CoCoA-5 - Bug #1514

Cocoa crashes when calling RingElems

22 Oct 2020 16:42 - Julian Danner

Status:	Closed	**Start date:**	22 Oct 2020
Priority:	High	**Due date:**	
Assignee:	John Abbott	**% Done:**	100%
Category:	enhancing/improving	**Estimated time:**	4.33 hours
Target version:	CoCoA-5.4.0	**Spent time:**	4.30 hours

Description

Hi,

I have a large polynomial system that has to be read from a file. In order to do so, I load all the polys as comma-separated-polynomials into a string and then call RingElems on it.
Unfortunately, this crashes CoCoA-5 on my machine if the file is large (~20MB) and the system has many indets (>>500).

Below you can find a script that generates a polynomial (linear) system for which cocoa also crashes on my notebook.

```
F2:=NewZZmod(2);
R::=F2[x[1..1000]];

ln:=20*250; --250 elems correspond to approx. 1MB file size
--generate large file
D:=OpenOFile("tmp");
for i:=1 to ln-1 do
  print Sprint(RandomLinearForm(R))+"," on D;
endfor;
print Sprint(RandomLinearForm(R)) on D;
close(D);

--read file with RingElems
D:=OpenIFile("tmp");
l:=GetLine(D);
L:=RingElems(R, l); -- here CoCoA crashes without any output...
close(D);
```

(Starting cocoa5 with --fullCoCoALibError only prints out Killed.)

This seems to be connected to the memory running out. Right after calling RingElems one can observe that the memory usage of cocoa5 increases dramatically.
Could there be some severe memory leak?

Related issues:

Related to CoCoA-5 - Feature #1516: substring function	**Closed**	**23 Oct 2020**
Related to CoCoA-5 - Slug #875: Interpreter is too slow reading a big polynomial	**In Progress**	**03 May 2016**
Related to CoCoA-5 - Design #1519: Interpreter fn Value::from can use std::move?	**New**	**26 Oct 2020**
Related to CoCoALib - Slug #1521: Unexpectedly slow example with larger types...	**New**	**26 Oct 2020**
Related to CoCoALib - Bug #1522: SEGV: avoid long linked lists of loaves in M...	**Closed**	**26 Oct 2020**

History

#1 - 23 Oct 2020 09:22 - John Abbott

- Category set to enhancing/improving

- Target version set to CoCoA-5.4.2

The test example as in the description took about 7 mins to generate the 20Mbyte file. That is disappointingly slow (why?)
The call to GetLine was quite quick.

The call to RingElems took 30-60s (I wasn't watching too closely), and required about 19Gbytes RAM (which does seem like too much).
The result is a list with 5000 elements containing a total of about 2.5*20^6 terms, and each term comprises a dense exponent vector (presumably occupying 4000bytes). This means that the total space occupied by the read list is at least 1.0*10^10 bytes, around 10Gbytes.

Why does reading need twice that amount of space?

#2 - 23 Oct 2020 10:10 - John Abbott

- Status changed from New to In Progress

- % Done changed from 0 to 10

I have just recompiled CoCoALib to use unsigned char as exponent type.
The program ran noticeably faster: about 3 mins to print out the polys.
Memory (according to top) was about 4.8Gbytes; *i.e.* about 1/4 the original RAM requirement, as expected.

#3 - 23 Oct 2020 10:23 - John Abbott

- Related to Feature #1516: substring function added

#4 - 23 Oct 2020 11:07 - John Abbott

- Related to Slug #875: Interpreter is too slow reading a big polynomial added

#5 - 23 Oct 2020 12:19 - John Abbott

- % Done changed from 10 to 20

I have modified the test program so that it first generates a list of random linear forms (and then prints it).

With unsigned char exps generation time is about 90s
With unsigned short exps generation time is about 158s (memory about 4.7G)
With unsigned int exps generation time is about 346s (and memory required is about 9.4G)

#6 - 23 Oct 2020 15:56 - John Abbott

I have just made a first run with debugger/profiler --> SEGV
Ooops!

#7 - 23 Oct 2020 20:48 - John Abbott

MemPool causes a SEGV! 8-O

I'll try without MemPool... if I can remember how.

ANS: configure with --threadsafe-hack; the SEGV goes away (not surprising since MemPool is not being used)
Let's see what valgrind says...

#8 - 26 Oct 2020 09:54 - John Abbott

- Related to Design #1519: Interpreter fn Value::from can use std::move? added

#9 - 26 Oct 2020 09:56 - John Abbott

- % Done changed from 20 to 50

I was not able to get the information I wanted from valgrind (probably my fault).

After some thinking (but less time than spent messing about with valgrind), I believe the cause of the "double-size" memory consumption is the Value::from function in the interpreter. This function makes a copy of the value computed by cocoalib: so at some point CoCoA will have **two copies** of the large list (*i.e.* about 19Gbytes, as observed).

I have created a new issue (#1519) about seeing whether std::move can be used to avoid making two copies.

#10 - 26 Oct 2020 12:05 - John Abbott

- Related to Slug #1521: Unexpectedly slow example with larger types for SmallExponent_t added

#11 - 26 Oct 2020 12:08 - John Abbott

- Status changed from In Progress to Feedback

- Assignee set to John Abbott

- % Done changed from 50 to 90

A temporary workaround for Julian is probably the following:

- obtain the CoCoA sources, modify include/CoCoA/config.H so that SmallExponent_t is defined to be unsigned short (or even Unsigned char)
- compile CoCoA for your machine, and use that version instead of the standard release.

As far as I can tell, the crash was simply due to running out of RAM (or swap space)

This matter is "pseudo-resolved": *i.e.* I have created a number of consequential issues derived from problems arising during the investigation here. So I am marking this issue as in "feedback" -- the workaround above should enable Julian to proceed.

#12 - 26 Oct 2020 13:00 - John Abbott

- Related to Bug #1522: SEGV: avoid long linked lists of loaves in MemPools added

#13 - 03 Dec 2020 21:31 - John Abbott

- Status changed from Feedback to Closed

- Target version changed from CoCoA-5.4.2 to CoCoA-5.4.0

- % Done changed from 90 to 100

- Estimated time set to 4.33 h

I now have a personal CoCoA version with the new MemPool code (see issue #1522 comment 10).
I have just tried the following test case:

```
use P ::= ZZ/(101)[x[1..1000]];
L := [RandomLinearForm(P) | i in 1..8000];
ciao;
```

Exponents were 32 bits; CoCoA occupied almost 30Gbytes when the list was complete.
CoCoA then ended cleanly (no errors or other nasty surprises).

I regard this issue as fully resolved. So closing.