CoCoALib - Design #1500

IsDivisible in a field?

05 Oct 2020 14:29 - John Abbott

Status:	Closed	**Start date:**	05 Oct 2020
Priority:	Normal	**Due date:**	
Assignee:	John Abbott	**% Done:**	100%
Category:	Safety	**Estimated time:**	3.99 hours
Target version:	CoCoALib-0.99800	**Spent time:**	3.95 hours

Description

What should IsDivisible(a,b) do with arguments in a field?

Currently it returns the same as not(b = 0).
This is mathematically correct, but I have just seen an example of writing a function where IsDivisible was called on elements of QQ... the programmer knew that the value was an integer but had overlooked that it was actually represented as a rational.

So... should IsDivisible give an error if handed elements of a field? If the user really wants to test not(b=0) then it is surely better to write it explicitly...?

Related issues:

Related to CoCoALib - Design #377: IsDivisible -- exact semantics?	**Closed**	**19 Jun 2013**
Related to CoCoALib - Design #1085: Fns with "OUT" args: should they give ERR...	**Closed**	**30 Jun 2017**

History

#1 - 05 Oct 2020 14:32 - John Abbott

This might be a bit like computing gcd: strictly we can define it for a field, but we opted instead to give an error.

The idea is to help the programmer avoid mistakes. What actually happened was that the discriminant of a polynomial in QQ[x] which has integer coeffs
is itself an integer, and then the program had to look for a special non-divisor of this integer... but instead the program entered an infinite loop (even though the code "looked correct").

#2 - 05 Oct 2020 14:33 - John Abbott

- Related to Design #377: IsDivisible -- exact semantics? added

#3 - 08 Oct 2020 12:35 - John Abbott

- Status changed from New to In Progress

- % Done changed from 0 to 10

I am becoming increasingly convinced that it is better to throw an error if the args are in a field (since it makes little sense to test for divisibility in a field).

Another related matter is whether IsDivisible should allow automatic ring conversion... perhaps that should be another issue?

#4 - 09 Oct 2020 09:19 - John Abbott

- Description updated

This is a bit less clear than I had previously thought. In the file ring.C the function IsDivisible (with 3 args) is used fairly widely: here is an example (around ring.C:700)

```
RingElem div_SameRing(ConstRefRingElem x, ConstRefRingElem y)
{
  const ring& Rx = owner(x);
  CoCoA_ASSERT(Rx == owner(y));
  if (IsZeroDivisor(y)) CoCoA_THROW_ERROR(ERR::DivByZero, "RingElem / RingElem");
  RingElem ans(Rx);
  if (!Rx->myIsDivisible(raw(ans), raw(x), raw(y)))
    CoCoA_THROW_ERROR(ERR::BadQuot, "RingElem / RingElem");
  return ans;
}
```

This would become annoyingly messy if we had to handle fields in a special way.

Perhaps the solution is that myIsDivisible with 3 args should have another name?

#5 - 09 Oct 2020 11:03 - John Abbott

I have tried implementing the change (*i.e.* IsDivisibile throws if given args in field).
Two tests fail: test-IsInteger1 and test-OrderedDomain1.

Mmmm. What to do?

#6 - 09 Oct 2020 20:06 - John Abbott

- Assignee set to John Abbott

- % Done changed from 10 to 20

A comment about the code excerpt in comment 4. The call to IsZeroDivisor is superfluous; perhaps it was put there to give a more informative error mesg?
Anyway, it would make more sense to call IsZeroDivisor **after** having established that division fails; certainly for arithmetic in ZZ/(N) attempting to divide.and testing for being a zero-divisor are largely the same computation, so testing IsZeroDivisor may almost double the computational cost in ZZ/(N).

For much the same reason, **it is not worth** having a non-virtual myIsDivisible which calls IsZeroDivisor, and if not then calls a virtual fn to do the actual work.

A possible solution is to give IsDivisible an optional 3rd param *e.g.* **PermitFieldElems**. In other words we offer two versions of IsDivisible: one which gives error when handed field elems, and one which does not. The default would be the version which does signal an error (to protect the unwary user).

#7 - 09 Oct 2020 21:19 - John Abbott

Also ex-RingElem1 fails...

#8 - 12 Oct 2020 20:52 - John Abbott

I still like the idea of a 3rd param (or perhaps two fns with similar names).
My preference is that a call like IsDivisible(x,y) gives error if the div-test is in a field; to permit testing also in a field the call should look "a bit more complicated" (but not too much).

An advantage of two different fns is that we do not need to define a new type to use as the 3rd param -- yes, it is effectively a boolean, but a new and specific type would make the code more readable.

If we opt for a 3rd param, there are two possible impls:

- **(A)** there are 2 poss values for the 3rd param (*e.g.* DisallowFields and AllowFields)
- **(B)** there is 1 poss value for the 3rd param (*e.g.* AllowFields)

Option **(B)** is perhaps marginally simpler to implement, but might make it slightly trickier if one wants to call a fn passing param saying which sort of div-test to perform [TBH I cannot imagine when one might want to do this]. With option **(A)** one could simply pass whichever of the two values is the desired one. If there are two distinct div-test fns, then the fn to use could be passed as param.

UPDATE: a possible alternative name could be **IsDivisible_AllowFields**; the name is long (probably a good thing), it is also clear (good!).

#9 - 14 Oct 2020 10:59 - John Abbott

Today my preference is for **IsDivisible** and **IsDivisible_AllowFields**.
I think these names are fairly clear, and would also be clear if they needed to be passed as a parameter.

Internal impl would probably use a fn IsDiv(a,b,bool) where bool indicates whether fields are allowed -- this design should maximise code sharing.

#10 - 14 Oct 2020 20:21 - John Abbott

- % Done changed from 20 to 50

Oh wow! There are a lot more IsDivisible finctions than I thought... :-(

SERIOUS QUESTION
What should the following call to IsDivisible do?

```
// Assume R1, R2, R3 are different rings: R3 can be promoted to R2 (not a field)
RingElem a(R1);
RingElem b = one(R2);
RingElem c = one(R3);
IsDivisible(a,b,c); // error or not?
```

The point is that **a is in the wrong ring.** Note that a = b/c; will succeed (and automatically change the ring of a to be R2).

#11 - 14 Oct 2020 21:21 - John Abbott

- Status changed from In Progress to Feedback

- % Done changed from 50 to 90

There are now 20 different IsDivisible functions (half of them are actually IsDivisible_AllowFields).

I have modified test-IsInteger1, test-OrderedDomain1 and ex-RingElem1 to call IsDivisible_AllowFields instead of IsDivisible. They all work now.

Regarding the question in comment 10: I have opted to make IsDivisible(a,b,c) behave like a = b/c if the division succeeds (otherwise a should remain unchanged, but I am not sure I want to guarantee that in the doc).

#12 - 23 Oct 2020 10:57 - John Abbott

- Status changed from Feedback to Closed

- % Done changed from 90 to 100

- Estimated time set to 3.99 h

#13 - 27 Oct 2020 10:22 - John Abbott

- Related to Design #1085: Fns with "OUT" args: should they give ERR::MixedRings? added