Proposal: Add methods to determine if it is an infinite range

Proposal to add methods to determine if it is an infinite range.

Current status

# begin / end return nil
p (..10).begin  # => nil
p (1..).end     # => nil

# But, first / last raise an exception
p (..10).first  # Error: in `first': cannot get the first element of beginless range (RangeError)
p (1..).last    # Error: in `last': cannot get the last element of endless range (RangeError)

# Returns Infinity if it is a Numeric
p (1..10).size  # => 10
p (1..).size    # => Infinity
p (..1).size    # => Infinity

# Otherwise returns nil
p ("a".."z").size  # => nil
p ("a"..).size     # => nil
p (.."z").size     # => nil

- (..,10).begin return nil, but (..,10).first raise an exception.
- (1..).size return Infinity, but ("a"..).size return nil.
- Behavior changes depending on the state of Range and the called method.
- It is difficult to determine if it is an infinite range.

Proposal methods

- Range#beginless?
  - return true if begin is nil
- Range#endless?
  - return true if end is nil
- Range#infinite?
  - return true if begin is nil or end is nil
- Range#finite?
  - return true if begin is not nil and end is not nil

Example

p (1..10).beginless?  # => false
p (1..10).endless?    # => false
p (1..10).infinite?   # => false
p (1..10).finite?     # => true

p (..10).beginless?  # => true
p (..10).endless?    # => false
p (..10).infinite?   # => true
p (..10).finite?     # => false
p (1..).beginless? #=> false
p (1..).endless? #=> true
p (1..).infinite? #=> true
p (1..).finite? #=> false

# NOTE: Float::INFINITY is not support
p (1..Float::INFINITY).beginless? #=> false
p (1..Float::INFINITY).endless? #=> false
p (1..Float::INFINITY).infinite? #=> false
p (1..Float::INFINITY).finite? #=> true

Use case

before

def search_in(range)
  query = "#/search"
  if !(range.begin.nil? || range.end.nil?)
    "#{query}?from=#{range.begin}&to=#{range.end}"
  elsif range.begin.nil?
    "#{query}?to=#{range.end}"  
  elsif range.end.nil?
    "#{query}?from=#{range.begin}"
  else
    query
  end
end

p search_in("2019-1-1".."2019-4-30")
# => "/search?from=2019-1-1&to=2019-4-30"

p search_in(.."2019-4-30")
# => "/search?to=2019-4-30"

p search_in("2019-5-1"..)
# => "/search?from=2019-5-1"

after

def search_in(range)
  query = "#/search"
  if range.finite?
    "#{query}?from=#{range.begin}&to=#{range.end}"
  elsif range.beginless?
    "#{query}?to=#{range.end}"
  elsif range.endless?
    "#{query}?from=#{range.begin}"
  else
    query
  end
end

p search_in("2019-1-1".."2019-4-30")
# => "/search?from=2019-1-1&to=2019-4-30"

p search_in(.."2019-4-30")
# => "/search?to=2019-4-30"

p search_in("2019-5-1"..)
# => "/search?from=2019-5-1"

Memo

- Check whether the tip is infinite only with nil.
- Float::INFINITY is not supported.
  - I think that there is no relation between Float::INFINITY and infinite Range.
I would like an opinion on how to determine if it is infinite.
- see range.begin.infinite? ?
- Whether there is a better name for the method name.
  - e.g. #infinite? to #infinity?.

Thank you.

github pull request: https://github.com/ruby/ruby/pull/2196

Related issues:
Related to Ruby master - Misc #16114: Naming of "beginless range"

History

#1 - 05/21/2019 08:16 AM - shevegen (Robert A. Heiler)
I think the proposal makes sense. The method names are a bit odd though.

I have no huge qualms with the inf* related methods, such as infinite? or finite? although it reads a bit odd; but I think that .beginless? and .endless? are very clumsy names.

Unfortunately I do not have better alternative suggestions either.

Perhaps it could suffice to only add e.g. ".infinite?" and we may not even need ".finite?".

The use case on the other hand is clear to me, IMO, in particular 
"(1..).size return Infinity, but ("a"..).size return nil." so it would make sense to give ruby users the ability to check for infinite in advance, rather than check for nil lateron.

I think that there is no relation between Float::INFINITY and infinite Range

I guess a/any concept of infinity should not necessarily be intrinsically tied down to a particular constant (e.g. Float::INFINITY) per se; in the latter case it would seem more like an implementation detail (to me), rather than a more general concept for/of infinity.

#2 - 05/21/2019 06:58 PM - Eregon (Benoit Daloze)
Why is range.begin.nil? || range.end.nil? not enough to check if beginless or endless Range?

Maybe we should integrate beginless/endless in pattern matching?

The use-case can be simplified quite a bit if we can assume false is not used as a Range endpoint:

```ruby
def search_in(range)
  query = "search"
  if range.begin && range.end
    "#{query}?from=#{range.begin}&to=#{range.end}"
  elsif range.end
    "#{query}?to=#{range.end}"
  elsif range.begin
    "#{query}?from=#{range.begin}"
  else
    query
  end
end
```

I think infinite? is a different concept, which already exists as Numeric#infinite?, and should definitely handle the Float::INFINITY case if introduced.

I think these are probably bugs:

```ruby
p ("a"..).size  # => nil
p (.."z").size  # => nil
```

and should return infinity.

#3 - 05/21/2019 11:13 PM - mame (Yusuke Endoh)
I have no strong opinion to the proposal itself, but I'm also skeptical to the motivating example. I'd like to write:

```ruby
def search_in(range)
```

03/19/2022
I have no idea when we want to check if a range is just infinite without checking the actual values.

Currently, "(a..".size #=> nil is intended because ("a".."z").size #=> nil. According to marcandre (Marc-Andre Lafortune)
https://bugs.ruby-lang.org/issues/14699#change-71575, there is no special reason why ("a".."z").size returns nil. So after ("a".."z").size is
implemented correctly, I agree that ("a".."z").size should return infinity.

#4 - 05/22/2019 01:14 AM - osyo (manga osyo)

Thanks comments!

I think infinite? is a different concept, which already exists as Numeric#infinite?, and should definitely handle the Float::INFINITY case if introduced

I also considered the following implementation.

class Range
  def beginless?
    self.begin.nil? || (self.begin.respond_to?(:infinite?) & self.begin.infinite?)
  end
end

MEMO: Numeric#infinite? should also consider returning something other than true / false.

So after ("a".."z").size is implemented correctly, I agree that ("a".."z").size should return infinity.

I'm concerned about the performance of ("a".."z").size. Can implement ("a".."z").size withO(1)?

#5 - 05/22/2019 01:22 AM - osyo (manga osyo)

How about doing this?

("a".."z").size #=> nil
(\.."z").size #=> Infinity
(\"a\"..\"z\").size #=> Infinity

#6 - 05/22/2019 02:54 AM - mame (Yusuke Endoh)

Do you mean that Range#infinite? is no longer needed if the behavior of ("a".."z").size is changed? If not, I think it is a different topic from this ticket.

#7 - 05/22/2019 08:56 AM - matz (Yukihiro Matsumoto)

I am against having finite? or infinite? methods. I don't think there's use-case for those methods. Meanwhile, I see the small possibility for the usage of beginless? and endless? methods. But I don't like the names.

Matz.

#8 - 08/22/2019 12:05 AM - mame (Yusuke Endoh)

- Related to Misc #16114: Naming of "beginless range" added

#9 - 08/22/2019 12:07 AM - mrkn (Kenta Murata)

Other candidates from mathematical terms:

- unbounded? for checking whether begin or end is nil
- lower_unbounded?, left_unbounded?, or unbounded_below? for checking whether begin is nil
- upper_unbounded?, right_unbounded?, or unbounded_above? for checking whether end is nil

#10 - 08/22/2019 03:23 AM - sawa (Tsuyoshi Sawada)

Having negative meaning in the method name would make it complicated. Having begin? and end? instead of beginless? and endless? would be more natural, and easy to comprehend.

In fact, the given use case:
is not straightforward; it is twisted. The intent of the elsif range.beginless? condition is actually to check whether the end exists, not whether the beginning is absent. Using begin? and end?, a more straightforward code can be written as follows (I also believe your last condition is redundant, and there is no case that corresponds to your else case).

def search_in(range)
    query = "/search"
    if range.finite?
        "#{query}?from=#{range.begin}&to=#{range.end}"
    elsif range.beginless?
        "#{query}?to=#{range.end}"
    elsif range.endless?
        "#{query}?from=#{range.begin}"  
    else
        query
        end
    end
end

Having begin? and end? instead of beginless? and endless? would be more natural, and easy to comprehend.

Agreed. Moreover, Range#begin and #end just work.