Ruby master - Feature #7511
short-circuiting logical implication operator
12/05/2012 06:40 AM - rits (First Last)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Assigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td>matz (Yukihiro Matsumoto)</td>
</tr>
<tr>
<td>Target version:</td>
<td>3.0</td>
</tr>
</tbody>
</table>

Description
I find I need logical implication (will use => here) in boolean expressions fairly often
and even though a => b is equivalent to !a || b, this substitute is more difficult to read and make sense of in long expressions

History

#1 - 12/05/2012 10:32 AM - naruse (Yui NARUSE)
How about a ? b : true

#2 - 12/05/2012 05:10 PM - rits (First Last)
naruse (Yui NARUSE) wrote:
How about a ? b : true

One can get along without the implication operator, but for better readability / quicker mental parsing, it would be better than the existing alternatives, including ternary.

#3 - 12/05/2012 09:45 PM - alexeymuranov (Alexey Muranov)
I have no opinion about whether it is needed, but a usual mathematical symbol would be x -> y. So it could be --> for boolean and -> for bitwise. But this may be confusing because of other existing or possible uses of ->, <= etc.

Maybe the comparison operator <= could be defined for bools:
false <= true  # => true
?

Then to write that a implies b, one could do:
!!a <= !!b

#4 - 12/05/2012 09:48 PM - charliesome (Charlie Somerville)
alexeymuranov: This syntax is ambiguous. For example: a -> b {}

I don't think this would be a worthwhile syntax addition. It isn't something I can imagine would see much use.

#5 - 12/06/2012 12:26 AM - rits (First Last)
charliesome (Charlie Somerville) wrote:
It isn't something I can imagine would see much use.

implication is very common, the presence or validity of something requiring some condition to hold.

if a && b && c && (d => e) && (f => g)
...
end

is a lot more clear than

if a && b && c && (d ? e : true) && (f || g)
...
end

especially if some of a, b, c, d, e, f, g are non trivial expressions
Please show a more concrete example.

--
Yusuke Endoh mame@tsg.ne.jp

#7 - 11/25/2013 05:11 AM - rits (First Last)
mame (Yusuke Endoh) wrote:

logical implication is a core operation of boolean algebra, do you also need concrete examples for || ?

ok if you insist, lets say your code needs to perform some optional work but successfully, i.e. the existence of the method must imply it's successful execution.

required_work && respond_to?(optional_work) => send(optional_work)

#8 - 12/18/2013 07:16 PM - duerst (Martin Dürst)

rits (First Last) wrote:

Logical implication is actually NOT a core operation of Boolean algebra. There are two wikipedia articles about Boolean algebra, http://en.wikipedia.org/wiki/Boolean_algebra and http://en.wikipedia.org/wiki/Boolean_algebra_%28structure%29. In both, and/or/not are the core operations. This corresponds well with the fact that these three operations are available in virtually any programming language.

On the other hand, implication doesn't appear in the later article, and only appears in the section on propositional logic in the former article. This corresponds well with the fact that there are very few programming languages that have an implication operator. Actually, I have to admit that I don't know a single programming language with such an operator. Do you?

    ok if you insist, lets say your code needs to perform some optional work but successfully, i.e. the existence of the method must imply it's successful execution.

    required_work && respond_to?(optional_work) => send(optional_work)

How frequent are such examples? I'd guess they are very infrequent. Also, implication used as an operation isn't that easy to understand because common sense doesn't easily grok the fact that false can imply anything.

And rewriting the above as

    required_work and (not respond_to?(optional_work) or send(optional_work))

isn't actually too difficult to understand: Either the object doesn't respond to 'optional_work', or the 'optional_work' method returns a trueish value. This gives the whole picture.

On the other hand, reading "respond_to?(optional_work) => send(optional_work)" directly as "the existence of the method must imply it's successful execution" leaves the reader with the job of filling in "but if the method doesn't exist, we are still good".

My guess is that these kinds of difficulties also contributed to the fact that virtually no programming language has such an operator.

#9 - 12/03/2014 03:32 AM - why-capslock-though (Alexander Moore-Niemi)
to be precise, implies is a derived operator, not a core one.

racket has this operator, as does eiffel, which supplies it to, afaik, help make design by contract easier (targeting assertions as a core responsibility of the language)

but given ruby has not committed to that, this would probably make more sense to recommend to contracts.ruby library :)

in the meantime consider just refining TrueClass and FalseClass to receive a custom defined xnor if you want to be able to readably infix your predicates, ie. (quick and dirty monkeypatch, use refinements instead!!)

2.1.3 :053 > false.class
then you could do p.xnor(q) which i think is more readable than using the other recommended method

#10 - 03/27/2018 08:32 PM - jaredbeck (Jared Beck)
The spirit of ruby is to make programmers happy. So, if logical implication would make programmers happy, we must consider it. It doesn't matter if it's infrequently used. For example, Array#zip is rarely used, but it makes me very happy when I use it.

To me, the primary concern is that ruby has already assigned meaning to most ASCII characters. So, perhaps implication should simply be implemented as a named method, rather than an operator. For example,

class TrueClass
  def implies?(other)
    !self || other
  end
end
true.implies?(false) #=> false
true.implies?(true) #=> true

This implementation will not hurt ruby, and will make programmers happy. What do you think?

#11 - 03/28/2018 02:31 AM - nobu (Nobuyoshi Nakada)
It doesn't short-circuit, the argument is evaluated always regardless the receiver.