CoCoALib - Design #1463

SmoothFactor: use FactorMultiplicity

16 Jun 2020 19:56 - John Abbott

Status:	Closed	**Start date:**	16 Jun 2020
Priority:	Low	**Due date:**	
Assignee:	John Abbott	**% Done:**	100%
Category:	Tidying	**Estimated time:**	5.77 hours
Target version:	CoCoALib-0.99800	**Spent time:**	5.75 hours

Description

SmoothFactor contains almost a duplicate implementation of FactorMultiplicity.

Tidy up: Make SmoothFactor call FactorMultiplicity.

Related issues:

Related to CoCoALib - Support #1499: factorization: allow zero as exponent?	**Closed**	**05 Oct 2020**

History

#1 - 17 Jun 2020 20:19 - John Abbott

- Status changed from New to In Progress

- Assignee set to John Abbott

- % Done changed from 0 to 20

The current implementation of FactorMultiplicity makes it hard to use inside SmoothFactor (because it returns only the multiplicity, and not N/p^k).

I now think that the interface to FactorMultiplicity should change, or there should be a new function.
The advantage of adding a new function is that it is backwards-compatible.

What should the new function do, and what should it be called?
I see two possible sensible signatures:

- **(A)** args (N,p) returns a factorization with fields myFactors=[p], myMultiplicities=[k] and myRemainingFactor=N/p^k
- **(B)** args (ref N,p) and changes value of N

(B) feels "simpler", but does change the value of a parameter (so there could be a question about exception safety)
(A) is possibly more consistent with other "factorization" functions, but its return value is new object (implying possible copying of some values).

Another question with **(A)** is what happens if the multiplicity is zero?
Currently the impl for factorization requires multiplicities to be positive (even though the error message seems to prohibit only negative multiplicities).

#2 - 25 Sep 2020 10:51 - John Abbott

- Status changed from In Progress to Resolved

- % Done changed from 20 to 70

I have made a reasonable impl more or less following **(B)**.

There is a new non-exported fn which computes multiplicity and modifies one of its args. Should this be made public?

Checked in. No new tests were added; maybe I should (*e.g.* for negative args?)

#3 - 05 Oct 2020 11:58 - John Abbott

- Status changed from Resolved to Feedback

- Target version changed from CoCoALib-0.99850 to CoCoALib-0.99800

- % Done changed from 70 to 90

Now I think it is better not to make **DivideOutMaxPower** public; we can wait until there is actual demand for it.

I have tried the following test:

```
n := factorial(10^6);
b := 3^41;
FactorMultiplicity(b,n); --> takes about 23s
FactorMultiplicity(3,n); --> takes about 13s
```

So I have added a CheckForInterrupt in the main loop of DivideOutMaxPower for BigInt,BigInt.
Perhaps the loop should be changed to mimic that for long,BigInt? It is unexpected that the second call to FactorMultiplicity is faster than the first!

PS if I increase n to factorial(2000000) the times become 99s and 57s which is about the same ratio as before --- huh???

#4 - 05 Oct 2020 13:24 - John Abbott

- Related to Support #1499: factorization: allow zero as exponent? added

#5 - 29 Oct 2020 15:10 - John Abbott

Oh no! I have hit a bug:

```
n := factorial(10^6);
FactorMultiplicity(3^40,n); --> takes about 15s.. why not 13?
FactorMultiplicity(3^39,n); --> uninterruptible infinite loop???  :-(
```

Bother!!

#6 - 29 Oct 2020 15:13 - John Abbott

FactorMultiplicity(3^39,factorial(21)) goes into infinite loop :-/

It gets worse...
FactorMultiplicity(9, factorial(21)) goes into infinite loop.... well, that's small enough to be debuggable.... and very embarrassing!

PS luckily there are no "watchers" for this issue ;-)

#7 - 29 Oct 2020 21:30 - John Abbott

I have fixed the bug: part of the code incorrectly assumed that the base was a prime number... ooops! (probably an old version)
All is OK now; I have added an "exbug" test.

#8 - 29 Oct 2020 21:44 - John Abbott

- Status changed from Feedback to Closed

- % Done changed from 90 to 100

- Estimated time set to 5.77 h