CoCoALib - Design #683

Module index component in internal compressed representation

14 Apr 2015 13:50 - Anna Maria Bigatti

Status:	Closed	**Start date:**	14 Apr 2015
Priority:	Normal	**Due date:**	
Assignee:	Anna Maria Bigatti	**% Done:**	100%
Category:	Documentation	**Estimated time:**	10.00 hours
Target version:	CoCoALib-0.99560	**Spent time:**	8.80 hours

Description

The internal representation of module in GBasis computations is through an embedding into a polynomial ring.
The index i is encoded as exponent "10000 - i" (I'm skipping all the details).
This causes problms when using CoCoALib with 8-bit exponents (unsigned char).
(Found by John Abbott while tackling a huge computation with small exponents)

Related issues:

Related to CoCoALib - Design #268: Exponent range (in power products)	**Closed**	**18 Oct 2012**
Related to CoCoALib - Feature #269: PPMonoids: check for exponent overflow in...	**Closed**	**18 Oct 2012**
Related to CoCoA-5 - Bug #275: Unhelpful error messages when SmallExponent_t ...	**Closed**	**15 Nov 2012**
Related to CoCoALib - Design #707: MatrixOrderingMod32749Impl: test and write...	**In Progress**	**15 May 2015**

History

#1 - 14 Apr 2015 13:57 - Anna Maria Bigatti

I think 10000 can be replaced by max_exp (how is it called in cocoalib?)
I don't think it is ever used in operations with other exponents, but in that case using max_exp instead of 10000 should break immediately ;-) Could you test it with all debugging on?

It is still ugly to use a hard-coded constant, but indeed max_exp would cover all cases which can be represented in that way.

#2 - 14 Apr 2015 15:20 - John Abbott

OK I can try doing the tests (sometime over the next few days).

#3 - 15 Apr 2015 20:22 - John Abbott

- Status changed from New to In Progress

- Target version set to CoCoALib-1.0

- % Done changed from 0 to 10

Is there a MaxExp function? At least one PPMonoid has no upper limit on exponents; what value should be used in that case? Perhaps 0? If so, the max number of components could then be arbitrarily set to 2^32-1 (or even just 1000000); it's hard to imagine how
anyone could compute with a larger module.

#4 - 15 May 2015 18:43 - Anna Maria Bigatti

- Assignee set to Anna Maria Bigatti

- % Done changed from 10 to 30

- Estimated time set to 10.00 h

Bug probably found.... my fault
John has implemented in GBEnv.C

```
/*static*/ const long GRingInfo::myMaxComponentIndex = numeric_limits<SmallExponent_t>::max()/2; // max num of
 compts -- depends on type SmallExponent_t
```

and this failed, showing that my guess for max_exp was wrong.
But this allowed making many tests very quickly (just compiling 1 .C file), so I found that the strange limit was **32748** instead (??!?!?). After one day of tests, and noticing how coherently it was going even when failing I decided to grep for **327**... and I found

```
  OrdvArith::MatrixOrderingMod32749Impl::MatrixOrderingMod32749Impl(long NumIndets, long GradingDim, const mat
rix& OrderMatrix):
```

where **32749** is the biggest prime with some guarantee for <I-cant-remember-what>, so the order matrix is actually over ZZ/(32749) instead of over ZZ. I think this was for fast arithmetic with matrix-defined orderings.
This seems to be the choice (instead of MatrixOrderingImpl) in

```
  OrdvArith::reference NewOrdvArith(const PPOrdering& PPO)
```

It is worth investigating whether it is worth it, safe, and write proper documentation... I'll do it (opening a new redmine issue)

#5 - 15 May 2015 19:28 - Anna Maria Bigatti

- Status changed from In Progress to Resolved

tested both with unsigned int and unsigned char: it works!!

```
  /*static*/ const long GRingInfo::myMaxComponentIndex = min((long)32748,(long)numeric_limits<SmallExponent_t>
::max()-1);
```

#6 - 25 Jul 2015 17:10 - John Abbott

The definition on line GBEnv.C:48 does not work with SmallExponent_t being unsigned long. JAA is fixing now.

#7 - 25 Jul 2015 17:34 - John Abbott

I shall check in shortly; anyway here is the modified line:

```
  /*static*/ const long GRingInfo::myMaxComponentIndex = min(32748ul,
                                                            min(static_cast<unsigned long>(numeric_limits<lon
g>::max()-1),
                                                                static_cast<unsigned long>(numeric_limits<Sma
llExponent_t>::max()-1))); // max num of compts -- depends on type SmallExponent_t
```

NOTE checked in now :-)

#8 - 23 Mar 2016 17:40 - Anna Maria Bigatti

- Target version changed from CoCoALib-1.0 to CoCoALib-0.99550 spring 2017

#9 - 02 Aug 2016 11:03 - John Abbott

- % Done changed from 30 to 80

#10 - 20 Sep 2016 18:20 - Anna Maria Bigatti

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

#11 - 15 Dec 2017 16:37 - John Abbott

- Status changed from Resolved to Closed

- % Done changed from 80 to 100

The code has been unchanged for over a year, and no problems have yet arisen (because no one has tried computing with a module having more than 32000 components?).

Someone ought to check whether there are safeguards against creating modules with too many components... sigh.
Next time, maybe?

I'm fed up with this issue. Closing.