Ruby - Bug #20970

`it /1/i` raises undefined method 'it' for main (NoMethodError) even if binding.local_variables includes `it`

12/19/2024 08:59 AM - tompng (tomoya ishida)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	ruby 3.4.0dev (2024-12-19T07:16:12Z master 335bba0fde) +PRISM [x86_64-linux]	**Backport:**	3.1: DONTNEED, 3.2: DONTNEED, 3.3: DONTNEED

Description

it parameter became a local variable with #20965, but it does not behave like local variable with --parser=prism

```
i=2
42.tap do
  p it # 42
  p local_variables # [:it, :i]
  p it /1/i # should be 21, got NoMethodError
end
```

It prints 42, [:it, :i], 21with--parser=parse.y`

Related issues:

Related to Ruby - Feature #18980: `it` as a default block parameter	**Closed**

History

#1 - 12/19/2024 10:39 AM - k0kubun (Takashi Kokubun)

- Related to Feature #18980: `it` as a default block parameter added

#2 - 12/19/2024 10:57 AM - zverok (Victor Shepelev)

Interesting. Before the change (on 3.4.0dev (2024-12-15T13:36:38Z master 366fd9642f)) it was an error both with Prism and parse.y:

```
$ ruby -e "i = 2; [1, 2, 3].each { it /1/i }"
-e:1:in 'block in <main>': undefined method 'it' for main (NoMethodError)
$ ruby --parser=parse.y -e "i = 2; [1, 2, 3].each { it /1/i }"
-e:1:in 'block in <main>': undefined method 'it' for main (NoMethodError)
```

As far as I can guess (by IRB syntax highlighting), it /1/i parses at once as it(/1/i) (with /1/i treated as a regexp literal). Reproduced in other ambiguous cases (that are resolved by local name presence, for all I can understand), say:

```
ruby -e "[1, 2, 3].each { it + 1 }"
#=> OK
$ ruby -e "[1, 2, 3].each { it +1 }"
-e:1:in 'block in <main>': undefined method 'it' for main (NoMethodError)
$ ruby -e "[1, 2, 3].each { |it| it +1 }"
#=> OK
```

I.e., ambiguous operators parsing decisions are made based on whether it is known to be a local var (then it is it.+(1)), or not known (then it is tentatively considered it(+1)).

#3 - 12/19/2024 10:58 AM - zverok (Victor Shepelev)

- Backport changed from 3.1: UNKNOWN, 3.2: UNKNOWN, 3.3: UNKNOWN to 3.1: DONTNEED, 3.2: DONTNEED, 3.3: DONTNEED

#4 - 12/19/2024 05:17 PM - alanwu (Alan Wu)

- Assignee set to prism

Prism outputs undesired parse tree with the example.

#5 - 12/20/2024 05:28 AM - tompng (tomoya ishida)

The behavior of nested it discussed in #20930 has changed with #20969

```
[1].each { p it; [5].each { p it } }
# 1, 5 # --parser=prism, (confirmed by matz at dev meeting?)
# 1, 1 # --parser=parse.y (result was 1, 5 before)
```

So I think there is a room for discussion to fix prism or to revert the change made to parse.y.

#6 - 12/20/2024 06:16 PM - alanwu (Alan Wu)

- Assignee deleted (prism)

Sorry, I was too quick to blame Prism. I agree that having it as a local brings hidden semantic complexity that warrent discussion. Whether an identifier is a local influences parsing, so it's important to know exactly when it becomes a local to the parser. Usually, assignments make locals, but the way parse.y has it currently, it becomes a local on read, so in the OP example the first it essentially does it => it (not it = it because that always sets nil) implicitly to influence lines below it. It's a sort of promotion-on-first-read, if you will, a brand new way to add a local in the grammar. It can be hard to parse code mentally and know that whether an it is reading a variable and promoting. To illustrate:

```
i=2
42.tap do
  p it /1/i rescue 0 # `it` treated as method call
  p it /1/i
# NoMethodError under parse.y(e23a60b92) surprising if you expect first `it` to refer to a variable
end
```

Essentially, with the promote-to-local-on-first-use behavior, the ambiguity inherent to the design of it infects subsequent lines in the same scope. (Maybe this is the same point raised in [ruby-core:120326])

@k0kubun (Takashi Kokubun) @nobu (Nobuyoshi Nakada) I feel 46fec0f62a1803d44edb8b06e39ac0f358e56670 makes a semantic change maybe too close to the release date. Do we really want this right now? How about a narrower fix for #20955? It's fair enough to say "just don't write code that trigger parse warnings", but it's worth thinking through the details of the design, I think.

#7 - 12/20/2024 11:15 PM - k0kubun (Takashi Kokubun)

> changed with #20969

I think you meant #20965 (https://github.com/ruby/ruby/pull/12398).

> Before the change (on 3.4.0dev (2024-12-15T13:36:38Z master 366fd9642f)) it was an error both with Prism and parse.y
> I feel 46fec0f62a1803d44edb8b06e39ac0f358e56670 makes a semantic change maybe too close to the release date. Do we really want this right now?

It's too close to the release date to discuss and change how it /1/i in the ticket description's code is parsed. Since Matz said "the current master's behavior is good" before https://github.com/ruby/ruby/pull/12398, for the 3.4.0 release, it /1/i should be parsed like before the PR. As @zverok (Victor Shepelev) said, both parsers used to raise NoMethodError. Prism is good; it didn't change the behavior on the PR. parse.y now parses it in it /1/i as a local variable ((it / 1) / i), but parse.y should raise a NoMethodError (it(/1/i)).

The same applies to [1].each { p it; [5].each { p it } }. Prism didn't change the behavior, but parse.y should be changed back to print 1 and 5.

@nobu (Nobuyoshi Nakada) said he's looking into the [1].each { p it; [5].each { p it } } issue last night, so I'd like to wait for his parse.y fix for now, hoping that the it /1/i case is fixed as well. We could revert https://github.com/ruby/ruby/pull/12398 as a last resort if we don't come up with a fix.

#8 - 12/21/2024 06:47 PM - kddnewton (Kevin Newton)

I'm not sure if I'm reading this correctly or not. Is there any action required on Prism's side as of right now? Happy to make changes, though we are getting close to the release.

#9 - 12/21/2024 10:01 PM - k0kubun (Takashi Kokubun)

No, Prism is all set from my perspective. Thank you for checking on it.

#10 - 12/23/2024 04:51 AM - k0kubun (Takashi Kokubun)

- Status changed from Open to Closed

So, since it's too close to the release, we reverted https://github.com/ruby/ruby/pull/12398 at https://github.com/ruby/ruby/pull/12418. The reported issue does not happen as of the current master.

#11 - 12/24/2024 12:52 PM - Eregon (Benoit Daloze)

k0kubun (Takashi Kokubun) wrote in #note-7:

> The same applies to [1].each { p it; [5].each { p it } }. Prism didn't change the behavior, but parse.y should be changed back to print 1 and 5.

Is there a test for this? I guess not as the original PR passed CI, it would be good to add it to avoid regressions there.
Testing cases of nested _1 and mixing it and _1 would be go as well (https://bugs.ruby-lang.org/issues/20930)

#12 - 12/24/2024 05:32 PM - k0kubun (Takashi Kokubun)

I think @yui-knk (Kaneko Yuichiro) already wrote one https://github.com/ruby/ruby/pull/12435.