Ruby master - Bug #9192

Inconsistent comparison between Float and BigDecimal

12/02/2013 09:01 AM - vatsu (Gustavo Sales)

Status:	Closed		
Priority:	Normal		
Assignee:	mrkn (Kenta Murata)		
Target version:			
ruby -v:	2.0.0p353 (2013-11-22 revision 43784) [x86_64-darwin11.4.2]	**Backport:**	2.1: UNKNOWN, 2.2: UNKNOWN, 2.3: UNKNOWN

Description

I was checking a possible inconsistency on ActiveModel and I found situations where comparing BigDecimal to Float differs to comparing Float to BigDecimal.

I have create a simple ruby script to exemplify the problem. The script is attach to this issue and output is as follows:

Instance number 706.05
###########
Instance number 706.06
Inconsistence:
0.70606E3 > 706.06

706.06 < 0.70606E3

Inconsistence:
0.70606E3 <= 706.06

706.06 >= 0.70606E3

###########
Instance number 706.07
Inconsistence:
0.70607E3 < 706.07

706.07 > 0.70607E3

Inconsistence:
0.70607E3 >= 706.07

706.07 <= 0.70607E3

###########
Instance number 706.08
###########

Related issues:

Related to Ruby master - Feature #8295: Float □ Rational □□□□□□□□□□□□□□ BigDec...	**Assigned**

Associated revisions

Revision 0727a22c - 12/08/2013 11:20 AM - mrkn (Kenta Murata)

- bigdecimal.c (BigDecimal_coerce): convert a Float to a BigDecimal instead of converting the receiver to a Float. [ruby-core:58756] [Bug #9192]
- test/bigdecimal/test_bigdecimal.rb: add tests for the above change.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@44073 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 44073 - 12/08/2013 11:20 AM - mrkn (Kenta Murata)

- bigdecimal.c (BigDecimal_coerce): convert a Float to a BigDecimal instead of converting the receiver to a Float.
  [ruby-core:58756] [Bug #9192]

- test/bigdecimal/test_bigdecimal.rb: add tests for the above change.

Revision 44073 - 12/08/2013 11:20 AM - mrkn (Kenta Murata)

- bigdecimal.c (BigDecimal_coerce): convert a Float to a BigDecimal instead of converting the receiver to a Float.
  [ruby-core:58756] [Bug #9192]

- test/bigdecimal/test_bigdecimal.rb: add tests for the above change.

Revision 44073 - 12/08/2013 11:20 AM - mrkn (Kenta Murata)

- bigdecimal.c (BigDecimal_coerce): convert a Float to a BigDecimal instead of converting the receiver to a Float.
  [ruby-core:58756] [Bug #9192]

- test/bigdecimal/test_bigdecimal.rb: add tests for the above change.

Revision 44073 - 12/08/2013 11:20 AM - mrkn (Kenta Murata)

- bigdecimal.c (BigDecimal_coerce): convert a Float to a BigDecimal instead of converting the receiver to a Float.
  [ruby-core:58756] [Bug #9192]

- test/bigdecimal/test_bigdecimal.rb: add tests for the above change.

Revision 44073 - 12/08/2013 11:20 AM - mrkn (Kenta Murata)

- bigdecimal.c (BigDecimal_coerce): convert a Float to a BigDecimal instead of converting the receiver to a Float.
  [ruby-core:58756] [Bug #9192]

- test/bigdecimal/test_bigdecimal.rb: add tests for the above change.

Revision 44073 - 12/08/2013 11:20 AM - mrkn (Kenta Murata)

- bigdecimal.c (BigDecimal_coerce): convert a Float to a BigDecimal instead of converting the receiver to a Float.
  [ruby-core:58756] [Bug #9192]

- test/bigdecimal/test_bigdecimal.rb: add tests for the above change.

History

#1 - 12/02/2013 01:50 PM - nobu (Nobuyoshi Nakada)

Sounds like the old usual floating point number error.

#2 - 12/02/2013 04:28 PM - vatsu (Gustavo Sales)

Regardless the floating point computational representation, shouldn't we expect the same result for following statements?

BigDecimal('706.06') <= 706.06

and

706.06 >= BigDecimal('706.06')

#3 - 12/02/2013 06:54 PM - duerst (Martin Dürst)

nobu (Nobuyoshi Nakada) wrote:

> Sounds like the old usual floating point number error.

I don't think so. BigDecimal('706.06') is exact. 706.06 is a float, and therefore can't be represented exactly. That means that it's either represented as too small or too big than 706.06. If it's represented as too small, then we should get false twice; if it's represented as too big, we should get true twice.

This would be "the old usual floating point number error" if Gustavo had asked why 706.06 != BigDecimal('706.06'), but that's not the question here.

#4 - 12/02/2013 10:52 PM - nobu (Nobuyoshi Nakada)

Sorry, I've not read your code carefully. # from my iPhone

I'll look it again tommorrow.

#5 - 12/03/2013 11:22 AM - vatsu (Gustavo Sales)

BTW, I am keen to help on solving this, but I would need guidance...

#6 - 12/03/2013 02:38 PM - nobu (Nobuyoshi Nakada)

- Category set to ext

- Status changed from Open to Assigned

- Assignee set to mrkn (Kenta Murata)

BigDecimal converts Float with to_r, and 706.06.to_r returns (1552642359815045/2199023255552).
But Float#to_r doesn't consider that Float is inexact, this is not equal to 706.06.
Converting by rb_flt_rationalize_with_prec() seems solving this issue, but makes BigDecimal test stuck.

#7 - 12/03/2013 02:57 PM - marcandre (Marc-Andre Lafortune)

duerst (Martin Dürst) wrote:

> nobu (Nobuyoshi Nakada) wrote:
>
> > Sounds like the old usual floating point number error.
>
> I don't think so. BigDecimal('706.06') is exact. 706.06 is a float, and therefore can't be represented exactly. That means that it's either represented as too small or too big than 706.06. If it's represented as too small, then we should get false twice; if it's represented as too big, we should get true twice.
> This would be "the old usual floating point number error" if Gustavo had asked why 706.06 != BigDecimal('706.06'), but that's not the question here.

I never thought about comparison of float with other numeric types, but should we convert everything to float and compare this way? This way we insure that:

```
any_big_decimal.to_f == any_big_decimal
```

Here, the float 706.06 stands for a small range of real numbers that includes BigDecimal('706.06'). Thus shouldn't we have 706.06 ==

BigDecimal('706.06')?

#8 - 12/04/2013 07:58 PM - duerst (Martin Dürst)

marcandre (Marc-Andre Lafortune) wrote:

> I never thought about comparison of float with other numeric types, but should we convert everything to float and compare this way? This way we insure that:
>
> ```
> any_big_decimal.to_f == any_big_decimal
> ```
>
> Here, the float 706.06 stands for a small range of real numbers that includes BigDecimal('706.06'). Thus shouldn't we have 706.06 == BigDecimal('706.06')?

This definitely would be a good solution for this specific example. But we have to (try to) make sure it's going to work consistently for other numeric types, too. For example, I tested with Rational(70606, 100). That compares equal with 706.06 (both ways). So it would indeed make sense to have BigDecimal('706.06') also compare equal with 706.06. It currently does if written 706.06 op BigDecimal('706.06') (op is any of <=>, <=, ==, >=, !=, <, >), but not the other way round.

But simply converting BigDecimal to Float may not make sense in general. Imagine a BigDecimal with much higher precision, like BigDecimal('706.06059999999999'), and some corresponding floats. We may want to actually say that the BigDecimal is bigger or smaller than the float created from the same string, because the float looses precision.

Currently, I get
706.06059999999999 <=> BigDecimal('706.06059999999999') #=> 0
BigDecimal('706.06059999999999') <=> 706.0605999999999 #=> -1
which is of course inconsistent. We should in any case make sure that
(a <=> b) + (b <=> a)
is always 0 wherever the comparison makes sense.

#9 - 12/05/2013 12:44 AM - marcandre (Marc-Andre Lafortune)

duerst (Martin Dürst) wrote:

> ...
> But simply converting BigDecimal to Float may not make sense in general. Imagine a BigDecimal with much higher precision, like BigDecimal('706.06059999999999'), and some corresponding floats. We may want to actually say that the BigDecimal is bigger or smaller than the float created from the same string, because the float looses precision.
> Currently, I get
> 706.06059999999999 <=> BigDecimal('706.06059999999999') #=> 0
> BigDecimal('706.06059999999999') <=> 706.0605999999999 #=> -1
> which is of course inconsistent. We should in any case make sure that
> (a <=> b) + (b <=> a)
> is always 0 wherever the comparison makes sense.

Agreed.

The float we write as 706.0606 represents a small range of real values that includes the real number 706.06059999999999. So the first result is correct:

```
706.0606 <=> BigDecimal('706.06059999999999') #=> 0, ok
```

The only strange effect is that we can get:

```
a == x # => true
b == x # => true
a == b # => false
```

when x is a Float and a or b are higher precision numerics.
This can be explained because the first two comparison involve floats and thus dealing with approximate values.

#10 - 12/05/2013 07:32 AM - vatsu (Gustavo Sales)

marcandre (Marc-Andre Lafortune) wrote:

> duerst (Martin Dürst) wrote:
>
> > ...
> > But simply converting BigDecimal to Float may not make sense in general. Imagine a BigDecimal with much higher precision, like BigDecimal('706.06059999999999'), and some corresponding floats. We may want to actually say that the BigDecimal is bigger or smaller than the float created from the same string, because the float looses precision.
> > Currently, I get
> > 706.06059999999999 <=> BigDecimal('706.06059999999999') #=> 0

> BigDecimal('706.06059999999999') <=> 706.0605999999999 #=> -1
> which is of course inconsistent. We should in any case make sure that
> (a <=> b) + (b <=> a)
> is always 0 wherever the comparison makes sense.

Agreed.

The float we write as 706.0606 represents a small range of real values that includes the real number 706.06059999999999. So the first result is correct:

```
706.0606 <=> BigDecimal('706.06059999999999') #=>  0, ok
```

The only strange effect is that we can get:

```
a == x # => true
b == x # => true
a == b # => false
```

when x is a Float and a or b are higher precision numerics.
This can be explained because the first two comparison involve floats and thus dealing with approximate values.

I don't think it's a good idea to convert to float. We, programmers, have to know how numbers are stored in memory and we also know how to deal with different number's precision. My only concern is about making comparison between any 2 Numeric subclasses consistent. In that sense, I agree with @duerst. But, it is just my opinion as ruby programmer that is starting to try to help on ruby-core.

#11 - 12/08/2013 08:20 PM - mrkn (Kenta Murata)

- Status changed from Assigned to Closed

- % Done changed from 0 to 100

This issue was solved with changeset r44073.
Gustavo, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

- bigdecimal.c (BigDecimal_coerce): convert a Float to a BigDecimal instead
  of converting the receiver to a Float.
  [ruby-core:58756] [Bug #9192]

- test/bigdecimal/test_bigdecimal.rb: add tests for the above change.

#12 - 12/10/2013 03:42 AM - marcandre (Marc-Andre Lafortune)

- Status changed from Closed to Open

Hi

1) I disagree with this change. Is it possible to know why you decided to ignore my recommendation without even commenting on it?

2) The implementation is buggy. For example:

```
BigDecimal('0.21611564636388508') == 0.21611564636388508 # => false, should be true
```

#13 - 12/10/2013 09:20 AM - vatsu (Gustavo Sales)

Thank you, mrkn!

#14 - 12/14/2013 12:25 AM - mrkn (Kenta Murata)

marcandre (Marc-Andre Lafortune) wrote:

> 1) I disagree with this change. Is it possible to know why you decided to ignore my recommendation without even commenting on it?

I'm sorry I didn't see your comments.
It is because there were only 3-4 comments on this issue when I started to tackle it.

> 2) The implementation is buggy. For example:

```
BigDecimal('0.21611564636388508') == 0.21611564636388508 # => false, should be true
```

Thank you to find this case. I'll read your above comments and fix this issue.

#15 - 02/28/2014 11:23 PM - knugie (Wolfgang Teuber)

Marc-Andre Lafortune wrote:

> 2) The implementation is buggy. For example:
>
> ```
> BigDecimal('0.21611564636388508') == 0.21611564636388508 # => false, should be true
> ```

You are exceeding the max precision Float::DIG (http://www.ruby-doc.org/core-2.0.0/Float.html#DIG => "Usually defaults to 15."). Your example uses 17 significant digits, which will ultimately lead to floating point number errors. The issue you described is not related to Gustavo's issue.

#16 - 03/01/2014 06:36 AM - marcandre (Marc-Andre Lafortune)

Wolfgang Teuber wrote:

> You are exceeding the max precision Float::DIG (http://www.ruby-doc.org/core-2.0.0/Float.html#DIG => "Usually defaults to 15."). Your example uses 17 significant digits, which will ultimately lead to floating point number errors.

The doc is misleading. Floats can take from 15 to 17 decimals to be described uniquely. See for example https://bugs.ruby-lang.org/issues/3273

The example float I gave does take 17 decimals. It's not possible to describe that float with less.

> The issue you described is not related to Gustavo's issue.

I find it difficult to respond when what I wrote seems to me to be so clear. r44073 is an attempt to fix Gustavo's issue. As I stated, I disagree with what r44073 wants to do, but even if it was decided it was the right thing to do, it doesn't even do it correctly, as shown in my counterexample.

Hope this helps.

#17 - 12/10/2016 07:23 AM - mrkn (Kenta Murata)

- Backport deleted (1.9.3: UNKNOWN, 2.0.0: UNKNOWN)

This is fixed in the latest master branch of ruby/bigdecimal.

#18 - 12/10/2016 07:23 AM - mrkn (Kenta Murata)

- Backport set to 2.1: UNKNOWN, 2.2: UNKNOWN, 2.3: UNKNOWN

- Status changed from Open to Closed

Files

compare_decimal_float.rb	529 Bytes	12/02/2013	vatsu (Gustavo Sales)