

$G_{ab} = 0 \rightarrow$ **Cactus Thorns via CA**

SH & Christiane Lechner

22/05/2002

aims

- derive 3+1 - split of complicated systems (i.e. CFE, . . .)

$G_{ab} = 0 \rightarrow$ **Cactus Thorns via CA**

SH & Christiane Lechner

22/05/2002

aims

- derive 3+1 - split of complicated systems (i.e. CFE, . . .)
- manipulation of evolution equations (i.e. change variables, add constraints)

$G_{ab} = 0 \rightarrow$ **Cactus Thorns via CA**

SH & Christiane Lechner

22/05/2002

aims

- derive 3+1 - split of complicated systems (i.e. CFE, . . .)
- manipulation of evolution equations (i.e. change variables, add constraints)
- derive constraint propagation equations and other associated systems, e.g. linearized

$G_{ab} = 0 \rightarrow$ Cactus Thorns via CA

SH & Christiane Lechner

22/05/2002

aims

- derive 3+1 - split of complicated systems (i.e. CFE, . . .)
- manipulation of evolution equations (i.e. change variables, add constraints)
- derive constraint propagation equations and other associated systems, e.g. linearized
- automatic generation of code (complete thorns and thorns consistently sharing information)
. . . CFE thorn ScriwalkerMoL has a very long `interface.ccl`

main design decisions

- do computations as transparently as possible

→ use abstract index tensor manipulation package: abstract index on input **and** output throughout

convert to components only at the end when producing code

main design decisions

- do computations as transparently as possible

→ use abstract index tensor manipulation package: abstract index on input and output throughout

convert to components only at the end when producing code

- prepare for sharing with other people: try to be clean, document, willing to share

choice of CA-package:

- Maxima (opensource version of Macsyma)

main design decisions

- do computations as transparently as possible

→ use abstract index tensor manipulation package: abstract index on input and output throughout

convert to components only at the end when producing code

- prepare for sharing with other people: try to be clean, document, willing to share

choice of CA-package:

- Maxima (opensourced version of Macsyma)
- Reduce ??

main design decisions

- do computations as transparently as possible

→ use abstract index tensor manipulation package: abstract index on input **and** output throughout

convert to components only at the end when producing code

- prepare for sharing with other people: try to be clean, document, willing to share

choice of CA-package:

- Maxima (opensourced version of Macsyma)
- Reduce ??
- Maple ?

main design decisions

- do computations as transparently as possible

→ use abstract index tensor manipulation package: abstract index on input **and** output throughout

convert to components only at the end when producing code

- prepare for sharing with other people: try to be clean, document, willing to share

choice of CA-package:

- Maxima (opensourced version of Macsyma)
- Reduce ??
- Maple ?
- Mathematica

main design decisions

- do computations as transparently as possible

→ use abstract index tensor manipulation package: abstract index on input **and** output throughout

convert to components only at the end when producing code

- prepare for sharing with other people: try to be clean, document, willing to share

choice of CA-package:

- Maxima (opensourced version of Macsyma)
- Reduce ??
- Maple ?
- Mathematica
 - Ricci: free!

main design decisions

- do computations as transparently as possible
 - use abstract index tensor manipulation package: abstract index on input and output throughout
 - convert to components only at the end when producing code
- prepare for sharing with other people: try to be clean, document, willing to share

choice of CA-package:

- Maxima (opensourced version of Macsyma)
- Reduce ??
- Maple ?
- Mathematica
 - Ricci: free!
 - **Mathtensor**: commercial, huge! (conversion of tensor expressions to/from Ricci possible)

Tensors in Mathtensor: $h[1a, 1b] \rightarrow h_{ab}$, $CD[Metricg[1a, 1b], 1c] \rightarrow 0$

what's there?

Decompose_3+1_Tools.nb

teach Mathtensor about geometry of 3+1 splits, e.g.

```
RuleUnique[Evaluate@rname, t[la_, lb_]n[ua_], 0, PairQ[la, ua]]
```

what's there?

Decompose_3+1_Tools.nb

teach Mathtensor about geometry of 3+1 splits, e.g.

```
RuleUnique[Evaluate@rname,t[1a_,1b_]n[ua_],0,PairQ[1a,ua]]
```

Initializations_3+1_CFE.nb

call functions from Decompose_3+1_Tools.nb to create spatial objects, declare names of metric, unit normal etc., and corresponding rules

what's there?

Decompose_3+1_Tools.nb

teach Mathtensor about geometry of 3+1 splits, e.g.

```
RuleUnique[Evaluate@rname, t[1a_, 1b_]n[ua_], 0, PairQ[1a, ua]]
```

Initializations_3+1_CFE.nb

call functions from Decompose_3+1_Tools.nb to create spatial objects, declare names of metric, unit normal etc., and corresponding rules

Gauss_Codazzi.nb

Gauss–Codazzi identities for chosen 3+1 variables

what's there?

Decompose_3+1_Tools.nb

teach Mathtensor about geometry of 3+1 splits, e.g.
`RuleUnique[Evaluate@rname, t[1a_, 1b_]n[ua_], 0, PairQ[1a, ua]]`

Initializations_3+1_CFE.nb

call functions from Decompose_3+1_Tools.nb to create spatial objects, declare names of metric, unit normal etc., and corresponding rules

Gauss_Codazzi.nb

Gauss–Codazzi identities for chosen 3+1 variables

3+1-Decomp_CFE.nb

derive 3+1 split of CFE (derived from $G_{ab} = 0$ with Mathtensor)

what's there?

- Decompose_3+1_Tools.nb** teach Mathtensor about geometry of 3+1 splits, e.g.
`RuleUnique[Evaluate@rname, t[la_, lb_]n[ua_], 0, PairQ[la, ua]]`
- Initializations_3+1_CFE.nb** call functions from Decompose_3+1_Tools.nb to create spatial objects, declare names of metric, unit normal etc., and corresponding rules
- Gauss_Codazzi.nb** Gauss–Codazzi identities for chosen 3+1 variables
- 3+1-Decomp_CFE.nb** derive 3+1 split of CFE (derived from $G_{ab} = 0$ with Mathtensor)
- Evol_Eqs_CFE.nb** derive component form of a given system of evolution equations, create MoL thorn with 2nd order centered differences, apply to CFE

what's there?

Decompose_3+1_Tools.nb	teach Mathtensor about geometry of 3+1 splits, e.g. <code>RuleUnique[Evaluate@rname, t[la_, lb_]n[ua_], 0, PairQ[la, ua]]</code>
Initializations_3+1_CFE.nb	call functions from Decompose_3+1_Tools.nb to create spatial objects, declare names of metric, unit normal etc., and corresponding rules
Gauss_Codazzi.nb	Gauss–Codazzi identities for chosen 3+1 variables
3+1-Decomp_CFE.nb	derive 3+1 split of CFE (derived from $G_{ab} = 0$ with Mathtensor)
Evol_Eqs_CFE.nb	derive component form of a given system of evolution equations, create MoL thorn with 2nd order centered differences, apply to CFE
ID_Thorn_CFE.nb	create a trivial initial data thorn for a list of variables, apply to CFE

what's there?

- Decompose_3+1_Tools.nb** teach Mathtensor about geometry of 3+1 splits, e.g.
`RuleUnique[Evaluate@rname, t[1a_, 1b_]n[ua_], 0, PairQ[1a, ua]]`
- Initializations_3+1_CFE.nb** call functions from Decompose_3+1_Tools.nb to create spatial objects, declare names of metric, unit normal etc., and corresponding rules
- Gauss_Codazzi.nb** Gauss–Codazzi identities for chosen 3+1 variables
- 3+1-Decomp_CFE.nb** derive 3+1 split of CFE (derived from $G_{ab} = 0$ with Mathtensor)
- Evol_Eqs_CFE.nb** derive component form of a given system of evolution equations, create MoL thorn with 2nd order centered differences, apply to CFE
- ID_Thorn_CFE.nb** create a trivial initial data thorn for a list of variables, apply to CFE
- code generation from lists of variables and equations does not require Mathtensor!

what's there?

Decompose_3+1_Tools.nb	teach Mathtensor about geometry of 3+1 splits, e.g. <code>RuleUnique[Evaluate@rname, t[1a_, 1b_]n[ua_], 0, PairQ[1a, ua]]</code>
Initializations_3+1_CFE.nb	call functions from Decompose_3+1_Tools.nb to create spatial objects, declare names of metric, unit normal etc., and corresponding rules
Gauss_Codazzi.nb	Gauss–Codazzi identities for chosen 3+1 variables
3+1-Decomp_CFE.nb	derive 3+1 split of CFE (derived from $G_{ab} = 0$ with Mathtensor)
Evol_Eqs_CFE.nb	derive component form of a given system of evolution equations, create MoL thorn with 2nd order centered differences, apply to CFE
ID_Thorn_CFE.nb	create a trivial initial data thorn for a list of variables, apply to CFE

code generation from lists of variables and equations does not require Mathtensor!

also done: 3+1 split of E&M, ADM, constraint propagation of E&M on general background

a few details . . .

- the hardest part: which rules are needed? – first find out how you would do the calculations by hand! – then translate to Mathtensor syntax.

a few details . . .

- the hardest part: which rules are needed? – first find out how you would do the calculations by hand! – then translate to Mathtensor syntax.
- derivation of constraint propagation system is particularly hard

a few details . . .

- the hardest part: which rules are needed? – first find out how you would do the calculations by hand! – then translate to Mathtensor syntax.
- derivation of constraint propagation system is particularly hard
- missing functionality: automatically read off hyperbolicity features, linearization, insertion of particular solutions/backgrounds

a few details . . .

- the hardest part: which rules are needed? – first find out how you would do the calculations by hand! – then translate to Mathtensor syntax.
- derivation of constraint propagation system is particularly hard
- missing functionality: automatically read off hyperbolicity features, linearization, insertion of particular solutions/backgrounds
- code generation scripts try to generate nicely formatted human readable code

a few details . . .

- the hardest part: which rules are needed? – first find out how you would do the calculations by hand! – then translate to Mathtensor syntax.
- derivation of constraint propagation system is particularly hard
- missing functionality: automatically read off hyperbolicity features, linearization, insertion of particular solutions/backgrounds
- code generation scripts try to generate nicely formatted human readable code
- code generation scripts do not assume a particular system of equations, set of variables!

a few details . . .

- the hardest part: which rules are needed? – first find out how you would do the calculations by hand! – then translate to Mathtensor syntax.
- derivation of constraint propagation system is particularly hard
- missing functionality: automatically read off hyperbolicity features, linearization, insertion of particular solutions/backgrounds
- code generation scripts try to generate nicely formatted human readable code
- code generation scripts do not assume a particular system of equations, set of variables!
- not yet done: optimisation!

a few details . . .

- the hardest part: which rules are needed? – first find out how you would do the calculations by hand! – then translate to Mathtensor syntax.
- derivation of constraint propagation system is particularly hard
- missing functionality: automatically read off hyperbolicity features, linearization, insertion of particular solutions/backgrounds
- code generation scripts try to generate nicely formatted human readable code
- code generation scripts do not assume a particular system of equations, set of variables!
- not yet done: optimisation!
- suggested things for other people to try out: code KST system, BSSN derivatives

ok, let's see some code ...