CoCoA-5 - Slug #798

use poly ring with many variables is too slow

09 Nov 2015 11:38 - John Abbott

Status:	Closed	**Start date:**	09 Nov 2015
Priority:	Normal	**Due date:**	
Assignee:	John Abbott	**% Done:**	100%
Category:	enhancing/improving	**Estimated time:**	0.00 hour
Target version:	CoCoA-5.2.2	**Spent time:**	3.15 hours

Description

Mario will need polyrings with many variables (>10000).
CoCoALib seems to have no "artificial limit"; but when I tried use QQ[x[1..10000]]; in CoCoA-5 it took almost 400s to create the ring.
That is too slow!

Related issues:

Related to CoCoALib - Feature #800: PPMonoidSparse: impl of sparse PPs	**Closed**	**09 Nov 2015**

History

#1 - 09 Nov 2015 11:41 - John Abbott

I suspect the main problem is that creating a poly ring also creates a std::vector containing each power-product as a length 1 polynomial. This is quite slow using a dense repr for the power-products.

Perhaps the default type of power product should change to a sparse PP repr when the number of indets exceeds a given limit (what value should this limit have?)

I also note that creating the ring also consumed about 1.5Gbytes of RAM. :-(

#2 - 09 Nov 2015 13:24 - John Abbott

- Status changed from New to In Progress

- % Done changed from 0 to 10

I have just tried NewPolyRing(RingQQ(),10000) in CoCoALib, and it took just 4s.
So why so much slower in CoCoA-5?

I have also tried using the profiler (on the flashy new Kassel machine), and "discovered" that the PPMonoid also creates a vector of PPs, one for each indet. Is this ever really useful? (see #799)

#3 - 09 Nov 2015 13:52 - John Abbott

I was forgetting that the use command in CoCoA-5 also assigns all the CoCoA-5 variables whose names coincide with those of the indets. It must be this which is causing CoCoA-5 to be so slow compared to CoCoALib!

More profiling to find out what is so costly, and how (if possible) this can be avoided or mitigated.

#4 - 09 Nov 2015 15:02 - John Abbott

Functions involved include:
RING::injectIndeterminates in Interpreter.C:456 which calls
allSymbolValues in Interpreter.C:434 which calls

createAllSymbolValues in Interpreter.C:440 which calls
ctor for RingElem from a symbol which calls
RingBase::myNew (from a symbol) which calls
AreDistinct as a "cheat" way of testing whether the symbol is in the ring.

AreDistinct sorts the vector then checks whether two adjacent symbols are equal, via op= which calls cmp for symbol.
op= computes its answer by calling symbol::myCmp.
symbol::myCmp is called about 6*10^9 times!! 8-0

This is unacceptable!

#5 - 09 Nov 2015 17:03 - John Abbott

In PolyRing.H there is a fn called IndetSymbol which apparently gives the symbol of the k-th indet. Presumably createAllSymbolValues could iterate over integer indices, rather than iterate over the symbols.

But how to descend into subrings?

#6 - 03 Dec 2015 11:10 - John Abbott

One possible approach is to devise a hash for symbol then just comparing hashes will probably show quickly that they are different. I wonder how much it would cost to compute a hash of a symbol?

Nevertheless I do feel that the interpreter should create the values by using indet rather than constructing an RingElem from a symbol; this should avoid doing many comparisons between symbol values.

NOTE: on the new Kassel machine I confirmed the cause of slowness:

```
t0 := CpuTime();
P ::= QQ[x[1..10000]];
TimeFrom(t0);  --> 0.73s
t1 := CpuTime();
use P;
TimeFrom(t1);  --> 126s
```

A further comparison: with 20000 indets instead of 10000 -- complexity is quadratic :-(

```
t0 := CpuTime();
P ::= QQ[x[1..20000]];
TimeFrom(t0);  --> 2.6s
t1 := CpuTime();
use P;
TimeFrom(t1);  --> 515s
```

#7 - 03 Dec 2015 11:44 - John Abbott

- Assignee set to John Abbott

- % Done changed from 10 to 50

I didn't sleep so well this last night (ate something which didn't agree with me), so today's a good day to hack away at the code of the interpreter.
Here is the new version of createAllSymbolValues; it "cheats" in that it uses the old slow method for symbols in the coeff ring, but uses a faster way for the indets.

```
void RING::createAllSymbolValues(const ring &R, std::vector<SymbolPair> &allIndets)
{
  vector<symbol> syms;
  if (IsPolyRing(R)) syms = symbols(CoeffRing(R)); else syms = symbols(R);
  BOOST_FOREACH(const symbol &s, syms)
    allIndets.push_back(make_pair(s, new RINGELEM(RingElem(R, s))));
  if (!IsPolyRing(R)) return;
  const long nvars = NumIndets(R);
  allIndets.reserve(allIndets.size() + nvars);
  for (long i=0; i < nvars; ++i)
    allIndets.push_back(make_pair(IndetSymbol(R,i), new RINGELEM(indet(R,i))));
}
```

Now the computation time for use P; is about 2.5s for 20000 indets! :-)

#8 - 03 Dec 2015 13:18 - John Abbott

- Status changed from In Progress to Resolved

- Target version changed from CoCoA-5.?.? to CoCoA-5.1.3/4 Jan 2016

- % Done changed from 50 to 80

The CoCoA-5 tests pass as they did before; and the speed test with 20000 indets seemed to go OK. So marking as resolved.

#9 - 17 Feb 2016 13:00 - John Abbott

- Target version changed from CoCoA-5.1.3/4 Jan 2016 to CoCoA-5.2.0 spring 2017

Not closing this issue; that hack I made doesn't seem all that convincing now. I guess it's OK in simple cases, but it should be properly checked -- that could take 2-3 days if any non-hackish changes are needed, so no time to do it now.

#10 - 27 Apr 2017 15:01 - John Abbott

- Target version changed from CoCoA-5.2.0 spring 2017 to CoCoA-5.2.2

I do not like the fact that the last comment expressed doubt... but obviously I was too lazy to give any details. As it will take time to check what the problem is (if any), I prefer to postpone this issue.

#11 - 13 Dec 2017 14:28 - John Abbott

- Status changed from Resolved to Closed

- % Done changed from 80 to 100

Presumably we've been using this code for a couple of years without any problems arising.

Indeed the solution is not perfect, but it seems to work well enough in the cases which actually arise. There will be problems (slugs) if someone wants to compute in the fraction field of a poly ring with many thousands of indets.

Closing.