CoCoALib - Design #789

NumTheory: behaviour of InvMod when inverse does not exist

26 Oct 2015 11:43 - John Abbott

Status:	Closed	**Start date:**	26 Oct 2015
Priority:	Normal	**Due date:**	
Assignee:	John Abbott	**% Done:**	100%
Category:	Safety	**Estimated time:**	5.55 hours
Target version:	CoCoALib-0.99560	**Spent time:**	5.35 hours

Description

Currently InvMod(r,m) returns 0 if there is no inverse of r modulo m.

Should it throw an exception (presumably DivByZero) instead?

History

#1 - 26 Oct 2015 11:54 - John Abbott

- Status changed from New to In Progress

- % Done changed from 0 to 50

Currently InvMod(r,m) returns 0 if no inverse exists. This is simple to implement, and can easily be used by the caller to detect when a "problem" has arisen.

Alternatively, InvMod could throw an exception when no inverse exists. This may be slightly more costly at run-time (especially if an exception is actually thrown). It is slightly harder for the caller to deal with a "problem" (since the exception must be caught in a try...catch block).

An advantage of throwing an exception is that there is less risk of a "nasty surprise". If the function "silently" returns 0, and the calling code does not check for this, then the calling code may well proceed, possibly producing a wrong answer without any warning. This cannot happen if an exception is thrown.

So I think that throwing an exception is safer.

Note that there is also the function ExtGcd which can be used to handle uniformly the cases where an inverse exists and those where one does not.

Making InvMod throw an exception is an interface change, so it could in principle break some existing code (but is there any outside CoCoALib?). I'll check CoCoALib.

#2 - 28 Oct 2015 18:29 - Anna Maria Bigatti

John Abbott wrote:

> Making InvMod throw an exception is an interface change, so it could in principle break some existing code (but is there any outside CoCoALib?).

There might be. We don't know... but that's because documentation is too good ;-)
I heard about a cocoalib user I never met (that reminds me i had to check his feedback and comments)

#3 - 16 Jun 2016 17:03 - John Abbott

- Assignee set to John Abbott

- Target version changed from CoCoALib-0.99540 Feb 2016 to CoCoALib-0.99550 spring 2017

We could also offer 2 functions: InvMod which throws, and InvModNoThrow which returns 0.

To maintain backward compatibility InvMod ought to be the name of the function which does not throw :-/

#4 - 25 Jun 2016 21:54 - John Abbott

In this case I think that the "backward compatibility" aspect is of relatively little importance.
If someone has used that aspect of (old) InvMod, and assuming we do offer InvModNoThrow then it is enough just to change the name of the fn being called (or change the code to catch ERR::DivByZero).

It seems to be too risky to allow the "standard" function InvMod to return a wrong answer "without warning" (*i.e.* throwing an exception).

#5 - 27 Jun 2016 14:05 - John Abbott

The actual implementations in NumTheory.C do throw rather than return 0 -- I do not know when the code was changed.

Perhaps I'll just update the documentation, and wait and see if anyone complains?

#6 - 27 Jun 2016 14:40 - Anna Maria Bigatti

Thinking about it: I would prefer to have 0 if it is relatively costly to determine if it is divisible.

We also have, in general

```
  bool IsDivisible(RingElem& lhs, ConstRefRingElem num, ConstRefRingElem den)
  bool IsDivisible(ConstRefRingElem num, ConstRefRingElem den)
```

In QuotientRing.C there is

```
 virtual bool myDivMod(RingElemRawPtr rawlhs, RingElemConstRawPtr rawnum, RingElemConstRawPtr rawden) const;
```

which
<quote>
return true iff quot exists & is unique; if true then lhs = num/den modulo I
</quote>
but that is a member function so we do not need full compatibility.

#7 - 27 Jun 2016 17:20 - John Abbott

The documentation was already updated to say that ERR:DivByZero is thrown if the inverse does not exist.

Note that the inverse exists iff gcd(value,modulus)==1. Computing the GCD should be slightly less costly than calling InvMod(value,modulus); the algorithms are virtually the same (but InvMod has to compute the values of a "cofactor" which will eventually give the final answer).

#8 - 27 Jun 2016 17:26 - John Abbott

- % Done changed from 50 to 60

We can indeed supply two (or more?) fns.
I like InvMod as it currently is: it can be used conveniently inside a formula (and throws in case of need).

A non-throwing version can be implemented in several ways:

- **(A)** just return 0 when inverse does not exist (*i.e.* like the old impl);
- **(B)** have a fn which returns a bool, and assigns the answer to a reference arg.

In case **(A)** possible names include InvModNoThrow, InvMod0, ...?
In case **(B)** possible names include IsInvMod, IsInvertibleMod, ...?

If we do this, JAA has a slight personal preference for a type **(A)** approach.

#9 - 27 Jun 2016 22:41 - John Abbott

Another possible interface is an optional 3rd arg to say not to throw: *e.g.* InvMod(res,mod,NoThrow). This avoids having to invent a new fn name; there will be a (tiny?) run-time cost in case the inverse does not exist (since the decision about what to do is made at run-time).

In any case, the internal implementation probably would be neatest if it did not throw "internally"; so just the publicly visible fns would potentially throw.
NOTE this is not what the impl currently does... sigh!

#10 - 28 Jun 2016 07:26 - Anna Maria Bigatti

John Abbott wrote:

> Another possible interface is an optional 3rd arg to say not to throw: *e.g.* InvMod(res,mod,NoThrow). This avoids having to invent a new name; there will be a (tiny?) run-time cost in case the inverse does not exist (since the decision about what to do is made at run-time).

I like that! and we can use it in similar cases.
Maybe we could have the NoThrow arg to be an enum with just one choice (usable in all CoCoALib), and therefore with no run-time cost. This is essentially identical to implementing InvModNoThrow(res,mod), but for some reason I have a preference for InvMod with two signatures.

> In any case, the internal implementation probably would be neatest if it did not throw "internally"; so just the publicly visible fns would potentially throw.
> **NOTE** this is not what the impl currently does... sigh!

#11 - 28 Jun 2016 19:41 - John Abbott

Here is another proposal for the interface:

- InvMod(r,m) throws if the inverse does not exist
- InvMod(r,m, flag) never throws; if the inverse does not exist, it returns 0 and sets flag (which is a bool&)

The order of the args in the second option might be different.

A possible disadvantage here is that the flag is of type bool (but we could make it a different type, perhaps an enum). With bool it is not clear to me if true means "no problem", or if it means "there was a problem".

Perhaps the NoThrow arg as suggested in comment 9 is clearer?
Comments? Opinions?

#12 - 28 Jun 2016 21:37 - Anna Maria Bigatti

John Abbott wrote:

> Perhaps the NoThrow arg as suggested in comment 9 is clearer?

clearer

#13 - 29 Jun 2016 12:42 - John Abbott

I have implemented enum ErrorActionIndicator { ThrowOnError, NoThrow };

And the functions now have a 3rd arg const ErrorActionIndicator ErrorAction = ThrowOnError.

To be honest the name of the enum is excessively long.
Also when an error is encountered the test to be made is something like if (ErrorAction == ThrowOnError). This is again rather long.

Should I aim for a more compact solution?

#14 - 29 Jun 2016 14:45 - Anna Maria Bigatti

I'd rather have (as I said in comment #10)

```
NoThrowMarker { NoThrow };
```

#15 - 29 Jun 2016 15:01 - John Abbott

@Anna: your approach is quite possible but it is more awkward than what I have implemented.

My approach has an optional 3rd arg (*i.e.* always present but with a default value); this means that I just had to modify each of the 5 signatures (and the corresponding impls slightly).

Your approach requires me to add 5 new signatures (for a total of 10); also the impls become slightly more indirect. To avoid duplicating code, the public fns would presumably call the "real" fns which actually do the work with a "boolean" flag saying what to do in case of error.

In summary: it can be done your way, but I cannot currently see any neat way of implementing it.

#16 - 29 Jun 2016 15:12 - Anna Maria Bigatti

John Abbott wrote:

> Your approach requires me to add 5 new signatures (for a total of 10);

good point!

#17 - 29 Jun 2016 15:15 - John Abbott

I have just checked in my impl. Perhaps Anna could have a look at it, and decide whether she thinks it is comprehensible enough.

I have done some tidying in NumTheory.C (*e.g.* removing useless arg checks).
Everything compiles, and all tests pass here :-)

#18 - 29 Jun 2016 15:18 - John Abbott

Currently the enum ErrorActionIndicator is defined inside NumTheory.H; probably it should be moved to utils.H so other files can use it...?

#19 - 29 Jun 2016 15:21 - Anna Maria Bigatti

John Abbott wrote:

> Currently the enum ErrorActionIndicator is defined inside NumTheory.H;

May we call it enum ThrowFlag {Throw, NoThrow}?

> probably it should be moved to utils.H so other files can use it...?

yes!

checked-out, compiles, run, comprehensible enough.

#20 - 29 Jun 2016 15:32 - John Abbott

I have just thought of an argument in favour of Anna's proposed design (namely, different signatures).
In C++11 there is a noexcept specifier which can be used to specify that a function does not throw. With Anna's proposal this can be used for the "NoThrow" versions of the fns; with my proposal it cannot be used at all.

I don't know what this really means in practice; supposedly noexcept functions can run a bit faster, and perhaps produce smaller compiled code.

I wonder what we should do?

#21 - 29 Jun 2016 17:42 - John Abbott

If we do adopt the idea of different signatures for throwing and non-throwing versions, which approach is better:

- **(A)** different fn names: InvMod(r,m) and InvModNoThrow(r,M)
- **(B)** extra arg: InvMod(r,m) and InvMod(r,m,NoThrow)

To me, approach **(A)** seems slightly simpler (but the user has to remember an extra function name, though it ought to be easy to remember). The name could be slightly different: InvMod_NoThrow.

Approach **(B)** might encourage us to be coherent if we need throwing and non-throwing versions of several functions -- can you think of any other candidate fns?

I wonder which approach would be easiest to document, or perhaps there is no real difference?

#22 - 29 Jun 2016 17:54 - John Abbott

Perhaps the best approach is not to worry about noexcept at the moment since we have no experience of the feature. We could note this issue as one to be revisited when we switch to C++11, and are familiar with its features.

What do you think?

Is the current CVS version acceptable for the time being?

#23 - 30 Jun 2016 13:59 - John Abbott

I am still a bit perplexed by this issue.

Having InvMod always throw seems to me to be the cleanest and safest solution.

An advantage of the non-throwing version is that a caller can easily then choose to throw its own exception: for instance InvMod is called inside PowerMod. Currently this calls the throwing version of InvMod, so the exception will report that it was thrown by InvMod; a previous version of PowerMod checked the return value of InvMod, and would throw if there was a problem -- the exception would then say that it was thrown by PowerMod. Is this difference important?

When I have to debug code which unexpectedly threw an exception, I normally set a break-point inside CoCoA::ThrowError and then "walk up the stack"; so it would make little difference which routine actually threw.

Undecided :-/

#24 - 16 Sep 2016 16:40 - John Abbott

- Target version changed from CoCoALib-0.99550 spring 2017 to CoCoALib-0.99560

Although this possibly already almost finished, I prefer to postpone it since there are some questions still unanswered.

It would be nice to be future-compatible with noexcept...

#25 - 08 Nov 2017 17:44 - John Abbott

- Status changed from In Progress to Closed

- % Done changed from 60 to 100

- Estimated time set to 5.55 h