CoCoALib - Design #429

factorization: public data fields, or mem fns allowing the fields to be updated.

28 Jan 2014 18:18 - John Abbott

Status:	Closed	**Start date:**	28 Jan 2014
Priority:	Normal	**Due date:**	
Assignee:	John Abbott	**% Done:**	100%
Category:	Safety	**Estimated time:**	4.01 hours
Target version:	CoCoALib-0.99533 Easter14	**Spent time:**	4.10 hours

Description

Christof was surprised to see that the data fields in a factorization are publicly modifiable. He suggested having member fns allowing the field to be updated in a coherent manner (*e.g.* guaranteeing that the factors lie in the same ring, and that there is a multiplicity for each factor)

Look at use of factorization and decide what mem fns would be needed for existing code to work if the data fields were private.

Related issues:

Related to CoCoALib - Support #391: Check consistency of template class facto...	**Closed**	**23 Jul 2013**
Related to CoCoALib - Feature #356: IsZeroDivisor	**Closed**	**24 May 2013**
Related to CoCoALib - Design #411: design of factorization template class	**Closed**	**23 Oct 2013**

History

#1 - 31 Jan 2014 20:25 - John Abbott

- Category set to Safety

- Status changed from New to In Progress

- % Done changed from 0 to 10

John and Anna talked about Christof's proposal, and both believe it is a good idea! :-)

John will look at existing code which uses factorization and then propose a proper, clean interface.

Some ideas:

- new fn to add factor+multiplicity pair (will check type compatibility, default mult=1)
- factors must be non-zero, and non-invertible
- NO further restriction on factors (may be repeats, may have common factors)
- unsure what restrictions on multiplicities (>0, >=0, no restriction?)
- Anna suggested using the same structure to represent ideal decompositions

We'll probably need some template specializations to handle the differences between BigInt and RingElem. Maybe more specializations if we choose to use the same structure for ideal decompositions.

#2 - 27 Mar 2014 11:30 - Christof Soeger

I updated nmzIntegrate to the changes, while looking at the header to figure out the new interface of the factorization I saw these lines:

```
///???    const T& myFactor(long i) const { return myFactorVec[i]; }
    const std::vector<T>& myFactors() const { return myFactorVec; }
///???    const long myMultiplicity(long i) const { return myMultiplicityVec[i]; }
```

I think it is a good idea to have them. Right now we write something like

```
for(i=0;i<FF.myFactors().size();++i){
    if(FF.myMultiplicities()[i]==1)
        G*=phi(FF.myFactors()[i]);
    else{
        G1=phi(FF.myFactors()[i]);
        for(int nn=0;nn<FF.myMultiplicities()[i];++nn)
            G*=G1;
    }
}
```

maybe also a direct size() function.

#3 - 27 Mar 2014 14:54 - John Abbott

- Assignee set to John Abbott

- Target version set to CoCoALib-0.99533 Easter14

- % Done changed from 10 to 20

The reason myFactor and myMultiplicity are commented out is because they offered little gain in readability; it seemed more readable to use const-ref aliases (though I'm not entirely happy with this solution).

Your example code would then look like this:

```
const vector<RingElem>& facs = FF.myFactors();  // handy alias
const vector<long>& mults = FF.myMultiplicities(); // handy alias
const long NumFacs = facs.size();
for(i=0;i<NumFacs;++i){
    if(mults[i]==1)
        G*=phi(facs[i]);
    else{
        G1=phi(facs[i]);
        for(int nn=0;nn<mults[i];++nn)
            G*=G1;
    }
}
```

In fact I'd be tempted to shorten the code to this:

```
const vector<RingElem>& facs = FF.myFactors();  // handy alias
const vector<long>& mults = FF.myMultiplicities(); // handy alias
const long NumFacs = facs.size();
for(i=0;i<NumFacs;++i)
{
  RingElem G1=phi(facs[i]);
  for (int nn=0;nn<mults[i];++nn)
    G*=G1;
}
```

Of course, readability is "in the eye of the beholder".

#4 - 27 Mar 2014 15:00 - John Abbott

The main reason I do not offer a len/size fn is that its semantics are unclear to me.

(A) Should it simply be the length of the vector myFactors?
(B) Or should it try to take into account myRemainingFactor (*e.g.* if it not a unit)

I think **(A)** is by far the more sensible/useful choice.

Some might even think that it should be
(C) the sum of the values in myMultiplicities.
Perhaps not entirely unreasonable, but not terribly useful either.

#5 - 27 Mar 2014 15:06 - John Abbott

I have added a note to the documentation about possibly having a function for changing the multiplicity of a factor.

Should the function let one change the multiplicity to zero? If so, what happens? Is that factor deleted from myFactors()? Or do we allow factors with zero multiplicity? [seems potentially troublesome]

#6 - 27 Mar 2014 16:37 - Christof Soeger

The const ref aliases are a good suggestion. myFactors(i) was also not the optimal syntax.
For the size, to me it was clear what I had in mind, but to avoid the ambiguity it is best to just do not have it.

Some other point I ran into during this, they concern the consistency tests.

We use the factorization for a polynomial g in Q[x]. In this case the factors in factor(g) will have integer coefficients and the denominator is collected in the remainingFactor. We use this feature. We convert the factors to Z[x] and then we continue to work with a factorization where the factors are in Z[x] and the remaining factor is in Q. This worked well so far, but now the new checks gives us errors. We can work around it and just do not use the remaining factor and just set it to one(RingZZ()).
The header says

```
    static void ourCheckCompatible(const T& fac, const T& RemFac); // give error if fac*RemFac cannot be compu
ted
```

but in fact it checks if the owner of the elements are the same. So it is the old problem of which values are allowed to be multiplied automatically.

But after I did this, the next problem I cannot figure out. Here our code:

```
  cout << "multis " << multiplicities << endl;
  cout << "remFac " << remainingFactor << " " << owner(remainingFactor) << endl;
  cout << "fac1 " << primeFactorsNonhom << " " << owner(primeFactorsNonhom[0]) << " " << owner(primeFactorsNon
hom[1]) << endl;
  factorization<RingElem> FFNonhom(primeFactorsNonhom,multiplicities,remainingFactor); // for output

  cout << "fac2 " << primeFactors << " " << owner(primeFactors[0]) << " " << owner(primeFactors[1]) << endl;
  cout << "test remFac " << one(owner(primeFactors[0])) << " " << owner(one(owner(primeFactors[0]))) << endl;
  factorization<RingElem> FF(primeFactors,multiplicities,one(owner(primeFactors[0]))); //remainingFactor); //
assembels the data
```

and the output:

```
multis [4,  1]
remFac -5 RingDistrMPolyInlPP(QQ, 4)
fac1 [x[2],  x[1]] RingDistrMPolyInlPP(QQ, 4) RingDistrMPolyInlPP(QQ, 4)
fac2 [x[2],  x[1]] RingDistrMPolyInlPP(ZZ, 4) RingDistrMPolyInlPP(ZZ, 4)
test remFac 1 RingDistrMPolyInlPP(ZZ, 4)
terminate called after throwing an instance of 'CoCoA::ErrorInfo'
  what():  NYI ideal of polynomials with coeffs not in a field

Program received signal SIGABRT, Aborted.
0x00000000007aca9b in raise (sig=<optimized out>) at ../nptl/sysdeps/unix/sysv/linux/pt-raise.c:42
42    ../nptl/sysdeps/unix/sysv/linux/pt-raise.c: No such file or directory.
(gdb) bt
#0  0x00000000007aca9b in raise (sig=<optimized out>) at ../nptl/sysdeps/unix/sysv/linux/pt-raise.c:42
#1  0x00000000007b712b in abort ()
#2  0x0000000000078787d in __gnu_cxx::__verbose_terminate_handler() ()
#3  0x0000000000784de6 in __cxxabiv1::__terminate(void (*)()) ()
#4  0x0000000000784e13 in std::terminate() ()
#5  0x0000000000783cbe in __cxa_throw ()
#6  0x00000000005678d2 in CoCoA::ThrowError(CoCoA::ErrorInfo const&) ()
#7  0x00000000005d5633 in CoCoA::SparsePolyRingBase::IdealImpl::IdealImpl(CoCoA::SparsePolyRing const&, std::v
ector<CoCoA::RingElem, std::allocator<CoCoA::RingElem> > const&) ()
#8  0x00000000005d5732 in CoCoA::SparsePolyRingBase::myIdealCtor(std::vector<CoCoA::RingElem, std::allocator<C
oCoA::RingElem> > const&) const ()
#9  0x00000000005a882c in CoCoA::ideal::ideal(CoCoA::RingElemAlias const&) ()
#10 0x0000000000059c465 in CoCoA::IsZeroDivisor(CoCoA::RingElemAlias const&) ()
#11 0x0000000000043012d in ourCheckNotZeroDiv (fac=...)
    at /home/csoeger/CoCoA/CoCoALib-0.99/include/CoCoA/factorization.H:101
#12 CoCoA::factorization<CoCoA::RingElem>::ourConsistencyCheck (factors=..., multiplicities=..., RemainingFact
or=...)
    at /home/csoeger/CoCoA/CoCoALib-0.99/include/CoCoA/factorization.H:165
#13 0x0000000000426123 in factorization (RemainingFactor=..., multiplicities=..., factors=..., this=0x7fffffff
d7b0)
    at /home/csoeger/CoCoA/CoCoALib-0.99/include/CoCoA/factorization.H:56
#14 integrate (project=..., pnm=..., do_leadCoeff=@0x7fffffffdead: true, homogeneous=@0xbbec80: 48,
    appendOutput=@0x7fffffffdeac: false) at nmzIntegral.C:229
```

The creation of the first factorization works fine. For the second I use a dummy RemainingFactor but then the ourCheckNotZeroDiv strangely fails as visible above.
Do you have any idea?

#7 - 27 Mar 2014 18:28 - John Abbott

- % Done changed from 20 to 50

I have fixed the problem with factorizations over ZZ[x].
[I have implemented IsIntegralDomain3]

Checked in to CVS.
Let me know if it works OK for you.

#8 - 27 Mar 2014 18:56 - Christof Soeger

Yes, that works. Thanks a lot! We can work with that solution.

How do you think about allowing ZZ[x] factors together with QQ remaniningFactor?
I think this is related to Feature #223.

#9 - 01 Apr 2014 15:35 - John Abbott

- Status changed from In Progress to Feedback

- % Done changed from 50 to 90

John and Anna are not convinced by the proposal to allow the factors to reside in different rings (even if CoCoALib will eventually allow arithmetic between elements of certain different rings).

JAA also notes that it is generally easier to start with a more restrictive implementation and then relax the restrictions (which typically won't break existing code), than it is to tighten restrictions (with the risk of breaking some existing code).

Bruns reports that anyway Normaliz works well enough with the current restrictions.

JAA will add some comments to the documentation about ideas for relaxing restrictions (so this discussion won't be forgotten).

#10 - 02 Apr 2014 16:54 - John Abbott

- Status changed from Feedback to Closed

- % Done changed from 90 to 100

Closing because it works, the code has been released, and we're happy with the new design.

#11 - 17 Apr 2014 09:50 - Anna Maria Bigatti

- Estimated time set to 4.01 h