Ruby master - Feature #7545

Make Range act as a "lazy ordered set"

12/12/2012 01:05 AM - alexeymuranov (Alexey Muranov)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	3.0

Description

=begin

Make Range act as a "lazy ordered set"

This replaces my older feature request #5534.

I propose the following new behavior of (({Range})):

```
(1..3).to_a              # => 1, 2, 3.to_a              # => [3, 2, 1]
'a'..'c'.to_a            # => ['a', 'b', 'c']
'c'..'a'.to_a            # => ['c', 'b', 'a']
1...1 == 3...3           # => true
1...1 == 'a'...'a'       # => true
1...1 == Range::EMPTY_SET # => true
1..2 == 1...3            # => false
1...3.include?(2.5)      # => true
```

Also, maybe in the future the following behavior can be possible:

```
r1 = Range.new('a', 'bc', :deglex)
r1.include?('abc') # => false
r1.to_a          # => ['a', 'b', ..., 'az', 'ba', 'bb', 'bc']
r2 = Range.new('a', 'bc', :lex)
r2.include?('abc') # => true
r2.to_a          # => Error
```

and this:

```
((0.5)..(5.5)).to_a(Integer) # => 1, 2, 3, 4, 5.each(Integer) do |i| puts i end
```

(i imagine this would require additions to the (({Integer})) class, so that it know more about "real" ranges).

=end

History

#1 - 12/12/2012 01:44 AM - drbrain (Eric Hodel)

- Target version set to 3.0

=begin
How would (({(3..1).to_a})) be implemented? The opposite uses #succ today.

Why should (({1...1})) equal (({3...3}))? I don't understand at all. This would break existing code.

In (({1...3.include?(2.5)})) is the change to Ruby syntax also part of the proposal, or a typo? (Currently NoMethodError is raised due to low precedence of (({...}). Is there something wrong with Range#cover?

What is a deglex? What is a lex?
=end

#2 - 12/12/2012 02:37 AM - alexeymuranov (Alexey Muranov)

drbrain (Eric Hodel) wrote:

> =begin
> How would (({(3..1).to_a})) be implemented? The opposite uses #succ today.

I propose the range to store a lazy ordered set, which is normally an interval in some bigger ordered set. I have not specified an implementation, but for example it can store the bounds, unless the set is empty, and some kind of identifiers for the ambient set and for the order used. Probably this will have little in common with the current implementation.

As 3 and 1 are integers and 3 > 1, i propose to interpret the literal 3..1 as an interval in the set of integers, with the lower bound 3, the upper bound 1, and the order reverse to the usual order (3 < 2 < 1), designated by a symbol :rev for example. I would propose to delegate the responsibility of converting 3..1 to an array to the Integer class (allowing also the explicit syntax (3..1).to_a(Integer)). The Integer class should know how to iterate over a range when the bounds and the order (direction) are given.

Actually, what did you mean by "opposite"?

> Why should (({1...1})) equal (({3...3}))? I don't understand at all. This would break existing code.

Because, for example, currently

(1...1).to_a # => .to_a # => .to_a # => []

More precisely, because they are the same as ordered sets: all three are the empty set.

I understand that this proposal changes the existing behavior and so can break existing code.
However, i do not imagine easily an example of code where someone relies on the fact that 1...1 is not the same as 3...3.

> In (({1...3.include?(2.5)})) is the change to Ruby syntax also part of the proposal, or a typo? (Currently NoMethodError is raised due to low precedence of (({...})). Is there something wrong with Range#cover?

Yes, there was a typo, i meant (1...3).include?(2.5), and it is not a change, it already works like this. It was just for an example, that this behavior should stay.

I did not talk about #cover? in this proposal, but in my opinion having both is redundant, i would prefer having just #include?, possibly with some options.

> What is a deglex? What is a lex?

I meant here different orders on the set of words, for example :lex for lexicographic, :deglex for graded lexicographic http://en.wikipedia.org/wiki/Monomial_order#Graded_lexicographic_order (there it is called "grlex").

Update: sorry, i am not sure i used the "deglex" term appropriately, i would need to check. I meant the order where the words are first compared by length and then lexicographically, this is the order that is currently used when converting 'a'..'ac' to an array, but curiously it does not work for ('b'..'ac').to_a (a bug?).

#3 - 12/12/2012 04:23 AM - drbrain (Eric Hodel)

alexeymuranov (Alexey Muranov) wrote:

> drbrain (Eric Hodel) wrote:
>
> > =begin
> > How would (({(3..1).to_a})) be implemented? The opposite uses #succ today.
>
> I propose the range to store a lazy ordered set,

Range is lazy today.

> which is normally an interval in some bigger ordered set. I have not specified an implementation, but for example it can store the bounds, unless the set is empty, and some kind of identifiers for the ambient set and for the order used. Probably this will have little in common with the current implementation.

If no bounds are stored for an empty set and you can't tell the difference between (1…1) and (3…3) what should Range#inspect do?

What is an ambient set?

Why is such a big change necessary?

> As 3 and 1 are integers and 3 > 1, i propose to interpret the literal 3..1 as an interval in the set of integers, with the lower bound 3, the upper bound 1, and the order reverse to the usual order (3 < 2 < 1), designated by a symbol :rev for example. I would propose to delegate the

> responsibility of converting 3..1 to an array to the Integer class (allowing also the explicit syntax (3..1).to_a(Integer)). The Integer class should know how to iterate over a range when the bounds and the order (direction) are given.

What is ":rev"? Revision? Reverse? Revise?

I think matz will object to such ambiguous short names.

> Actually, what did you mean by "opposite"?

(1…3).to_a uses Integer#succ to build the Array.

> > Why should (({1...1})) equal (({3...3}))? I don't understand at all. This would break existing code.

> Because, for example, currently
>
> (1...1).to_a # => .to_a # => .to_a # => []
>
> More precisely, because they are the same as ordered sets: all three are the empty set.

Range and Set are different concepts (and classes). I don't see why you would define equality on Set based on some other class (Array, here). This is not good OO design.

> I understand that this proposal changes the existing behavior and so can break existing code.
> However, i do not imagine easily an example of code where someone relies on the fact that 1...1 is not the same as 3...3.

I do not either, but these things happen.

> > In (({1...3.include?(2.5)})) is the change to Ruby syntax also part of the proposal, or a typo? (Currently NoMethodError is raised due to low precedence of (({...}). Is there something wrong with Range#cover?

> Yes, there was a typo, i meant (1...3).include?(2.5), and it is not a change, it already works like this. It was just for an example, that this behavior should stay.
>
> I did not talk about #cover? in this proposal, but in my opinion having both is redundant, i would prefer having just #include?, possibly with some options.

How does this fit with existing implementations of #include?

> > What is a deglex? What is a lex?

> I meant here different orders on the set of words, for example :lex for lexicographic, :deglex for graded lexicographic http://en.wikipedia.org/wiki/Monomial_order#Graded_lexicographic_order (there it is called "grlex").
>
> Update: sorry, i am not sure i used the "deglex" term appropriately, i would need to check. I meant the order where the words are first compared by length and then lexicographically, this is the order that is currently used when converting 'a'..'ac' to an array, but curiously it does not work for ('b'..'ac').to_a (a bug?).

Again, I don't think matz would approve of these short names. They are unintuitive.

#4 - 12/12/2012 05:01 AM - alexeymuranov (Alexey Muranov)

drbrain (Eric Hodel) wrote:

> alexeymuranov (Alexey Muranov) wrote:
>
> > I propose the range to store a lazy ordered set,
>
> Range is lazy today.

But it is not a set, it seems to be just a pair of bounds, like a 2-element array. I propose to change the semantics, and hence behavior in applications like Array#slice.

> > which is normally an interval in some bigger ordered set. I have not specified an implementation, but for example it can store the bounds, unless the set is empty, and some kind of identifiers for the ambient set and for the order used. Probably this will have little in common with

> the current implementation.

> If no bounds are stored for an empty set and you can't tell the difference between (1…1) and (3…3) what should Range#inspect do?

Just print that it is the empty range. I suggested a constant name Range::EMPTY_SET

> What is an ambient set?

I meant that to store 1..3 or 3..1 or 1...(3.5), it is enough to store the bounds, assume that the the interval is a subset of the reals (so that floats, rationals, and others could be tested for inclusion, for 'a'..'c' the natural ambient set would be the set of words), and remember the order (the usual or the reverse).

> Why is such a big change necessary?

This would make the Range class quite intelligent. I do not know if it is necessary.

> As 3 and 1 are integers and 3 > 1, i propose to interpret the literal 3..1 as an interval in the set of integers, with the lower bound 3, the upper bound 1, and the order reverse to the usual order (3 < 2 < 1), designated by a symbol :rev for example. I would propose to delegate the responsibility of converting 3..1 to an array to the Integer class (allowing also the explicit syntax (3..1).to_a(Integer)). The Integer class should know how to iterate over a range when the bounds and the order (direction) are given.

> What is ":rev"? Revision? Reverse? Revise?

I meant reverse order, it was just an example.

> Range and Set are different concepts (and classes). I don't see why you would define equality on Set based on some other class (Array, here). This is not good OO design.

Set class only works to store finite sets, while Range class can be used to represent infinite sets, like the intervals [0, ∞), or 0, 1.

For the equality of 1...1 and 3...3, i meant that they both model the same object: the empty set, this was the actual reason.

> I did not talk about #cover? in this proposal, but in my opinion having both is redundant, i would prefer having just #include?, possibly with some options.

> How does this fit with existing implementations of #include?

I do not know. I was thinking about an abstraction of Range class.

#5 - 12/12/2012 06:33 AM - phluid61 (Matthew Kerwin)

alexeymuranov (Alexey Muranov) wrote:

> But it is not a set, it seems to be just a pair of bounds, like a 2-element array. I propose to change the semantics, and hence behavior in applications like Array#slice.

You say that "[range] is not a set" but you propose making it one? Why not instead create a class (e.g. LazyBoundedSet, which may or may not be backed by a Range at your discretion) which explicitly *is* a set, and doesn't require breaking existing functionality, let alone breaking Rubyists' current understanding?

> For the equality of 1...1 and 3...3, i meant that they both model the same object: the empty set, this was the actual reason.

Point in case: they model the same (empty) set, but they are different ranges.

#6 - 12/12/2012 06:37 AM - drbrain (Eric Hodel)

alexeymuranov (Alexey Muranov) wrote:

> But it is not a set, it seems to be just a pair of bounds, like a 2-element array. I propose to change the semantics, and hence behavior in applications like Array#slice.

How is this beneficial to Rubyists?

> If no bounds are stored for an empty set and you can't tell the difference between (1…1) and (3…3) what should Range#inspect do?

Just print that it is the empty range. I suggested a constant name Range::EMPTY_SET

These don't seem useful:

```
p (1…1) #=> Range::EMPTY_SET
p (3…3) #=> Range::EMPTY_SET
p ('a'…'b') #=> Range::EMPTY_SET
```

I can't think of any other built-in class that behaves this way.

Discarding the beginning and ending of the Range is very distasteful. I don't approve.

> What is an ambient set?

I meant that to store 1..3 or 3..1 or 1...(3.5), it is enough to store the bounds, assume that the the interval is a subset of the reals (so that floats, rationals, and others could be tested for inclusion, for 'a'..'c' the natural ambient set would be the set of words), and remember the order (the usual or the reverse).

Range already stores only the bounds. For Numeric bounds, Range#include? and Range#cover? already assume the interval is a subset of the reals (with the exception that the lower bound must be the lower number).

Can you show value for allowing the lower bound to be the higher number?

> Why is such a big change necessary?

This would make the Range class quite intelligent. I do not know if it is necessary.

Since you do not know if it is necessary I think this should be rejected.

> Range and Set are different concepts (and classes). I don't see why you would define equality on Set based on some other class (Array, here). This is not good OO design.

Set class only works to store finite sets, while Range class can be used to represent infinite sets, like the intervals [0, ∞), or 0, 1.

As I stated above, no change to ruby is necessary.

```
p (1...Float::INFINITY).include? Float::INFINITY #=> false
p (0..1).include? 0.5 #=> true
```

For the equality of 1...1 and 3...3, i meant that they both model the same object: the empty set, this was the actual reason.

Purposeful information loss is a bad policy, I think this proposal should be rejected.

> I did not talk about #cover? in this proposal, but in my opinion having both is redundant, i would prefer having just #include?, possibly with some options.

> How does this fit with existing implementations of #include?

I do not know. I was thinking about an abstraction of Range class.

Since you do not know, I think this proposal should be rejected.