

## CoCoALib - Bug #1585

### CRASH/ABORT: GMP overflow

22 Mar 2021 09:37 - John Abbott

<b>Status:</b>	Closed	<b>Start date:</b>	22 Mar 2021
<b>Priority:</b>	High	<b>Due date:</b>	
<b>Assignee:</b>	John Abbott	<b>% Done:</b>	100%
<b>Category:</b>	Safety	<b>Estimated time:</b>	6.66 hours
<b>Target version:</b>	CoCoALib-0.99800	<b>Spent time:</b>	6.65 hours
<b>Description</b>			
The following causes an abort:  <pre>3^(3^(3^3));</pre> Problem originally reported by Julian Danner (via email). Placed under CoCoALib since the problem surely exists in CoCoALib.  How to handle neatly all possible cases of GMP overflow?			

#### History

##### #1 - 22 Mar 2021 09:40 - John Abbott

I have given this **high priority** because an abort is a nasty failure... everything is lost.

How to handle all possible cases of overflow in GMP? I hope there is some mechanism built in; otherwise it will be quite tedious having to code special checks before each GMP call.

Quite why Julian wanted to compute  $3^{(3^{(3^3)})}$  he did not say.

##### #2 - 22 Mar 2021 09:55 - John Abbott

I have written to the GMP people asking whether there is some way to specify an alternative way to handle overflow. An old post on StackOverflow suggests that there was none 8 years ago.

##### #3 - 23 Mar 2021 10:12 - John Abbott

- Status changed from *New* to *Resolved*

- % Done changed from 0 to 60

No reply from the GMP people. I looked at the code, and it seems that the error is actually signalled inside a memory allocator. It could be that GMP is "crash-only code" (see Wikipedia).

Anyway, I have now implemented an overflow check before actually asking GMP to compute a power. This involved choosing an arbitrary *overflow bound* (not sure how portable that really is). Implementing the check turned out to be more tedious than I had imagined. As a side effect, large powers of -1,0,1 are now allowed.

This is what CoCoA now gives:

```
/**/ 3^(3^(3^3));  
--> ERROR: Exponent is too large  
--> [CoCoALib] power(BigInt, BigInt)  
--> 3^(3^(3^3));  
--> ^
```

Presumably an overflow check should also be made for multiplication of BigInt values... (sigh)  
In which case the arbitrary *overflow bound* must be made more global.

#### #4 - 23 Mar 2021 11:19 - John Abbott

Here is a "torture test" using multiplication (on my machine):

```
/**/ N := 3^5000000000;  
/**/ N := N*N;  
/**/ N := N*N;  
/**/ N := N*N;  
/**/ N := N*N;  
/**/ FloatStr(N);  
--> ERROR: Exponent is too large  
--> [CoCoALib] power  
--> FloatStr(N);  
--> ^^^^^^^^^^^  
/**/ N := N*N;  
/**/ N := N*N;  
/**/ FloorLog2(N);  
50718800023  
/**/ FloatStr(It/2^30);  
47.236  
/**/ t0 := CpuTime(); FloorLog2(N); TimeFrom(t0);  
50718800023  
2.526  
/**/ N := N*N;
```

Process cocoa5 killed

The arbitrary limit on size in power caused FloatStr to fail; not sure how to work around that without introducing unnecessary complication.  
I was surprised to see that FloorLog2 took so long... I wonder why?

Don't know what actually caused the process to die: top reported that the process was using 41Gbyte virtual memory, but it still seemed to be running (until suddenly it wasn't any more).

## #5 - 23 Mar 2021 13:39 - John Abbott

- Assignee set to John Abbott

I have increased the arbitrary limit so that the example in comment 4 above now runs without hiccups (except for the final "killed").

I now think that the KISS approach is indeed to impose an arbitrary limit (on bitlength of BigInt values), and then modify also multiplication. A quick grep showed that `mpz_mul` (or variants) are called in many places; the neatest solution would be to have a new function `mpz_mul_with_overflow_check`.

Should the bitlength limit be an arg or a global? ... globals are ugly (esp. in multithreaded settings)

I'll try suggesting to the GMP people that some new functions which check for overflow be added... probably much easier said than done :-/

## #6 - 02 Apr 2021 10:27 - John Abbott

Which functions should test for overflow?  
power and multiplication certainly. Which others?

```
factorial(10^12); --> BOOM  
factorial(10^11); --> takes a long time, and cannot be interrupted (presumably will eventually crash)
```

fibonacci and binomial are also like "power" functions.

```
fibonacci(2*10^11); --> BOOM  
fibonacci(10^11); --> takes a long time, and cannot be interrupted (presumably will eventually crash);
```

binomial is probably harder to handle: the 1st arg can be very large, and the computation will finish if the 2nd arg is small. What to do? Estimate the log?

## #7 - 18 Apr 2021 22:51 - John Abbott

I have put a first version of `mpz_mul` with overflow check in `utils-gmp`. A quick suggests that it is working as expected.

We need also an overflow check for `mpq`... sigh.

Still plenty more work to be done before it is ready to be released.

## #8 - 19 Apr 2021 12:03 - John Abbott

How to test if the overflow detection mechanism is working properly (*i.e.* throws an exception instead of causing a crash)?

Here is an idea:

```
ZZx ::= ZZ[x];
QQx ::= QQ[x];
N := 2^K; // K so large that N*N causes overflow
N1 := N+1; // odd number
// The following should trigger overflow
N*N;
N1*N1;
N^2;
lcm(N, N1);
RingElem(ZZx, N)^2;
RingElem(QQx, N)^2;
(1/RingElem(QQx, N))^2; // "underflow"
1/RingElem(QQx, N) + 1/RingElem(QQx, N1); // denom is N*N1
RingElem(QQx, N) + 1/RingElem(QQx, N); // numer is N^2+1
```

Here are some cases where overflow should not occur (but a naive impl might cause overflow):

```
N3 := 2^K; // K such that N^3 causes overflow but N^2 does not
N3odd := N3-1;
lcm(N^2, N);
lcm(N^2, N^2);
// and similar operations in ZZx
```

## #9 - 19 Apr 2021 20:55 - John Abbott

There are very many cases to consider, esp. if we want to cover all (or almost all) possibilities.

Now I am more inclined to cover just those cases where a "typo" could lead to a crash.  
This could include powering (already fixed), maybe binomials and fibonacci nums?

Anna thinks that operations such as multiplication will probably be slow enough that they act as a warning...

**#10 - 05 May 2021 15:46 - John Abbott**

I have put in *ad hoc* protections for factorial and binomial.  
Still need to decide how high to set the overflow trigger; can the user change it?

**#11 - 15 Sep 2021 11:56 - John Abbott**

I have discussed this with Anna; and had also written to the GMP people some time ago.

Overall conclusion: it is **very hard** (imposs) to protect against all possible forms of overflow (and could be costly).

So we shall "protect" just a few functions where a simple mistake could trigger GMP overflow (and a dead CoCoA/CoCoALib program).  
Which functions? Probably just **power** and **factorial** initially (maybe also **binomial**?).

**#12 - 20 Sep 2021 20:52 - John Abbott**

- % Done changed from 60 to 80

I have put a new constant called **OVERFLOW\_BITS** in **config.H**, and modified the impls so that they use this constant (I also reworked the impls for power).

There remains the problem for FloatStr... mmm

**#13 - 20 Sep 2021 21:33 - John Abbott**

I have worked around the FloatStr problem (actually in FloorLogBase) mentioned in comment 4 above.  
I just call the GMP power function directly, rather than check that the args are sane and do not cause overflow.

Might there be other cases where someone wants to compute a large power knowing that it cannot be too large?  
Should there be an option to turn off overflow checking for power? Or perhaps for all checked functions?  
I could make the overflow limit settable at run time (with a sane default initial value).  
Better KISS? Try what we have, and see if that causes trouble.

**#14 - 22 Sep 2021 17:45 - John Abbott**

- Status changed from Resolved to Closed

- % Done changed from 80 to 100

- Estimated time set to 6.66 h

As observed in comment 11, it would be very hard to try to prevent overflow in all cases.  
It thus seems reasonable to limit to a few "likely" cases initially, and potentially extend the set of cases covered if that seems necessary.

KISS and close.

**NOTE** I have increased the overflow limit to 4G bits (previously was 1G bit. No real reason, just 1G bits seemed perhaps a bit low.