Ruby - Feature #16260

Symbol#to_proc behaves like lambda, but doesn't aknowledge it

10/18/2019 09:21 AM - zverok (Victor Shepelev)

Status:	Closed
Priority:	Normal
Assignee:	nobu (Nobuyoshi Nakada)
Target version:	3.0

Description

Seems that Symbol#to_proc returns Proc that has lambda semantics:

```
proc = :+.to_proc
proc.call(1, 2)   # => 3
proc.call([1, 2]) # ArgumentError (wrong number of arguments (given 0, expected 1))
```

But if you ask...

```
proc.lambda? # => false
```

That seems to be an inconsistency, which I'd like to clarify. There are obviously two ways to fix it:

1. Make it respond true to lambda? (and mention the semantics in docs)
2. Make it behave like non-lambda.

The second one seems to produce some useful behavior:

```
# Currently:
[1, 2].zip([3, 4]).map(&:+) # ArgumentError (wrong number of arguments (given 0, expected 1))

# With non-lambda:
class Symbol
  def to_proc
    proc { |o, *a| o.send(self, *a) }
  end
end

[1, 2].zip([3, 4]).map(&:+) # => [4, 6]
```

Probably all of it was discussed when Symbol#to_proc was introduced, but as old NEWS-files doesn't link to tickets/discussions, I can't find the reasoning for current behavior.

Associated revisions

Revision f0b815dc670b61eba1daaa67a8613ac431d32b16 - 02/19/2020 06:46 AM - nobu (Nobuyoshi Nakada)

Proc made by Symbol#to_proc should be a lambda [Bug #16260]

Revision f0b815dc670b61eba1daaa67a8613ac431d32b16 - 02/19/2020 06:46 AM - nobu (Nobuyoshi Nakada)

Proc made by Symbol#to_proc should be a lambda [Bug #16260]

Revision f0b815dc - 02/19/2020 06:46 AM - nobu (Nobuyoshi Nakada)

Proc made by Symbol#to_proc should be a lambda [Bug #16260]

Revision 5cab86f3b0725457be3c50d3cab43b04bea53290 - 02/21/2020 03:30 PM - nobu (Nobuyoshi Nakada)

Proc made by Symbol#to_proc should be a lambda [Bug #16260]

Revision 5cab86f3b0725457be3c50d3cab43b04bea53290 - 02/21/2020 03:30 PM - nobu (Nobuyoshi Nakada)

Proc made by Symbol#to_proc should be a lambda [Bug #16260]

Revision 5cab86f3 - 02/21/2020 03:30 PM - nobu (Nobuyoshi Nakada)

Proc made by Symbol#to_proc should be a lambda [Bug #16260]

Revision 8c5ca318cbe57269f144a4d0822c5283c1fd4e1a - 02/21/2020 03:45 PM - nobu (Nobuyoshi Nakada)

Proc made by Symbol#to_proc should be a lambda [Bug #16260]

With refinements, too.

Revision 8c5ca318cbe57269f144a4d0822c5283c1fd4e1a - 02/21/2020 03:45 PM - nobu (Nobuyoshi Nakada)

Proc made by Symbol#to_proc should be a lambda [Bug #16260]

With refinements, too.

Revision 8c5ca318 - 02/21/2020 03:45 PM - nobu (Nobuyoshi Nakada)

Proc made by Symbol#to_proc should be a lambda [Bug #16260]

With refinements, too.

History

#1 - 11/27/2019 06:05 PM - Eregon (Benoit Daloze)

I think we should just return true for lambda?.

Proc has extra confusing behavior, e.g., #16166.

#2 - 11/28/2019 05:22 AM - nobu (Nobuyoshi Nakada)

https://github.com/ruby/ruby/pull/2708

#3 - 12/19/2019 03:15 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Rejected

As a symbol proc cannot know the method to be invoked, so now I think it cannot be lambda.
In the case :+, it looks like a lambda, but it is not always true.

#4 - 12/19/2019 03:26 AM - mame (Yusuke Endoh)

Just curious: How do you want to use the result of lambda?? Even if it returns true, we may pass an arbitrary number of arguments: lambda {|*a| ... }.
I think that lambda? is useless except debugging.

#5 - 12/19/2019 09:23 AM - zverok (Victor Shepelev)

> As a symbol proc cannot know the method to be invoked, so now I think it cannot be lambda.
> In the case :+, it looks like a lambda, but it is **not always true.**

@nobu (Nobuyoshi Nakada), I am not sure I get it right. Can you please show when it is not true?..
For as far as I can understand, there are two distinctions of lambda:

1. Its return returns from lambda itself, not enclosing scope
2. It treats parameters strictly, without implicit unpacking/optionality

Now, :+.to_proc behaves this way:

```
PLUS = :+.to_proc
PLUS.call(1, 2)
# => 3
PLUS.call([1, 2])
# ArgumentError (wrong number of arguments (given 0, expected 1))
# Tried to call [1, 2].+(), not 1.+(2), so no unpacking
```

Whilst lambda would behave this way:

```
PLUS_L = lambda { |obj, *rest| obj.send(:+, *rest) }
PLUS_L.call(1, 2)
# => 3
PLUS_L.call([1, 2])
# ArgumentError (wrong number of arguments (given 0, expected 1))

# Explicit return:
lambda { |obj, *rest| return obj.send(:+, *rest) }.call(1, 2)
# => 3
```

....and proc will behave this way:

```
PLUS_P = lambda { |obj, *rest| obj.send(:+, *rest) }
PLUS_P.call(1, 2)
# => 3
PLUS_P.call([1, 2])
# => 3
# Implicit unpacking

# Explicit return:
proc { |obj, *rest| return obj.send(:+, *rest) }.call(1, 2)
# --- returns from the enclosing scope
```

So, :<sym>.to_proc behaves *exactly* like lambda, and *nothing* like proc.

The only thing that differs from the equivalent lambda is...

```
PLUS.parameters # => [[:rest]]
PLUS_L.parameters # => [[:req, :obj], [:rest, :rest]]
```

(which is ideally to be fixed too, as in fact the first parameter is indeed mandatory.)

Can you please show me the case when :<sym>.to_proc does NOT behave like lambda?..

> Just curious: How do you want to use the result of lambda??

@mame (Yusuke Endoh) For explanatory and educational purposes, at least. For example, in this article, I am showing some funny examples, and to explain why this works:

```
[1, 2, 3].zip([4, 4, 4]).map { |a, b| a + b }
```

...and this not:

```
[1, 2, 3].zip([4, 4, 4]).map(&:+)
```

...I'd like to just say "because :+.to_proc is a lambda, as you can see", but what I really need to say is "becuase :+.to_proc doesn't unpacks arguments, behaving like lambda... though it doesn't aknowledge it is"".

So, yep, debugging, explaining, teaching, this kind of things.

#6 - 12/19/2019 09:31 AM - Eregon (Benoit Daloze)

- Status changed from Rejected to Open

I agree with @zverok (Victor Shepelev) here, a method behaves as a lambda, and doesn't unpack arguments (except a few special methods that specifically do that).

@nobu (Nobuyoshi Nakada) I think we should merge your PR. Could you show an example of a Symbol#to_proc Proc that behaves like a proc and not a lambda? I think that's only rare exceptions (due to that method semantic, not due to the generated Proc), and so Symbol#to_proc should acknowledge it's a lambda.

#7 - 12/26/2019 02:35 AM - mame (Yusuke Endoh)

- Tracker changed from Misc to Feature

- Assignee set to nobu (Nobuyoshi Nakada)

- Target version set to 36

At the previous meeting, matz said it should return true. Will do.

#8 - 02/19/2020 07:15 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Closed

Applied in changeset git|f0b815dc670b61eba1daaa67a8613ac431d32b16.

Proc made by Symbol#to_proc should be a lambda [Bug #16260]

#9 - 09/29/2020 03:37 AM - hsbt (Hiroshi SHIBATA)

- Target version changed from 36 to 3.0