CoCoALib - Feature #248

IsDivisible for RingElem with nice interface

01 Oct 2012 13:03 - Christof Soeger

Status:	Closed	**Start date:**	01 Oct 2012
Priority:	Normal	**Due date:**	
Assignee:	Anna Maria Bigatti	**% Done:**	100%
Category:	New Function	**Estimated time:**	5.00 hours
Target version:	CoCoALib-0.99532	**Spent time:**	5.00 hours

Description

The function

```
R->myIsDivisible(r, r1, r2) -- r = r1/r2, and returns true iff division was exact
```

is of general use to programmers to avoid code like

```
if (IsDivisible(r1,r2)) { r = r1/r2 };
```

which basically would do the computation twice.

2014-03 added 3 arg syntax **IsDivisible(r, r1, r2)**

Related issues:

Related to CoCoALib - Design #377: IsDivisible -- exact semantics?	**Closed**	**19 Jun 2013**

History

#1 - 17 Oct 2012 19:05 - Anna Maria Bigatti

Maybe we could add the interface **bool IsDivisible(r, r1, r2)** which has a similar feeling to **bool IsInteger(n, x)**.

We have a similar situation (double interface) in

```
  bool IsIndet(long& index, ConstRefPPMonoidElem pp);
  bool IsIndet(ConstRefPPMonoidElem pp);
```

John?

#2 - 17 Oct 2012 20:33 - John Abbott

Anna's proposal **bool IsDivisible(r, r1, r2)** is reasonable and fits in with what we have done in similar cases so far.

Another possibility is **STRUCT IsDivisible(r1, r2)** where STRUCT is a new type, presumably containing two fields (one boolean, the other a RingElem). We can define an automatic conversion from STRUCT to bool which simply takes the value of the boolean field. Probably we could define automatic conversion to RingElem too (which would give an error if the boolean is not true). Note that C++11 allows variables to be declared with type auto which could be helpful in this case.

JAA thinks that the second suggestion is possibly more "mathematical" but may actually be more cumbersome in use. Here is a silly example of how use of the two approaches might look:

```
RingElem r;
if (!IsDivisible(r, r1, r2)) CoCoA_ERROR("Cannot divide");
cout << "Result is " << r << endl;
```

```
STRUCT ans = IsDivisible(r1, r2);
if (!ans) CoCoA_ERROR("Cannot divide");
cout << "Result is " << ans.myQuot << endl;
```

Yet another possibility is a function **RingElem divide(r1, r2)** which returns either the correct quotient or zero. This might make it easy to write compact code, but I'm not sure it's such a good idea:

```
RingElem r = divide(r1, r2);
if (IsZero(r)) CoCoA_ERROR("Cannot divide");
cout << "Result is " << r << endl;
```

Comments? Suggestions? Preferences?

#3 - 17 Oct 2012 20:37 - John Abbott

It is not really important, but I will mention that in some cases one can test divisibility much more cheaply than computing the quotient. However, I suspect that such cases would happen only rarely in programs using CoCoALib.

#4 - 18 Oct 2012 07:39 - Anna Maria Bigatti

John Abbott wrote:

> It is not really important, but I will mention that in some cases one can test divisibility much more cheaply than computing the quotient.

That's why I suggest to have both interfaces: **IsDivisible(r1,r2)** and **IsDivisible(r,r1,r2)**.
If the user does not need to use r1/r2 he will prefer the first.
But If he does he will compute it anyway.

I expect we might have other functions which may benefit from this double interface.

#5 - 20 Oct 2012 11:38 - John Abbott

OK, we can add **IsDivisible(r, r1, r2)** as Anna proposes; I'm not convinced it is the best possible interface, but at the moment I cannot produce an alternative which is clearly superior.

Question: what does this function do to the value in **r** if the quotient does not exist?

I see four likely possibilities:

1. the value in r is left unchanged
2. the value in r is set to zero (in which ring?)
3. only some vague guarantee (*e.g.* might change, might not)
4. the value in r is set to *invalid* [requires a *philosophical* change in CoCoALib]

In some ways (1) is the obvious choice (it makes me think of *exception safety*, but is actually unrelated).

For some reason (2) appeals to me; I think the ring should be the "bigger" of the owners of r1 and r2.

JAA does not like (3), the vague guarantee.

I'm not sure where idea (4) came from (perhaps a small piece of undigested dinner? - as Scrooge would say). We could also define an "invalid" ring where practically every operation produces an error; this might be handy during development/debugging if users can be warned that an uninitialized RingElem is being operated upon. Of course, it is not mathematically clean.

As usual... comments? opinions? better ideas?

#6 - 22 Oct 2012 17:52 - Anna Maria Bigatti

John Abbott wrote:

> Question: what does this function do to the value in **r** if the quotient does not exist?
> 1 the value in r is left unchanged

That's my preference, and that's what **IsInteger** does.

#7 - 22 Oct 2012 17:53 - Anna Maria Bigatti

- Category set to New Function

- Target version set to CoCoALib-0.9953

#8 - 31 Oct 2012 13:34 - Anna Maria Bigatti

Currently the function
bool IsDivisible(ConstRefRingElem num, ConstRefRingElem den)
returns **false** if den=0.
I think it should return and error anyway.

#9 - 31 Oct 2012 13:55 - John Abbott

The main question is precisely when should IsDivisible return false?

Assuming x,y,z are all RingElem...
(A) if x = y/z; would throw "inexact division" then IsDivisible will return false (and not throw)
(B) if x = y/z; would throw "division by zero-divisor" then currently IsDivisible returns false
(C) if x = y/z; would throw any other exception then IsDivisible should throw the same exception (or one which is equivalent).

The question is whether in case (B) IsDivisible should absorb the error and return false or whether it should throw "division by zero-divisor".

Just for information: the GMP function mpz_divisible_p returns false if the divisor is zero (except for 0/0 where it returns true). I note that GMP is written in C, so there is no concept of exceptions. Actually attempting to divide by zero produces a "hardware exception".

What should we do? And why?

#10 - 23 May 2013 08:12 - Anna Maria Bigatti

- Status changed from New to In Progress

- % Done changed from 0 to 50

#11 - 03 Jun 2013 17:29 - John Abbott

- Target version changed from CoCoALib-0.9953 to CoCoALib-0.99534 Seoul14

Might be able to talk about this in Osnabruck...

#12 - 29 Oct 2013 14:57 - Anna Maria Bigatti

- Target version changed from CoCoALib-0.99534 Seoul14 to CoCoALib-0.99532

#13 - 28 Jan 2014 12:13 - Anna Maria Bigatti

- Status changed from In Progress to Feedback

- Assignee set to Anna Maria Bigatti

- % Done changed from 50 to 80

All done (I think)

#14 - 01 Apr 2014 19:11 - Anna Maria Bigatti

- Status changed from Feedback to Closed

- % Done changed from 80 to 100

- Estimated time set to 4.00 h

#15 - 01 Apr 2014 19:13 - Anna Maria Bigatti

- Estimated time changed from 4.00 h to 5.00 h