

Ruby master - Bug #14704

Module#ancestors looks wrong when a module is both included and prepended in the same class.

04/20/2018 03:23 PM - knknkn1162 (Kenta Nakajima)

Status: Closed	
Priority: Normal	
Assignee:	
Target version:	
ruby -v: ruby 2.6.0dev (2018-04-20 trunk 63212) [x86_64-darwin17]	Backport: 2.3: UNKNOWN, 2.4: UNKNOWN, 2.5: UNKNOWN

Description

Module#ancestors looks wrong when a module is both included and prepended in the same class. Here is the example script:

```
module M3; end
module M1
  include M3
end

module M2
  prepend M3
end

class Sub
  include M1
  include M2
end

# [Sub, M1, M3, M2, Object, Kernel, BasicObject]
p Sub.ancestors
```

The output is expected to be [Sub, M2, M1, M3, Object, Kernel, BasicObject] or [Sub, M3, M2, M1, Object, Kernel, BasicObject] or [Sub, M3, M2, M1, M3, Object, Kernel, BasicObject], but the actual is [Sub, M1, M3, M2, Object, Kernel, BasicObject].

When the M1 and M2 module aren't included or prepended at all like the below script, the result is [Sub, M2, M1, Object, Kernel, BasicObject]. In the first example, the position of the M2 module seems to be wrong.

```
module M1; end
module M2; end

class Sub
  include M1
  include M2
end

# [Sub, M2, M1, Object, Kernel, BasicObject]
p Sub.ancestors
```

Related issues:

Related to Ruby master - Bug #7844: include/prepend satisfiable module depend...

Closed

History

#1 - 10/17/2019 10:55 PM - jeremyevans0 (Jeremy Evans)

- Related to Bug #7844: include/prepend satisfiable module dependencies are not satisfied added

#2 - 11/20/2020 09:28 PM - jeremyevans0 (Jeremy Evans)

- Status changed from Open to Closed

The reason for this behavior is, at the point of the Sub.include M2 call, Sub.ancestors is [Sub, M1, M3, Object, Kernel, BasicObject] and M2.ancestors is [M3, M2]. So Sub.include M2 looks in the ancestry tree for M3, since that is the first ancestor of M2. It finds the ancestor already exists, so it does

not add it. Then it adds the next ancestor, M2, directly after. Hence why you get M1, M3, M2 in that order.

So this behavior isn't a bug, it's just how Module#include works. There's not a way to handle all cases perfectly. You either have to tradeoff on the order or allow modules to be added more than once:

- M1, M3, M2 (current behavior) M1 appears before M3, M2 appears after M3, as you would expect since M1 includes M3 and M2 prepends M3.
- M2, M1, M3: M3 comes after M2 even though M2 prepends M3.
- M3, M2, M1: M3 comes before M1 even though M1 includes M3. Requires moving the M3 iclass from after M1 to before M3 (include never moves positions of existing ancestors).
- M3, M2, M1, M3: M3 appears multiple times in ancestry list.

If we are going to change the behavior, only M3, M2, M1, M3 appears a reasonable candidate, and that would be a feature request to change include to add a module even though the module is already in the receiver's ancestors. I think that approach is likely to cause backwards compatibility issues.

I'm going to close this now. If you would like this reopened as a feature request to allow include to insert modules that are already in the ancestry list, please respond.