CoCoALib - Bug #853

NearestInt can needlessly throw InsufficientPrecision

23 Mar 2016 16:52 - John Abbott

Status:	Closed	**Start date:**	23 Mar 2016
Priority:	Normal	**Due date:**	
Assignee:	John Abbott	**% Done:**	100%
Category:	Maths Bugs	**Estimated time:**	7.70 hours
Target version:	CoCoALib-0.99550 spring 2017	**Spent time:**	7.55 hours

Description

The current impl of NearestInt with an arg from a RingTwinFloat can throw InsufficientPrecision when it should not.

The cause is a speculative call to IsRational which does not catch the error.

Fix impl of NearestInt.

Related issues:

Related to CoCoALib - Support #696: test-OrderedRing: activate or eliminate?	**Closed**	**08 May 2015**
Related to CoCoALib - Bug #858: floor for TwinFloat can produce ERR::SERIOUS	**Closed**	**26 Mar 2016**

History

#1 - 23 Mar 2016 16:53 - John Abbott

- Related to Support #696: test-OrderedRing: activate or eliminate? added

#2 - 23 Mar 2016 17:12 - John Abbott

- Status changed from New to In Progress

- Assignee set to John Abbott

- % Done changed from 0 to 10

What properties should the result of NearestInt have?

- an obvious property is abs(NearestInt(X) - X) <= 1/2

What should happen to halves?

Recall that there is a function called RoundDiv(N,D) and a closely related function round(q) for a rational number. It seems reasonable to expect that NearestInt(X) = round(Q) if IsRational(Q,X); or equivalently, given a rational number Q then round(Q) = NearestInt(RingElem(R,Q)) for any ring R where NearestInt(R,Q) is defined.

How to implement NearestInt so that it is surely compatible with round?
The hard case is when the ring is RingTwinFloat.

#3 - 23 Mar 2016 17:29 - John Abbott

Which is the better design?

- **(A)** A normal C++ fn (as currently implemented), or
- **(B)** mem fns for the rings which actually need to implement it.

An advantage of **(A)** is that all the code is together so it should be (relatively) easy to see that halves are rounded consistently for all rings.

An advantage of **(B)** is that the impls can exploit knowledge of the explicit internal representation of the values.
A disadvantage of **(B)** is that every ring must have an impl of myNearestInt even when this makes no sense; of course, there can be a default impl which gives an error (but that is a bit ugly).

#4 - 25 Mar 2016 14:54 - John Abbott

Currently the only ordered domains are: RingZZ, RingQQ (since a FractionField of an ordered domain is an ordered domain), and RingTwinFloat.

A disadvantage of implementation approach **(A)** in the previous comment is that the function must (really ought to be) be reviewed every time a new ordered domain is added to make sure that it works properly also for the new ring. JAA is now inclined to think that this disadvantage outweighs the disadvantages of approach **(B)**.
[of course this means rewriting the current implementation]

#5 - 25 Mar 2016 14:59 - John Abbott

I am thinking of making the test for NearestInt of twin float values work as follows:
fix an integer N, for ever smaller positive values of eps verify that after setting x = N+eps we have that NearestInt(x) == N gives exactly the same response as N <= x && x < N+1. Repeat also for values of eps = 1- eta where eta is small and positive.

Do the test for N small and positive, large and positive, small (abs val) and negative, large (abs val) and negative.

#6 - 25 Mar 2016 15:12 - Anna Maria Bigatti

John Abbott wrote:

> Which is the better design?
>
> - **(A)** A normal C++ fn (as currently implemented), or
> - **(B)** mem fns for the rings which actually need to implement it.

can't we have a member function with an abstract implementation (or error), and concrete implementations only for special cases (i.e. TwinFloat)?

#7 - 25 Mar 2016 22:04 - John Abbott

I think I have fixed the problem. I have completely rewritten the code to do with floor, ceil and NearestInt.

Part of the problem was a test something like abs(x-N) <= 1/2 where the problem was x-N would throw InsuffPrec because the result could not be calculated with the precision required by the twin-float ring (even though it could be calculated quite accurately enough to recognize that the difference was less than 1/2.

A new, clearer test now passes as I expect it to. So marking this as in feedback.

#8 - 25 Mar 2016 22:11 - John Abbott

- Status changed from In Progress to Feedback

- % Done changed from 10 to 90

I followed design **(B)**; we just have to be careful if we want to change policy about how halves are handled. Perhaps some vagueness in exactly how they are handled is not such a bad thing?

#9 - 26 Mar 2016 11:03 - John Abbott

- Related to Bug #858: floor for TwinFloat can produce ERR::SERIOUS added

#10 - 30 Mar 2016 18:23 - John Abbott

- Estimated time set to 7.70 h

I have cleaned the impl considerably; it is now obviously correct (but possibly slower than necessary as a wasteful temporary is created).

Maybe I'll try a version without the temporary to see if it is usefully faster (but the code will be less clear).

#11 - 25 Jun 2016 12:19 - John Abbott

- Status changed from Feedback to Closed

- % Done changed from 90 to 100

This has been in feedback for 3 months without any reports of problems.
Apparently the (new-ish?) test test-OrderedRing2.C covers the problems reported here.
Closing!

#12 - 05 Oct 2016 16:33 - John Abbott

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