CoCoALib - Design #769

CoCoALib cone

03 Sep 2015 16:50 - Anna Maria Bigatti

Status:	In Progress	**Start date:**	03 Sep 2015
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	10%
Category:	Data Structures	**Estimated time:**	0.00 hour
Target version:	CoCoALib-1.0	**Spent time:**	0.85 hour

Description

Make a cone type which includes Normaliz and Gfan cones.

Related issues:

Related to CoCoALib - Feature #762: ExternalLib-GFan: first prototype	**Closed**	**28 Aug 2015**
Related to CoCoALib - Feature #210: Normaliz: "double" cone for speed and safety	**Closed**	**25 Jul 2012**
Related to CoCoA-5 - Feature #770: CoCoA type "cone"	**New**	**03 Sep 2015**

History

#1 - 03 Sep 2015 17:01 - Anna Maria Bigatti

- Subject changed from CoCoA cone to CoCoALib cone

#2 - 04 Sep 2015 15:19 - John Abbott

Normaliz 2.99 now includes the "double cone" idea we had used in CoCoALib; this means that the CoCoALib impl can be simpler.

Normaliz and Gfanlib have different approaches to the interpretation of "const":
Normaliz regards "const" as meaning immutable memory image;
Gfan regards "const" as meaning that the cone being represented doesnot change (but the in-memory representation may change).

Is there are problem with these differing viewpoints?
Perhaps it can be solved by using "mutable" (as we already do for normaliz cones)

#3 - 04 Sep 2015 17:49 - John Abbott

- Status changed from New to In Progress

- % Done changed from 0 to 10

Here is a suggested design.

A CoCoALib cone object typically contains both a normaliz cone and a gfan cone. Obviously both fields can be present only if both extlibs are present.

If neither extlib is present then probably the CoCoALib cone class should not be defined (since it cannot be used for anything, and probably would have no usable ctor).

If just one of normaliz/gfan is present then a CoCoALib cone would be just a wrapper for the cone type of the extlib. If both are present then a CoCoALib cone will contain one cone from each of the two extlibs.

When both are present we could:

- **(A)** require that both cones are initialized (with mathematically equivalent values), or
- **(B)** we could initialize just one of the cones and keep an enum (or 2 flags) to say which cone(s) are initialized.

Approach (A) looks to be potentially more costly since both cones must be initialized whenever a CoCoALib cone is created.
With approach (B) it is not so clear which of the two cones should be initialized upon construction; moreover the implementation may have to create "on the fly" the absent cone type if an operation requiring that sort of cone is attempted. Perhaps the type of cone to be initialized could be "guessed"

from the ctor parameters: if they are gfan-like then make a gfan cone, otherwise make a normalize cone.

Implementing methods on a CoCoALib cone could be "interesting" since the implementation has to be valid in the 3 cases:

- gfan only
- normaliz only
- both gfan and normaliz

In particular what should happen to a function which is available in just one of the extlibs?

#4 - 04 Sep 2015 20:14 - John Abbott

Anders pointed out that normaliz's interpretation of "const" is probably more threadsafe than gfan's

#5 - 06 Sep 2015 14:04 - Christof Soeger

I think the CoCoALib cone should always exist. Even if it can do basically nothing. But otherwise a user would have to make two distinctions
1. the cone exists
2. the special method of the cone works

And maybe you want to add functionality to that cone which does not depend on one of the two libs (maybe another). Okay this is not really a problem.

"const": CoCoALib (and gfan) decided to use const when the mathematical meaning does not change, so we should also use that for this cone.

BTW: Scott Meyers comment on the threadsafety of const member functions in his video Effective Modern C++ (Part 6 of 6). But a discussion about that should go in another issue.

#6 - 27 Jan 2020 15:32 - John Abbott

What is the status of this issue? Can we close it?