Rational number literal

I would like to propose a new literal syntax for rational numbers. The implementation is available in my github repository: https://github.com/mrkn/ruby/commit/8ca0c9a53593e55d67f509fc403df616e2276e3a

This patch implements a notation that consists of an integer, "//", and another integer, in a row. The first integer is the numerator, and the second is the denominator. Whitespaces are permitted between them.

For example:
1 // 2 == Rational(1, 2)
1 // 1 == Rational(1, 1)
0 // 1 == Rational(0, 1)

"0 // 0" occurs syntax error.

I think this new syntax isn't conflict with an empty regexp because this implementation doesn't treat // as a binary operator.

Associated revisions

Revision e06407cf - 08/01/2013 02:58 PM - mrkn (Kenta Murata)

- rational.c (rb_flt_rationalize_with_prec): new public C function to rationalize a Float instance with a precision.
- rational.c (rb_flt_rationalize): new public C function to rationalize a Float instance. A precision is calculated from the given float number.
- include/ruby/intern.h: Add rb_flt_rationalize_with_prec and rb_flt_rationalize.
- parse.y: implement number literal suffixes, ‘r’ and ‘i’.
  [ruby-core:55096] [Feature #8430]
- bootstrap/test_literal_suffix.rb: add tests for parser to scan number literals with the above tsuffixes.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@42311 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 42311 - 08/01/2013 02:58 PM - mrkn (Kenta Murata)

- rational.c (rb_flt_rationalize_with_prec): new public C function to rationalize a Float instance with a precision.
- rational.c (rb_flt_rationalize): new public C function to rationalize a Float instance. A precision is calculated from the given float number.
- include/ruby/intern.h: Add rb_flt_rationalize_with_prec and rb_flt_rationalize.
- parse.y: implement number literal suffixes, 'r' and 'i'.
  [ruby-core:55096] [Feature #8430]

- bootstrap/test/test_literal_suffix.rb: add tests for parser to scan number literals with the above tsuffixes.

Revision 42311 - 08/01/2013 02:58 PM - mrkn (Kenta Murata)

- rational.c (rb_flt_rationalize_with_prec): new public C function to rationalize a Float instance with a precision.

- rational.c (rb_flt_rationalize): new public C function to rationalize a Float instance. A precision is calculated from the given float number.

- include/ruby/intern.h: Add rb_flt_rationalize_with_prec and rb_flt_rationalize.

- parse.y: implement number literal suffixes, 'r' and 'i'.
  [ruby-core:55096] [Feature #8430]

- bootstrap/test/test_literal_suffix.rb: add tests for parser to scan number literals with the above tsuffixes.

Revision 42311 - 08/01/2013 02:58 PM - mrkn (Kenta Murata)

- rational.c (rb_flt_rationalize_with_prec): new public C function to rationalize a Float instance with a precision.

- rational.c (rb_flt_rationalize): new public C function to rationalize a Float instance. A precision is calculated from the given float number.

- include/ruby/intern.h: Add rb_flt_rationalize_with_prec and rb_flt_rationalize.

- parse.y: implement number literal suffixes, 'r' and 'i'.
  [ruby-core:55096] [Feature #8430]

- bootstrap/test/test_literal_suffix.rb: add tests for parser to scan number literals with the above tsuffixes.

Revision 42311 - 08/01/2013 02:58 PM - mrkn (Kenta Murata)

- rational.c (rb_flt_rationalize_with_prec): new public C function to rationalize a Float instance with a precision.

- rational.c (rb_flt_rationalize): new public C function to rationalize a Float instance. A precision is calculated from the given float number.

- include/ruby/intern.h: Add rb_flt_rationalize_with_prec and rb_flt_rationalize.

- parse.y: implement number literal suffixes, 'r' and 'i'.
  [ruby-core:55096] [Feature #8430]

- bootstrap/test/test_literal_suffix.rb: add tests for parser to scan number literals with the above tsuffixes.

08/23/2022
rational.c (rb_flt_rationalize_with_prec): new public C function to rationalize a Float instance with a precision.

rational.c (rb_flt_rationalize): new public C function to rationalize a Float instance. A precision is calculated from the given float number.

include/ruby/intern.h: Add rb_flt_rationalize_with_prec and rb_flt_rationalize.

parse.y: implement number literal suffixes, ‘r’ and ‘i’. [ruby-core:55096] [Feature #8430]

bootstrap/test_literal_suffix.rb: add tests for parser to scan number literals with the above suffixes.

Revision 42311 - 08/01/2013 02:58 PM - mrkn (Kenta Murata)

rational.c (rb_flt_rationalize_with_prec): new public C function to rationalize a Float instance with a precision.

rational.c (rb_flt_rationalize): new public C function to rationalize a Float instance. A precision is calculated from the given float number.

include/ruby/intern.h: Add rb_flt_rationalize_with_prec and rb_flt_rationalize.

parse.y: implement number literal suffixes, ‘r’ and ‘i’. [ruby-core:55096] [Feature #8430]

bootstrap/test_literal_suffix.rb: add tests for parser to scan number literals with the above suffixes.

History

#1 - 05/21/2013 10:43 PM - mrkn (Kenta Murata)
I updated the patch for fixing negative rational numbers:
https://github.com/mrkn/ruby/commit/b16f4da890e1cb68058098612b7d38fa44182151

#2 - 05/22/2013 11:08 AM - nobu (Nobuyoshi Nakada)
Shouldn't tDIV2 be only if IS_SPCARG(c)?

#3 - 05/22/2013 11:49 AM - knu (Akinori MUSHA)
I think it should be added as an operator rather than a literal notation.

A literal should not look like an expression, or it will fail you when you find out you have to give up the // notation in order to constify a numerator and/or a denominator of a rational literal like that.

Also, I guess runtime cost would not change much if it were introduced as operator.

#4 - 05/22/2013 12:26 PM - charliesome (Charlie Somerville)

Also, I guess runtime cost would not change much if it were introduced as operator.

Rational is immutable, so if it was introduced as a literal, the same Rational instance could be re-used, similar to how symbols and fixnums work.
If // was introduced as an operator, a new object would need to be allocated each time, similar to strings.

#5 - 05/22/2013 01:45 PM - phluid61 (Matthew Kerwin)
charliesome (Charlie Somerville) wrote:

Also, I guess runtime cost would not change much if it were introduced as operator.

Rational is immutable, so if it was introduced as a literal, the same Rational instance could be re-used, similar to how symbols and fixnums work.

If // was introduced as an operator, a new object would need to be allocated each time, similar to strings.

At the risk of asking something stupid, could it be both? I'm thinking of unary minus / literal negative number. (Assuming there's such a thing in Ruby as a literal negative (?))

#6 - 05/22/2013 05:15 PM - nobu (Nobuyoshi Nakada)
I also prefer an operator.

https://github.com/nobu/ruby/commit/7ff6073d70306999c0def3387387649ccd13f9d6

#7 - 05/22/2013 05:21 PM - mrkn (Kenta Murata)
knu (Akinori MUSHA) wrote:

I think it should be added as an operator rather than a literal notation.

A literal should not look like an expression, or it will fail you when you find out you have to give up the // notation in order to constify a numerator and/or a denominator of a rational literal like that.

I made another implementation of the rational number literal implemented in token-level. The implementation is available in https://github.com/mrkn/ruby/commit/f0bf41b6593866b82ab0068e6a66ce7c12748aec
Whitespaces around of // aren't permitted in this implementation.

#8 - 05/22/2013 05:25 PM - mrkn (Kenta Murata)
nobu (Nobuyoshi Nakada) wrote:

I also prefer an operator.

https://github.com/nobu/ruby/commit/7ff6073d70306999c0def3387387649ccd13f9d6

Pretty nice!

I love this if introducing // as a operator is accepted.

#9 - 05/29/2013 09:47 AM - mrkn (Kenta Murata)
Yesterday I ask matz about merging it to trunk, he said it is ok with experimental mark like refinements.

And I confirmed that nobu's quo-operator doesn't conflict with empty regexps:

$ .prefix/bin/ruby -e 'f=1; i=2; p(f //i)'

(1)

$ .prefix/bin/ruby -e 'def f(a); p a; end; i=2; p(f //i)'

//i

//i

#10 - 05/29/2013 05:36 PM - nobu (Nobuyoshi Nakada)

$ .prefix/bin/ruby -e 'foo.split // bar'

It is "(((foo.split(/) bar))" currently, but it would be "(((foo.split/or)) bar)" with the patch.

Of course it's possible to address it heuristically, but I wonder if it's good or worth to do.
nobu (Nobuyoshi Nakada) wrote:

Of course it's possible to address it heuristically, but I wonder if it's good or worth to do.

How about a different symbol? Since $\{(\)}$ is the standard mathematical symbol for ratio, why not define $\{([\pm\d+:\d+])\} as an immediate Rational, and/or $\{(\}) as an operator?

As an operator it conflicts with short symbol/hash syntax only in the case that the left-hand operand is a tIDENTIFIER (like local variable/function) and there is no whitespace before the $\{(\})$. Not sure if that's better or worse than the empty regexp conflict.

phluid61 (Matthew Kerwin) wrote:

How about a different symbol? Since $\{(\})$ is the standard mathematical symbol for ratio, why not define $\{([\pm\d+:\d+])\} as an immediate Rational, and/or $\{(\}) as an operator?

It collides to the conditional operator, ?; for example:

expr ? 1:2

As I discussed with matz and akr today, the token-level implementation of $\//$ doesn't introduce incompatibility, so it can be introduced safely.

different symbol changes the interpretation.

http://www.jaro.in/international-MBA-1-year.html

headius (Charles Nutter) wrote:

Not sure if the debate still rages, but something that keeps the numerator and denominator together in a single "literal" seems better to me. Why not add another % literal?

%R[1,2]

There's no other literals that are produced via a magic infix operator, and it seems confusing to me.

headius (Charles Nutter) wrote:

Not sure if the debate still rages, but something that keeps the numerator and denominator together in a single "literal" seems better to me. Why not add another % literal?

%R[1,2]

There's no other literals that are produced via a magic infix operator, and it seems confusing to me.

+1 this seems to introduce the least potential for confusion and backwards-incompatibility. Question: at what place(s) would a negative sign be allowed? %R[-1,2], %R[1,-2], etc.?

mrkn (Kenta Murata) wrote:

%R[1,2]

It isn't similar to a fraction.

mrkn (Kenta Murata) wrote:

%R[1,2]
It isn't similar to a fraction.

Does it have to be?

How about %R{1/2} then?

#18 - 06/25/2013 05:49 AM - headius (Charles Nutter)

phluid61 (Matthew Kerwin) wrote:

headius (Charles Nutter) wrote:

%R{1/2}

+1 this seems to introduce the least potential for confusion and backwards-incompatibility. Question: at what place(s) would a negative sign be allowed? %R{-1,2}, %R{1,-2}, etc.?

I suppose anywhere you can pass them to Rational's constructor...so both numerator and denominator.

If the slash syntax is more to @mrkn's liking, these examples would be %R{-1/2} and %R{1/-2}.

Note that since Rational() takes a string, the slashing %R format fits well the other % formats, in that % formats wrap something string-like that's then processed into a more specific data type, cacheable as a literal object in many cases.

I really do not like the // magic infix literal format.

#19 - 06/28/2013 10:40 AM - mrkn (Kenta Murata)

headius (Charles Nutter) wrote:

If the slash syntax is more to @mrkn's liking, these examples would be %R{-1/2} and %R{1/-2}.

I don't hate this form. It is better than %R{1,2}.
But I think 1/2 looks like a fraction than %R{1}/2).

#20 - 06/28/2013 11:02 AM - phluid61 (Matthew Kerwin)

mrkn (Kenta Murata) wrote:

headius (Charles Nutter) wrote:

If the slash syntax is more to @mrkn's liking, these examples would be %R{-1/2} and %R{1/-2}.

I don't hate this form. It is better than %R{1,2}.
But I think 1/2 looks like a fraction than %R{1}/2).

Slightly bike-shedding, but I have issues with // as an operator. For one, it immediately screams "comment" at me, even though ruby comments use #, simply because so many other languages use it. Then after that, I wonder at the relationship between / and // when compared to * and ** (i.e. is // meant to mean some sort of multi-phased divide, or a logarithm?)

The reason I particularly like %R{1}/2 is that it contains the existing division operator, 1/2, without any modification, so at a glance you can see that 1/2 and %R{1}/2 are in some way equivalent, but the %R{} around it adds some flavourful difference.

The %R format also lends itself to interpolation, if such is deemed to be useful, e.g. %R{#foo}/2), which could arguably be more or less useful than variable operands.

#21 - 07/27/2013 04:16 PM - matz (Yukihiro Matsumoto)

The final idea was "1r" to be "Rational(1,1)". We also accept the idea of "1i" as "Complex(0,1)".

Matz.

#22 - 07/27/2013 04:29 PM - mrkn (Kenta Murata)

matz (Yukihiro Matsumoto) wrote:

The final idea was "1r" to be "Rational(1,1)". We also accept the idea of "1i" as "Complex(0,1)".
Additionaly them, "1.2r" as "Rational(12, 10)" is also accepted.
But the exponential form with "r" suffix like "1e-5r" is not accepted because it can make us confusing.

#23 - 07/28/2013 12:09 AM - mrkn (Kenta Murata)
- Status changed from Open to Assigned
- Assignee changed from matz (Yukihiro Matsumoto) to mrkn (Kenta Murata)

#24 - 08/01/2013 04:53 PM - takuto_h (Takuto Hayashi)
- File ratio_lit.patch added

Hello.
I wrote a patch for the rational number literal and recognized that a part of the proposed feature is confusable.

If we accept "1.2r" as "Rational(12, 10)"
1/3r      #=> (1/3)
0.4/1.2r  #=> 0.33333333333333337
I think this feature's point is that "1/3r" can be seen as "1/3" followed by "r",
so it can make us confusing that "0.4/1.2" followed by "r" is not a rational number.

The attached file is a patch which accept "3r" and doesn't accept "1.2r".
The implementation is also available at: https://github.com/takuto-h/ruby/commit/6827699e642c3afdf57af35af481377a0038a402

Thank you.

#25 - 08/01/2013 10:05 PM - mame (Yusuke Endoh)
takuto_h (Takuto Hayashi) wrote:

Hello.
I wrote a patch for the rational number literal

Great.

and recognized that a part of the proposed feature is confusable.

If we accept "1.2r" as "Rational(12, 10)"
1/3r      #=> (1/3)
0.4/1.2r  #=> 0.33333333333333337
I think this feature's point is that "1/3r" can be seen as "1/3" followed by "r",
so it can make us confusing that "0.4/1.2" followed by "r" is not a rational number.

IMO, it does not matter because we don't usually write a rational whose numerator and denominator are decimal.
Also, it is very clear and reasonable what happens.

--
Yusuke Endoh mame@tsg.ne.jp

#26 - 08/01/2013 11:58 PM - mrkn (Kenta Murata)
- Status changed from Assigned to Closed
- % Done changed from 0 to 100

This issue was solved with changeset r42311.
Kenta, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

- rational.c (rb_flt_rationalize_with_prec): new public C function
to rationalize a Float instance with a precision.
- rational.c (rb_flt_rationalize): new public C function
to rationalize a Float instance. A precision is calculated from
the given float number.
- include/ruby/intern.h: Add rb_flt_rationalize_with_prec and rb_flt_rationalize.
- parse.y: implement number literal suffixes, ‘r’ and ‘i’.
  [ruby-core:55096] [Feature #8430]
- bootstrap/test/test_literal_suffix.rb: add tests for parser to scan number literals with the above tsuffixes.

#27 - 08/02/2013 12:19 AM - mrkn (Kenta Murata)
takuto_h (Takuto Hayashi) wrote:

Hello.
I wrote a patch for the rational number literal and recognized that a part of the proposed feature is confusable.

Thank you. But I'd already have a patch made by me on the last Sunday. Although I've committed the changes based on my patch, I really appreciate your contribution for this issue.

#28 - 08/02/2013 01:59 AM - david_macmahon (David MacMahon)
On Aug 1, 2013, at 12:53 AM, takuto_h (Takuto Hayashi) wrote:

If we accept "1.2r" as "Rational(12, 10)":
  1/3r   #=> (1/3)
  0.4/1.2r #=> 0.33333333333333337

I think this feature's point is that "1/3r" can be seen as "1/3" followed by "r", so it can make us confusing that "0.4/1.2" followed by "r" is not a rational number.

I haven't looked at the implementation, but my understanding was that 1/3r was seen as 1 divided by Rational(3,1). With that interpretation, I think 0.4/1.2r is not confusing: 0.4/1.2r is a Float (i.e. 0.4) divided by a Rational (i.e. Rational(12,10) or 1.2r). Float divided by Rational gives Float. To end up with a Rational result, use 0.4r/1.2r.

The attached file is a patch which accept "3r" and doesn't accept "1.2r".

I very much like the idea of "1.2r", let's please keep accepting 1.2r.

Thanks,
Dave

#29 - 08/10/2016 02:54 AM - shyouhei (Shyouhei Urabe)
- Related to deleted (Feature #8643: Add Binding.from_hash)

Files
ratio_lit.patch 2.11 KB 08/01/2013 takuto_h (Takuto Hayashi)