Ruby - Bug #20478

Circular parameter syntax error rules

05/08/2024 04:07 PM - kddnewton (Kevin Newton)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:		**Backport:**	3.1: UNKNOWN, 3.2: UNKNOWN, 3.3: UNKNOWN

Description

I would like to revisit https://bugs.ruby-lang.org/issues/16343.

These cases are syntax errors:

```
def foo(bar = -> { bar }) end        # no lambda parameters
def foo(bar = ->() { bar }) end      # no lambda parameters
def foo(bar = baz { bar }) end       # no block parameters
def foo(bar = baz { _1 + bar }) end # parameters, but no pipes
def foo(bar = baz { it + bar }) end # parameters, but no pipes
```

These cases are not syntax errors:

```
def foo(bar = ->(baz) { bar }) end   # lambda parameters
def foo(bar = baz { || bar }) end    # no block parameters but empty pipes
def foo(bar = baz { |qux| bar }) end # block parameters
```

I don't think these rules are very intuitive, and they feel somewhat arbitrary. I would like to suggest we change them to be either:

- Syntax error is raised if the parameter is ever read in its default value at any scope depth
- Syntax error is raised if the parameter is ever read in its default value at depth 0

Either one is fine by me, but gating the syntax error based on the presence of pipes is really confusing.

Related issues:

Related to Ruby - Bug #16343: Inconsistent behavior of 'circular argument ref...	**Closed**

History

#1 - 05/10/2024 07:48 AM - nobu (Nobuyoshi Nakada)

Given that these Procs are only created if the argument bar is not assigned, should they all be syntax errors?

#2 - 05/10/2024 01:44 PM - kddnewton (Kevin Newton)

Yes, I very much think they should all be syntax errors.

#3 - 05/12/2024 03:57 AM - nobu (Nobuyoshi Nakada)

Even this should be a syntax error?

```
def foo(bar = ->(baz = bar) {}) end
```

That means it needs to manage the list of yet-unusable variables, not only tracking single variable.

#4 - 05/13/2024 04:42 PM - kddnewton (Kevin Newton)

I figured that was already happening for the "unused" warning.

#5 - 05/14/2024 01:21 AM - byroot (Jean Boussier)

- Related to Bug #16343: Inconsistent behavior of 'circular argument reference' error added

#6 - 05/20/2024 02:05 PM - kddnewton (Kevin Newton)

@nobu (Nobuyoshi Nakada) another option would be to delete those tests and leave it up to the parser instead of forcing parse.y to implement it.

Specifically I'm talking about:

```
    o = Object.new
    assert_warn("") do
      o.instance_eval("def foo(var: bar {|  | var}) var end")
    end
```

```
    o = Object.new
    assert_warn("") do
      o.instance_eval("def foo(var: bar {|| var}) var end")
    end
```

and

```
    o = Object.new
    assert_warn("") do
      o.instance_eval("def foo(var = bar {|  | var}) var end")
    end
```

```
    o = Object.new
    assert_warn("") do
      o.instance_eval("def foo(var = bar {|| var}) var end")
    end
```

If it's too complicated to implement in parse.y, then removing these tests would be a good compromise. These tests themselves are the issue blocking me.

#7 - 05/23/2024 05:37 PM - kddnewton (Kevin Newton)

If we go with only syntax errors at depth 0, then this:

```
def foo(bar = baz { bar }) end
```

should not be a syntax error either. I think that makes sense, because the baz method could use instance_exec/instance_eval so we don't know if bar is going to be the same variable here or not.

#8 - 05/23/2024 05:39 PM - kddnewton (Kevin Newton)

Also:

```
def foo(bar = ->    { bar }) end
def foo(bar = ->( ) { bar }) end
def foo(bar = ->(_) { bar }) end
```

Two of these are a syntax error, but I think either all of them should be or none of them should be.

#9 - 06/06/2024 09:10 AM - mame (Yusuke Endoh)

Discussed at the dev meeting. @matz (Yukihiro Matsumoto) said all cases should be accepted with no syntax error. So def foo(bar = bar) = bar; foo will return nil with no warning and error.

#10 - 06/06/2024 08:30 PM - kddnewton (Kevin Newton)

- Status changed from Open to Closed

Merged.

#11 - 12/20/2024 09:16 AM - Earlopain (Earlopain _)

This used to emit a warning since all the way back from Ruby 2.2, before it was invalid syntax. Should the warning be reintroduced?