Ruby master - Bug #17105

A single `return` can return to two different places in a proc inside a lambda inside a method

08/06/2020 10:54 AM - Eregon (Benoit Daloze)

Status: Open
Priority: Normal
Assignee:
Target version:
ruby -v:
Backport: 2.5: UNKNOWN, 2.6: UNKNOWN, 2.7: UNKNOWN

Description

A single `return` in the source code might return to 2 different lexical places. That seems wrong to me, as AFAIK all other control flow language constructs always jump to a single place.

```ruby
def m(call_proc)
  r = -> {
    # This single return in the source might exit the lambda or the method!
    proc = Proc.new { return :return }

    if call_proc
      proc.call
    else
      proc
    end
  }.

  if call_proc
    [:after_in_method, r]
  else
    r.call
  end
end
```

```
p m(true)  # => [:after_in_method, :return]
p m(false) # :return
```

We're trying to figure out the semantics of `return` inside a proc in https://github.com/oracle/truffleruby/issues/1488#issuecomment-669185675 and this behavior doesn't seem to make much sense.

headius (Charles Nutter) also seems to agree:

I would consider that behavior to be incorrect; once the proc has escaped from the lambda, its return target is no longer valid. It should not return to a different place.

https://github.com/jruby/jruby/issues/6350#issuecomment-669603740

So:

- is this behavior intentional? or is it a bug?
- what are actually the semantics of return inside a proc?

The semantics seem incredibly complicated to a point developers have no idea where return actually goes. Also it must get even more complicated if one defines a lambda method as the block in lambda { return } is then non-deterministically a proc or lambda.

History

#1 - 08/06/2020 11:00 AM - Eregon (Benoit Daloze)
I should also note some of these semantics might significantly harm the performance of Ruby.
CRuby seems to walk the stack on every return.
On others VMs there need to be some extra logic to find if the frame to return to is still on the stack.
It's already quite complicated but then if return can go to two places, it becomes a huge mess.

#2 - 08/06/2020 02:34 PM - Hanmac (Hans Mackowiak)
i think this is by design:
https://www.rubyguides.com/2016/02/ruby-procs-and-lambdas/
A lambda will return normally, like a regular method.
But a proc will try to return from the current context.
Procs return from the current method, while lambdas return from the lambda itself.

#3 - 08/06/2020 02:43 PM - chrisseaton (Chris Seaton)
Hans I don't think anyone is debating the basic idea of what return in a proc or lambda does - I think we're talking about the edge-case for a proc in a return in the example above, which isn't explained by the text you have.

#4 - 08/06/2020 07:09 PM - Dan0042 (Daniel DeLorme)
I think the behavior makes sense to some extent, because the proc is within 2 nested contexts. Since the proc is within the lambda context, calling it in the lambda returns from the lambda. And since the proc is also within the method context, calling it in the method returns from the method.
The call_proc branching logic makes this look more complicated than it really is, but if you separate the logic I feel the behavior is rather reasonable.
What do you think should be the behavior of m2 below?

```ruby
def m1
  r = -> {
    proc = Proc.new { return :return }
    proc.call #return from lambda
    :after_in_lambda
  }.call
  [:after_in_method, r]
end

def m2
  r = -> {
    proc = Proc.new { return :return }
  }.call
  r.call #return from method
  :never_reached
end

p m1 #=> [:after_in_method, :return]
p m2 #=> :return
```

#5 - 08/06/2020 11:47 PM - Eregon (Benoit Daloze)
IMHO it should be a LocalJumpError. The Proc should return to the lambda, that's syntactically the closest scope it should return to.
Since it's not possible to return to it (the lambda is no longer on stack), it should be a LocalJumpError.

#6 - 08/09/2020 10:41 AM - shyouhei (Shyouhei Urabe)
+1 to Eregon (Benoit Daloze)'s interpretation. Current behaviour is at least very cryptic.

#7 - 08/12/2020 09:29 PM - headius (Charles Nutter)
Just to be clear I am +1 on single return target, as described here: https://github.com/jruby/jruby/issues/6350#issuecomment-669603740
In addition to the confusing (and possibly inefficient) behavior that results from having two possible return targets, there's also a bug potential here if someone "accidentally" allows a proc containing a return to escape from its lambda container. Rather than returning from the lambda as it should have done, it will now return from the next "returnable" scope, and likely interrupt execution in an unexpected way.
I would challenge anyone to explain why the current behavior should exist, since I can't think of a single valid use case. If there's no use case for a confusing "feature", we should remove it.

#8 - 08/31/2020 09:00 AM - matz (Yukihiro Matsumoto)
It is intentional since 1.6.0. But I am OK with making m2 raise LocalJumpError. Ask ko1 (Koichi Sasada) about migration.

Matz.