# CoCoALib - Design #785

## finite fields: global register of fields already created?

12 Oct 2015 11:44 - John Abbott

| | | | |
|---|---|---|---|
| **Status:** | New | **Start date:** | 12 Oct 2015 |
| **Priority:** | Normal | **Due date:** | |
| **Assignee:** | | **% Done:** | 0% |
| **Category:** | Improving | **Estimated time:** | 0.00 hour |
| **Target version:** | CoCoALib-1.0 | **Spent time:** | 0.95 hour |

**Description**

The rings ZZ and QQ are global and unique -- you cannot (any more) create two copies of ZZ or of QQ.

The small prime finite fields are not unique: if you call twice NewZZmod(5) it will give two distinct rings both implementing the field of 5 elements. Yet it is well known that a finite field of given cardinality is essentially unique -- all such fields are isomorphic.

It would be more natural if there were a function FiniteField(5) which always returns the same ring. Perhaps also for FiniteField(25) which is not a "prime field".

Can this be done (threadsafely)? Do we want it?

**Related issues:**

| | | |
|---|---|---|
| Related to CoCoALib - Feature #664: Impl small non-prime finite fields (using... | **Resolved** | **11 Feb 2015** |
| Related to CoCoALib - Feature #482: Unique copies of rings -- smart ctor | **In Progress** | **19 Mar 2014** |
| Related to CoCoALib - Feature #180: GlobalManager: registration of global var... | **Closed** | **07 Jun 2012** |
| Related to CoCoALib - Design #763: GlobalManager: initialization compatible w... | **In Progress** | **01 Sep 2015** |
| Is duplicate of CoCoALib - Feature #281: Store unique copy of FF(p) in Global... | **New** | **03 Dec 2012** |

### History

#### #1 - 12 Oct 2015 11:49 - John Abbott

A global register of finite fields would imply that once a field has been created it cannot later be destroyed (unless we are sure that no reference to that field still remains). NB **A global datastructure needs special handing to be threadsafe.**

Algorithms which use chinese remaindering may internally create very many finite fields; to avoid filling up the global register, it would be better if these "temporary" finite fields were not added to the register. This implies having two distinct ways of creating finite fields: one which uses the global register, and the other which does not.

#### #2 - 12 Oct 2015 16:14 - John Abbott

JAA and Renzo discussed the idea of automatically creating all small, prime finite fields (up to size 32767, say) during initialization. This would make having a single unique copy easy to achieve -- we just mimic what we have done for RingZZ and RingQQ.

JAA discarded the suggestion because, in any case, if we adopt the idea that each (prime) finite field is to be unique then this must apply all (prime) finite fields, and not just those up to some arbitrary limit. So some mechanism must be devised for other (prime) finite fields which will have to be created on-the-fly, in which case the same mechanism might just as well be used for all (prime) finite fields.

Also the idea of creating around 3500 rings during initialization means every program using CoCoALib has to pay the time and memory cost of the construction. It also seems a shame to require a minimum RAM "footprint" of 16Mbytes (assuming 4K minimum loaf size -- see #786).

#### #3 - 12 Oct 2015 16:58 - John Abbott

Presumably the global register will comprise two std::map objects: one for characteristics which fit into long, and one for larger characteristics.

Presumably the functions which access and update the global register will have to use a threadsafe trick like that described in #784.

Will it be possibile to remove rings from the register?  If so, how and when?

#### #4 - 13 Oct 2015 17:35 - Anna Maria Bigatti

John Abbott wrote:

> Presumably the global register will comprise two std::map objects: one for characteristics which fit into long, and one for larger characteristics.
>
> Presumably the functions which access and update the global register will have to use a threadsafe trick like that described in #784.

I expect there is no (time) problem in locking the map when creating a new ring (that should be a rare event...)

> Will it be possibile to remove rings from the register?  If so, how and when?

This is potentially dangerous, or not?

#### #5 - 15 Oct 2015 16:36 - John Abbott

I modified my prototype so that it registered the std::map for global destruction, then realised that the order of destruction of the rings inside the std::map is unknown.  For basic finite fields (no alg extn) this is not a problem as the different rings are entirely independent of each other; as soon as algebraic extensions are considered dependencies appear, and the rings must be destroyed in the correct order (reverse order of construction).

THIS IS A PROBLEM!

#### #6 - 15 Oct 2015 17:27 - Anna Maria Bigatti

John Abbott wrote:

> and the rings must be destroyed in the correct order (reverse order of construction).
> THIS IS A PROBLEM!

Yes indeed...
I've got this idea (would it work?):
Register is
+ an array/vector<ring> (naturally sorted by RingID).
+ a map<long> indexed by characteristic, giving the array position.
The destructor destroys the rings in reversed order. (The map is just a map of long)

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

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