Ruby master - Bug #8883

Rational canonicalization unexpectedly converts to Fixnum

09/10/2013 08:41 AM - melquiades (Paul Cantrell)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	ruby 2.0.0p247 (2013-06-27 revision 41674) [x86_64-darwin12.3.0]	**Backport:**	1.9.3: UNKNOWN, 2.0.0: DONE

Description

The documentation for Rational (http://www.ruby-doc.org/core-2.0.0/Rational.html) states that the result of creating or doing arithmetic on Rationals returns Rationals, as one would expect. Examples from the docs:

```
Rational(1)      #=> (1/1)
3.to_r           #=> (3/1)
Rational(-2, 9) * Rational(-9, 2)  #=> (1/1)
```

These all work as documented in 1.9. In 2.0, however, they all return Fixnum:

```
Rational(1)      #=> 1
3.to_r           #=> 3
Rational(-2, 9) * Rational(-9, 2)  #=> 1
```

This leads to unexpected behavior:

```
Rational(2) / Rational(3)  # => 0  ...but returns (2/3) in 1.9
```

That behavior is potentially dangerous. Math that may *usually* work, but suddenly start suffering from truncation errors depending on intermediate results. For example:

def should_always_return_one(a, b, c)
(Rational(a, c) + Rational(b, c)) / (a + b) * c
end

Under 1.9:

```
should_always_return_one(2, 3, 7) #=> (1/1)
should_always_return_one(2, 4, 7) #=> (1/1)
should_always_return_one(2, 5, 7) #=> (1/1)
should_always_return_one(2, 6, 7) #=> (1/1)
```

Under 2.0:

```
should_always_return_one(2, 3, 7) #=> 1
should_always_return_one(2, 4, 7) #=> 1
should_always_return_one(2, 5, 7) #=> 0   Oops!
should_always_return_one(2, 6, 7) #=> 1
```

Either the docs are wrong, or this is a bug. I vote bug. Whether arithmetic expressions truncate the result should not depend on whether intermediate values just happen to be integers! Such behavior renders Rational almost too dangerous to use in situations where exact results are required. (Yes, I realize that requiring 'mathn' fixes this, but even with such a workaround as an option, this is dangerously broken. See also #2121.) Note that floating point arithmetic does not exhibit this behavior.

Associated revisions

Revision 86ffd21c - 11/03/2013 12:35 AM - nobu (Nobuyoshi Nakada)

load.c: defer initalization of static-linked-ext

- load.c (rb_feature_p): deal with default loadable suffixes.
- load.c (load_lock): initialize statically linked extensions.
- load.c (search_required, rb_require_safe): deal with statically linked extensions.

- load.c (ruby_init_ext): defer initalization of statically linked extensions until required actually. [Bug #8883]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@43514 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 43514 - 11/03/2013 12:35 AM - nobu (Nobuyoshi Nakada)

load.c: defer initalization of static-linked-ext

- load.c (rb_feature_p): deal with default loadable suffixes.
- load.c (load_lock): initialize statically linked extensions.
- load.c (search_required, rb_require_safe): deal with statically linked extensions.
- load.c (ruby_init_ext): defer initalization of statically linked extensions until required actually. [Bug #8883]

Revision 43514 - 11/03/2013 12:35 AM - nobu (Nobuyoshi Nakada)

load.c: defer initalization of static-linked-ext

- load.c (rb_feature_p): deal with default loadable suffixes.
- load.c (load_lock): initialize statically linked extensions.
- load.c (search_required, rb_require_safe): deal with statically linked extensions.
- load.c (ruby_init_ext): defer initalization of statically linked extensions until required actually. [Bug #8883]

Revision 43514 - 11/03/2013 12:35 AM - nobu (Nobuyoshi Nakada)

load.c: defer initalization of static-linked-ext

- load.c (rb_feature_p): deal with default loadable suffixes.
- load.c (load_lock): initialize statically linked extensions.
- load.c (search_required, rb_require_safe): deal with statically linked extensions.
- load.c (ruby_init_ext): defer initalization of statically linked extensions until required actually. [Bug #8883]

Revision 43514 - 11/03/2013 12:35 AM - nobu (Nobuyoshi Nakada)

load.c: defer initalization of static-linked-ext

- load.c (rb_feature_p): deal with default loadable suffixes.
- load.c (load_lock): initialize statically linked extensions.
- load.c (search_required, rb_require_safe): deal with statically linked extensions.
- load.c (ruby_init_ext): defer initalization of statically linked extensions until required actually. [Bug #8883]

Revision 43514 - 11/03/2013 12:35 AM - nobu (Nobuyoshi Nakada)

load.c: defer initalization of static-linked-ext

- load.c (rb_feature_p): deal with default loadable suffixes.
- load.c (load_lock): initialize statically linked extensions.
- load.c (search_required, rb_require_safe): deal with statically linked extensions.
- load.c (ruby_init_ext): defer initalization of statically linked extensions until required actually. [Bug #8883]

Revision 43514 - 11/03/2013 12:35 AM - nobu (Nobuyoshi Nakada)

load.c: defer initalization of static-linked-ext

- load.c (rb_feature_p): deal with default loadable suffixes.
- load.c (load_lock): initialize statically linked extensions.
- load.c (search_required, rb_require_safe): deal with statically linked extensions.
- load.c (ruby_init_ext): defer initalization of statically linked extensions until required actually. [Bug #8883]

Revision 71f8a0d2 - 11/12/2013 02:55 PM - nagachika (Tomoyuki Chikanaga)

merge revision(s) 43449,43514,43525: [Backport #8879] [Backport #8883]

```
    * load.c (ruby_init_ext): share feature names between frame name and
      provided features.

    * load.c (rb_feature_p): deal with default loadable suffixes.

    * load.c (load_lock): initialize statically linked extensions.

    * load.c (search_required, rb_require_safe): deal with statically
      linked extensions.

    * load.c (ruby_init_ext): defer initalization of statically linked
      extensions until required actually.  [Bug #8883]
    * load.c (ruby_init_ext): defer initialization of statically linked
```

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/branches/ruby_2_0_0@43656 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

History

#1 - 09/10/2013 10:12 AM - phluid61 (Matthew Kerwin)

=begin
I can't reproduce this in any version of Ruby that I have installed. What is your Ruby 2.0 patch level?

$ ruby2.0 -ve 'p Rational(2), Rational(3), Rational(2)/Rational(3)'
ruby 2.0.0p247 (2013-06-27 revision 41674) x86_64-linux
(3/1)
(2/3)
$ ruby2.0 -ve 'p (Rational(2,7)+Rational(5,7))/((2+5)/7)'
ruby 2.0.0p247 (2013-06-27 revision 41674) x86_64-linux
$ ruby2.1 -ve 'p Rational(2), Rational(3), Rational(2)/Rational(3)'
ruby 2.1.0dev (2013-08-27 trunk 42696) x86_64-linux
(3/1)
(2/3)
$ ruby2.1 -ve 'p (Rational(2,7)+Rational(5,7))/((2+5)/7)'
ruby 2.1.0dev (2013-08-27 trunk 42696) x86_64-linux

Edit: same behaviour on a fresh build of ruby 2.1.0dev (2013-09-10 trunk 42900) [x86_64-linux]
=end

#2 - 09/10/2013 01:38 PM - nobu (Nobuyoshi Nakada)

- Category changed from core to lib

=begin
Rather, it seems caused by ((%mathn%)).

$ ~/ruby/1.9.3/bin/ruby -e 'p Rational(2)*Rational(1,2)'
(1/1)
$ ~/ruby/1.9.3/bin/ruby -rmathn -e 'p Rational(2)*Rational(1,2)'
1

$ ~/ruby/2.0.0/bin/ruby -e 'p Rational(2)*Rational(1,2)'
(1/1)
$ ~/ruby/2.0.0/bin/ruby -rmathn -e 'p Rational(2)*Rational(1,2)'
1
=end

#3 - 09/10/2013 01:56 PM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Rejected

Requiring only 'mathn/rational' causes this behavior.
It's a bug to use 'mathn/rational' solely.

#4 - 09/11/2013 12:53 AM - david_macmahon (David MacMahon)

But your previous example required just mathn:

$ ruby -rmathn -e 'p Rational(2,1)'
2

It seems like a mathn bug to me.

Dave

On Sep 9, 2013, at 9:56 PM, nobu (Nobuyoshi Nakada) wrote:

> Issue #8883 has been updated by nobu (Nobuyoshi Nakada).
>
> Status changed from Open to Rejected
>
> Requiring only 'mathn/rational' causes this behavior.
>
> # It's a bug to use 'mathn/rational' solely.
>
> Bug #8883: Rational canonicalization unexpectedly converts to Fixnum

> https://bugs.ruby-lang.org/issues/8883#change-41708
>
> Author: melquiades (Paul Cantrell)
> Status: Rejected
> Priority: Normal
> Assignee:
> Category: lib
> Target version:
> ruby -v: ruby 2.0.0p247 (2013-06-27 revision 41674) [x86_64-darwin12.3.0]
> Backport: 1.9.3: UNKNOWN, 2.0.0: UNKNOWN
>
> The documentation for Rational (http://www.ruby-doc.org/core-2.0.0/Rational.html) states that the result of creating or doing arithmetic on Rationals returns Rationals, as one would expect. Examples from the docs:
>
> ```
> Rational(1)      #=> (1/1)
> 3.to_r          #=> (3/1)
> Rational(-2, 9) * Rational(-9, 2)  #=> (1/1)
> ```
>
> These all work as documented in 1.9. In 2.0, however, they all return Fixnum:
>
> ```
> Rational(1)      #=> 1
> 3.to_r          #=> 3
> Rational(-2, 9) * Rational(-9, 2)  #=> 1
> ```
>
> This leads to unexpected behavior:
>
> ```
> Rational(2) / Rational(3)  # => 0  ...but returns (2/3) in 1.9
> ```
>
> That behavior is potentially dangerous. Math that may *usually* work, but suddenly start suffering from truncation errors depending on intermediate results. For example:
>
> ```
> def should_always_return_one(a, b, c)
> (Rational(a, c) + Rational(b, c)) / (a + b) * c
> end
> ```
>
> Under 1.9:
>
> ```
> should_always_return_one(2, 3, 7) #=> (1/1)
> should_always_return_one(2, 4, 7) #=> (1/1)
> should_always_return_one(2, 5, 7) #=> (1/1)
> should_always_return_one(2, 6, 7) #=> (1/1)
> ```
>
> Under 2.0:
>
> ```
> should_always_return_one(2, 3, 7) #=> 1
> should_always_return_one(2, 4, 7) #=> 1
> should_always_return_one(2, 5, 7) #=> 0   Oops!
> should_always_return_one(2, 6, 7) #=> 1
> ```
>
> Either the docs are wrong, or this is a bug. I vote bug. Whether arithmetic expressions truncate the result should not depend on whether intermediate values just happen to be integers! Such behavior renders Rational almost too dangerous to use in situations where exact results are required. (Yes, I realize that requiring 'mathn' fixes this, but even with such a workaround as an option, this is dangerously broken. See also #2121.) Note that floating point arithmetic does <u>not</u> exhibit this behavior.
>
> --
> http://bugs.ruby-lang.org/

#5 - 09/11/2013 07:26 AM - marcandre (Marc-Andre Lafortune)

- Status changed from Rejected to Open

david_macmahon (David MacMahon) wrote:

> But your previous example required just mathn:
>
> ```
> $ ruby -rmathn -e 'p Rational(2,1)'
> 2
> ```
>
> It seems like a mathn bug to me.

Agreed.

#6 - 09/11/2013 07:28 AM - marcandre (Marc-Andre Lafortune)

- Status changed from Open to Rejected

Mmm, sorry, misread.

I think the idea is that the buggy part (Rational(2) / Rational(3) # => 0) won't happen if you require 'mathn'

#7 - 09/11/2013 08:53 AM - david_macmahon (David MacMahon)

OK. I agree that requiring mathn avoids that buggy part. Thanks for clarifying. I guess I'm just a little uncomfortable with Rationals and Fixnums being promoted/demoted as needed, but maybe it's all OK and I'm just being paranoid.

While playing around with this, I see that integer Floats also have some special handling:

```
# Without mathn...

$ ruby -e 'p [1/2.0, 1/2.5]'
[0.5, 0.4]

# With mathn...

$ ruby -r mathn -e 'p [1/2.0, 1/2.5]'
[(1/2), 0.4]
```

Oddly though, this can result in non-reduced Rationals:

```
$ ruby -r mathn -e 'p [2/2.0, 2/2.5]'
[(2/2), 0.8]
```

Weird.

Also, why do integer Floats not get changed to Fixnums like Rational and Complex do?

```
$ ruby -r mathn -e 'p Rational(1).class'
Fixnum

$ ruby -r mathn -e 'p Complex(1).class'
Fixnum

$ ruby -r mathn -e 'p Float(1).class'
Float
```

Thanks,
Dave

On Sep 10, 2013, at 3:28 PM, marcandre (Marc-Andre Lafortune) wrote:

> Issue #8883 has been updated by marcandre (Marc-Andre Lafortune).
>
> Status changed from Open to Rejected
>
> Mmm, sorry, misread.
>
> I think the idea is that the buggy part (Rational(2) / Rational(3) # => 0) won't happen if you require 'mathn'
>
> ---
>
> Bug #8883: Rational canonicalization unexpectedly converts to Fixnum
> https://bugs.ruby-lang.org/issues/8883#change-41727
>
> Author: melquiades (Paul Cantrell)
> Status: Rejected
> Priority: Normal
> Assignee:
> Category: lib
> Target version:
> ruby -v: ruby 2.0.0p247 (2013-06-27 revision 41674) [x86_64-darwin12.3.0]
> Backport: 1.9.3: UNKNOWN, 2.0.0: UNKNOWN
>
> The documentation for Rational (http://www.ruby-doc.org/core-2.0.0/Rational.html) states that the result of creating or doing arithmetic on Rationals returns Rationals, as one would expect. Examples from the docs:
>
> ```
> Rational(1)      #=> (1/1)
> 3.to_r          #=> (3/1)
> Rational(-2, 9) * Rational(-9, 2)  #=> (1/1)
> ```

> These all work as documented in 1.9. In 2.0, however, they all return Fixnum:
>
> ```
> Rational(1)      #=> 1
> 3.to_r           #=> 3
> Rational(-2, 9) * Rational(-9, 2)  #=> 1
> ```
>
> This leads to unexpected behavior:
>
> Rational(2) / Rational(3) # => 0 ...but returns (2/3) in 1.9
>
> That behavior is potentially dangerous. Math that may *usually* work, but suddenly start suffering from truncation errors depending on intermediate results. For example:
>
> ```
> def should_always_return_one(a, b, c)
> (Rational(a, c) + Rational(b, c)) / (a + b) * c
> end
> ```
>
> Under 1.9:
>
> ```
> should_always_return_one(2, 3, 7) #=> (1/1)
> should_always_return_one(2, 4, 7) #=> (1/1)
> should_always_return_one(2, 5, 7) #=> (1/1)
> should_always_return_one(2, 6, 7) #=> (1/1)
> ```
>
> Under 2.0:
>
> ```
> should_always_return_one(2, 3, 7) #=> 1
> should_always_return_one(2, 4, 7) #=> 1
> should_always_return_one(2, 5, 7) #=> 0   Oops!
> should_always_return_one(2, 6, 7) #=> 1
> ```
>
> Either the docs are wrong, or this is a bug. I vote bug. Whether arithmetic expressions truncate the result should not depend on whether intermediate values just happen to be integers! Such behavior renders Rational almost too dangerous to use in situations where exact results are required. (Yes, I realize that requiring 'mathn' fixes this, but even with such a workaround as an option, this is dangerously broken. See also #2121.) Note that floating point arithmetic does not exhibit this behavior.
>
> --
> http://bugs.ruby-lang.org/

#8 - 09/11/2013 01:09 PM - marcandre (Marc-Andre Lafortune)

david_macmahon (David MacMahon) wrote:

> While playing around with this, I see that integer Floats also have some special handling:

Right. Floats are inexact while Integers & Rational are exact, and so are Complex with exact components. Rational(1/1) and 1 should yield the same mathematical result, but with floats that can be tricky. For example there are infinitely many different bigdecimals that will map to 1.0 (say 1.000....1 and 1.000...2 with enough zeros), but they don't behave exactly the same way, for example if you substract 1), so we can't freely map them.

> Oddly though, this can result in non-reduced Rationals:
>
> ```
> $ ruby -r mathn -e 'p [2/2.0, 2/2.5]'
> [(2/2), 0.8]
> ```

Oh oh, that's a bug. It's not even related to 'mathn'. I opened a new issue about this: https://bugs.ruby-lang.org/issues/8894

#9 - 09/11/2013 01:29 PM - david_macmahon (David MacMahon)

On Sep 10, 2013, at 9:09 PM, marcandre (Marc-Andre Lafortune) wrote:

> Issue #8883 has been updated by marcandre (Marc-Andre Lafortune).
>
> david_macmahon (David MacMahon) wrote:
>
> > While playing around with this, I see that integer Floats also have some special handling:
>
> Right. Floats are inexact while Integers & Rational are exact, and so are Complex with exact components. Rational(1/1) and 1 should yield the same mathematical result, but with floats that can be tricky. For example there are infinitely many different bigdecimals that will map to 1.0 (say 1.000....1 and 1.000...2 with enough zeros), but they don't behave exactly the same way, for example if you substract 1), so we can't freely map them.

That's all fine from a numerical/mathematical point of view, but it still seems like there is something missing from the duck typing:

$ ruby -e 'p (1/1.0).nan?'
false

$ ruby -r mathn -e 'p (1/1.1).nan?'
false

$ ruby -r mathn -e 'p (1/1.0).nan?'
-e:1:in <main>': undefined methodnan?' for (1/1):Rational (NoMethodError)

Though admittedly this is getting a bit far from the OP.

> > Oddly though, this can result in non-reduced Rationals:
> >
> > $ ruby -r mathn -e 'p [2/2.0, 2/2.5]'
> > [(2/2), 0.8]
>
> Oh oh, that's a bug. It's not even related to 'mathn'. I opened a new issue about this: https://bugs.ruby-lang.org/issues/8894

Thanks!

Dave

#10 - 10/27/2013 11:44 AM - melquiades (Paul Cantrell)

Somewhere in all the discussion, the actual bug got lost. This issue shouldn't be closed.

To clarify:

(1) The bug occurs when you do not include mathn, and has nothing to do with mathn.

(2) The bug occurs when you include nothing at all:

```
$ ~/.rvm/rubies/ruby-2.0.0-p247/bin/ruby -e 'p Rational(2) / Rational(3)'
0
```

That is clearly wrong. The should_always_return_one example demonstrates why this behavior is terribly dangerous, and is probably causing mathematically incorrect results in production code right now for poor unsuspecting souls out there in the world.

(3) The bug does not occur in 1.9.3:

```
$  ~/.rvm/rubies/ruby-1.9.3-p448/bin/ruby -e 'p Rational(2) / Rational(3)'
(2/3)
```

Bottom line: promoting the results of Rational calculations to Fixnum is never safe without mathn, not ever, and Ruby should never do it.

#11 - 10/28/2013 10:38 AM - marcandre (Marc-Andre Lafortune)

Hi

melquiades (Paul Cantrell) wrote:

> (2) The bug occurs when you include nothing at all:
>
> ```
> $ ~/.rvm/rubies/ruby-2.0.0-p247/bin/ruby -e 'p Rational(2) / Rational(3)'
> 0
> ```

I can't reproduce this, with ruby 2.0.0p247, p195 nor trunk.

#12 - 10/28/2013 12:48 PM - nagachika (Tomoyuki Chikanaga)

Hello, melquiades

Don't you build your binary with --with-static-linked-ext ?
A similar issue is reported when extension library mathn/rational is statically linked.
See #8879

If so, require "mathn" explicitly ease the problem.

#13 - 11/03/2013 11:10 AM - nobu (Nobuyoshi Nakada)

- Status changed from Rejected to Closed

#14 - 11/13/2013 12:05 AM - nagachika (Tomoyuki Chikanaga)

- Backport changed from 1.9.3: UNKNOWN, 2.0.0: UNKNOWN to 1.9.3: UNKNOWN, 2.0.0: DONE

r43449, r43514 and r43525 are backported to ruby_2_0_0 at r43656.

#15 - 11/19/2013 04:43 PM - melquiades (Paul Cantrell)

nagachika (Tomoyuki Chikanaga): Yes, your guess is correct. I am using rvm, which passes --with-static-linked-ext.

I verified that patch 43656 does indeed fix the issue:

```
$ rvm install 2.0.0-patch43656 --patch changeset_r43656.diff
...
 ~/.rvm/rubies/ruby-2.0.0-p247-patch43656/bin/ruby -e 'p Rational(2) / Rational(3)'
(2/3)
```

Hooray!

(Apologies for my slow responses. Apparently I'm not receiving email notifications on this thread, despite having watched it.)