# CoCoALib - Feature #586

## BigInt ctor from a machine integer

11 Jul 2014 15:23 - John Abbott

| | | | |
|---|---|---|---|
| **Status:** | Rejected | **Start date:** | 11 Jul 2014 |
| **Priority:** | High | **Due date:** | |
| **Assignee:** | John Abbott | **% Done:** | 100% |
| **Category:** | Safety | **Estimated time:** | 4.40 hours |
| **Target version:** | CoCoALib-0.99540 Feb 2016 | **Spent time:** | 4.65 hours |

**Description**

Christof asks why the ctor for a BigInt from a machine integer is explicit rather than implicit.
He also suggests the we make it implicit (for the convenience of the user).

### History

#### #1 - 11 Jul 2014 15:26 - John Abbott

I've just removed the explicit keyword from the BigInt ctor. CoCoALib compiled fine (no problems with ambiguity); same also for the tests (which all passed).

#### #2 - 11 Jul 2014 15:42 - John Abbott

*- Status changed from New to In Progress*

*- % Done changed from 0 to 10*

Advantages:

- in IntOperations and NumTheory we can reduce the number of signatures to some functions (*e.g.* IsExactRoot and PowerMod)
- anyone writing a new function for CoCoALib needs only use the type BigInt for formal parameters with integer values (rather than having signatures for both BigInt and MachineInt -- if there are k integer params then there must be 2^k signatures)

Disadvantages:

- sometimes there might be an unexpected conversion from a machine integer to a BigInt, *e.g.* if we had forgotten to implement a specific version of a function for machine integers.

#### #3 - 14 Jul 2014 14:25 - John Abbott

*- % Done changed from 10 to 20*

*- Estimated time set to 4.40 h*

I suggest that I check in the version of BigInt.H that I have (with the ctor from MachineInt being implicit). That way we can all try it for a few days to see if any unexpected consquences arise.

Any objections?

#### #4 - 15 Jul 2014 13:40 - John Abbott

*- Assignee set to John Abbott*

*- % Done changed from 20 to 30*

There were no objections (in the last 24 hours), so I'll check in.  If any problems arise, it's trivial to revert the code.

#### #5 - 27 Jul 2014 12:57 - John Abbott

*- Target version changed from CoCoALib-0.99534 Seoul14 to CoCoALib-1.0*

*- % Done changed from 30 to 40*

No complaints from anyone after checking in, so I regard the idea as safe and sound.

Postponing to "after Seoul" because the code works fine as is, and I prefer to do the necessary tidying (*i.e.* removal of pointless fn signatures) without haste.  Also the tidying does not really produce any visible benefit which we could show at Seoul.

It'd be nice to finish this soon after Seoul, so I've left the priority as high.

#### #6 - 08 Oct 2015 11:02 - John Abbott

It seems that I had forgotten about this task.  I will try to finish it now.

#### #7 - 08 Oct 2015 12:04 - John Abbott

As a check I reinserted the explicit keyword and tried to compile: two (minor) problems arose in NumTheory.C where I checked for equality to 0 and to 1 of MachineInt variable -- the code should have been checking a long value which had been extracted from the MachineInt, so technically the mistakes were "slugs" (logically correct, but needelessly inefficient).

I also checked to see which functions in IntOperations.H would benefit from the automatic conversion from a machine integer to a BigInt and there were only a few.  Many of the functions were able to be more efficient if one of the args was a machine integer.

Sometimes a return value could be made simpler *i.e.* remove the explicit call to the BigInt ctor... but perhaps I prefer to see that there is a conversion when creating the return value.

So now I'm not so convinced of the possible benefits of automatic conversion.

#### #8 - 08 Oct 2015 12:18 - John Abbott

There could be some minor benefit to NumTheory.H especially PowerMod which has 8 different signatures to cover all possible combinations, even though I expect that only 2 of them would suffice for almost all uses: "all MachineInt" and "all BigInt".

Similarly gcd and lcm could perhaps be reduced to 2 signatures.

Overall, I now think that the potential benefit would be fairly slight... but Christof did point out that it would make life easier for contributors.

Undecided; it is probably not that important either way.

#### #9 - 08 Oct 2015 15:13 - John Abbott

On several occasions the design of CoCoALib has preferred the path of "ease of use" over that of "absolute speed".  Since this issue is really about

such a choice, it would make sense to prefer also here "ease of use" over "absolute speed". Indeed if there is some case where a significant speed penalty arises, we can always make a separate function for MachineInt. The only tricky aspect is determining when a "significant speed penalty" occurs -- my guess is that most such cases have already been handled in IntOperations.

This point of view supports removing explicit.

#### #10 - 08 Oct 2015 15:33 - John Abbott

Here are the fns which would benefit from having implicit conversion to BigInt:

- ILogBase --> need just 1 signature
- mantissa, exponent, NumDigits would also work for machine ints
- RoundDiv --> just 2 signatures
- iroot --> just 1 or 2 signatures
- IsExactIroot --> just 2 signatures
- IsSquare and IsPower --> just 1 signature each

- PowerMod
- lcm --> just two signatures

#### #11 - 08 Oct 2015 16:33 - John Abbott

I have found a catch...
If a and b are variables of type int, and the only signature for lcm expects two BigInt values then lcm(a,b) does not compile!

The problem is that there is no direct "conversion" from int to BigInt; instead, as currently implemented, the int must first be converted to MachineInt and then to BigInt. However C++ allows only a single stage of conversion when doing automatic conversions.

The obvious solution is to define direct conversions (*i.e.* new ctors) to BigInt from each of the built-in integral types in the same way we have done for the ctors of MachineInt. Tedious!

#### #12 - 08 Oct 2015 16:58 - Anna Maria Bigatti

John Abbott wrote:

> However C++ allows only a single stage of conversion when doing automatic conversions.

well, I'm pleased to hear that...

> The obvious solution is to define direct conversions (*i.e.* new ctors) to BigInt from each of the built-in integral types in the same way we have done for the ctors of MachineInt. Tedious!

Tedious, but it makes sense.
One option is to write them and leave them commented out. Meanwhile we try to decide if we really want them or not. Maybe also doing some speed tests.

#### #13 - 08 Oct 2015 17:33 - John Abbott

I have just encountered a bigger problem...
Line 167 of BigInt.C wants to evaluate the expression N > 0 where N is of type BigInt. The compiler complains that this is ambiguous: it matches equally well operator>(BigInt,BigInt) and operator>(BigInt,MachineInt)

Right now I do not see any good way out of this:

- **(A)** undo: *i.e.* no implicit coversion from MachineInt to BigInt
- **(B)** eliminate operator>(BigInt,MachineInt)

Option **(A)** takes us back to known territory... it's not too bad.
Option **(B)** means that every comparison between a BigInt and a machine integer will construct a BigInt object... not too tempting :-(

#### #14 - 08 Oct 2015 17:51 - Anna Maria Bigatti

John Abbott wrote:

> Right now I do not see any good way out of this:
>
> - **(A)** undo: *i.e.* no implicit coversion from MachineInt to BigInt
> - **(B)** eliminate operator>(BigInt,MachineInt)

(A) undo

#### #15 - 08 Oct 2015 17:52 - John Abbott

There is an option **(C)** but it is not realistic: I could define operator> between BigInt and each of the actual machine integer types. Or I could even define a template function: template <typename T> bool operator>(BigInt,T) but that is pretty ugly, and might well require a lot of code to be put into IntOperations.H

#### #16 - 08 Oct 2015 17:53 - John Abbott

I'll think about it overnight, but suspect that "undo" is the best (=least bad) solution.

#### #17 - 09 Oct 2015 09:47 - John Abbott

*- % Done changed from 40 to 50*

I've decided for option **(A)** "undo"; the others seem to be too much hassle+risk.

I'll comment out the new BigInt ctors, check-in the files. No doubt the commented out code will be eliminated sooner or later.

Sorry Christof: I cannot figure out how to achieve what you want in a reasonable way.

#### #18 - 09 Oct 2015 11:32 - Christof Soeger

So to make sure I got it right. You implemented the constructures BigInt(long) and so on, and now get this ambiguity?
I didn't know you have this MachineInt type. In this case I also see no easy solution.

Maybe you then can add a MachineInt version to some functions. I don't remember where I wanted to have this, maybe multiplying a RingElem with or taking some power, or constructing a RingElem.

#### #19 - 09 Oct 2015 14:09 - John Abbott

*- Target version changed from CoCoALib-1.0 to CoCoALib-0.99540 Feb 2016*

Christof:
Yes, I implemented 8 new ctors for @BigInt direct from the various C++ machine integer types; and yes, these then created ambiguities since most functions which expect "integer" arguments are written with formal parameters of type both MachineInt and BigInt (*i.e.* two separate functions).

It is possible that some functions have only a BigInt interface: this is either an oversight, or because there is already a standard function for machine integers.

You are right that it would be easier if the ctor for BigInt from a machine integer were "implicit" (*i.e.* not explicit), but I just don't see how to achieve this without causing some trouble somewhere.

I created the MachineInt class to avoid the nasty C++ rule about automatic silent conversion between built-in integers types, most especially the value-changing conversions between signed and unsigned types.

#### #20 - 09 Oct 2015 20:18 - John Abbott

*- Status changed from In Progress to Rejected*

*- % Done changed from 50 to 100*

Checked in the commented out defns of ctors for BigInt direct from machine integer types.

Marking as "Rejected" since the desired change cannot be done.