CoCoALib - Feature #342

Remove denominators: QQ[x] -> ZZ[x] (and PushBack(coeff, PP))

17 Apr 2013 17:32 - Anna Maria Bigatti

Status:	Closed	**Start date:**	17 Apr 2013
Priority:	Normal	**Due date:**	
Assignee:	John Abbott	**% Done:**	100%
Category:	New Function	**Estimated time:**	3.00 hours
Target version:	CoCoALib-0.99533 Easter14	**Spent time:**	3.00 hours

Description

For many computations QQ[x] it is very convenient to remove denominators and perfom computations in ZZ[x].
There are already some implementations (e.g. WithoutDenominators, hidden in TmpGReductor.C, and another by W.Bruns).

Design a **flexible** and **efficient** function doing this, and document it clearly (explaining why QQ[x] is intrinsically more costly that ZZ[x])

Scope for improvement: copy the PP instead of recreating it using exponents (function for doing so is not yet implemented)

Related issues:

Related to CoCoALib - Feature #253: W.Bruns's wish list	**Closed**	**04 Oct 2012**
Related to CoCoA - Support #425: Osnabrueck 2014-01	**Closed**	**27 Jan 2014**

History

#1 - 29 Oct 2013 15:19 - Anna Maria Bigatti

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

#2 - 30 Oct 2013 13:32 - Winfried Bruns

This is the first i am using redmine actively.

In my version of QQ[x] -> Z[x] I am using PushBack to build the target polynomial (also for th converse operation). One of the arguments is the exponent vector. Does it get decomposed and reconstructed, or is it simply copied?

Winfried

#3 - 30 Oct 2013 18:50 - Anna Maria Bigatti

- Subject changed from Remove denominators: QQ[x] -> ZZ[x] to Remove denominators: QQ[x] -> ZZ[x] (and PushBack(coeff, PP))

- Status changed from New to In Progress

- % Done changed from 0 to 10

Winfried Bruns wrote:

> In my version of QQ[x] -> Z[x] I am using PushBack to build the target polynomial (also for th converse operation). One of the arguments is the exponent vector. Does it get decomposed and reconstructed, or is it simply copied?

The function hidden in TmpGReductor.C uses PolyRingHom which (I think) is even "worse" than decomposing the PP into the exponents and reconstructing it.
My guess is that we can implement PushBack accepting a PP and making a copy of the PP instead of passing through the exponents. (and change **PolyRingHom** too)
The philosophical question really is: the PP
(1) must be in the same PPMonoid?
(2) may be in any PPMonoid and silently converted?
version (2) is easier for the user, but hides the fact that it is much more costly if one has the wrong PPMonoid. Moreover in that case it is more

efficient if
So I vote for (1).
The follow-ups of this new **PushBack** implementation are so many that it is quite scary to start ;-)

#4 - 30 Oct 2013 19:04 - Winfried Bruns

Dear Anna,

I agree that (1) is the better choice. Given that PushBack copies the exponent vector, I think it is the best solution for the replacement of coefficients. It is clear that every term must be touched, and I don't think it is possible to exchenge the coefficient in a given term.

Winfried

#5 - 30 Oct 2013 21:43 - John Abbott

I, too, favour proposal (1):

- it is far simpler to implement
- it is simple to understand the restriction
- it should have good execution speed
- there will be no questions about the PP ordering being right (& no overflow problems converting the PP)
- it (probably?) suffices in most cases (???)

If proposal (1) turns out to be too restrictive, we can always change to (2) and all existing code will still work.

The KISS philosophy!

#6 - 20 Nov 2013 19:51 - John Abbott

- % Done changed from 10 to 50

PushFront and PushBack for PPs have already been implemented (no idea when), but they were not documented. I have now added the documentation.

#7 - 29 Jan 2014 14:48 - John Abbott

- % Done changed from 50 to 60

Regarding the conversion QQ[x] to ZZ[x] clearing denominators, JAA thinks the following is a reasonable approach:

1. Compute RingElem D = CommonDenom(f);
2. Rescale: f *= D;
3. Apply a (partial) poly algebra hom from QQ[x] to ZZ[x]

I suppose we could wrap this into a function, perhaps called CommonNumer???

What do you think?

20140130 I now think that **ClearDenom** might be a better name :-)

#8 - 30 Jan 2014 17:18 - John Abbott

- Assignee set to John Abbott

- % Done changed from 60 to 80

Anna pointed out that is already a fn **ClearDenom** which does almost what is wanted: it multiplies the polynomial by the CommonDenom (but does not move it to a new ring).

Below is an example of a function which uses **ClearDenom** to rescale the polynomial, and then copies it into the new ring (which must have the same PPMonoid).

```
RingElem ClearDenom2(const SparsePolyRing& Zx, const RingElem& f)
{
  CoCoA_ASSERT(PPM(Zx) == PPM(AsSparsePolyRing(owner(f))));
  RingElem g = ClearDenom(f);
  RingElem ans(Zx);
  for (SparsePolyIter it=BeginIter(g); !IsEnded(it); ++it)
    PushBack(ans, num(coeff(it)), PP(it));
  return ans;
}
```

A similar routine which avoid creating g, and which supplies a rescaled coeff takes about 0.7 times as long as this fn (on longer polys).

#9 - 08 Feb 2014 10:43 - Winfried Bruns

This function seems to be optimal.

What is the authorative function for the converse transformation ZZ[X] --> QQ[X] ?

#10 - 12 Feb 2014 16:30 - Winfried Bruns

My non-authorative solution is

```
RingElem makeQQCoeff(const RingElem& F, const SparsePolyRing R){
// F is a polynomial over RingZZ
// This function converts it into a polynomial over RingQQ
    SparsePolyIter mon=BeginIter(F); // go over the given polynomial
    RingElem G(zero(R));
    for (; !IsEnded(mon); ++mon){
        PushBack(G,RingElem(RingQQ(),coeff(mon)),PP(mon));
    }
    return(G);
}
```

Winfried

#11 - 01 Apr 2014 19:29 - Anna Maria Bigatti

- Target version changed from CoCoALib-0.99532 to CoCoALib-0.99533 Easter14

#12 - 15 Apr 2014 15:39 - John Abbott

- Status changed from In Progress to Closed

- % Done changed from 80 to 100

A new **ClearDenom** function has been added to SparsePolyRing; it follows very closely the code ClearDenom2 I wrote in comment-8.

Closing.

#13 - 17 Apr 2014 09:23 - Anna Maria Bigatti

- Estimated time set to 3.00 h