CoCoALib - Feature #221

Better RingElems

08 Aug 2012 20:22 - John Abbott

Status:	Closed	**Start date:**	08 Aug 2012
Priority:	High	**Due date:**	
Assignee:	John Abbott	**% Done:**	100%
Category:	Various	**Estimated time:**	0.00 hour
Target version:	CoCoALib-0.9952	**Spent time:**	8.80 hours

Description

Bruns suggests:

- allow default ctor for RingElem (value could be 0 in ZZ)
- allow assignment of RingElem to change rings (no technical objections)

JAA estimates: both changes should be simple to achieve, and will probably not introduce any technical problems.

Note that my suggestion for the default value of a RingElem should work well if/when we introduce automatic mapping of RingElem values

Related issues:

Related to CoCoALib - Feature #223: Automatic mapping of RingElems	**Closed**	**08 Aug 2012**
Related to CoCoALib - Feature #244: Rings: default ctor & assignment	**Closed**	**28 Sep 2012**
Related to CoCoALib - Feature #253: W.Bruns's wish list	**Closed**	**04 Oct 2012**
Related to CoCoALib - Feature #261: Review the utility of RefRingElem	**Closed**	**10 Oct 2012**

History

#1 - 08 Aug 2012 20:47 - John Abbott

At the moment JAA **sees no advantage** to the current "restrictive" regime for RingElem values.

JAA is not sure if he recalls why we introduced the restriction that a RingElem is eternally associated with a single specific ring. He thinks it may have been vaguely "for reasons of safety".

If we really want to preserve the current restrictions, then JAA notes that it would be technically fairly simple to use a global flag to say whether to be restrictive or permissive.

Since CoCoA-5 does not impose restrictions about "changing ring", it seems hard to justify saying that we need such a restriction in CoCoALib.

#2 - 03 Sep 2012 18:01 - Anna Maria Bigatti

- Category set to Various

The "safety" in assigning only RingElems in the same ring was (probably) to prevent mixing rings in one algorithm.
Anyway I see no strong problem in allowing the change of ring in an assignment.
The only problem I see here is in this example:

```
RingElem f(Qxy);
f = 3;
```

Is now f in ZZ or Qxy?

Of course one could say: "Simple, put it in ZZ and allow mixed arithmetics!" (i.e. between different rings), but that's like opening a can of worms...

#3 - 03 Sep 2012 18:35 - John Abbott

What should the following code excerpts do?

Case (A)

```
RingElem r; // by default an element of RingZZ
r = BigRat(2,1);
```

Case (B)

```
RingElem r; // by default an element of RingZZ
r = BigRat(2,3);
```

Assignment from a BigRat is allowed, and JAA believes it would be inconvenient to forbid it.

Case (A) should surely work; the only question is to which ring r belongs after the assignment (namely ZZ or QQ).
Case (B) might work or might give an error; if it works then after the assignment x can no longer belong to RingZZ, so would presumably belong to RingQQ.

We can make the ring evident by using an explicit constructor like this:

```
x = RingElem(RingQQ(), BigRat(2,3));
```

NB we cannot mimic CoCoA-5 because there the type of the value changes from RINGELEM to RAT.

#4 - 24 Sep 2012 15:45 - John Abbott

This may not be the correct place, but I wanted to note that until we have automatic conversions (if ever), an equality test between elements of

different rings will produce an error!

Here is an example scenario (which is not too far-fetched).

```
RingElem x(R);
RingElem y(R);
if (x == y) ... // no error here
x = 1;  // to which ring does x belong now?  Either R or ZZ
if (x == y) ... // this will give error if RingOf(x) is not R
```

This makes me think that (the details of) the current issue are closely related to whether we intend implementing automatic "promotion" of ring elements.

I suggest the following rule/guideline:

- assignment changes the ring if and only if the assigned value indicates **explicitly** the ring to use

(*i.e.* in practice the only possibility currently is when the RHS is itself a RingElem).

#5 - 25 Sep 2012 15:56 - John Abbott

Anna, Christof and John all agree that:

- the default ctor for RingElem produces 0 in ZZ
- the rule/guideline in note-4 is good.

We propose implementing in the near future.

#6 - 25 Sep 2012 16:00 - John Abbott

- Status changed from New to In Progress

- Assignee set to John Abbott

- Priority changed from Normal to High

- Target version set to CoCoALib-0.9952

- % Done changed from 0 to 50

#7 - 26 Sep 2012 12:26 - John Abbott

If we allow "mixed ring" assignment then we should also allow "mixed ring" **swap**.

JAA would expect **swap** to work as if it were implemented like this:

```
void swap(A,B)
{
  tmp = A;
  A = B;
  B = tmp;
}
```

#8 - 26 Sep 2012 12:31 - Anna Maria Bigatti

John Abbott wrote:

> If we allow "mixed ring" assignment then we should also allow "mixed ring" **swap**.

I agree

#9 - 28 Sep 2012 14:20 - John Abbott

Case (A) We have decided that this code:

```
  RingElem r;
```

initializes r as being the zero of **ZZ**

Case (B) What should this code do?

```
  RingElem r = 3;
```

Case (C) And what about this?

```
  RingElem r(3);
```

If I recall well, the rules of C++ say that cases **(B)** and **(C)** either both fail to compile or both succeed (since they trigger effectively the same ctor).

#10 - 28 Sep 2012 17:08 - John Abbott

To implement assignment of RingElem which can change the ring, I need to allow **ring** to be assigned -- currently assignment is disabled.

Any objections to allowing assignment of values of type ring?

JAA does not anticipate any *nasty surprises* suddenly appearing.

#11 - 28 Sep 2012 17:33 - Anna Maria Bigatti

John Abbott wrote:

> Any objections to allowing assignment of values of type ring?

ok for me: in fact I had to do use auto-prt tricks to delay initialization somewhere.
I could clean them up :-)

#12 - 02 Oct 2012 12:21 - John Abbott

Note that assignment to a **RefRingElem** cannot change the ring to which it belongs because it may well be a reference to a value inside some other homogeneous structure (*e.g.* a matrix). Perhaps the choice of name RefRingElem is not ideal any more?

#13 - 02 Oct 2012 17:42 - John Abbott

Non homogeneous assignment of RingElem is trickier than I thought (because of the inheritance from **RefRingElem**).

Currently I'm stuck.

#14 - 03 Oct 2012 18:28 - John Abbott

- % Done changed from 50 to 70

I've solved yesterday's problem (but still have not understood what caused it).

I was about to update the documentation but then noticed that it is not clear what the following should do:

```
RingElem x0;  // this *is* defined
RingElem x1(2);
RingElem x2 = 2;
RingElem x3 = BigRat(2,1);
RingElem x4 = BigRat(2,3);
```

We now have that x0 will be initialized to 0 in ZZ; all the others will not currently compile.

When no ring is specified in the default ctor, the system automatically chooses **ZZ**. So should it automatically choose **ZZ** whenever no ring is explicitly given to a ctor? If we adopt this rule then the line with x4 will compile but produce a run-time error.

We could also allow **QQ** to be chosen automatically, so that the x4 line would work. Though if **QQ** may also be chosen, it is not clear to me in which ring x3 will lie. Its initial value happens to be in **ZZ**, but it was a value of type BigRat.

Opinions? Ideas?

2013-02-18: DECISION according to the code and doc, I decided to KISS: if no ring is specified then the ring used is ZZ (so the x4 line above will cause a run-time error).

#15 - 05 Oct 2012 17:29 - John Abbott

2013-02-18: The problem presented in this post has been resolved by removing the class RefRingElem (see #261)

There is a potential problem with "ring changing" assignment/swap and RefRingElem. The difficulty is highlighted by the following two examples:

(A) This example has been valid for some time, and will print 2

```
RingElem x(R,1);
RingElem y(R,2);
RefRingElem z(x);
x = y;
cout << "z=" << z << endl;
```

(B) We get **DANGEROUS** behaviour if the rings of x and y are different:

```
RingElem x(R1,1);
RingElem y(R2,2);
RefRingElem z(x);
x = y;
cout << "z=" << z << endl;
```

The assignment to x deletes the old value contained, so z subsequently contains a dangling reference/pointer to the space where the original value of x was stored!

Recall that the ring to which z belongs cannot be changed, and it would require a significant redesign to produce a mechanism for informing z (and any other references) that it is now dangling.

What to do???

#16 - 26 Oct 2012 17:14 - John Abbott

- Status changed from In Progress to Resolved

- % Done changed from 70 to 90

Updated the doc for RingElem.

Might be necessary to revise the example progs?

#17 - 18 Feb 2013 12:57 - John Abbott

- Status changed from Resolved to Closed

No problems have surfaced in 4 months; the doc seems to be correct too.
So I'm closing this issue.

#18 - 18 Feb 2013 13:04 - John Abbott

- % Done changed from 90 to 100