CoCoA-5 - Bug #110

Surprise return type for GCD of a list of ints

19 Mar 2012 16:19 - John Abbott

Status:	Closed	**Start date:**	19 Mar 2012
Priority:	Normal	**Due date:**	
Assignee:	John Abbott	**% Done:**	100%
Category:	CoCoA-4 function to be added	**Estimated time:**	15.00 hours
Target version:	CoCoA-5.0.3	**Spent time:**	11.50 hours

Description

JAA expected GCD of a list of INTs to produce an INT; instead it produces a RINGELEM (in ZZ)

```
N := gcd([2,4,6]);
Type(N); --> RINGELEM
RingOf(N); --> ZZ
```

The behaviour comes from the fact that gcd uses the same code as is used for reading matrix elements which always produces a list of RINGELEMs (in ZZ or QQ if all entries are ringless).

Anna also asked what result I would expect if the list contained some RAT values (all of whose denominators are 1). It seems rather hard to justify giving an INT as the result in that case.

A possible solution could be to have a separate function (i.e. different name) for computing GCDs of INTs (which could auto-convert a RAT with denom=1 into an INT).

The hard part will probably be decising what strategy to adopt; implementation will likely be simple(-ish) but tedious.

Related issues:

Related to CoCoA-5 - Bug #158: May AsRAT produce an INT?	**Closed**	**14 May 2012**

History

#1 - 19 Mar 2012 16:45 - John Abbott

The function AsINT can be used as a workaround:

```
L := [2,4,6];
N := AsINT(gcd(L));
Type(N); --> INT
```

#2 - 14 May 2012 12:14 - John Abbott

Today some students passed with a CoCoA-5 problem which derived from the unexpected return type of **gcd** applied to a list of INT.

JAA suggests that **gcd** of list of INT give result INT (non-negative); if there is any RAT in the list then an error is produced.

#3 - 14 May 2012 12:23 - John Abbott

JAA proposes that reading of homogeneous lists have three possible return types:

- list of INT
- list of RAT

- list of RINGELEM (with the guarantee that all elements are in the same ring)

Not sure what the empty list should give; probably either error or list of INT (because INT is always convertible into any of the other types).

It is not entirely clear how best to achieve this in C++ since functions have to have a fixed return type.

#4 - 14 May 2012 14:06 - John Abbott

JAA thinks that introducing a separate name for the function for computing GCD of integers is a **bad** idea because it places an extra (unnecessary?) burden on the user.

#5 - 14 May 2012 15:03 - John Abbott

Anna thinks she can fix the reading of homogeneous lists; she will look into it in the next few days.

#6 - 15 May 2012 15:32 - Anna Maria Bigatti

- Category set to CoCoA-4 function to be added

- Status changed from New to Resolved

- Assignee set to John Abbott

- Target version set to CoCoA-5.0.3

- % Done changed from 0 to 60

#7 - 15 May 2012 16:17 - Anna Maria Bigatti

- % Done changed from 60 to 100

The code now works: **type(gcd([2,3,4])** is of type INT.
Also lcm has been fixed accordingly.

It does not seem that there are other functions taking list of INT or list RingElem as argument.

Still to be fixed: rational input

```
/**/ lcm(1,x);
ERROR: Expecting type INT, but found type RINGELEM
lcm(1,x);
      ^
```

#8 - 15 May 2012 16:17 - Anna Maria Bigatti

- % Done changed from 100 to 80

#9 - 15 May 2012 16:52 - Anna Maria Bigatti

In CoCoALib we have the function myGcdInField, so we can compute gcd of elements in RingQQ. So, what shouldwe do in CoCoA-5?

#10 - 16 May 2012 15:01 - John Abbott

JAA modified RingBase::myGcdInField to give error rather than produce a result of 0/1. All CoCoALib tests passed, but the SourceAnna test in CoCoA-5 failed because it explicitly tests gcd between two elements of QQ.

Mathematically a field is a GCD domain. However I cannot think of a situation where it would be correct and convenient to compute a GCD in a field. If the field is a fraction field then at times it would be handy to consider its elements as elements of the base ring (*e.g.* think of computing the "content" of a polynomial in QQ[x] which actually lies in the natural image of ZZ[x]) but then the 0-or-1 definition of GCD does not produce the desired result.

Maybe we could define gcd in a special way in fraction fields; but frankly, this seems to be a recipe for confusion!

However, if we regard fields as not "gcd domains" then what name should be use to mean "gcd domain but not a field"? **TrueGCDDomain**, **ProperGCDDomain**, **GCDRing** (with the implication that "ring" means "not field") It would be nice to have a short and unambiguous name.

#11 - 16 May 2012 15:31 - John Abbott

There is one other imstance where myGCDInField is called.
ex-RingElem1 attempts to compute a GCD in a field (after having checked that it satisfies IsGCDDomain).

#12 - 16 May 2012 15:36 - John Abbott

JAA thinks that

> **we can compute GCDs between values of type RAT**

should be equivalent to

> **we can compute GCDs between RINGELEM values in QQ**

At the moment JAA thinks the answer should be "no" in both cases, but is willing to listen to other opinions.

#13 - 16 May 2012 16:15 - Anna Maria Bigatti

> JAA thinks that
> **we can compute GCDs between values of type RAT**
> should be equivalent to
> **we can compute GCDs between RINGELEM values in QQ**
> At the moment JAA thinks the answer should be "no" in both cases, but is willing to listen to other opinions.

I agree.

I think that **myGCDInField** was written for printing simplified polys like

```
/**/ R ::= QQ[a,b];
/**/ K := NewFractionField(R);
/**/ Use P ::= K[x,y];
/**/ (3*a)*x/3;
a*x
```

.... or not?

#14 - 16 May 2012 18:20 - John Abbott

JAA believes that **myGcdInField** was created for the following reason.

In CoCoALib fields satisfy **IsGCDDomain** because mathematically this is true; this means that CoCoALib must be capable of computing gcds of elements from a field. Thus I implemented for each field the almost trivial function for computing gcds in fields. Later I realized that all these separate imlementations were essentially identical; so I "moved them up" to a common definition in **RingBase::myGcdInField**. Note that this member function is **protected** because it was intended to be accessible only in concrete ring implementations. In fact a quick "grep" reveals that it is called only from inside the implementations of **myGcd** in concrete ring classes which implement fields.

If we decide to forbid computing gcds of elements in a field then the function **RingBase::myGcdInField** can be removed (or replaced by a fn which simply gives a "cannot compute GCD in field" error).

Note that in a **QuotientRing** it can be determined only at run-time whether the ring is actually a field.

#15 - 17 May 2012 17:33 - Anna Maria Bigatti

John Abbott wrote:

> However, if we regard fields as not "gcd domains" then what name should be use to mean "gcd domain but not a field"? **TrueGCDDomain**, **ProperGCDDomain**, **GCDRing** (with the implication that "ring" means "not field") It would be nice to have a short and unambiguous name.

Another suggestion: **IsValidGCDDomain** (by L.Robbiano)

#16 - 17 May 2012 18:12 - John Abbott

Anna Maria Bigatti wrote:

> Another suggestion: **IsValidGCDDomain** (by L.Robbiano)

I'm not very convinced; I'd prefer **IsTrueGCDDomain**.
Another suggestion is **IsNontrivGCDDomain** (where Nontriv stands for non-trivial) -- it's very precise, but rather cumbersome.

Almost all our suggestions are rather long; perhaps this does not matter so much? Maybe we should just choose one, and possibly change it later if it turns out to be too cumbersome/unreadable/...?

#17 - 17 May 2012 18:32 - Anna Maria Bigatti

> Almost all our suggestions are rather long; perhaps this does not matter so much? Maybe we should just choose one, and possibly change it later if it turns out to be too cumbersome/unreadable/...?

So the best is **IsTrueGCDDomain** it is the shortest of the type Is***GCDDomain (and I think we should stick to GCDDomain)

#18 - 17 May 2012 21:41 - John Abbott

Anna Maria Bigatti wrote:

> So the best is **IsTrueGCDDomain** it is the shortest of the type Is***GCDDomain (and I think we should stick to GCDDomain)

Is that choice OK with Robbiano too?
That name has the advantage that it is compatible with an error symbol that I had already defined (if you recall...)

Here's another candidate (long but very clear) **IsGCDDomainNotField**

#19 - 28 May 2012 11:57 - John Abbott

- Status changed from Resolved to Closed

- % Done changed from 80 to 100

Implemented **IsTrueGCDDomain**; the main cost was sorting out the consequential changes. Eliminated ERR::NotGCDDomain; its function is now subsumed by ERR::NotTrueGCDDomain (which already existed).

I opted for a default definition (in **ring.C**) of "not is a field".

Changed the documentation.

#20 - 09 Nov 2015 15:32 - John Abbott

- Subject changed from Surprise return type for GCD fo a list of ints to Surprise return type for GCD of a list of ints