CoCoALib - Feature #482

Unique copies of rings -- smart ctor

19 Mar 2014 21:12 - John Abbott

Status:	In Progress	**Start date:**	19 Mar 2014
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	10%
Category:	Safety	**Estimated time:**	0.00 hour
Target version:	CoCoALib-1.0	**Spent time:**	1.75 hour

Description

After some experience using CoCoALib, it has been useful that ZZ and QQ are unique (*i.e.* repeated calls to the ctor produce the same identical ring).

It would be nice to extend this notion of a **unique copy** to other rings too; obvious candidates are the small prime finite fields.

Is it technically possible to do the same for more complex rings (such as QQ[x])?
If so, it is feasible (without excessive overheads even in multithreaded execution)?
If so, do we actually want to do this?

Related issues:

Related to CoCoA-5 - Design #483: Unique copies of rings in CoCoA-5	**New**	**19 Mar 2014**
Related to CoCoALib - Design #647: Unique copies of free modules?	**New**	**10 Nov 2014**
Related to CoCoALib - Design #785: finite fields: global register of fields a...	**New**	**12 Oct 2015**
Related to CoCoALib - Feature #390: Store unique copy of QQ[t_1..t_n] (RingQQ...	**Closed**	**23 Jul 2013**
Related to CoCoALib - Design #787: Remove refcounts from RingElem?	**New**	**16 Oct 2015**
Related to CoCoALib - Design #763: GlobalManager: initialization compatible w...	**In Progress**	**01 Sep 2015**

History

#1 - 19 Mar 2014 21:22 - John Abbott

- Category set to Safety

Having unique copies of rings implies a centralized global registry, and this might adversely affect multithreaded execution -- at least for the creation of a new ring, but code which creates lots of different rings is "asking for trouble anyway".

Do we want to be able to remove rings from the centralized registry?
Is this technically possible in a multithreaded setting without incurring bad overhead?
Does it matter much?
JAA thinks the biggest rings (in terms of RAM) are the small finite fields.

The registry must be emptied (destroyed) by GlobalManager during final tidy up; the order of destruction is important!
If rings cannot be removed then the registry is a sort of "memory leak" -- it never frees resources that are no longer needed [think of temporary rings used inside some algorithms].

#2 - 19 Mar 2014 21:28 - John Abbott

If a ring pseudo-ctor is to be smart (and return an existing ring if possible) then how smart should it be?

Consider two poly rings with the same indets, the only difference are the order matrices which happen to define the same ordering. Will the smart ctor recognise this?

What about QQ[x,y],lex and QQ[y,x],mat([[0,1],[1,0]])?
COMMENT (2015-10-16) the two rings above are clearly different since indet(P,0) prints out differently!

Probably a pragmatic defn of *the same* is best: the ctor args have to be equal.

#3 - 20 Mar 2014 09:34 - Anna Maria Bigatti

- Target version set to CoCoALib-0.99534 Seoul14

My guess is that it will be impossible to deal with under multithreading, and, if implemented, we must guarantee the same behaviour in single and multithread environment.
Consider the case of Chinese Remandering...

#4 - 09 Jul 2014 18:07 - John Abbott

- Target version changed from CoCoALib-0.99534 Seoul14 to CoCoALib-1.0

#5 - 16 Oct 2015 12:27 - John Abbott

I am considering as a "first step" towards possibly implementing unique copies of (some) rings to implement a global registry of all types of ring (not just finite fields).

I think it may also be useful to be able to create "temporary" rings which are not inserted into the global registry (*e.g.* inside CRT-based algorithms), so that the registry does not fill up with "junk".

There are some questions related to destruction of rings:

- when is a ring destroyed?
- if a ring can be destroyed before the end of the program, what happens to its slot in the registry?
- are rings in the registry to be destroyed when the dtor for GlobalManager is called? [presumably "yes", if they have not already been destroyed]
- if rings in the registry are to be destroyed then they must be destroyed in reverse order to insertion [so presumably the registry will be a stack?]

#6 - 20 Oct 2015 14:33 - John Abbott

- Status changed from New to In Progress

- % Done changed from 0 to 10

Continuing with the assumption that rings in the registry will be destroyed (in reverse) order when the GlobalManager goes out of scope...

- **(A)** assuming that a smart pseudo-ctor will return an existing ring if a suitable one already exists, how is the pseudo-ctor going to find out whether a suitable ring already exists in the register?
- **(B)** if "temporary rings" can also be created (*i.e.* ones not placed in the register) then the pseudo-ctor must be told whether to create a temporary or registered ring -- this could be via an argument, or there could be two different pseudo-ctors.

For **(B)**, if there is a single pseudo-ctor, what type will it return? The type must be able to represent both registered rings and temporary rings.

For **(A)**, for the case of rings of the form ZZ/(n) we can easily use a std::map<long,ring*> to find out if the ring has already been registered. We will also need a std::map<BigInt, ring*> for large characteristic.

What should be the second type param in the std::map@s above? A plain pointer is probably sufficient. Note that a @std::stack by default bases itself on a std::deque, and that push_back on a deque (potentially) invalidates all iterators; so it would not be safe to use iterators to "point" to elements of the stack. Perhaps a std::stack based on a std::list would be better?

#7 - 21 Mar 2016 15:14 - John Abbott

- Related to Design #763: GlobalManager: initialization compatible with initialization of external libs added