Ruby master - Bug #17007

SystemStackError when using super inside Module included and lexically inside refinement

07/03/2020 12:32 PM - Eregon (Benoit Daloze)

Status:	Assigned		
Priority:	Normal		
Assignee:	shugo (Shugo Maeda)		
Target version:			
ruby -v:	ruby 2.7.1p83 (2020-03-31 revision a0c7c23c9c) [x86_64-linux]	**Backport:**	2.5: UNKNOWN, 2.6: UNKNOWN, 2.7: UNKNOWN

Description

```
class C
  def foo
    ["C"]
  end
end

refinement = Module.new do
  R = refine C do
    def foo
      ["R"] + super
    end

    include Module.new {
      def foo
        ["M"] + super
      end
    }
  end
end

using refinement
p C.new.foo
```

gives

```
$ ruby bug_refine_super.rb
Traceback (most recent call last):
    10920: from bug_refine_super.rb:22:in `<main>'
    10919: from bug_refine_super.rb:10:in `foo'
    10918: from bug_refine_super.rb:15:in `foo'
    10917: from bug_refine_super.rb:10:in `foo'
    10916: from bug_refine_super.rb:15:in `foo'
    10915: from bug_refine_super.rb:10:in `foo'
    10914: from bug_refine_super.rb:15:in `foo'
    10913: from bug_refine_super.rb:10:in `foo'
     ... 10908 levels...
        4: from bug_refine_super.rb:15:in `foo'
        3: from bug_refine_super.rb:10:in `foo'
        2: from bug_refine_super.rb:15:in `foo'
        1: from bug_refine_super.rb:10:in `foo'
bug_refine_super.rb:15:in `foo': stack level too deep (SystemStackError)
```

OTOH defining the module lexically outside of refine works:

```
m = Module.new {
  def foo
    ["M"] + super
  end
}
refinement = Module.new do
  R = refine C do
```

```
    def foo
      ["R"] + super
    end

    include m
  end
end

# result: ["R", "M", "C"]
```

Related issues:

Related to Ruby master - Bug #16977: Ambiguous lookup super for refinements	**Closed**

Associated revisions

Revision eeef16e1 - 07/27/2020 03:18 PM - jeremyevans (Jeremy Evans)

Prevent SystemStackError when calling super in module with activated refinement

Without this, if a refinement defines a method that calls super and
includes a module with a module that calls super and has a activated
refinement at the point super is called, the module method super call
will end up calling back into the refinement method, creating a loop.

Fixes [Bug #17007]

History

#1 - 07/03/2020 12:33 PM - Eregon (Benoit Daloze)

- Related to Bug #16977: Ambiguous lookup super for refinements added

#2 - 07/10/2020 10:20 PM - jeremyevans0 (Jeremy Evans)

After more analysis, I've determined that this issue is due to CREF handling in method lookup in search_refined_method. It fails in the lexical case because the module is affected by the refinement (refinement blocks have an implicit using of the refinement themselves, I guess).

You can reproduce the SystemStackError outside of the lexical case by having the refinement activated at the point of method definition:

```
class C
  def foo
    ["C"]
  end
end

module M; end

refinement = Module.new do
  R = refine C do
    def foo
      ["R"] + super
    end

    include M
  end
end

using refinement
M.define_method(:foo){["M"] + super()}
p C.new.foo
```

You can fix the issue by skipping any refined method during super lookup if the current method also uses the same refinement. I implemented this in a pull request: https://github.com/ruby/ruby/pull/3309.

#3 - 07/27/2020 03:18 PM - jeremyevans (Jeremy Evans)

- Status changed from Open to Closed

Applied in changeset git|eeef16e190cdabc2ba474622720f8e3df7bac43b.

Prevent SystemStackError when calling super in module with activated refinement

Without this, if a refinement defines a method that calls super and

includes a module with a module that calls super and has a activated refinement at the point super is called, the module method super call will end up calling back into the refinement method, creating a loop.

Fixes [Bug #17007]

#4 - 07/27/2020 07:30 PM - Eregon (Benoit Daloze)

Thanks for the fix!
BTW this changes the behavior on a new spec, is that intended? (result is [:A, :C] instead of [:A, :LOCAL, :C] on < 2.8)
https://github.com/ruby/spec/commit/b0da11b52560860e844470d145acee0ff4d4acea?w=1

#5 - 07/27/2020 09:02 PM - jeremyevans0 (Jeremy Evans)

Eregon (Benoit Daloze) wrote in #note-4:

> Thanks for the fix!
> BTW this changes the behavior on a new spec, is that intended? (result is [:A, :C] instead of [:A, :LOCAL, :C] on < 2.8)
> https://github.com/ruby/spec/commit/b0da11b52560860e844470d145acee0ff4d4acea?w=1

It's a consequence of the fix (skips the currently activated refinement during super), but I wouldn't say it is intended. Ideally, the fix would only affect looping cases. Unfortunately, I'm not sure if there is a better way to detect whether looping during super would occur. I'm also not sure whether the approach I committed can detect all cases of looping (there may be other ways to introduce looping during super).

Maybe the refinements spec needs to be changed if we want to forbid looping, spelling out how to handle super in modules included in refinements where the refinements are activated at the point of super call.

#6 - 09/22/2020 07:07 PM - jeremyevans0 (Jeremy Evans)

- Status changed from Closed to Assigned

To restore compatibility with Ruby 2.7 and fix #17182, I've reverted eeef16e190cdabc2ba474622720f8e3df7bac43b at 179384a66862d5ef7413b6f4850b97d0becf4ec9.

shugo (Shugo Maeda), I would appreciate your input here on how refinements should be handled in this case.