Gaalop Guide

Christian Steinmetz
Patrick Charrier
Dietmar Hildenbrand

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Contents

1 First steps on Gaalop

1.1 Installation

1. Install Maxima (`http://maxima.sourceforge.net/download.html`) optionally. This step is recommended but not necessary.
2. Unzip the downloaded Gaalop binary zip archive (`http://www.gaalop.de/download/gaalop-2-0-1-bin/`)
3. Ensure, you already have installed Java Runtime Environment (JRE - downloadable from `http://www.oracle.com/technetwork/java/javase/downloads/index.html`).
4. Run Gaalop. Under Windows, this is done via the *start.bat* in the root directory. Under Linux, start via *start.sh*.

1.2 Configuration

1.2.1 Maxima

1. Click on the *Configure*-Button
2. If you don't want to use maxima, then you can go to step 3. If you want to use maxima, do the following steps: Select the *GlobalSettings*-Tab. You have to ensure, that
   a) the *maximaCommand* is correct
      i. on Windows, the Maxima-Path depends on the installation directory of Maxima. The command is usually like this: `C:\ProgramFiles(x86)\Maxima\bin\maxima.bat`;
      ii. on Linux, the standard value can be used: `/usr/bin/maxima`
   b) the *optMaxima* checkbox is selected.
3. The configurations are stored when terminating Gaalop in the *gaalop.xml*, so you must configure Gaalop only once.

1.2.2 Algebra

Starting Gaalop for the first time, the conformal geometric algebra (5d) is used by default. However, Gaalop is able to handle arbitrary algebras. It is able to read in built-in and external algebra definition files, see section **??** for details. The stable build-in algebras are: 2d,3d,4d,5d,6d,9d.

If you want to integrate an own algebra, create a directory with the algebra name. Create in this directory 2 files:

- definition.csv
- macros.clu

Insert into the "definition.csv" file the following contents:

- set of all base vectors, starting with the base vector 1 and separated commas. This is baseVectorSet1.
- the basis transformation from baseVectorSet1 to baseVectorSet2 for each element of baseVectorSet2.
- set of all base vectors, starting with the base vector 1 and separated commas. This is baseVectorSet2.
- signature of the algebra in baseVectorSet1
- the basis transformation from baseVectorSet2 to baseVectorSet1 for each element of baseVectorSet1.

Line 2,3,5 can be empty, if no basis transformation exits.

The following shows an example for the conformal geometric algebra (5d):

```
1,e1,e2,e3,einf,e0
ep=0.5*einf-e0,em=0.5*einf+e0
1,e1,e2,e3,ep,em
e1=1,e2=1,e3=1,ep=1,em=-1
```

```
e0=0.5*em-0.5*ep,einf=em+ep
```

Insert into the "macros.clu" file all the macros of your geometric algebra, at least it must contain following macros:

- "createPoint" with the arguments x,y,z when using visualizer. It returns the point representation of a point given by the three coordinates x,y,z.
- "Dual" with the arguments m. This macro dualizes multivector m.

To integrate your geometric algebra in Gaalop, select in the Configuration Panel of Gaalop the tab "Algebra" and type in the directory path of the now created directory.

After a restart of Gaalop, you should be able to select your algebra in the combobox "Algebra to use" at the right side of the Gaalop window.

The *userMacroFilePath* property can be used for automatically integrating a user-specific macro file. These macros are inserted at the beginning of each compiling CLUScript. If this property is empty, no user-specific macro file is used.

1.3 First compiling

1. Press *New File* button to type in your CluScript Code or press *Open File* button to load an existing CLUScript file.
2. Ensure that *de.gaalop.tba.Plugin* is selected in the drop box right next to the *Configure* button.
3. Click on the *Optimize* button to select the CodeGenerator of your choice (usually *To C/C++*).
4. Note, that the GAPP CodeGenerator is usable only, if *de.gaalop.gapp.Plugin* is selected in the drop box.

1.4 Known issues

- If the CLUScript contains a *DefVarsN3* or a similar CLUCalc command, then compilation failures.

2 Further improvements with Gaalop

2.1 Pragmas

Pragma:	output
Syntax:	//#pragma output *variable* $blade_1\ blade_2 \ldots blade_n$
Description:	If a question mark is set at the specified variable, only the specified blade is calculated
Arguments:	*variable*: The variable $blade_n$: The blades, which should be calculated only.
Example:	//#pragma output mv 1.0 e3 e1^e2

Pragma:	onlyEvaluate
Syntax:	//#pragma onlyEvaluate $var_1\ var_2 \ldots var_n$
Description:	If a variable of the specified list is assigned, then the value is not inserted in later uses. If a component is not later used, then it is not calculated. This is the main difference to usage of the question mark, where all blades are calculated.
Arguments:	var_i: The *i*-th variable
Example:	//#pragma onlyEvaluate mv1 mv2;

3 Built-in algebras

This section gives a brief description of the build-in algebras, especially their definition and built-in macros.

3.1 5d

This algebra is also known as *conformal geometric algebra* $G_{4,1}$.

3.1.1 Definition

Built-in basis: $\{e_1, e_2, e_3, e_\infty, e_0\}$
Signature: $e_1^2 = 1, e_2^2 = 1, e_3^2 = 1, e_\infty^2 = 0, e_0^2 = 0$

Founding basis: $\{e_1, e_2, e_3, e_+, e_-\}$
Signature: $e_1^2 = 1, e_2^2 = 1, e_3^2 = 1, e_+^2 = 1, e_-^2 = -1$

Map transformations from Built-in basis to Founding basis:
$e_\infty = e_- + e_+$
$e_0 = \frac{1}{2}(e_- - e_+)$

3.1.2 Built-in macros

Creators

Syntax: VecN3(x,y,z)
Description: Creates a conformal point with the coordinates x, y, z.
This method equals to the method implemented in CLUCalc.
Arguments: x: The x-coordinate
y: The y-coordinate
z: The z-coordinate
Returns: The created conformal point

Syntax: createPoint(x,y,z)
Description: Creates a conformal point with the coordinates x, y, z.
Arguments: x: The x-coordinate
y: The y-coordinate
z: The z-coordinate
Returns: The created conformal point

Syntax: SphereN3(centre, radius)
Description: Creates a sphere from a given centre and a given radius
Arguments: *centre*: The centre (conformal) point of the sphere
radius: The radius of the sphere
Returns: The created sphere

Syntax: SphereN3(centre_x, centre_y, centre_z, radius)
Description: Creates a sphere from given centre coordinates and a given radius
Arguments: x: The x-coordinate
y: The y-coordinate
z: The z-coordinate
radius: The radius of the sphere
Returns: The created sphere

Versors

Syntax: RotorN3(x,y,z,angle)
Description: Creates a rotor, which rotates along an axis (defined by x,y,z) with an angle
Arguments: *x*: The x-component of the axis direction vector
y: The y-component of the axis direction vector
z: The z-component of the axis direction vector
angle: The angle in radians
Returns: The created rotator

Syntax: TranslatorN3(x,y,z)
Description: Creates a translator, which translates along a vector (defined by x,y,z)
Arguments: *x*: The x-component of the translation vector
y: The y-component of the translation vector
z: The z-component of the translation vector
Returns: The created translator

Extractors

Syntax: ExtractFirstPoint(pp)
Description: Extracts the first point of a given point pair
Arguments: *pp*: The point pair
Returns: The first point of the given point pair

Syntax: ExtractSecondPoint(pp)
Description: Extracts the second point of a given point pair
Arguments: *pp*: The point pair
Returns: The second point of the given point pair

Operators

Syntax: Dual(mv)
Description: Dualizes a given multivector
Arguments: *mv*: The multivector to be dualized
Returns: A dualized version of the given multivector

Syntax: Normalize(mv)
Description: Normalizes a given multivector *mv*
Arguments: *mv*: The multivector to be normalized
Returns: A normalized version of *mv*