemannCD[-a, -b, -c, -d] RiemannCD[a, f, b, h] /. RuleTwoRiemann2
```

Out[846]=

$$\frac{1}{2}\, g^{fe}\, g^{hg}\, R[\nabla]_{abcd}\, R[\nabla]_{eg}{}^{ab}$$

RiemannRule9d:

```
In[847]:=
      RiemannCD[-a, -b, -c, -d] RiemannCD[a, f, g, b] /. RuleTwoRiemann2
```

Out[847]=

$$-\frac{1}{2}\, g^{fe}\, g^{gh}\, R[\nabla]_{abcd}\, R[\nabla]_{eh}{}^{ab}$$

There are seven more rules involving two curvature tensors and derivatives, corresponding to five independent rules:

```
RuleD01TwoRiemann
RuleD11TwoRiemann
RuleD2RicciRiemann
RuleD2RicciDRiemann
RuleD11bisTwoRiemann
```

```
In[848]:=
      RuleD01TwoRiemann = MakeRule[{CD[-e]@RiemannCD[a, f, b, c] RiemannCD[-a, -b, -c, -d],
          -1 / 2 CD[-e]@RiemannCD[h1, f, h2, h3] RiemannCD[-h1, -d, -h2, -h3]},
         UseSymmetries → True, MetricOn → All];

In[849]:=
      RuleD11TwoRiemann =
        MakeRule[{CD[-e]@RiemannCD[-a, -b, -c, -d] CD[-f]@RiemannCD[a, g, b, c],
          -1 / 2 CD[-e]@RiemannCD[-h1, -d, -h2, -h3] CD[-f]@RiemannCD[h1, g, h2, h3]},
         UseSymmetries → True, MetricOn → All];

In[850]:=
      RuleD2RicciRiemann = MakeRule[{CD[-d]@CD[-c]@RicciCD[-a, -b] RiemannCD[a, d, b, c],
          CD[-h4]@CD[-h3]@RicciCD[-h1, -h2] RiemannCD[h1, h3, h2, h4]},
         UseSymmetries → True, MetricOn → All];

In[851]:=
      RuleD2RicciDRiemann =
        MakeRule[{CD[-e]@RiemannCD[a, d, b, c] CD[-d]@CD[-c]@RicciCD[-a, -b],
          CD[-e]@RiemannCD[h1, h2, h3, h4] CD[-h4]@CD[-h2]@RicciCD[-h1, -h3]},
         UseSymmetries → True, MetricOn → All];
```

This rule has a typo in MathTensor’s book. I assume the index c on the RHS is actually f:

```
In[852]:=
      RuleD11bisTwoRiemann =
        MakeRule[{CD[-g]@RiemannCD[-a, -b, -c, -d] CD[g]@RiemannCD[c, e, d, f],
          1 / 2 CD[-h3]@RiemannCD[-h1, -h2, -a, -b] CD[h3]@RiemannCD[h1, h2, f, e]},
         UseSymmetries → True, MetricOn → All];
```

RiemannRule34:

```
In[853]:=
      CD[-e]@RiemannCD[a, f, b, c] RiemannCD[-a, -b, -c, -d] /. RuleD01TwoRiemann
```

Out[853]=

$$-\frac{1}{2} R[\nabla]_{adbc} (\nabla_e R[\nabla]^{afbc})$$

RiemannRule35:

```
In[854]:=
      CD[-e]@RiemannCD[-a, -b, -c, -d] CD[-f]@RiemannCD[a, g, b, c] /. RuleD11TwoRiemann
```

Out[854]=

$$-\frac{1}{2} (\nabla_e R[\nabla]_{adbc}) (\nabla_f R[\nabla]^{agbc})$$

RiemannRule36:

```
In[855]:=
      CD[-e]@RiemannCD[a, c, b, f] RiemannCD[-a, -b, -c, -d] /. RuleD01TwoRiemann
```

Out[855]=

$$\frac{1}{2} R[\nabla]_{adbc} (\nabla_e R[\nabla]^{afbc})$$

RiemannRule37:

```
In[856]:=
      CD[-e]@RiemannCD[-a, -b, -c, -d] CD[-f]@RiemannCD[a, c, b, g] /. RuleD11TwoRiemann
Out[856]=
```

$$\frac{1}{2}\,(\nabla_e R[\nabla]_{adbc})\,(\nabla_f R[\nabla]^{agbc})$$

RiemannRule38:

```
In[857]:=
      CD[-g]@RiemannCD[-a, -b, -c, -d] CD[g]@RiemannCD[c, e, d, f] /. RuleD11bisTwoRiemann
Out[857]=
```

$$\frac{1}{2}\,(\nabla_g R[\nabla]_{cdab})\,(\nabla^g R[\nabla]^{cdfe})$$

RiemannRule39:

```
In[858]:=
      CD[-e]@RiemannCD[a, d, b, c] CD[-d]@CD[-c]@RicciCD[-a, -b] /. RuleD2RicciDRiemann
Out[858]=
```

$$(\nabla_d \nabla_b R[\nabla]_{ac})\,(\nabla_e R[\nabla]^{abcd})$$

RiemannRule40:

```
In[859]:=
      CD[-d]@CD[-c]@RicciCD[-a, -b] RiemannCD[a, d, b, c] /. RuleD2RicciRiemann
Out[859]=
```

$$R[\nabla]^{acbd}\,(\nabla_d \nabla_c R[\nabla]_{ab})$$

Finally we study six rules with products of three curvature tensors, only four of which are independent:

`RuleRicciTwoRiemann`

`RuleThreeRiemann`

`RuleThreeRiemannB`

`RuleThreeRiemannC`

Constructing the independent rules takes rather long because there are lots of equivalent cases:

```
In[860]:=
      AbsoluteTiming[Length[RuleRicciTwoRiemann =
         MakeRule[{RicciCD[-a, -b] RiemannCD[-c, -d, -e, a] RiemannCD[b, c, d, e],
           -1/2 RicciCD[-h1, -h2] RiemannCD[-h3, -h4, -h5, h1] RiemannCD[h2, h5, h3, h4]},
          UseSymmetries → True, MetricOn → All]]]
Out[860]=
      {12.024198 Second, 4096}
```

```
In[861]:=
     AbsoluteTiming[Length[RuleThreeRiemann =
        MakeRule[{RiemannCD[-a, -b, -c, -d] RiemannCD[-e, -f, a, b] RiemannCD[c, e, d, f],
          1 / 2 RiemannCD[-h1, -h2, -h3, -h4] RiemannCD[-h5, -h6, h1, h2]
           RiemannCD[h3, h4, h5, h6]}, UseSymmetries → True, MetricOn → All]]]

Out[861]=
     {122.842100 Second, 32768}

In[862]:=
     AbsoluteTiming[Length[RuleThreeRiemannB =
        MakeRule[{RiemannCD[-a, -b, -c, -d] RiemannCD[-e, a, -f, b] RiemannCD[c, e, d, f],
          1 / 4 RiemannCD[-h1, -h2, -h3, -h4] RiemannCD[-h5, -h6, h3, h4]
           RiemannCD[h1, h2, h5, h6]}, UseSymmetries → True, MetricOn → All]]]

Out[862]=
     {122.398527 Second, 32768}

In[863]:=
     AbsoluteTiming[Length[RuleThreeRiemannC =
        MakeRule[{RiemannCD[-a, -b, -c, -d] RiemannCD[-e, a, -f, c] RiemannCD[b, f, d, e],
          RiemannCD[-h1, -h2, -h3, -h4] RiemannCD[-h5, h1, -h6, h3] RiemannCD[h2, h5,
             h4, h6] - 1 / 4 RiemannCD[-h1, -h2, -h3, -h4] RiemannCD[-h5, -h6, h1, h2]
            RiemannCD[h3, h4, h5, h6]}, UseSymmetries → True, MetricOn → All]]]

Out[863]=
     {200.198221 Second, 32768}
```

RiemannRule10:

```
In[864]:=
     RicciCD[-a, -b] RiemannCD[-c, -d, -e, a] RiemannCD[b, c, d, e] /. RuleRicciTwoRiemann

Out[864]=
```

$$-\frac{1}{2} R[\nabla]_{ab} R[\nabla]^{becd} R[\nabla]_{cde}{}^{a}$$

RiemannRule11:

```
In[865]:=
     RiemannCD[-a, -b, -c, -d] RiemannCD[-e, -f, a, b] RiemannCD[c, e, d, f] /.
      RuleThreeRiemann

Out[865]=
```

$$\frac{1}{2} R[\nabla]_{abcd} R[\nabla]^{cdef} R[\nabla]_{ef}{}^{ab}$$

RiemannRule12:

```
In[866]:=
     RiemannCD[-a, -b, -c, -d] RiemannCD[-e, a, -f, b] RiemannCD[c, e, d, f] /.
      RuleThreeRiemannB

Out[866]=
```

$$\frac{1}{4} R[\nabla]_{abcd} R[\nabla]^{abef} R[\nabla]_{ef}{}^{cd}$$

RiemannRule13:

```
In[867]:=
      RiemannCD[-a, -b, -c, -d] RiemannCD[-e, a, -f, b] RiemannCD[c, d, e, f] /.
       RuleThreeRiemann
```

Out[867]=

$$\frac{1}{2} R[\nabla]_{abcd}\, R[\nabla]^{cdef}\, R[\nabla]_{ef}{}^{ab}$$

RiemannRule14:

```
In[868]:=
      RiemannCD[-a, -b, -c, -d] RiemannCD[-e, a, -f, c] RiemannCD[b, d, e, f] /.
       RuleThreeRiemannB
```

Out[868]=

$$\frac{1}{4} R[\nabla]_{abcd}\, R[\nabla]^{abef}\, R[\nabla]_{ef}{}^{cd}$$

RiemannRule15:

```
In[869]:=
      RiemannCD[-a, -b, -c, -d] RiemannCD[-e, a, -f, c] RiemannCD[b, f, d, e] /.
       RuleThreeRiemannC
```

Out[869]=

$$R[\nabla]_{abcd}\, R[\nabla]^{bedf}\, R[\nabla]_{e}{}^{a}{}_{f}{}^{c} - \frac{1}{4} R[\nabla]_{abcd}\, R[\nabla]^{cdef}\, R[\nabla]_{ef}{}^{ab}$$

9. Manipulation of expressions

9.1. Find indices

Given an expression, very often we want to extract all indices of the expression, or those of certain type, or character or state. We have already seen how to select indices of any type and any character. In this subsection we shall define functions to extract all indices according to a number of ‘ ‘selectors´ ´.

There are two sets of functions. There is first the internal function `FindIndices`, which returns all indices of an expression, without first evaluating it, and its relatives `FindFreeIndices`, `FindDummyIndices` and `Find-BlockedIndices`. There is then the user driver for finding indices, called `IndicesOf`, which admits a number of criteria for index selection, and evaluates its input (as the user typically expects).

Let us start with this expression

```
In[870]:=
      expr = 4 T[a, b, -c, Dir[v[d]]] v[-a] r[]^2 CD[-e][S[LI[1], -b, d]]
```

Out[870]=

$$4\, r^2\, T^{ab}{}_{cv}\, v_a\, (\nabla_e S^1{}_b{}^d)$$

The three `Find*Indices` functions have attribute `HoldFirst`, and hence require the use of `Evaluate` when we do not feed the expression itself:

```
In[871]:=
      FindIndices[Evaluate[expr]]

Out[871]=
      {a, b, -c, Dir[v^d], -a, LI[1], -b, d, -e}

In[872]:=
      FindFreeIndices[Evaluate[expr]]

Out[872]=
      {-c, d, -e}

In[873]:=
      FindDummyIndices[Evaluate[expr]]

Out[873]=
      {a, b}

In[874]:=
      FindBlockedIndices[Evaluate[expr]]

Out[874]=
      {Dir[v^d], LI[1]}
```

This attribute is given to be able to extract indices of expressions before they get evaluated:

```
In[875]:=
      n[a] n[-a]

Out[875]=
      -1

In[876]:=
      FindIndices[n[a] n[-a]]

Out[876]=
      {a, -a}
```

Note that the returned lists of indices have head `IndexList`, in order to avoid confusion with basis and component indices:

```
In[877]:=
      InputForm[%]

Out[877]//InputForm=
     IndexList[a, -a]
```

`FindIndices`	Get all indices of an expression
`FindFreeIndices`	Get all free indices of an expression
`FindDummyIndices`	Get all dummy indices of an expression
`FindBlockedIndices`	Get all blocked indices of an expression
`IndexList`	Head for a list of indices

Finding indices. Internal functions

Then we have the user driver, with two pairs of brackets:

With no selectors, we get all indices in the expression:

```
In[878]:=
      IndicesOf[][expr]

Out[878]=
      {a, b, -c, Dir[v^d], -a, LI[1], -b, d, -e}
```

We can select indices by type:

```
In[879]:=
      IndicesOf[AIndex][expr]

Out[879]=
      {a, b, -c, -a, -b, d, -e}

In[880]:=
      IndicesOf[DIndex][expr]

Out[880]=
      {Dir[v^d]}

In[881]:=
      IndicesOf[LIndex][expr]

Out[881]=
      {LI[1]}
```

or by character:

```
In[882]:=
      IndicesOf[Up][expr]

Out[882]=
      {a, b, LI[1], d}

In[883]:=
      IndicesOf[Down][expr]

Out[883]=
      {-c, Dir[v^d], -a, -b, -e}
```

or by state:

```
In[884]:=
      IndicesOf[Free][expr]

Out[884]=
      {-c, d, -e}

In[885]:=
      IndicesOf[Dummy][expr]

Out[885]=
      {-a, a, -b, b}
```

```
In[886]:=
      IndicesOf[Blocked][expr]
```

```
Out[886]=
      {Dir[v^d], LI[1]}
```

We can also specify a vbundle (or a basis):

```
In[887]:=
      IndicesOf[TangentM3][expr]
```

```
Out[887]=
      {a, b, -c, Dir[v^d], -a, -b, d, -e}
```

or a tensor:

```
In[888]:=
      IndicesOf[S][expr]
```

```
Out[888]=
      {-b, d, LI[1]}
```

We can combine several selectors, in two possible ways: a list of selectors represents the union, and a sequence represents the intersection.

Directions or labels:

```
In[889]:=
      IndicesOf[{DIndex, LIndex}][expr]
```

```
Out[889]=
      {Dir[v^d], LI[1]}
```

Free indices on the tensor S:

```
In[890]:=
      IndicesOf[Free, S][expr]
```

```
Out[890]=
      {d}
```

Members of a selectors list are conmutative, but not members of a sequence: the construction of the final list is performed from left to right. Compare with the previous result:

```
In[891]:=
      IndicesOf[S, Free][expr]
```

```
Out[891]=
      {-b, d}
```

This powerful function is very flexible and has more possibilities once we allow the presence of bases (see xCobaDoc.nb).

`IndicesOf`	Get all indices of an expression according to a number of selecting criteria

Finding indices. User driver

9.2. Replace indices

A second, commonly required operation is the change of indices. In principle, the user should never need to replace indices directly in an expression. But sometimes it is needed, either because two expressions with different free indices must be compared, or to avoid problems with the dummy indices. This subsection introduces the basic function `ReplaceIndex`, and five more functions based on it.

Replacing indices is a basic critical operation: done without care would inmediately lead to wrong results. Whenever possible, we recommend the use of delta contractions to change indices, which is a safer and mathematically clearer operation.

Above all, avoid the temptation of changing indices using normal rules, unless you are sure that index−collisions will not appear.

Let us go back to our expression:

```
In[892]:=
        expr
```

Out[892]=

$$4\,r^2\,T^{ab}{}_{cv}\,v_a\,(\nabla_e S1_b{}^d)$$

For similar reasons, `ReplaceIndex` has attribute `HoldFirst`, and hence we need `Evaluate`. We can replace a single index:

```
In[893]:=
        ReplaceIndex[Evaluate[expr], a → f]
```

Out[893]=

$$4\,r^2\,T^{fb}{}_{cv}\,v_a\,(\nabla_e S1_b{}^d)$$

Note that the paired index `−a` has not been replaced. We need separate rules for that (this gives greater flexibility):

```
In[894]:=
        ReplaceIndex[Evaluate[expr], {a → f, -a → -f}]
```

Out[894]=

$$4\,r^2\,T^{fb}{}_{cv}\,v_f\,(\nabla_e S1_b{}^d)$$

We can change the character of an index, or even the type:

```
In[895]:=
        ReplaceIndex[Evaluate[expr], {a → -f, -a → f, Dir[v[d]] → LI[2]}]
```

Out[895]=

$$4\,r^2\,T_f{}^b{}_c{}^2\,v^f\,(\nabla_e S1_b{}^d)$$

If the rules do not apply, nothing happens:

```
In[896]:=
        ReplaceIndex[Evaluate[expr], {f → a}]
```

Out[896]=

$$4\,r^2\,T^{ab}{}_{cv}\,v_a\,(\nabla_e S1_b{}^d)$$

Pattern indices can be changed, but this should never be required:

```
In[897]:=
      ReplaceIndex[T[a_], a → b]

      ReplaceIndex::nopat :
       Replaced pattern a_. From General. This should not happen: contact JMM.
```

Out[897]=

T^{b}

The replacement is performed with great care. In the following list T is a tensor, but X is not a tensor:

```
In[898]:=
      ReplaceIndex[{a, T[a], T[-a], X[a]}, a → b]
```

Out[898]=

$\{a, T^{b}, T_{a}, X[a]\}$

As I said, the user should try to refrain from using `ReplaceIndex`. If a free index must be changed, use contractions with delta tensors: if we want to change from `T[a]` to `T[b]` we can do this:

```
In[899]:=
      T[a] delta[-a, b]
```

Out[899]=

T^{b}

In some cases we need to change the dummy indices of an expression. For example when we want to multiply a previ− ous expression by itself. `ReplaceDummies` is the function which calls `ReplaceIndex` to manipulate all possible cases with dummies. A particular case, always required in the canonicalization process, is implemented in `SameDummies`.

Suppose now this scalar expression:

```
In[900]:=
      expr = T[a, b] T[-a, -b]
```

Out[900]=

$T_{ab} T^{ab}$

This is incorrect:

```
In[901]:=
      expr expr
```

Out[901]=

${T_{ab}}^{2} {T^{ab}}^{2}$

The following is correct. `ReplaceDummies` changes the dummy by the given indices

```
In[902]:=
      expr ReplaceDummies[expr, IndexList[e, f]]
```

Out[902]=

$T_{ab} T^{ab} T_{ef} T^{ef}$

If not enough indices (or no index at all) are given, the dummies are replaced by unique dollar−indices:

```
In[903]:=
      expr ReplaceDummies[expr]
```

Out[903]=

$T_{ab} T^{ab} T_{cd} T^{cd}$

```
In[904]:=
      IndicesOf[][%]

Out[904]=
      {-a, -b, a, b, -h$108555, -h$108556, h$108555, h$108556}
```

A particular case of dummy manipulation is the minimization of the number of dummies in a sum of terms. This is achieved using `SameDummies`, which is simply a combined call to `FindDummyIndices` and `ReplaceDummies`.

```
In[905]:=
      SameDummies[T[a, b, d, -d] v[-b] + T[a, c, e, -e] v[-c]]
```

Out[905]=

$2 T^{abc}{}_{c} v_{b}$

In parallel, sometimes we need to change the free indices of an expression. This happens for example when we need to compare the results of several independent computations. If we do not know the index translation table which brings one expression to the other, then we need to search among all permutations, what could take quite some time...

This changes the free indices a,c to b,e. The index b was a dummy in the original expression, and hence it must be replaced (by a unique dollar−dummy) before actually changing the free indices:

```
In[906]:=
      ChangeFreeIndices[T[a, b, c, d, -b, -d], {b, e}]
```

Out[906]=

$T^{baed}{}_{ad}$

It is not obvious to see whether these two expressions are the same or not under a renaming of free indices:

```
In[907]:=
      expr1 = 4 U[-a, -b, -c] U[b, -d, -e] T[c, d]
      expr2 = 2 U[-d, -b, -a] U[a, -f, g] T[d, -g]
```

Out[907]=

$4 T^{cd} U_{abc} U^{b}{}_{de}$

Out[908]=

$2 T^{d}{}_{g} U^{a}{}_{f}{}^{g} U_{dba}$

The function `EqualExpressionsQ` gives `True` if one is simply a constant times the other, and `False` otherwise:

```
In[909]:=
      EqualExpressionsQ[expr1, expr2]

         LEFT  == -2 RIGHT   with rules: {-a → -b, -e → -f}

Out[909]=
      True
```

```
In[910]:=
      EqualExpressionsQ[expr1, T[] expr2]

Out[910]=
      False
```

Finally, we introduce the operation of index splitting. The key idea is the concept of a splitrule of the form `a -> IndexList[ a1, a2, ... ]`. The function `SplitIndex` is rather efficient because it uses `ReplaceIndex` only once, replacing the index by a temporary variable, which is later used to do all other replacements with `ReplaceAll`, which is much faster.

We can construct a list of replacements as follows:

```
In[911]:=
      SplitIndex[expr1, -a → IndexList[-A, -B, -C]]

Out[911]=
```
$\{4\, T^{cd}\, U_{Abc}\, U^{b}{}_{de},\ 4\, T^{cd}\, U^{b}{}_{de}\, U_{Bbc},\ 4\, T^{cd}\, U^{b}{}_{de}\, U_{Cbc}\}$

or an array:

```
In[912]:=
      SplitIndex[expr1, {-a → IndexList[-A, -B, -C], -b → IndexList[D, F, G, H]}]

Out[912]=
```
$$\{\{4\, T^{cd}\, U_{A}{}^{D}{}_{c}\, U^{b}{}_{de},\ 4\, T^{cd}\, U_{A}{}^{F}{}_{c}\, U^{b}{}_{de},\ 4\, T^{cd}\, U_{A}{}^{G}{}_{c}\, U^{b}{}_{de},\ 4\, T^{cd}\, U_{A}{}^{H}{}_{c}\, U^{b}{}_{de}\},$$
$$\{4\, T^{cd}\, U^{b}{}_{de}\, U_{B}{}^{D}{}_{c},\ 4\, T^{cd}\, U^{b}{}_{de}\, U_{B}{}^{F}{}_{c},\ 4\, T^{cd}\, U^{b}{}_{de}\, U_{B}{}^{G}{}_{c},\ 4\, T^{cd}\, U^{b}{}_{de}\, U_{B}{}^{H}{}_{c}\},$$
$$\{4\, T^{cd}\, U^{b}{}_{de}\, U_{C}{}^{D}{}_{c},\ 4\, T^{cd}\, U^{b}{}_{de}\, U_{C}{}^{F}{}_{c},\ 4\, T^{cd}\, U^{b}{}_{de}\, U_{C}{}^{G}{}_{c},\ 4\, T^{cd}\, U^{b}{}_{de}\, U_{C}{}^{H}{}_{c}\}\}$$

```
In[913]:=
      Dimensions[%]

Out[913]=
      {3, 4}
```

There is also a combined notation for multiple expansions over the same range. Again, this is performed through a single call to `ReplaceIndex`.

```
In[914]:=
      SplitIndex[expr1, IndexList[-a, -b] → IndexList[LI[1], LI[2], LI[3]]]

Out[914]=
```
$$\{\{4\, T^{cd}\, U^{b}{}_{de}\, U^{11}{}_{c},\ 4\, T^{cd}\, U^{b}{}_{de}\, U^{12}{}_{c},\ 4\, T^{cd}\, U^{b}{}_{de}\, U^{13}{}_{c}\},$$
$$\{4\, T^{cd}\, U^{b}{}_{de}\, U^{21}{}_{c},\ 4\, T^{cd}\, U^{b}{}_{de}\, U^{22}{}_{c},\ 4\, T^{cd}\, U^{b}{}_{de}\, U^{23}{}_{c}\},$$
$$\{4\, T^{cd}\, U^{b}{}_{de}\, U^{31}{}_{c},\ 4\, T^{cd}\, U^{b}{}_{de}\, U^{32}{}_{c},\ 4\, T^{cd}\, U^{b}{}_{de}\, U^{33}{}_{c}\}\}$$

`ReplaceIndex`	Replace an index by another index in an expression
`ReplaceDummies`	Replace dummy indices by new dummy indices
`SameDummies`	Minimize number of different dummy indices among several terms of a sum
`ChangeFreeIndices`	Change free indices of an expression
`EqualExpressionsQ`	Check whether two expressions are proportional to each other apart from a renaming of free indices
`SplitIndex`	Return a list of expressions where an index has been replace by different possibilities in a list
`TraceDummy`	Return a sum expressions where a dummy pair has been replace by different possibilities in a list

Index replacements.

9.3. Scalar and scalar functions

Another important problem is the following. In the same way that we are using `Times` for a product of tensors, we shall use `Power`, `Sqrt`, `Exp` and other functions acting on scalar tensor expressions. There is a problem: there are built−in definitions for those functions that could be incorrect for tensors. For example we have `(a b)^n = a^n b^n`. However `(v[a] v[−a])^n` is not equal to `v[a]^n v[−a]^n`. To prevent those problems we have two options: we could define new functions IndexPower, IndexSqrt, etc, or we can introduce a new head `Scalar` such that the argu− ments of those functions must be always wrapped with that head.

`Scalar`	Wrapper to isolate scalar expressions
`PutScalar`	Wrap scalars with the head Scalar
`BreakScalars`	Break Scalar expressions into irreducible Scalar expressions
`NoScalar`	Remove all Scalar heads, introducing new dummies wherever needed

Scalar expressions.

The direct use of `Power` could lead to unconsistent expressions:

```
In[915]:=
      (v[a] v[-a])^2
```

Out[915]=

$$v_a^{\,2}\, v^{a\,2}$$

Instead, the right way to write it would be

```
In[916]:=
      Scalar[v[a] v[-a]]^2
```

Out[916]=

$$\text{Scalar}[v_a\, v^a]^2$$

`xTensor‘` can work with those objects as usual:

```
In[917]:=
      Scalar[v[a] v[-a]]^2 Scalar[v[b] v[-b]]^3 / Scalar[v[c] v[-c]]^4
```

Out[917]=

$$\frac{\text{Scalar}[v_a\, v^a]^2\ \text{Scalar}[v_b\, v^b]^3}{\text{Scalar}[v_c\, v^c]^4}$$

```
In[918]:=
      Simplification[%]
```

Out[918]=

$$\text{Scalar}[v_a\, v^a]$$

Note that `Scalar` effectively shields its argument from the rest of an expression and therefore there can be repeated indices:

```
In[919]:=
     v[a] Scalar[v[a] v[-a]]
```

Out[919]=

$\text{Scalar}[v_a\, v^a]\, v^a$

```
In[920]:=
     Validate[%]
```

Out[920]=

$\text{Scalar}[v_a\, v^a]\, v^a$

By default the function `Simplification` does not add new `Scalar` heads, because that takes some time for large expressions:

```
In[921]:=
     v[b] v[-b] / Scalar[v[a] v[-a]] // Simplification
```

Out[921]=

$$\frac{v_b\, v^b}{\text{Scalar}[v_a\, v^a]}$$

You can force it using `PutScalar`:

```
In[922]:=
     PutScalar[%]
```

Out[922]=

$$\frac{\text{Scalar}[v_b\, v^b]}{\text{Scalar}[v_a\, v^a]}$$

```
In[923]:=
     Simplification[%]
```

Out[923]=

1

Two other functions to manipulate Scalar expressions are `BreakScalars` and `NoScalar`:

```
In[924]:=
     Scalar[v[a] v[-a] v[b] v[-b]]
```

Out[924]=

$\text{Scalar}[v_a\, v^a\, v_b\, v^b]$

```
In[925]:=
     % // BreakScalars
```

Out[925]=

$\text{Scalar}[v_a\, v^a]\, \text{Scalar}[v_b\, v^b]$

```
In[926]:=
     % // Simplification
```

Out[926]=

$\text{Scalar}[v_a\, v^a]^2$

```
In[927]:=
      % // NoScalar
```

Out[927]=

$v_a v^a v_b v^b$

`DefScalarFunction`	Define an inert head
`UndefScalarFunction`	Undefine an inert head
`$ScalarFunctions`	List of defined inert heads

Definition of a scalar function.

A scalar function is one that only accepts scalar arguments and returns scalars. Some of them are

```
In[928]:=
      $ScalarFunctions

Out[928]=
      {Exp, Log, Sin, Cos, Tan, Csc, Sec, Cot, Power, Factorial}
```

and we can define new scalar functions:

We define a scalar function SF:

```
In[929]:=
      DefScalarFunction[SF]

         ** DefScalarFunction: Defining scalar function SF.

In[930]:=
      ScalarQ[SF[v[a] v[-a]]]

Out[930]=
      True

In[931]:=
      $CovDs

Out[931]=
      {PD, Cd, ICD, CD}

In[932]:=
      CD[-a][1 / SF[Scalar[v[a] v[-a]]]]
```

Out[932]=

$$-\frac{(\nabla_a \mathrm{Scalar}[v_a\ v^a])\ \mathrm{SF}'[\mathrm{Scalar}[v_a\ v^a]]}{\mathrm{SF}[\mathrm{Scalar}[v_a\ v^a]]^2}$$

9.4. Inert heads

Sometimes we need to wrap tensors with some head. This is an extremely general concept and it is prepared to play the role of an arbitary head with the only property of being "transparent" to the symmetry and canonicalization routines. `Dagger` and `ERROR` are both inert heads.

A prototypical example could be traceless, transverse tensors: We define an inert head called `TTPart`:

```
In[933]:=
      DefInertHead[TTPart]

          ** DefInertHead: Defining inert head TTPart.

In[934]:=
      TTPart[U[-b, -c, -a]] // ToCanonical

Out[934]=
      TTPart[U_abc]

In[935]:=
      UpVectorQ[TTPart[v[a]]]

Out[935]=
      True

In[936]:=
      UndefInertHead[TTPart]

          ** UndefInertHead: Undefined inert head TTPart
```

With help from Guillaume Faye, this concept has been improved in version 0.9.2 of `xTensor'` in three different ways:

1) There is an option `LinearQ` to make the inert head linear with respect to constants. The default value is `False`.

```
In[937]:=
      DefInertHead[lhead, LinearQ → True]

          ** DefInertHead: Defining inert head lhead.

In[938]:=
      LinearQ[lhead]

Out[938]=
      True

In[939]:=
      lhead[3 T[a, b] + S[a, b]]

Out[939]=
      lhead[S^ab] + 3 lhead[T^ab]
```

2) There is an option `ContractThrough` to specify a list of metrics, projectors and/or the delta tensor which can be contracted through the inert head. In this example only the delta tensor can be contracted:

```
In[940]:=
      DefInertHead[chead, ContractThrough → {delta}]

          ** DefInertHead: Defining inert head chead.

In[941]:=
      DefInertHead[mhead, ContractThrough → {delta, metricg}]

          ** DefInertHead: Defining inert head mhead.
```

```
In[942]:=
      ?mhead

         Global`mhead

         mhead /: ContractThroughQ[mhead, delta] = True

         mhead /: ContractThroughQ[mhead, metricg] = True

         InertHeadQ[mhead] ^= True

         Info[mhead] ^= {inert head, }

         LinearQ[mhead] ^= False

         PrintAs[mhead] ^= mhead
```

Compare now these cases:

```
In[943]:=
      delta[a, -b] {lhead[T[b]], chead[T[b]], mhead[T[b]]}
```

Out[943]=

$\{\delta^a{}_b\ \mathrm{lhead}[T^b],\ \mathrm{chead}[T^a],\ \mathrm{mhead}[T^a]\}$

```
In[944]:=
      ContractMetric[metricg[-a, -b] {lhead[T[b]], chead[T[b]], mhead[T[b]]}]
```

Out[944]=

$\{\mathrm{lhead}[T^b]\ g_{ab},\ \mathrm{chead}[T^b]\ g_{ab},\ \mathrm{mhead}[T_a]\}$

3) It is possible to have additional arguments, including arguments with synchronized lists of indices. This does not even have to be specified as an argument. The indices must be surrounded with the head `IndexList`. The system keeps a one to one relation between the indices in the tensorial expression and the indices in the lists of arguments. This correspondence is based on the slots: for example in the following expression the IndexList marks the {3, 2} slots of the tensor `U`, and this is kept through the use of the `xTensor`` functions:

```
In[945]:=
      expr = U[-a, -b, d] mhead[U[b, c, a], {a, c}, IndexList[a, c]];

In[946]:=
      ReplaceIndex[Evaluate[expr], {c → f}] // InputForm

Out[946]//InputForm=
     mhead[U[b, f, a], {a, c}, IndexList[a, f]]*U[-a, -b, d]

In[947]:=
      ToCanonical[expr] // InputForm

Out[947]//InputForm=
     mhead[U[c, -a, -b], {a, c}, IndexList[-b, -a]]*U[d, a, b]

In[948]:=
      metricg[-f, -c] expr // ContractMetric // InputForm

Out[948]//InputForm=
     mhead[U[b, -f, a], {a, c}, IndexList[a, -f]]*U[-a, -b, d]
```

Perhaps there should be something more general than slot−syncronization, but we do not know of a complete set of possibilities, and no other case has been asked for until now.

`DefInertHead`	Define an inert head
`UndefInertHead`	Undefine an inert head
`$InertHeads`	List of defined inert heads
`LinearQ`	Boolean option to specify whether an inert head is linear or not
`LinearQ`	Detect whether an inert head is linear
`ContractThrough`	Option to specify which metrics, projectors or delta can be contracted through the inert head
`ContractThroughQ`	Detect whether a metric, projector or delta can be contracted through an inert head

Definition of an inert head.

9.5. Monomials

A monomial is defined as a product of tensors such that it cannot be further reduced into products with different dum−mies. The function `BreakInMonomials` decomposes an arbitrary expression in monomials with (inert) head `Monomial`.

We break this expressions in monomials. Note that scalars are left outside:

```
In[949]:=
      7 r[]^2 T[a, b, c] U[-b, -c, -e] v[e] T[f, g, h] v[-g] v[-h]
```

Out[949]=

$7\, r^2\, T^{abc}\, T^{fgh}\, U_{bce}\, v^e\, v_g\, v_h$

```
In[950]:=
      BreakInMonomials[%]
```

Out[950]=

$7\,\mathrm{Monomial}[T^{abc}\, U_{bce}\, v^e]\,\mathrm{Monomial}[T^{fgh}\, v_g\, v_h]\, r^2$

Now for example each monomial could be canonicalized independently:

```
In[951]:=
      % /. expr_Monomial :→ ToCanonical[expr]
```

Out[951]=

$7\,\mathrm{Monomial}[T^{abc}\, U_{bce}\, v^e]\,\mathrm{Monomial}[T^{fgh}\, v_g\, v_h]\, r^2$

`Monomial`	Head for a monomial expression
`BreakInMonomials`	Separate a term in monomials

Monomials.

9.6. Symmetrization of an expression

`ImposeSymmetry`	Symmetrize an expression as given by a permutations group
`Symmetrize`	Symmetrize a set of indices
`Antisymmetrize`	Antisymmetrize a set of indices
`PairSymmetrize`	Symmetrize a set of pairs of indices
`PairAntisymmetrize`	Antisymmetrize a set of pairs of indices

Symmetrization

Symmetrization of an expression under a given group of permutations. Numbers on the third argument refer to positions in the second argument of `ImposeSymmetry`. Note that `ToCanonical` is always applied after the symmetrization functions in order to take into account the antisymmetry of the tensor `U`. It is not automatically implemented into `ImposeSymmetry` because for large numbers of indices it would take ages to canonicalize all terms!

```
In[952]:=
      ImposeSymmetry[U[-a, -b, -c], {-a, -b}, Symmetric[{1, 2}]] // ToCanonical

Out[952]=
      0
```

```
In[953]:=
      ImposeSymmetry[U[-a, -b, -c], {-a, -b}, Antisymmetric[{1, 2}]] // ToCanonical

Out[953]=
```

U_{abc}

```
In[954]:=
      ImposeSymmetry[U[-a, -b, -c] U[-d, -e, -f],
        {-a, -d}, Antisymmetric[{1, 2}]] // ToCanonical

Out[954]=
```

$$-\frac{1}{2} U_{aef} U_{bcd} + \frac{1}{2} U_{abc} U_{def}$$

```
In[955]:=
      Symmetrize[U[-a, -b, -c] U[-d, -e, -f], {-a, -d}] // ToCanonical

Out[955]=
```

$$\frac{1}{2} U_{aef} U_{bcd} + \frac{1}{2} U_{abc} U_{def}$$

```
In[956]:=
      Symmetrize[U[-a, -b, -c] U[-d, -e, -f], {-e, -d}] // ToCanonical

Out[956]=
      0
```

```
In[957]:=
      Antisymmetrize[U[-a, -b, -c] U[-d, -e, -f], {-a, -d}] // ToCanonical

Out[957]=
```

$$-\frac{1}{2} U_{aef} U_{bcd} + \frac{1}{2} U_{abc} U_{def}$$

In[958]:=
```
Antisymmetrize[U[-a, -b, -c] U[-d, -e, -f], {-e, -d}] // ToCanonical
```
Out[958]=
$$U_{abc}\,U_{def}$$

In[959]:=
```
ps = PairSymmetrize[U[-a, -b, -c] U[-d, -e, -f],
   {{-a, -b}, {-c, -d}, {-e, -f}}] // ToCanonical
```
Out[959]=
$$\frac{1}{6}\,U_{aef}\,U_{bcd}+\frac{1}{6}\,U_{acd}\,U_{bef}+\frac{1}{6}\,U_{abf}\,U_{cde}+\frac{1}{6}\,U_{abe}\,U_{cdf}+\frac{1}{6}\,U_{abd}\,U_{cef}+\frac{1}{6}\,U_{abc}\,U_{def}$$

In[960]:=
```
pa = PairAntisymmetrize[U[-a, -b, -c] U[-d, -e, -f],
   {{-a, -b}, {-c, -d}, {-e, -f}}] // ToCanonical
```
Out[960]=
$$\frac{1}{6}\,U_{aef}\,U_{bcd}-\frac{1}{6}\,U_{acd}\,U_{bef}+\frac{1}{6}\,U_{abf}\,U_{cde}-\frac{1}{6}\,U_{abe}\,U_{cdf}-\frac{1}{6}\,U_{abd}\,U_{cef}+\frac{1}{6}\,U_{abc}\,U_{def}$$

In[961]:=
```
ps - ReplaceIndex[Evaluate[ps], {-a → -c, -b → -d, -c → -a, -d → -b}] // Simplification
```
Out[961]=
```
0
```

In[962]:=
```
pa + ReplaceIndex[Evaluate[pa], {-a → -c, -b → -d, -c → -a, -d → -b}] // Simplification
```
Out[962]=
```
0
```

9.7. Symmetric Trace−Free part

The command STFPart returns the symmetric, trace−free part of an expression with only free indices. In order for a symmetric object to have a trace we need to use a metric.

This is the STF part of a 3−index tensor without symmetry, and with respect to the metricg:

In[963]:=
```
STFPart[T[a, b, c], metricg]
```
Out[963]=
$$\frac{1}{6}\,(T^{abc}+T^{acb}+T^{bac}+T^{bca}+T^{cab}+T^{cba})+\frac{1}{10}$$
$$\Big(-\frac{1}{6}\,g^{bc}\,g_{de}\,(T^{ade}+T^{aed}+T^{dae}+T^{dea}+T^{ead}+T^{eda})-\frac{1}{6}\,g^{cb}\,g_{de}\,(T^{ade}+T^{aed}+T^{dae}+T^{dea}+T^{ead}+T^{ed}$$
$$\frac{1}{6}\,g^{ac}\,g_{de}\,(T^{bde}+T^{bed}+T^{dbe}+T^{deb}+T^{ebd}+T^{edb})-\frac{1}{6}\,g^{ca}\,g_{de}\,(T^{bde}+T^{bed}+T^{dbe}+T^{deb}+T^{ebd}+T^{ed}$$
$$\frac{1}{6}\,g^{ab}\,g_{de}\,(T^{cde}+T^{ced}+T^{dce}+T^{dec}+T^{ecd}+T^{edc})-\frac{1}{6}\,g^{ba}\,g_{de}\,(T^{cde}+T^{ced}+T^{dce}+T^{dec}+T^{ecd}+T^{ed}$$

The result is not simplified at all. We must ask for explicit metric contraction and simplification:

```
In[964]:=
      % // ContractMetric // Simplification
```

Out[964]=

$$\frac{1}{30}\left(5\,T^{abc}+5\,T^{acb}-2\,g^{bc}\,T^{ad}{}_{d}+5\,T^{bac}+5\,T^{bca}-2\,g^{ac}\,T^{bd}{}_{d}+5\,T^{cab}+5\,T^{cba}-2\,g^{ab}\,T^{cd}{}_{d}-2\,g^{bc}\,T^{da}{}_{d}-2\,g^{ac}\,T^{db}{}_{d}-2\,g^{ab}\,T^{dc}{}_{d}-2\,g^{bc}\,T^{d}{}_{d}{}^{a}-2\,g^{ac}\,T^{d}{}_{d}{}^{b}-2\,g^{ab}\,T^{d}{}_{d}{}^{c}\right)$$

9.8. Mapping at specified positions

We use the package `ExpressionManipulation`` (written by David Park & Ted Ersek) to have full control on the positions of an expression

Suppose we have an expression like this, which is clearly `0` because `U` is antisymmetric and `S` is symmetric:

```
In[965]:=
      expr = U[a, b, c] v[-a] v[-b] v[-c] +
        U[a, b, c] v[-a] S[-b, -c] + TT[-b, -c] v[b] v[c] - TT[-d, -e] v[d] v[e]
```

Out[965]=

$$S_{bc}\,U^{abc}\,v_a+U^{abc}\,v_a\,v_b\,v_c+\tau_{bc}\,v^b\,v^c-\tau_{de}\,v^d\,v^e$$

```
In[966]:=
      Simplification[expr]

Out[966]=
      0
```

Each object in the expression can be identified in a tree−like form by its position. In this case there are four positions at level 1:

```
In[967]:=
      expr // ColorTerms
```

Out[967]=

$$\boxed{\{1\}}\;S_{bc}\,U^{abc}\,v_a+\boxed{\{2\}}\;U^{abc}\,v_a\,v_b\,v_c+\boxed{\{3\}}\;\tau_{bc}\,v^b\,v^c+\boxed{\{4\}}\;-\tau_{de}\,v^d\,v^e$$

We can evaluate a given function at one or several positions:

```
In[968]:=
      MapAt[Simplification, expr, {2}]
```

Out[968]=

$$S_{bc}\,U^{abc}\,v_a+\tau_{bc}\,v^b\,v^c-\tau_{de}\,v^d\,v^e$$

```
In[969]:=
      MapAt[Simplification, expr, {{1}, {2}}]
```

Out[969]=

$$\tau_{bc}\,v^b\,v^c-\tau_{de}\,v^d\,v^e$$

In order to map a function over several terms simultaneously we need to make use of the concept of `ExtendedPosition` (also from the `ExpressionManipulation`` package) and the new function `NewMapAt`. See notes for `ExtendedPosition` and `eP`.

```
In[970]:=
      MapAt[f, expr, {{3}, {4}}]
```

Out[970]=

$f[\tau_{bc} v^b v^c] + f[-\tau_{de} v^d v^e] + S_{bc} U^{abc} v_a + U^{abc} v_a v_b v_c$

```
In[971]:=
      NewMapAt[f, expr, eP[{}, {3, 4}]]
```

Out[971]=

$f[\tau_{bc} v^b v^c - \tau_{de} v^d v^e] + S_{bc} U^{abc} v_a + U^{abc} v_a v_b v_c$

We can also localize a given pattern:

```
In[972]:=
      expr // ColorPositionsOfPattern[U[__]]
```

Out[972]=

$S_{bc} \left(\boxed{\{1, 2\}}\ U^{abc}\right) v_a + \left(\boxed{\{2, 1\}}\ U^{abc}\right) v_a v_b v_c + \tau_{bc} v^b v^c - \tau_{de} v^d v^e$

or map a function onto the occurrences of that pattern:

```
In[973]:=
      positions = Position[expr, U[__]]
```

Out[973]=

```
      {{1, 2}, {2, 1}}
```

```
In[974]:=
      MapAt[f, expr, positions]
```

Out[974]=

$f[U^{abc}] S_{bc} v_a + f[U^{abc}] v_a v_b v_c + \tau_{bc} v^b v^c - \tau_{de} v^d v^e$

In case the expression is held, we can use `EvaluateAt` (also from the package `ExpressionManipulation``) instead of `MapAt`:

```
In[975]:=
      Hold[U[a, b, c] v[-a] v[-b] v[-c] + U[a, b, c] v[-a] S[-b, -c] +
        TT[-b, -c] v[b] v[c] - TT[-d, -e] v[d] v[e] + 1 + 2]
```

Out[975]=

$\mathrm{Hold}[U^{abc} v_a v_b v_c + U^{abc} v_a S_{bc} + \tau_{bc} v^b v^c - \tau_{de} v^d v^e + 1 + 2]$

```
In[976]:=
      MapAt[Simplification, %, {1, 1}]
```

Out[976]=

$\mathrm{Hold}[\mathrm{Simplification}[U^{abc} v_a v_b v_c] + U^{abc} v_a S_{bc} + \tau_{bc} v^b v^c - \tau_{de} v^d v^e + 1 + 2]$

```
In[977]:=
      EvaluateAt[{1, 1}, Simplification][%%]
```

Out[977]=

$$\text{Hold}[0 + U^{abc}\, v_a\, S_{bc} + \tau_{bc}\, v^b\, v^c - \tau_{de}\, v^d\, v^e + 1 + 2]$$

9.9. Coefficients

Given an expression, sometimes we need to get the (possibly indexed) coefficient multiplying a given tensor. When the objects have symmetries the coefficient might be more complicated than expected.

`IndexCoefficient`	Find the coefficient of an indexed expression

Coefficients of indexed expressions.

The basic relation is this:

```
In[978]:=
      IndexCoefficient[T[a, b], T[c, d]]
```

Out[978]=

$$\delta^a{}_c\, \delta^b{}_d$$

Metrics and symmetries must be taken into account:

```
In[979]:=
      IndexCoefficient[T[-a, -b], T[c, d]]
```

Out[979]=

$$g_{ac}\, g_{bd}$$

```
In[980]:=
      IndexCoefficient[metricg[-a, -b], metricg[-c, -d]]
```

Out[980]=

$$\frac{1}{2}\left(\delta_a{}^d\, \delta_b{}^c + \delta_a{}^c\, \delta_b{}^d\right)$$

```
In[981]:=
      IndexCoefficient[T[a, b] v[-a] v[-b], v[-c]] // Simplification
```

Out[981]=

$$\frac{1}{2}\left(T^{ac} + T^{ca}\right) v_a$$

```
In[982]:=
      IndexCoefficient[T[a, b] v[-a] v[-b], v[-c] v[-d]]
```

Out[982]=

$$\frac{T^{cd}}{2} + \frac{T^{dc}}{2}$$

Imitating Coefficient, dependencies must be explicit:

```
In[983]:=
      IndexCoefficient[T[a, b], metricg[a, b]]

Out[983]=
      0
```

Frequently we need further manipulation of the result:

```
In[984]:=
      IndexCoefficient[T[a, b] v[-b] v[c] + T[d, -d] v[a] v[c], v[e]]

Out[984]=
```

$$\frac{1}{2}\left(\delta^c{}_e\, T^{ab}\, v_b + g_{be}\, T^{ab}\, v^c\right) + \frac{1}{2}\left(\delta^c{}_e\, T^b{}_b\, v^a + \delta^a{}_e\, T^b{}_b\, v^c\right)$$

```
In[985]:=
      % // ContractMetric // Simplification

Out[985]=
```

$$\frac{1}{2}\left(\delta^c{}_e\,\left(T^b{}_b\, v^a + T^{ab}\, v_b\right) + \left(T^a{}_e + \delta^a{}_e\, T^b{}_b\right) v^c\right)$$

A closely related function is `IndexCollect`, which acts as a simple recursive driver for `IndexCoefficient`. This pair has been designed to follow closely the *Mathematica* pair `Collect / Coefficient`.

`IndexCollect`	Collect terms in an indexed expression

Suppose an expression like this:

```
In[986]:=
      expr = T[a, b, -c] v[c] + U[a, b, c, d] v[-c] v[-d] + metricg[a, b]

Out[986]=
```

$$g^{ab} + T^{ab}{}_c\, v^c + U^{abcd}\, v_c\, v_d$$

```
In[987]:=
      IndexCollect[expr, {v[d], v[e]}, Simplification]

Out[987]=
```

$$g^{ab} + v^d\left(T^{ab}{}_d + \frac{1}{2}\left(U^{ab}{}_{de} + U^{ab}{}_{ed}\right) v^e\right)$$

9.10. Tensor equations

It is not simple to deal with tensor equations because there are many things to worry about simultaneously. Currently `xTensor‘` only works with linear equations where the x tensor is not contracted. The concept of equality (==) has been generalized to include tensor equalities.

From the point of view of *Mathematica* these two expressions are not related:

```
In[988]:=
      T[a, b] v[-b] == T[a, c] v[-c]
```

Out[988]=

$T^{ab}\, v_b == T^{ac}\, v_c$

However `xTensor'` recognizes them as equal:

```
In[989]:=
      Simplification[%]

Out[989]=
      True
```

These two are clearly different:

```
In[990]:=
      Simplification[T[a, b] v[-b] == T[-a, c] v[-c]]

Out[990]=
      False
```

Solving equations. The answer is a tensor rule. Options to `IndexSolve` are actually options to `MakeRule`:

```
In[991]:=
      rule = IndexSolve[v[a] v[-a] T[b, c, d] == v[b] v[c] v[a] S[-a, d],
        T[b, c, d], MetricOn → {b}, ContractMetrics → True]
```

Out[991]=

$\left\{\text{HoldPattern}\left[T^{\underline{b}\,\underline{c}\,\underline{d}}\right] :\!\to \text{Module}\left[\{a\},\ \frac{S^{d}{}_{a}\, v^b\, v^c\, v^a}{\text{Scalar}[v_a\, v^a]}\right]\right\}$

```
In[992]:=
      {T[a, b, c], T[-a, b, c], T[a, -b, c], T[-a, -b, c]} /. rule
```

Out[992]=

$\left\{\frac{S^{c}{}_{d}\, v^a\, v^b\, v^d}{\text{Scalar}[v_a\, v^a]},\ \frac{S^{c}{}_{d}\, v_a\, v^b\, v^d}{\text{Scalar}[v_a\, v^a]},\ T^{a}{}_{b}{}^{c},\ T_{ab}{}^{c}\right\}$

■ 10. Final comments

10.1. ToDo list. Version

There are many things that `xTensor'` cannot do yet. Here there is a list with some of them:

− Differential forms

− Spinors (In progress, in collaboration with A. García−Parrado)

− Grasmann variables

− Product of manifolds with non−zero non−diagonal boxes in the metric (In progress, based on unpublished notes by G. Faye)

Note: For further information about `xTensor‘`, and to be kept informed about new releases, you may contact the author electronically at jmm@iem.cfmac.csic.es. This is xTensorDoc.nb, the docfile of `xTensor‘`, currently in version 0.9.5.

10.2. Frequently Asked Questions

1. Why is delta contraction automatic? I want to have full control on every single detail of the process.

Why is metric contraction not automatic? It is boring to ask continuously for trivial tasks.

Those are two opposite approaches to the issue of automatic simplification. Too little automatic simplification makes the computa− tions hard and slow, while too much automatic simplification makes the process too restrictive and the result seems just magic. Following *Mathematica*, we have tried to automatize all those things which are commonly required, like contraction of any tensor with delta, or the expansion of derivatives of products using the Leibnitz rule. However, most of the processes are not automatic: simplification, metric contraction, commutation of derivatives, etc.

There are some processes for which the user can choose whether they are automatic or not: look for the pairs commandStart / commandStop.

2. I want to define several tensors at the same time. Why not allowing this syntax?

```
DefTensor[{T[a], S[b,c], R[−a,−b,−c,−d}, M]
```

We do not include new definitions which do not save time or space with respect to their *Mathematica* counterparts. This helps learning *Mathematica*, and save space in `xTensor‘`. In that particular example we recommend to use

```
Map[ DefTensor[#, M]&, {T[a], S[b,c], R[−a,−b,−c,−d]} ]
```

or simply threading

```
Thread[ DefTensor[{T[a], S[b,c], R[−a,−b,−c,−d}, M] ]
```

3. In *MathTensor* many commands have an additional argument to map them only to a certain part of an expression. Why don’t you have the same thing?

The answer is identical to that of question 2. Use the syntax `MapAt[f, expr, n]` to map the function `f` on the `n`−th term of `expr`.

10.3. Statistics

Collection of public symbols in `xTensor‘`:

```
In[993]:=
     Names["xAct`xTensor`*"]

Out[993]=
     {ABIndexQ, AbstractIndex, AbstractIndexQ, Acceleration, AChristoffel, AddIndices,
      AIndex, AIndexQ, AllowUpperDerivatives, Antihermitian, Antisymmetrize,
      AnyDependencies, AnyIndices, AutomaticRules, BaseOfVBundle, Basis, BasisQ,
      BCIndexQ, BIndex, BIndexQ, Blocked, BlockedQ, Bracket, BracketToCovD,
      BreakChristoffel, BreakInMonomials, BreakScalars, CDIndexQ, ChangeCovD,
      ChangeCurvature, ChangeFreeIndices, ChangeIndex, ChangeTorsion, Chart, ChartQ,
      CheckZeroDerivative, CheckZeroDerivativeStart, CheckZeroDerivativeStop,
      Christoffel, ChristoffelToGradMetric, ChristoffelToMetric, CIndex, CIndexForm,
      CIndexQ, CircleDot, ColorPositionsOfPattern, ColorTerms, CommuteCovDs,
      CommutePDs, ConstantMetric, ConstantQ, ConstantSymbol, ConstantSymbolQ,
      ContractCurvature, ContractDir, ContractMetric, ContractMetrics, ContractThrough,
      ContractThroughQ, CovD, CovDOfMetric, CovDQ, CovDToChristoffel, Curvature,
      CurvatureQ, CurvatureRelations, Dagger, DaggerIndex, DaggerQ, DefAbstractIndex,
      DefConstantSymbol, DefCovD, DefInertHead, DefManifold, DefMetric, DefParameter,
      DefProductMetric, DefScalarFunction, DefTensor, DefVBundle, delta, DependenciesOf,
      DependenciesOfTensor, DimOfManifold, DimOfVBundle, DIndex, DIndexQ, Dir,
      Disclaimer, DisjointManifoldsQ, DisorderedPairQ, DoubleRightTee, Down, DownIndex,
      DownIndexQ, DownVectorQ, Dummy, DummyIn, EIndexQ, Einstein, EinsteinToRicci,
      epsilon, EqualExpressionsQ, ERROR, ExpandChristoffel, ExpandGdelta,
      ExpandProductMetric, ExtendedFrom, ExtrinsicK, ExtrinsicKToGradNormal,
      FindAllOfType, FindBlockedIndices, FindDummyIndices, FindFreeIndices, FindIndices,
      FirstDerQ, FlatMetric, FlatMetricQ, ForceSymmetries, Free, FRiemann, FrobeniusQ,
      FromMetric, Gdelta, GetIndicesOfVBundle, GIndexQ, GiveOutputString, GivePerm,
      GiveSymbol, GradMetricToChristoffel, GradNormalToExtrinsicK, Hermitian, HostsOf,
      Imaginary, ImposeSymmetry, IndexCoefficient, IndexCollect, IndexForm, IndexList,
      IndexOrderedQ, IndexRange, IndexRule, IndexRuleDelayed, IndexSet, IndexSetDelayed,
      IndexSolve, IndexSort, IndicesOf, IndicesOfVBundle, InducedDecomposition,
      InducedFrom, InertHead, InertHeadQ, Info, Inv, IsIndexOf, Labels, LI, LieD,
      LieDToCovD, LIndex, LIndexQ, LinearQ, MakeRule, Manifold, ManifoldOfCovD,
      ManifoldQ, ManifoldsOf, Master, MasterOf, Metric, MetricEndowedQ, MetricOfCovD,
      MetricOn, MetricQ, MetricScalar, MetricsOfVBundle, MetricToProjector, Monomial,
      NewIndexIn, NoScalar, Notation, NumberOfArguments, OrthogonalTo, OverDerivatives,
      OverDot, PairAntisymmetrize, PairQ, PairSymmetrize, ParamD, Parameter, ParameterQ,
      ParametersOf, PatternIndex, PatternIndices, PD, PermuteIndices, PIndex, PIndexQ,
      PrintAs, ProjectDerivative, ProjectedWith, Projector, ProjectorToMetric,
      ProjectWith, ProtectNewSymbol, PutScalar, RemoveIndices, ReplaceDummies,
      ReplaceIndex, Ricci, RicciScalar, RicciToEinstein, RicciToTFRicci, Riemann,
      RiemannToChristoffel, RiemannToWeyl, RightTeeArrow, SameDummies, Scalar,
      ScalarFunction, ScalarFunctionQ, ScalarQ, ScreenDollarIndices, SeparateDir,
      SeparateMetric, ServantsOf, SetCharacters, SetIndexSortPriorities, SetOrthogonal,
      SetProjected, SignatureOfMetric, SignDetOfMetric, Simplification, SlotsOfTensor,
      SortCovDs, SortCovDsStart, SortCovDsStop, SplitIndex, STFPart, SubdummiesIn,
      SubmanifoldQ, SubmanifoldsOfManifold, SubvbundleQ, SubvbundlesOfVBundle,
      SymbolOfCovD, Symmetrize, Symmetry, SymmetryGroupOfTensor, SymmetryOf,
      SymmetryTableauxOfTensor, Tangent, TangentBundleOfManifold, Tensor, TensorID,
      TestIndices, TFRicci, TFRicciToRicci, ToCanonical, Torsion, TorsionQ,
      TorsionToChristoffel, TraceDummy, TraceProductDummy, TransposeDagger,
      Undef, UndefAbstractIndex, UndefConstantSymbol, UndefCovD, UndefInertHead,
      UndefManifold, UndefMetric, UndefParameter, UndefScalarFunction, UndefTensor,
      UndefVBundle, Up, UpIndex, UpIndexQ, UpVectorQ, UseMetricOnVBundle, UseSymmetries,
      Validate, ValidateSymbolInSession, VanishingQ, VarD, VBundle, VBundleOfIndex,
      VBundleOfMetric, VBundleQ, VBundlesOfCovD, VectorOfInducedMetric, VisitorsOf,
      WeightedWithBasis, WeightOf, WeightOfTensor, Weyl, WeylToRiemann, xSort,
      xTensorFormStart, xTensorFormStop, xTensorQ, Zero, $AbstractIndices,
      $AccelerationSign, $Bases, $Charts, $CheckZeroDerivativeVerbose, $CIndexForm,
      $CommuteCovDsOnScalars, $CommuteFreeIndices, $ComputeNewDummies, $ConstantSymbols,
      $CovDFormat, $CovDs, $epsilonSign, $ExtrinsicKSign, $FindIndicesAcceptedHeads,
      $InertHeads, $Manifolds, $Metrics, $MixedDers, $Parameters, $ProductManifolds,
      $ProductMetrics, $ProtectNewSymbols, $ReadingVerbose, $RicciSign,
      $RiemannSign, $Rules, $ScalarFunctions, $SumVBundles, $Tensors, $TorsionSign,
      $TraceDummyVerbose, $VBundles, $Version, $xPermVersionExpected}
```

```
In[994]:=
     Length[%]

Out[994]=
     338

In[995]:=
     TimeUsed[]

Out[995]=
     534.165

In[996]:=
     MemoryInUse[] / 1000 / 1024 // N

Out[996]=
     437.512

In[997]:=
     MaxMemoryUsed[] / 1000 / 1024 // N

Out[997]=
     559.742
```