Ruby - Bug #7331

Set the precedence of unary `-` equal to the precedence `-`, same for `+`

11/12/2012 02:20 AM - alexeymuranov (Alexey Muranov)

Status:	Rejected		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	1.9.3	**Backport:**	

Description

=begin
I will not be surprised if this proposal is rejected, because the compatibility would be a mess, but i feel i need to start a discussion. This continues the discussion in #7328.

To the best of my knowledge, in mathematics the unary minus has the same precedence as the binary one. However, in the expression

- a * b

Ruby computes first (({-a})), and then the product. This means that if i want (for whatever reason) to compute the expression in the natural order, i have to put the parentheses:

- (a * b)

which looks very strange to me. I would consider this a bug.
=end

History

#1 - 11/12/2012 09:52 AM - matz (Yukihiro Matsumoto)

- Status changed from Open to Rejected

unary minus has been higher precedence in the long history of programming languages.
probably it's related to negative number literals.

Matz.

#2 - 11/20/2012 10:40 AM - nobu (Nobuyoshi Nakada)

Possibly, emitting a warning against the visual mismatch?
https://github.com/nobu/ruby/compare/warn_unary_space

#3 - 11/20/2012 06:43 PM - alexeymuranov (Alexey Muranov)

@nobu (Nobuyoshi Nakada) : nice idea.

By the way, as this proposal is rejected, i do not personally oppose anymore to giving the unary - and + the highest precedence (#7328). I've read that previously the main reason of giving the binary power operator ** higher precedence than the unary - was that "people with mathematical background wanted so". If this is the case, i might be able to convince them to change their mind. Probably many of them are just unaware how - a * b is parsed in Ruby, and that this behavior is not going to change.

#4 - 11/21/2012 10:04 PM - stomar (Marcus Stollsteimer)

=begin
This has no relevance to the issue, but I think your point of view is wrong. :-)
You should do it the "mathematical" way not only for people with mathematical background but even more so for "normal" people.

My common sense as a teacher leads me to believe that a "normal" person would tend to just take a formula from a textbook, e. g. to calculate compound interest or whatever, without ever thinking about subleties like operator precedence.
A mathematician is much more likely to make sure what the exact precedence rules in the used programming language are, or to use parentheses just to make sure, and he would also have less difficulty in adapting to a different set of rules.

I observe a related phenomenon with students that happily calculate the volume of a sphere with (({4 / 3 * Math::PI * radius**3})), not thinking about subleties like integer division.

So, stay as close to the mathematical rules as possible. (And not primarily for the mathematician's sake!)
=end

#5 - 11/21/2012 10:51 PM - alexeymuranov (Alexey Muranov)

@stomar (Marcus Stollsteimer), do i understand correctly that you insist on reopening this feature request?

I wouldn't mind it too, but apparently some people prefer to be able to write -2 * -2 instead of the proper way (-2) * (-2).

#6 - 11/22/2012 01:20 AM - stomar (Marcus Stollsteimer)

No, absolutely not! Such a change would certainly have a bigger impact than I could imagine.

I just wanted to make a point for not deviating further from mathematical rules by changing -2**2 to result in 4.

#7 - 11/22/2012 02:29 AM - alexeymuranov (Alexey Muranov)

@stomar (Marcus Stollsteimer), as mathematical parsing rules are not and apparently will not be observed in Ruby, i do not understand very well your point of view.

Either a programmer/user knows the operation precedence is Ruby, or not. Unless you want to encourage use of Ruby without understanding how an expression the user writes will be parsed, let us assume that the user is required to know that when he/she writes -2 * -2, this actually means (-2) * (-2) for Ruby, and will be executed accordingly.

Do you think that remembering in the same time that -2 ** -2 means -(2 ** (-2)) and not (-2) ** (-2) will be easy? Does this parsing
-2 * -2 -> (-2) * (-2)
-2 ** -2 -> -(2 ** (-2))
make more mathematical sense than this one
-2 * -2 -> (-2) * (-2)
-2 ** -2 -> (-2) ** (-2)
?

To me, it would be easier to remember that Ruby alway implicitly adds parentheses around an expression of the form -a, than to remember that the unary minus has higher precedence than the binary one, and that for some reason the order of precedence of unary - and binary * is inverted.

I do not quite understand in what metric the current Ruby parsing is closer to the subset of "mathematically natural ones" than would be the one proposed in #7328, given that the present proposal stays rejected. :)

#8 - 11/22/2012 07:15 AM - stomar (Marcus Stollsteimer)

Generally, adding "wrong" syntax just for consistency is IMO no option.

But, looking over the thread again, I don't see your problem in the first place:

- 2 * 3 means (-1) * 2 * 3 = (- 2) * 3 = - (2 * 3), which all results in -6, so what's wrong with that?
  Furthermore, -2 * -2 is mathematically invalid syntax, you would have to write -2 * (-2),
  but here also there is no ambiguity involved, - (2 * (-2)) = 4 = (-2) * (-2).

The only thing that seems worth mentioning is that Ruby allows the shortcut 2 * - 3 for 2 * (-3).

#9 - 11/22/2012 07:50 AM - alexeymuranov (Alexey Muranov)

stomar (Marcus Stollsteimer) wrote:

> Generally, adding "wrong" syntax just for consistency is IMO no option.

Well, when the wrong syntax is already present (- a * b is (- a) * b) i do not think #7328 would make the syntax as a whole "more wrong" (IMO it would even make it right in some sense).

> But, looking over the thread again, I don't see your problem in the first place:
>
> - 2 * 3 means (-1) * 2 * 3 = (- 2) * 3 = - (2 * 3), which all results in -6, so what's wrong with that?

The question is about parsing, not about the arithmetic. There is (i believe) no Ruby specification saying that (-a) * b must produce the same result and side-effects as -(a * b). If it was a part of Ruby specification, i wouldn't have reported this issue.

> Furthermore, -2 * -2 is mathematically invalid syntax, you would have to write -2 * (-2),
> but here also there is no ambiguity involved, - (2 * (-2)) = 4 = (-2) * (-2).

In Ruby, -a * -b is allowed, and means (-a) * (-b), not -(a * (-b)). Hence an ambiguity for an unsuspecting user.

> The only thing that seems worth mentioning is that Ruby allows the shortcut 2 * - 3 for 2 * (-3).

Yes, this is worth mentioning :).

If there are other arguments, i think it would be better to continues the discussion in the thread for #7328.

#10 - 11/22/2012 04:17 PM - stomar (Marcus Stollsteimer)

> Well, when the wrong syntax is already present (- a * b is (- a) * b)
> i do not think #7328 would make the syntax as a whole "more wrong"
> (IMO it would even make it right in some sense).

1. Changing the precedence of **' and unary -' would affect the results and is in discrepancy with mathematical rules.
2. Changing the precedence of *' and unary -' would not affect the results and would not have any benefits.
3. The current parsing is **NOT** wrong, since -a = (-1) * a, so - a * b = (-1) * a * b, and you have essentially the same precedence, so the "natural" way is to go from left to right.

So most essentially, there is no argument for your statement that - (a * b) is a more "natural" order of evaluation than (-a) * b, which was the starting point of your "bug" report.

#11 - 11/22/2012 06:08 PM - alexeymuranov (Alexey Muranov)

stomar (Marcus Stollsteimer) wrote:

> 1. Changing the precedence of **' and unary -' would affect the results and is in discrepancy with mathematical rules.

There are already discrepancies with mathematical rules, like allowing 2 * -3.

> 1. Changing the precedence of *' and unary -' would not affect the results and would not have any benefits.

My request was about changing parsing, i agree that for integers it would not affect the result. The benefit would be to eliminate a discrepancy with mathematical parsing rules and to not mislead a new user who is likely to believe that - a * b *means* - (a * b). In mathematics, (-1) * 2 * 3 = (- 2) * 3 = - (2 * 3) because it is a property of integers, but - 2 * 3 is a shorthand syntax for - (2 * 3).

> 1. The current parsing is **NOT** wrong, since -a = (-1) * a, so - a * b = (-1) * a * b, and you have essentially the same precedence, so the "natural" way is to go from left to right.

You are talking about the result (for integers), not about parsing. How is the current parsing not mathematically wrong if - a * b is parsed as (- a) * b? (I repeat, i am talking about parsing.)

#12 - 11/22/2012 09:15 PM - alexeymuranov (Alexey Muranov)

I would propose the following experiment to evaluate which parsing would be less surprising to a new Ruby user :).

Take 10 new Ruby users who do not know about this discussion and ask them to insert parentheses into the following arithmetic expressions in the way they believe Ruby does it, without experimenting with the interpreter:
- 2 * - 2
- 2 ** - 2
- 2 * - 2 ** - 2
- 2 ** - 2 * - 2

#13 - 11/22/2012 09:33 PM - trans (Thomas Sawyer)

I have come to the opposite opinion. I do not think Ruby must reflect mathematical notation exactly. If you think about it, mathematical notations can be somewhat loose, a bit more like natural language rather then a rigorous syntax. There are plenty of examples.

Personally I'd even rather see #** move down a notch so that unary operators always take precedence. I find such a simplification makes it easier to read the code --official math parsing be damned.

#14 - 11/22/2012 11:30 PM - stomar (Marcus Stollsteimer)

@alexeymuranov: I understood that you were talking about parsing.

I refute your primary supposition. In the mathematical expression -2·3, the unary `-' can be interpreted both as a multiplication, (-1)·2·3, and as subtraction, 0 - 2·3. Both interpretations are equally valid. Which one you choose has no effect on the result.

I think there is no sense in discussing or speculating further, as long as you do not provide a citation that confirms your mere **assumption** that the "mathematical parsing rule" (I doubt there is one) is -(a * b). And which one feels more "natural" to you (or to any Ruby user) is no argument at all.

And BTW, I do not think anyone of us is really qualified enough in this field to decide on this issue.

#15 - 11/23/2012 12:32 AM - alexeymuranov (Alexey Muranov)

stomar (Marcus Stollsteimer) wrote:

> I refute your primary supposition. In the mathematical expression -2·3, the unary `-' can be interpreted both as a multiplication, (-1)·2·3, and as subtraction, 0 - 2·3. Both interpretations are equally valid. Which one you choose has no effect on the result.

It has to be interpreted one way or the other. This is what operation precedence is for. This is, for example, how Model Theory deals with formulas without all explicit parentheses. This is what parsing is in the sense of formal grammar. I believe that in mathematical language -2·3 is parsed (-(2·3)), as if 0 was taken out of (0-(2·3)).

@trans, this is why mathematical notation is not loose, because there is operation precedence taught in elementary school.

> I think there is no sense in discussing or speculating further, as long as you do not provide a citation that confirms your mere **assumption** that the "mathematical parsing rule" (I doubt there is one) is -(a * b). And which one feels more "natural" to you (or to any Ruby user) is no argument at all.

I will think about a reference, it seems that Google doesn't know :).

> And BTW, I do not think anyone of us is really qualified enough in this field to decide on this issue.

What field are you talking about? I think i am qualified in mathematics, but Ruby is not mathematics, and it is not up to me to decide. I do not mind learning something new anyway.

#16 - 11/23/2012 02:09 AM - stomar (Marcus Stollsteimer)

> I believe that in mathematical language -2·3 is parsed (-(2·3)),
> as if 0 was taken out of (0-(2·3)).

Exactly, you *believe*, I do believe differently (namely that there is no general agreement on the precedence because it has no consequences in mathematics).

We're both believers, but do not know...

#17 - 03/02/2013 03:29 AM - alexeymuranov (Alexey Muranov)

After some thinking, i want to add my last word on this :). I have not seen any "official" rule on parsing "- 2 * 3" (unless i was taught this in elementary school but not sure). However, i'll try to explain my reasons for believing that "- (2 * 3)" should be the "right" parsing.

In my opinion, it is not a proper way to teach a second-grade child to evaluate "- 2 * 3" by saying:
Hey! (or Yo!) Multiplication is associative and distributes over addition and subtraction, whether you compute "- 2 * 3" as "- (2 * 3)", as "(-2) * 3", or as "((-1)*2)*3", it will be all the same!

Because if i do, they may remain wondering what "- 2 * 3" *means* and *how* to *evaluate* it. So, i would simply say that the minus in front stands for "0 -", and the evaluation rules are the usual ones: first multiplications and divisions from left to right, then additions and subtractions from left to right. Which gives

- 2 * 3 = 0 - 2 * 3 = 0 - (2 * 3) = - (2 * 3).

Returning back to parsing expressions in programming languages, the reason to parse "2 * - 3" as "2 * (- 3)" is not the operator precedence, but the fact that this expression is "invalid" as written, and fortunately, as "-" can be a unary prefix but "*" cannot be a unary suffix, there exists a unique way to make this expression valid: add parentheses around "- 3".

Just to be sure that my opinion on parsing arithmetic unary minus is not a complete non-sense, i asked two colleagues across the hall about their opinions. First they said that in "- 2 * 3" parentheses do not matter, and i can put them any way i like. So i started doubting if my opinion is not a complete nonsense. However then i changed my question into asking how to teach a child to compute and how to parse an expression in a programming language, and explained my point of view, and they both agreed. (But maybe Marcus would be able to convince them otherwise.)