TopcomInterface

A package to communicate with the software Topcom

2021.08.12

12 August 2021

Martin Bies

Martin Bies
Email: martin.bies@alumni.uni-heidelberg.de
Homepage: https://martinbies.github.io/
Address: Mathematical Institute
University of Oxford
Andrew Wiles Building
Radcliffe Observatory Quarter
Woodstock Road
Oxford OX2 6GG
United Kingdom

Contents

Chapter 1

Introduction

1.1 What is the goal of the TopcomInterface package?

TopcomInterface provides an interface, to communicate with the software Topcom via gap. Thereby, for example, triangulations of toric varieties can be computed directly via gap.

Chapter 2

Functionality of Topcom

2.1 Functions to communicate with Topcom

2.1.1 points2chiro (for IsList, IsList, IsList)

▷ `points2chiro(`*`List1, List2, List3`*`)` (operation)

Returns: String

The first two lists are the input required by topcom. The third is a list of strings, consisting of the options supported by topcom.

2.1.2 points2chiro (for IsList)

▷ `points2chiro(`*`List1`*`)` (operation)

Returns: String

Convenience method of the above with List2 = [], List3 = []

2.1.3 chiro2dual (for IsString, IsList, IsList)

▷ `chiro2dual(`*`String, List2, List3`*`)` (operation)

Returns: String

The first argument is a string encoding the chiro and the second a list encoding an (optional) sample triangulation. The third argument is a list of strings, consisting of the options supported by topcom.

2.1.4 chiro2dual (for IsString)

▷ `chiro2dual(`*`String`*`)` (operation)

Returns: String

Convenience method of the above with List2 = [], List3 = []

2.1.5 chiro2circuits (for IsString, IsList, IsList)

▷ `chiro2circuits(`*`String, List2, List3`*`)` (operation)

Returns: String

The first argument is a string encoding the chiro and the second a list encoding an (optional) sample triangulation. The third argument is a list of strings, consisting of the options supported by topcom.

2.1.6 chiro2circuits (for IsString)

▷ `chiro2circuits(String)` (operation)

Returns: String

Convenience method of the above with List2 = [], List3 = []

2.1.7 chiro2cocircuits (for IsString, IsList, IsList)

▷ `chiro2cocircuits(String, List2, List3)` (operation)

Returns: String

The first argument is a string encoding the chiro and the second a list encoding an (optional) sample triangulation. The third argument is a list of strings, consisting of the options supported by topcom.

2.1.8 chiro2cocircuits (for IsString)

▷ `chiro2cocircuits(String)` (operation)

Returns: String

Convenience method of the above with List2 = [], List3 = []

2.1.9 cocircuits2facets (for IsString, IsList, IsList)

▷ `cocircuits2facets(String, List2, List3)` (operation)

Returns: String

The first argument is a string encoding the chiro and the second a list encoding an (optional) sample triangulation. The third argument is a list of strings, consisting of the options supported by topcom.

2.1.10 cocircuits2facets (for IsString)

▷ `cocircuits2facets(String)` (operation)

Returns: String

Convenience method of the above with List2 = [], List3 = []

2.1.11 points2facets (for IsList, IsList, IsList)

▷ `points2facets(List1, List2, List3)` (operation)

Returns: String

The first two lists are the input required by topcom. The third is a list of strings, consisting of the options supported by topcom.

2.1.12 points2facets (for IsList)

▷ `points2facets(List1)` (operation)

Returns: String

Convenience method of the above with List2 = [], List3 = []

2.1.13 points2nflips (for IsList, IsList, IsList)

▷ `points2nflips(List1, List2, List3)` (operation)

Returns: Integer

The first two lists are the input required by topcom. The third is a list of strings, consisting of the options supported by topcom.

2.1.14 points2nflips (for IsList)

▷ `points2nflips(List1)` (operation)

Returns: Integer

Convenience method of the above with List2 = [], List3 = []

2.1.15 points2flips (for IsList, IsList, IsList)

▷ `points2flips(List1, List2, List3)` (operation)

Returns: String

The first two lists are the input required by topcom. The third is a list of strings, consisting of the options supported by topcom.

2.1.16 points2flips (for IsList)

▷ `points2flips(List1)` (operation)

Returns: String

Convenience method of the above with List2 = [], List3 = []

2.1.17 chiro2placingtriang (for IsString, IsList, IsList)

▷ `chiro2placingtriang(String, List2, List3)` (operation)

Returns: String

The first argument is a string encoding the chiro and the second a list encoding an (optional) sample triangulation. The third argument is a list of strings, consisting of the options supported by topcom.

2.1.18 chiro2placingtriang (for IsString)

▷ `chiro2placingtriang(String)` (operation)

Returns: String

Convenience method of the above with List2 = [], List3 = []

2.1.19 points2placingtriang (for IsList, IsList, IsList)

▷ points2placingtriang(*List1, List2, List3*) (operation)

Returns: List

The first two lists are the input required by topcom. The third is a list of strings, consisting of the options supported by topcom.

2.1.20 points2placingtriang (for IsList)

▷ points2placingtriang(*List1*) (operation)

Returns: List

Convenience method of the above with List2 = [], List3 = []

2.1.21 chiro2finetriang (for IsString, IsList, IsList)

▷ chiro2finetriang(*String, List2, List3*) (operation)

Returns: String

The first argument is a string encoding the chiro and the second a list encoding an (optional) sample triangulation. The third argument is a list of strings, consisting of the options supported by topcom.

2.1.22 chiro2finetriang (for IsString)

▷ chiro2finetriang(*String*) (operation)

Returns: String

Convenience method of the above with List2 = [], List3 = []

2.1.23 points2finetriang (for IsList, IsList, IsList)

▷ points2finetriang(*List1, List2, List3*) (operation)

Returns: List

The first two lists are the input required by topcom. The third is a list of strings, consisting of the options supported by topcom.

2.1.24 points2finetriang (for IsList)

▷ points2finetriang(*List1*) (operation)

Returns: List

Convenience method of the above with List2 = [], List3 = []

2.1.25 chiro2triangs (for IsString, IsList, IsList)

▷ chiro2triangs(*String, List2, List3*) (operation)

Returns: String

The first argument is a string encoding the chiro and the second a list encoding an (optional) sample triangulation. The third argument is a list of strings, consisting of the options supported by topcom.

2.1.26 chiro2triangs (for IsString)

▷ `chiro2triangs(`*`String`*`)` (operation)

Returns: String

Convenience method of the above with List2 = [], List3 = []

2.1.27 points2triangs (for IsList, IsList, IsList)

▷ `points2triangs(`*`List1, List2, List3`*`)` (operation)

Returns: String

The first two lists are the input required by topcom. The third is a list of strings, consisting of the options supported by topcom.

2.1.28 points2triangs (for IsList)

▷ `points2triangs(`*`List1`*`)` (operation)

Returns: String

Convenience method of the above with List2 = [], List3 = []

2.1.29 chiro2ntriangs (for IsString, IsList, IsList)

▷ `chiro2ntriangs(`*`String, List2, List3`*`)` (operation)

Returns: String

The first argument is a string encoding the chiro and the second a list encoding an (optional) sample triangulation. The third argument is a list of strings, consisting of the options supported by topcom.

2.1.30 chiro2ntriangs (for IsString)

▷ `chiro2ntriangs(`*`String`*`)` (operation)

Returns: String

Convenience method of the above with List2 = [], List3 = []

2.1.31 points2ntriangs (for IsList, IsList, IsList)

▷ `points2ntriangs(`*`List1, List2, List3`*`)` (operation)

Returns: List

The first two lists are the input required by topcom. The third is a list of strings, consisting of the options supported by topcom.

2.1.32 points2ntriangs (for IsList)

▷ `points2ntriangs(`*`List1`*`)` (operation)

Returns: List

Convenience method of the above with List2 = [], List3 = []

2.1.33 chiro2finetriangs (for IsString, IsList, IsList)

▷ `chiro2finetriangs(`*`String, List2, List3`*`)` (operation)

Returns: String

The first argument is a string encoding the chiro and the second a list encoding an (optional) sample triangulation. The third argument is a list of strings, consisting of the options supported by topcom.

2.1.34 chiro2finetriangs (for IsString)

▷ `chiro2finetriangs(`*`String`*`)` (operation)

Returns: String

Convenience method of the above with List2 = [], List3 = []

2.1.35 points2finetriangs (for IsList, IsList, IsList)

▷ `points2finetriangs(`*`List1, List2, List3`*`)` (operation)

Returns: List

The first two lists are the input required by topcom. The third is a list of strings, consisting of the options supported by topcom.

2.1.36 points2finetriangs (for IsList)

▷ `points2finetriangs(`*`List1`*`)` (operation)

Returns: List

Convenience method of the above with List2 = [], List3 = []

2.1.37 chiro2nfinetriangs (for IsString, IsList, IsList)

▷ `chiro2nfinetriangs(`*`String, List2, List3`*`)` (operation)

Returns: String

The first argument is a string encoding the chiro and the second a list encoding an (optional) sample triangulation. The third argument is a list of strings, consisting of the options supported by topcom.

2.1.38 chiro2nfinetriangs (for IsString)

▷ `chiro2nfinetriangs(`*`String`*`)` (operation)

Returns: String

Convenience method of the above with List2 = [], List3 = []

2.1.39 points2nfinetriangs (for IsList, IsList, IsList)

▷ `points2nfinetriangs(`*`List1, List2, List3`*`)` (operation)

Returns: List

The first two lists are the input required by topcom. The third is a list of strings, consisting of the options supported by topcom.

2.1.40 points2nfinetriangs (for IsList)

▷ `points2nfinetriangs(`*`List1`*`)` (operation)

Returns: List

Convenience method of the above with List2 = [], List3 = []

2.1.41 chiro2alltriangs (for IsString, IsList, IsList)

▷ `chiro2alltriangs(`*`String, List2, List3`*`)` (operation)

Returns: String

The first argument is a string encoding the chiro and the second a list encoding an (optional) sample triangulation. The third argument is a list of strings, consisting of the options supported by topcom.

2.1.42 chiro2alltriangs (for IsString)

▷ `chiro2alltriangs(`*`String`*`)` (operation)

Returns: String

Convenience method of the above with List2 = [], List3 = []

2.1.43 points2alltriangs (for IsList, IsList, IsList)

▷ `points2alltriangs(`*`List1, List2, List3`*`)` (operation)

Returns: List

The first two lists are the input required by topcom. The third is a list of strings, consisting of the options supported by topcom.

2.1.44 points2alltriangs (for IsList)

▷ `points2alltriangs(`*`List1`*`)` (operation)

Returns: List

Convenience method of the above with List2 = [], List3 = []

2.1.45 chiro2nalltriangs (for IsString, IsList, IsList)

▷ `chiro2nalltriangs(`*`String, List2, List3`*`)` (operation)

Returns: String

The first argument is a string encoding the chiro and the second a list encoding an (optional) sample triangulation. The third argument is a list of strings, consisting of the options supported by topcom.

2.1.46 chiro2nalltriangs (for IsString)

▷ `chiro2nalltriangs(`*`String`*`)` (operation)

Returns: String

Convenience method of the above with List2 = [], List3 = []

2.1.47 points2nalltriangs (for IsList, IsList, IsList)

▷ `points2nalltriangs(`*List1, List2, List3*`)` (operation)

Returns: List

The first two lists are the input required by topcom. The third is a list of strings, consisting of the options supported by topcom.

2.1.48 points2nalltriangs (for IsList)

▷ `points2nalltriangs(`*List1*`)` (operation)

Returns: List

Convenience method of the above with List2 = [], List3 = []

2.1.49 chiro2allfinetriangs (for IsString, IsList, IsList)

▷ `chiro2allfinetriangs(`*String, List2, List3*`)` (operation)

Returns: String

The first argument is a string encoding the chiro and the second a list encoding an (optional) sample triangulation. The third argument is a list of strings, consisting of the options supported by topcom.

2.1.50 chiro2allfinetriangs (for IsString)

▷ `chiro2allfinetriangs(`*String*`)` (operation)

Returns: String

Convenience method of the above with List2 = [], List3 = []

2.1.51 points2allfinetriangs (for IsList, IsList, IsList)

▷ `points2allfinetriangs(`*List1, List2, List3*`)` (operation)

Returns: List

The first two lists are the input required by topcom. The third is a list of strings, consisting of the options supported by topcom.

2.1.52 points2allfinetriangs (for IsList)

▷ `points2allfinetriangs(`*List1*`)` (operation)

Returns: List

Convenience method of the above with List2 = [], List3 = []

2.1.53 chiro2nallfinetriangs (for IsString, IsList, IsList)

▷ `chiro2nallfinetriangs(`*String, List2, List3*`)` (operation)

Returns: String

The first argument is a string encoding the chiro and the second a list encoding an (optional) sample triangulation. The third argument is a list of strings, consisting of the options supported by topcom.

2.1.54 chiro2nallfinetriangs (for IsString)

▷ `chiro2nallfinetriangs(`*`String`*`)` (operation)

Returns: String

Convenience method of the above with List2 = [], List3 = []

2.1.55 points2nallfinetriangs (for IsList, IsList, IsList)

▷ `points2nallfinetriangs(`*`List1, List2, List3`*`)` (operation)

Returns: List

The first two lists are the input required by topcom. The third is a list of strings, consisting of the options supported by topcom.

2.1.56 points2nallfinetriangs (for IsList)

▷ `points2nallfinetriangs(`*`List1`*`)` (operation)

Returns: List

Convenience method of the above with List2 = [], List3 = []

2.2 Functionality of Topcom: Examples

Example

```
gap> rays := [[1,0],[0,1],[-1,-1]];
[ [ 1, 0 ], [ 0, 1 ], [ -1, -1 ] ]
gap> chiro := Concatenation(
>               "12, 4:-+--+++---++---++-+---++-+++-----++--++-++++--++",
>               "---+++-+++--+---++---++--++-++++--+---++-+++--+++--++-",
>               "-+----+-+++--+++--++--+----+---++--++-++++-++--+-----+",
>               "-----++---++---+++-+++--+---++---++--++-++++--+---++-+",
>               "++--+++--++--+----+-+++--+++--++--+----+---++--++-++++",
>               "---++-+++++-+++++--++++---++-+++--+++--++--+----+-+++-",
>               "-+++--++--+----+---++--++-++++---++-+++++-+++++--+++--",
>               "-++---++--++-++++-+++--++--+----+++--+-----+-----++--+",
>               "++--++--+----+++-+-----+-----++----++-+++++-+++++--++-",
>                                                    "-------++--++-" );
"12, 4:-+--+++---++---++-+---++-+++-----++--++-++++--++---+++-+++--+--\
-++---++--++-++++--+---++-+++--+++--++--+----+-+++--+++--++--+----+---\
++--++-++++-++--+-----+-----++---++---+++-+++--+---++---++--++-++++--+\
---++-+++--+++--++--+----+-+++--+++--++--+----+---++--++-++++---++-+++\
++-+++++--++++---++-+++--+++--++--+----+-+++--+++--++--+----+---++--++\
-++++---++-++++-++++--+++---++---++--++-++++-+++--++--+----+++--+---\
--+-----++--+++--++--+----+++-+-----+-----++----++-+++++-+++++--++---\
----++--++-"
gap> points2chiro( rays );
"3,2:+-+[]"
gap> chiro2dual( chiro );
"12,8:0++----++-+--+-++--++--+---++-+++---+--++--+---++-++++-++--+++--\
+-++-++--++-++---+--+++--+--++---++-+--+---++---++-+--+--++-+++--++-++\
---+--+++--+--++---++-+--+---++---++-+--+--++-++++--+++--+-++-++--+--+\
--+--+-++--+--+++-++---++-++--+++--+-++-+++--+++--+-++-++--+----++---+\
+-+--+--++-++-++-+++--++---++-+--+--++-++-++-+++-++-++---++--+--+++-++\
```

```
---++---++---++-+--+---++---++-+--+--++-++++--+++--+-++-++--+--+--+---\
++--+++--+-++-++--+--+--+---+--+--++--++---++-+--+-+--+--+--+----++-++\
---++-++---"
gap> chiro2circuits( points2chiro( rays ) );
"3,2:[[[0,1,2],[]]]"
gap> chiro2cocircuits( points2chiro( rays ) );
"3,2:[[[0],[1]][[1],[2]][[0],[2]]]"
gap> result := cocircuits2facets( chiro2cocircuits( chiro ) );
""
gap> points2facets( rays );
"3,2:[]"
gap> points2nflips( rays );
3
gap> points2flips( rays );
"[3,2:[[[0,1,2],[]]->0]]"
gap> chiro2placingtriang( chiro );
[ [ [ 0, 1, 2, 3 ], [ 1, 2, 3, 4 ], [ 0, 1, 3, 4 ], [ 0, 1, 2, 4 ],
    [ 0, 2, 3, 5 ], [ 0, 3, 4, 6 ], [ 0, 2, 4, 6 ], [ 0, 3, 5, 6 ],
    [ 0, 2, 5, 6 ], [ 2, 3, 4, 7 ], [ 2, 3, 5, 7 ], [ 3, 4, 7, 8 ],
    [ 2, 4, 7, 8 ], [ 3, 5, 7, 8 ], [ 2, 5, 7, 8 ], [ 3, 4, 8, 9 ],
    [ 2, 4, 8, 9 ], [ 3, 5, 8, 9 ], [ 2, 5, 8, 9 ], [ 3, 4, 6, 10 ],
    [ 2, 4, 6, 10 ], [ 3, 5, 6, 10 ], [ 2, 5, 6, 10 ] ] ]
gap> points2placingtriang( rays );
[ [ [ 0, 1 ], [ 1, 2 ], [ 0, 2 ] ] ]
gap> chiro2finetriang( chiro );
[ [ [ 0, 1, 2, 3 ], [ 1, 2, 3, 4 ], [ 0, 1, 3, 4 ], [ 0, 1, 2, 4 ],
    [ 0, 2, 3, 5 ], [ 0, 3, 4, 6 ], [ 0, 2, 4, 6 ], [ 0, 3, 5, 6 ],
    [ 0, 2, 5, 6 ], [ 2, 3, 4, 7 ], [ 2, 3, 5, 7 ], [ 3, 4, 7, 8 ],
    [ 2, 4, 7, 8 ], [ 3, 5, 7, 8 ], [ 2, 5, 7, 8 ], [ 3, 4, 8, 9 ],
    [ 2, 4, 8, 9 ], [ 3, 5, 8, 9 ], [ 2, 5, 8, 9 ], [ 3, 4, 6, 10 ],
    [ 2, 4, 6, 10 ], [ 3, 5, 6, 10 ], [ 2, 5, 6, 11 ], [ 2, 6, 10, 11 ],
    [ 5, 6, 10, 11 ], [ 2, 5, 10, 11 ] ] ]
gap> points2finetriang( rays );
[ [ [ 0, 1 ], [ 1, 2 ], [ 0, 2 ] ] ]
gap> chiro2triangs( points2chiro( rays ) );
"T[0]:=[0->3,2:[[0,1],[1,2],[0,2]]];T[1]:=[1->3,2:[]];"
gap> points2triangs( rays );
"T[0]:=[0->3,2:[[0,1],[1,2],[0,2]]];T[1]:=[1->3,2:[]];"
gap> chiro2ntriangs( points2chiro( rays ) );
2
gap> points2ntriangs( rays );
2
gap> chiro2finetriangs( points2chiro( rays ) );
"T[0]:=[0->3,2:[[0,1],[1,2],[0,2]]];T[1]:=[1->3,2:[]];"
gap> points2finetriang( rays );
[ [ [ 0, 1 ], [ 1, 2 ], [ 0, 2 ] ] ]
gap> chiro2nfinetriangs( points2chiro( rays ) );
2
gap> points2nfinetriangs( rays );
2
gap> chiro2alltriangs( points2chiro( rays ) );
[ [ [ 0, 1 ], [ 0, 2 ], [ 1, 2 ] ] ]
```

```
gap> points2alltriangs( rays );
[ [ [ 0, 1 ], [ 0, 2 ], [ 1, 2 ] ] ]
gap> chiro2nalltriangs( points2chiro( rays ) );
1
gap> points2nalltriangs( rays );
1
gap> chiro2allfinetriangs( points2chiro( rays ) );
[ [ [ 0, 1 ], [ 0, 2 ], [ 1, 2 ] ] ]
gap> points2allfinetriangs( rays );
[ [ [ 0, 1 ], [ 0, 2 ], [ 1, 2 ] ] ]
gap> points2allfinetriangs( rays, [], ["regular"] );
[ [ [ 0, 1 ], [ 0, 2 ], [ 1, 2 ] ] ]
gap> chiro2nallfinetriangs( points2chiro( rays ) );
1
gap> points2nallfinetriangs( rays );
1
gap> points2nallfinetriangs( rays, [], [] );
1
gap> rays2 := [ [0,0,1], [1,0,1], [2,0,1], [0,1,1],
>            [1,1,1], [2,1,1], [0,2,1], [1,2,1], [2,2,1], ];
[ [ 0, 0, 1 ], [ 1, 0, 1 ], [ 2, 0, 1 ], [ 0, 1, 1 ], [ 1, 1, 1 ],
  [ 2, 1, 1 ], [ 0, 2, 1 ], [ 1, 2, 1 ], [ 2, 2, 1 ] ]
gap> sample_triang2 := [ [2,1,0,5,4,3,8,7,6], [0,3,6,1,4,7,2,5,8] ];
[ [ 2, 1, 0, 5, 4, 3, 8, 7, 6 ], [ 0, 3, 6, 1, 4, 7, 2, 5, 8 ] ]
gap> points2ntriangs( rays2, sample_triang2, [] );
69
```

Chapter 3

Interface to Topcom

3.1 Finding the TopcomDirectory

3.1.1 FindTopcomDirectory

▷ `FindTopcomDirectory(`*`none`*`)` (operation)

Returns: the corresponding directory

This operation identifies the location where the topcom operations are stored.

3.2 Executing topcom

3.2.1 ExecuteTopcomForPoints (for IsDirectory, IsString, IsList, IsList, IsList)

▷ `ExecuteTopcomForPoints(`*`A, Directory, a, string, and, three, lists`*`)` (operation)

Returns: the corresponding quantity as computed by Topcom as a string

This operation executes topcom with five pieces of input information. The first is the directory of topcom, the second the name of the binary that is to be executed within topcom, the third is a list of points, the fourth a list containing a seed triangulation (this is optional) and the fifth a number of options (also optional). In case no seed triangulation or option is to be used, the empty list should be handed to the gap method.

3.2.2 ExecuteTopcomForChiro (for IsDirectory, IsString, IsString, IsList, IsList)

▷ `ExecuteTopcomForChiro(`*`A, Directory, a, string, a, string, and, two, lists`*`)` (operation)

Returns: the corresponding quantity as computed by Topcom as a string

This operation executes topcom with five pieces of input information. The first is the directory of topcom, the second the name of the binary that is to be executed within topcom, the third is a string encoding a chiro, the fourth a list containing a seed triangulation (this is optional) and the fifth a number of options (also optional). In case no seed triangulation or option is to be used, the empty list should be handed to the gap method.

Index