Ruby master - Feature #7738

Deprecate Set#+ as an alias of Set#|, use it for symmetric difference. Introduce Hash#| for Hash#reverse_merge in Rails.

01/24/2013 08:36 PM - alexeymuranov (Alexey Muranov)

Status: Rejected
Priority: Normal
Assignee: knu (Akinori MUSHA)
Target version: 3.0

Description
=begin
I am almost sure this will be rejected, but i want to try anyway. My goal is to propose a more efficient and consistent use of binary operator symbols for some classes, in this case for ({{Set}}) and ({{Hash}}).

I propose to deprecate ({{Set#+}}) as an alias of ({{Set#|}}), and use ({{Set#+}}) later for the (symmetric difference) of sets. I think that operator symbols like ({{+}}) and ({{|}}) are too precious to alias one another.

Currently ({{Set#+}}) is probably the only use of ({{#+}}) for an operation which is not (injective) in each of the arguments: for sets ({{a}}), ({{b}}), ({{c}}), the equality

a + b == a + c

does not currently imply

b == c

The natural binary operation on sets that is injective in each argument is the (symmetric difference), it corresponds to the bitwise XOR. I have also noticed that the "((+))" for sets is used in "Lectures on Ergodic Theory" by P. Halmos to denote the symmetric difference.

I also suggest to define ({{Hash#|}}) as Hash#reverse_merge in Rails, in my opinion this would correspond nicely to ({{Set#|}}).
=end

History
#1 - 01/24/2013 09:20 PM - trans (Thomas Sawyer)
It would help if you gave some examples of what "symmetric difference" actually meant.

Put the last "I also suggest" in another issue. Ok?

#2 - 01/24/2013 09:53 PM - alexeymuranov (Alexey Muranov)
Sorry for not being clear. The symmetric difference of A and B is the set of all those elements which belong to exactly one of the two sets A or B. It is like the bitwise "exclusive or".

This would mean:
Set[a, b] + Set[b, c] # => #

Ok, i'll open a separate issue for Hash#. (Update: #7739)

#3 - 01/24/2013 11:42 PM - trans (Thomas Sawyer)
I see. Thanks.

Well, i hate to mention b/c i don't like it, but Ruby uses ^ for xor. So wouldn't that be the more appropriate operator for this? Using + for this seems strange.

#4 - 01/25/2013 07:08 PM - alexeymuranov (Alexey Muranov)
I agree that having the same symbol for the symmetric difference as for XOR would be nice, given that there are not so many different ASCII symbols. However, the main part of my request is to not use + for the union of sets. This use is inconsistent with the set theory notation (or i have not yet met + for the union), and is not very consistent in my opinion with other uses of + in Ruby. Also it looks redundant to alias one operator symbol with another: a half of rubists will be using one, and the other half will be using another.

If Set#+ is used, in my opinion it should be the symmetric difference. (The problem with this is that i cannot come up with a good use case for the
symmetric difference operation. I was thinking about flipping a subset of binary options, but do not see how it can be useful.)

About the inconsistency with the Fixnum#^ (i don't like #^ for XOR either, i proposed #<> some time but it was rejected), i can say that this is in a sense a secondary operation on Fixnum, which violates to some extent the Single Responsibility Principle. In my opinion, for bitwise operations it would be natural to have some BitSet class, and use, for example, BitSet#+ for XOR. There seem to be no special reason for Fixnum to be stored internally in the binary form, there existed ternary computers once upon a time: http://en.wikipedia.org/wiki/Setun

#5 - 02/20/2013 08:11 PM - alexeymuranov (Alexey Muranov)
In fact, sets (or rather subsets of a given set) form a Boolean ring (http://en.wikipedia.org/wiki/Boolean_ring) under the following two operations:

- the product is the set intersection and
- the sum is the set symmetric difference. However, according to the Wikipedia article, it is accepted to use + for the union (which is not the ring addition) in this case, contrary to what i thought.

So maybe it is alright to have it this way in Ruby, except that there are two symbols for the same operation.

#6 - 02/22/2013 09:05 AM - ko1 (Koichi Sasada)
- Category set to lib
- Assignee set to knu (Akinori MUSHA)

#7 - 08/13/2013 06:05 PM - alexeymuranov (Alexey Muranov)
I can hardly come up with a meaningful example that would use the symmetric difference of sets. So i propose to consider instead redefining Set#+ as the disjoint union of sets: it would return the union of sets if the sets are disjoint, and raise an error otherwise.

#8 - 08/13/2013 08:32 PM - knu (Akinori MUSHA)
=begin
I think it is unusual in ruby for an operator to raise an exception depending on the combination of the values of operands.

It is like (((x + 1))) raising an exception when (((x))) is (((3))) while it successfully returns (((5))) when (((x))) is (((4))).
I can't imagine how it could be used usefully and properly.

Now that we have Set#disjoint?, guarding (((x | y))) with (((if x.disjoint?(y)))) is much more straightforward and handy than having to rescue (((x | y))) in case it raises an exception.
=end

#9 - 08/13/2013 10:32 PM - alexeymuranov (Alexey Muranov)
As i said, i will probably be surprised if this proposal is accepted :). However, i would like to answer:

1. 1 / x returns a result if x = 1, but raises an exception if x = 0.
2. I was proposing to change the behavior of Set#+, and leave Set# as it is.

I am not sure my proposal is very useful, but here are some motivations:

1. No duplicate methods under different names (some are using Set#, others Set#+ for exactly same function).
2. More consistency of meaning of #+ between different classes.

I do not know a very good use case of the new proposed function (raising an exception), but here is what comes to mind: combining two collections of objects where each object is expected to belong to exactly one collection. This would ensure that for two sets a and b, a + b - b will either return a, or raise an exception.

But i admit these are not very important, feel free to close.

#10 - 08/14/2013 03:05 PM - knu (Akinori MUSHA)
- Status changed from Open to Rejected

Case frozen until someone comes up with real use cases and a proposal as the solution to them.