Ruby - Feature #5310

Integral objects

09/13/2011 10:15 AM - mrkn (Kenta Murata)

Status:	Assigned
Priority:	Normal
Assignee:	mrkn (Kenta Murata)
Target version:	

Description

I believe it is ambiguous what object can behave as an integral number.
I don't think the current use of Object#to_int isn't appropriate for this purpose.

The most understandable example is Float#to_int.
It should raise error for all float values because they always have uncertainty,
but it doesn't and returns an integral part of it.

I propose to change the use of Object#to_int for the next release of Ruby.
I recommend the following specification changes:

(1) Remove to_int method from Float and BigDecimal.
(2) Rational#to_int returns an Integer only if its denominator is 1. Otherwise, it raises an appropriate error.
(3) Complex#to_int returns the result of to_int of its real part only if its imaginary part is exactly zero (0.0 isn't exactly zero).

If anyone have another idea, please give me your comment.

Related issues:

Related to Ruby - Bug #1792: Fixnum#& □□□Rational □□□□□□□□	**Closed**	**07/19/2009**
Related to Ruby - Feature #6973: Add an #integral? method to Numeric to test ...	**Assigned**	

History

#1 - 09/13/2011 10:23 AM - mrkn (Kenta Murata)

- Target version changed from 2.0.0 to 3.0

#2 - 09/13/2011 11:23 AM - brixen (Brian Shirai)

On Mon, Sep 12, 2011 at 6:15 PM, Kenta Murata muraken@gmail.com wrote:

> Issue #5310 has been reported by Kenta Murata.
>
> ---
>
> Feature #5310: Integral objects
> http://redmine.ruby-lang.org/issues/5310
>
> Author: Kenta Murata
> Status: Open
> Priority: Normal
> Assignee:
> Category: core
> Target version: 1.9.x
>
> I believe it is ambiguous what object can behave as an integral number.
> I don't think the current use of Object#to_int isn't appropriate for this purpose.
>
> The most understandable example is Float#to_int.
> It should raise error for all float values because they always have uncertainty,
> but it doesn't and returns an integral part of it.
>
> I propose to change the use of Object#to_int for the next release of Ruby.
> I recommend the following specification changes:
>
> (1) Remove to_int method from Float and BigDecimal.
> (2) Rational#to_int returns an Integer only if its denominator is 1. Otherwise, it raises an appropriate error.
> (3) Complex#to_int returns the result of to_int of its real part only if its imaginary part is exactly zero (0.0 isn't exactly zero).
>
> If anyone have another idea, please give me your comment.

I strongly disagree with this proposal.

Any object should be allowed to participate in integral operations
based on the object's implementing #to_int. When object A requests
that object B represent itself as an integral value, it is up to
object B to do so, or not do so. The only thing that object A should
require is that the value returned from #to_int be an integral value.
The object A should have no say is how the object B represents itself.
To do so is to 1) severely break encapsulation; 2) impose ad hoc
type/class requirements that break ducktyping; 3) create more brittle,
non-OO code.

Should it not be clear, allow me to reiterate that I am adamantly
opposed to this change.

Cheers,
Brian

> --
> http://redmine.ruby-lang.org

#3 - 09/13/2011 11:53 AM - mrkn (Kenta Murata)

I believe you are misreading of the topic.

On Tuesday, September 13, 2011 at 11:03 , brian ford wrote:

> Any object should be allowed to participate in integral operations
> based on the object's implementing #to_int.

My proposal doesn't disturb that, and I don't want to interfere that.

I want to allow anyone to create original "integral" numbers which can behave alike Fixnum and Bignum.
Unfortunately, to_int is currently used for converting to an Integer from a non-integral, inexact number like a Float.

--
Kenta Murata
Sent with Sparrow (http://www.sparrowmailapp.com)

#4 - 09/14/2011 06:29 AM - brixen (Brian Shirai)

Hi,

On Mon, Sep 12, 2011 at 7:30 PM, Kenta Murata muraken@gmail.com wrote:

> Â I believe you are misreading of the topic.

There is some inconsistency between your proposal and what has been implemented:

integral.rb

class Numberish
def initialize(value)
@value

#5 - 09/14/2011 09:23 AM - mrkn (Kenta Murata)

Hi,

On Wednesday, September 14, 2011 at 06:23 , brian ford wrote:

> There is some inconsistency between your proposal and what has been implemented:

We can change the implementation according to the proposal if accepted.

> A Float value is a machine approximation of a mathematical real
> number. A BigDecimal is an exact representation of a real number. The
> mathematical real numbers embed the integers.

You are not right.
A BigDecimal is a floating-point number same as a Float except for internal representation.
So, A BigDecimal is also approximation of a real number,
in other words, a BigDecimal has error as well as a Float does.
It is right understanding because I am the master of bigdecimal.

> It is untrue that Float numbers cannot be consistently represented as
> integral values. It is merely up to the language to define them as
> such. Ruby already takes liberties with defining mathematical
> operations (see http://redmine.ruby-lang.org/issues/3289).

I know. I suggest to change for number system.

> To remove #to_int from Float and BigDecimal and partially from
> Rational and Complex introduces typing concepts where none are needed,
> breaks consistent polymorphism, and breaks compatibility with 1.8 and
> prior 1.9.

Yes, my proposal introduces incompatibility, so I propose this for 2.0.

--
Kenta Murata
Sent with Sparrow (http://www.sparrowmailapp.com)

#6 - 09/14/2011 10:53 AM - brixen (Brian Shirai)

On Tue, Sep 13, 2011 at 5:18 PM, Kenta Murata muraken@gmail.com wrote:

> Hi,
>
> On Wednesday, September 14, 2011 at 06:23 , brian ford wrote:
>
> > There is some inconsistency between your proposal and what has been implemented:
>
> We can change the implementation according to the proposal if accepted.

But you have already changed the implementation and your change is
inconsistent with what you are claiming in your response. If you are
not intending to exclude other objects implementing #to_int, then why
did you implement it that way? That's confusing.

> > A Float value is a machine approximation of a mathematical real
> > number. A BigDecimal is an exact representation of a real number. The
> > mathematical real numbers embed the integers.
>
> You are not right.
> A BigDecimal is a floating-point number same as a Float except for internal representation.
> So, A BigDecimal is also approximation of a real number,
> in other words, a BigDecimal has error as well as a Float does.
> It is right understanding because I am the master of bigdecimal.

BigDecimal is an arbitrary precision floating-point library. There are
other arbitrary precision floating-point libraries. The
characteristics of BigDecimal really have nothing to do with this
discussion anyway.

A real-number approximation can be easily represented as an integral
value any number of ways. It can be consistently represented using any
one of those any number of ways. A real-number approximation is no
less and no more an integral value than the Numberish object in my
example. There is no reason to introduce this arbitrary distinction in
Ruby. I could just as easily define Numberish as:

```
class Numberish
def initialize(value)
@value = value
end
```

def to_int
@value.to_i # or *anything* else, even just 1
end
end

n = Numberish 4.2

Why is this change needed? Please don't reiterate this argument about
imprecise floating-point values. What problems does this change fix?

Thanks,
Brian

> > It is untrue that Float numbers cannot be consistently represented as
> > integral values. It is merely up to the language to define them as
> > such. Ruby already takes liberties with defining mathematical
> > operations (see http://redmine.ruby-lang.org/issues/3289).
>
> I know. I suggest to change for number system.
>
> > To remove #to_int from Float and BigDecimal and partially from
> > Rational and Complex introduces typing concepts where none are needed,
> > breaks consistent polymorphism, and breaks compatibility with 1.8 and
> > prior 1.9.
>
> Yes, my proposal introduces incompatibility, so I propose this for 2.0.
>
> --
> Kenta Murata
> Sent with Sparrow (http://www.sparrowmailapp.com)

#7 - 09/14/2011 05:53 PM - matz (Yukihiro Matsumoto)

Hi,

I strongly disagree to use to_int (currently working for integer
conversion) as integral conversion. Note that I don't disagree (yet)
to introduce concept of integrals to Ruby in the future. But
recycling name is not ideal.

```
matz.
```

#8 - 09/15/2011 03:53 AM - brixen (Brian Shirai)

Hi Matz,

On Wed, Sep 14, 2011 at 1:31 AM, Yukihiro Matsumoto matz@ruby-lang.org wrote:

> Hi,
>
> I strongly disagree to use to_int (currently working for integer
> conversion) as integral conversion. Â Note that I don't disagree (yet)
> to introduce concept of integrals to Ruby in the future. Â But
> recycling name is not ideal.

Could you explain what you mean by integer conversion versus integral
conversion?

I'm still completely lost on what problem this proposal is intending to fix.

Thanks,
Brian

#9 - 09/18/2011 07:07 PM - alexeymuranov (Alexey Muranov)

Hello, i also do not understand very well the issue.

Am i right that it has to do with the difference between #to_int and #to_i methods, similar to the difference between #to_ary and #to_a methods? Do i understand correctly that #to_a is a conversion to an Array of anything that can be converted, and #to_ary is an "easy conversion to Array" reserved for objects that are essentially arrays, and similarly #to_int is reserved for objects that are essentially integers?

In this case i agree with the proposal: to convert a Float to Integer, only #to_i should be allowed, because Floats store approximate values, and Integers store exact values, so Floats are not "essentially" Integers.

I think it would be nice if the rule for using #to_ary and #to_int was the following: conversion back and forth (if a corresponding inverse conversion exists) should always return the same value.
Also the conversion should whenever possible commute with some operations: sum, concatenation, etc.
Currently i get: (10000000000000000.to_f+1.to_f).to_int #=> 10000000000000000
(but of course #to_f is not an "easy conversion", so this in not wrong).
Probably this is impossible to observe exactly, but maybe to some extent?
This is just an idea.

Alexey.

#10 - 09/18/2011 09:16 PM - alexeymuranov (Alexey Muranov)

@Brian, if i understood correctly, the proposal is intending to fix the problem that Float and BigDecimal should not respond to #to_int, and in some other cases #to_int should raise an Error on some inputs.
Sorry, this sounds like a tautology :).

Alexey.

#11 - 10/18/2011 09:16 AM - naruse (Yui NARUSE)

- Project changed from Ruby to 14

- Category deleted (core)

- Target version deleted (3.0)

#12 - 10/23/2011 05:21 PM - naruse (Yui NARUSE)

- Project changed from 14 to Ruby

#13 - 03/27/2012 03:04 AM - mame (Yusuke Endoh)

- Status changed from Open to Assigned

- Assignee set to mrkn (Kenta Murata)

Hello, Mrkn-san

Could you tell me the status?

--
Yusuke Endoh mame@tsg.ne.jp

#14 - 11/20/2012 09:48 PM - mame (Yusuke Endoh)

- Target version set to 2.6

#15 - 12/25/2017 06:15 PM - naruse (Yui NARUSE)

- Target version deleted (2.6)