Ruby - Bug #4443

odd evaluation order in a multiple assignment

02/24/2011 11:28 PM - mame (Yusuke Endoh)

Status:	Closed		
Priority:	Normal		
Assignee:	ko1 (Koichi Sasada)		
Target version:			
ruby -v:	-	**Backport:**	

Description

遠藤です。core に投げてしまったようなので登録し直し。

Ruby は左から右に評価が進むと信じていたのですが、多重代入で裏切られました。

```
  def foo
    p :foo
    []
  end
  def bar
    p :bar
  end

  x, foo[0] = bar, 0
```

bar より foo が左にあるので、:foo ´:bar の順に出力されることを期待するのですが、なんと :bar ´:foo になります。

具体的に何が困るかというと、例えば

```
  obj, obj.foo = obj.foo, obj
```

には swap を期待するわけですが、そうなりません。こういうコードは実際に、木の回転などを実装するときにしばしば書きたくなります。この挙動に気がついたのも splay tree を実装していたときでした。こんなの:

```
  t.left, t.left.right, t = t.left.right, t, t.left
```

1.9 系列で修正すべきとまでは思いませんが、2.0 で直る可能性はあるでしょうか。

IRC で話したら「それで普通」みたいな反応もありましたが、

```
  foo[0] = bar
```

はちゃんと :foo ´:bar の順に出ます。

--
Yusuke Endoh mame@tsg.ne.jp

Related issues:

Related to Ruby - Bug #15928: Constant declaration does not conform to JIS 30...	**Closed**	
Is duplicate of Ruby - Bug #4440: odd evaluation order in a multiple assignment	**Closed**	**02/24/2011**

Associated revisions

Revision 50c54d40 - 04/21/2021 05:49 PM - jeremyevans (Jeremy Evans)

Evaluate multiple assignment left hand side before right hand side

In regular assignment, Ruby evaluates the left hand side before
the right hand side. For example:

```
foo[0] = bar
```

Calls foo, then bar, then []= on the result of foo.

Previously, multiple assignment didn't work this way. If you did:

```
abc.def, foo[0] = bar, baz
```

Ruby would previously call bar, then baz, then abc, then def= on the result of abc, then foo, then []= on the result of foo.

This change makes multiple assignment similar to single assignment, changing the evaluation order of the above multiple assignment code to calling abc, then foo, then bar, then baz, then def= on the result of abc, then []= on the result of foo.

Implementing this is challenging with the stack-based virtual machine. We need to keep track of all of the left hand side attribute setter receivers and setter arguments, and then keep track of the stack level while handling the assignment processing, so we can issue the appropriate topn instructions to get the receiver. Here's an example of how the multiple assignment is executed, showing the stack and instructions:

```
self                                    # putself
abc                                     # send
abc, self                               # putself
abc, foo                                # send
abc, foo, 0                             # putobject 0
abc, foo, 0, [bar, baz]                 # evaluate RHS
abc, foo, 0, [bar, baz], baz, bar       # expandarray
abc, foo, 0, [bar, baz], baz, bar, abc  # topn 5
abc, foo, 0, [bar, baz], baz, abc, bar  # swap
abc, foo, 0, [bar, baz], baz, def=      # send
abc, foo, 0, [bar, baz], baz            # pop
abc, foo, 0, [bar, baz], baz, foo       # topn 3
abc, foo, 0, [bar, baz], baz, foo, 0    # topn 3
abc, foo, 0, [bar, baz], baz, foo, 0, baz # topn 2
abc, foo, 0, [bar, baz], baz, []=       # send
abc, foo, 0, [bar, baz], baz            # pop
abc, foo, 0, [bar, baz]                 # pop
[bar, baz], foo, 0, [bar, baz]          # setn 3
[bar, baz], foo, 0                      # pop
[bar, baz], foo                         # pop
[bar, baz]                              # pop
```

As multiple assignment must deal with splats, post args, and any level of nesting, it gets quite a bit more complex than this in non-trivial cases. To handle this, struct masgn_state is added to keep track of the overall state of the mass assignment, which stores a linked list of struct masgn_attrasgn, one for each assigned attribute.

This adds a new optimization that replaces a topn 1/pop instruction combination with a single swap instruction for multiple assignment to non-aref attributes.

This new approach isn't compatible with one of the optimizations previously used, in the case where the multiple assignment return value was not needed, there was no lhs splat, and one of the left hand side used an attribute setter. This removes that optimization. Removing the optimization allowed for removing the POP_ELEMENT and adjust_stack functions.

This adds a benchmark to measure how much slower multiple assignment is with the correct evaluation order.

This benchmark shows:

- 4-9% decrease for attribute sets
- 14-23% decrease for array member sets
- Basically same speed for local variable sets

Importantly, it shows no significant difference between the popped (where return value of the multiple assignment is not needed) and !popped (where return value of the multiple assignment is needed) cases for attribute and array member sets. This indicates the previous optimization, which was dropped in the evaluation order fix and only affected the popped case, is not important to performance.

Fixes [Bug #4443]

History

#1 - 03/26/2011 10:25 PM - shyouhei (Shyouhei Urabe)

- Status changed from Open to Assigned

#2 - 06/11/2011 03:49 PM - ko1 (Koichi Sasada)

まつもとさん，こちらいかがでしょうか．
直せと言われたら私なのかなぁ．

#3 - 06/11/2011 04:04 PM - mame (Yusuke Endoh)

http://redmine.ruby-lang.org/issues/4440
に matz の返事があります。

> 優先順位は高くありませんが、直すべきだと思います。
>
> とはいうものの、1.8のころからこうだったのですし、直すのが難しいのも確
> かなのですが。

確か redmine の更新直後で、インターフェイスの違いにハマってチケット登録に失敗してしまったのでした。

--
Yusuke Endoh mame@tsg.ne.jp

#4 - 06/11/2011 10:29 PM - matz (Yukihiro Matsumoto)

- ruby -v changed from ruby 1.9.2p0 (2010-08-18 revision 29036) [i686-linux] to -

まつもと ゆきひろです

In message "Re: [ruby-dev:43724] [Ruby 1.9 - Bug #4443] odd evaluation order in a multiple assignment"
on Sat, 11 Jun 2011 15:49:30 +0900, Koichi Sasada redmine@ruby-lang.org writes:

> まつもとさん，こちらいかがでしょうか．
> 直せと言われたら私なのかなぁ．

すでに遠藤さんが指摘してくださいましたが、直せるものなら直し
たいと思ってます。

#5 - 06/11/2011 10:29 PM - matz (Yukihiro Matsumoto)

まつもと ゆきひろです

In message "Re: [ruby-dev:43724] [Ruby 1.9 - Bug #4443] odd evaluation order in a multiple assignment"
on Sat, 11 Jun 2011 15:49:30 +0900, Koichi Sasada redmine@ruby-lang.org writes:

|まつもとさん，こちらいかがでしょうか．
|直せと言われたら私なのかなぁ．

すでに遠藤さんが指摘してくださいましたが、直せるものなら直し
たいと思ってます。

#6 - 10/18/2011 09:16 AM - naruse (Yui NARUSE)

- Project changed from Ruby to 14

- Target version deleted (3.0)

#7 - 10/23/2011 05:21 PM - naruse (Yui NARUSE)

- Project changed from 14 to Ruby

#8 - 04/10/2012 06:35 PM - matz (Yukihiro Matsumoto)

Cから受け継いだ代入の評価順が「おかしい」のが原因である(本来は a → b と表記すべきか)ことを考えると、
むしろ foo[0] = bar が :bar, :fooと動作するようにすべきでしょうか。

いや、単なる思いつきなのですが。

ちなみに今調べたら mruby は foo[0] = bar が :bar, :fooと動作しますね。

Matz.

#9 - 04/10/2012 08:57 PM - mame (Yusuke Endoh)

まあ、それはそれでいいかなと思います。
ちなみに ISO とかの標準的にはどうなってるんでしょう?

--
Yusuke Endoh mame@tsg.ne.jp

#10 - 04/27/2012 02:28 PM - matz (Yukihiro Matsumoto)

手元にあるJIS x3017のドラフトを見ると現状の評価順(多重代入の場合には右辺が先)が記述してありますね(11.4.2.4)°
どうしたもんだか。

#11 - 07/14/2012 02:51 PM - ko1 (Koichi Sasada)

- Assignee changed from matz (Yukihiro Matsumoto) to ko1 (Koichi Sasada)

#12 - 07/14/2012 02:56 PM - ko1 (Koichi Sasada)

- Status changed from Assigned to Closed

duplicated (http://bugs.ruby-lang.org/issues/4443)

#13 - 10/26/2012 05:33 AM - nahi (Hiroshi Nakamura)

- Status changed from Closed to Open

It looks to be closed by mistake.

#14 - 10/30/2012 09:08 AM - ko1 (Koichi Sasada)

- Category set to core

- Target version set to 2.6

...これバグなんだっけ? □feature のような気もしますが.
2.0 には無理っぽいので，next minor にしておきます.

#15 - 08/21/2015 09:09 PM - ko1 (Koichi Sasada)

- Description updated

#16 - 01/31/2017 09:01 AM - ko1 (Koichi Sasada)

- Description updated

今更ですが、考えます（いつだろう...）。

#17 - 04/28/2017 01:45 PM - shyouhei (Shyouhei Urabe)

- Status changed from Open to Assigned

#18 - 01/24/2018 08:36 AM - akr (Akira Tanaka)

関連すると思われるささださんの昔のメールを見つけました: ruby-dev:31579

```
１）　右辺が重複しないローカル変数の列だったら今まで通り
□　（副作用はないので問題ない）
２）　そうじゃなければ，色々コストをかけて順番通りに実行
```

```
ということにしたいと思います．多分，（１）　がほとんどではないかと期待．
ネストした多重代入は　（１）　にはあてはまりませんが，まぁしょうがない．
```

#19 - 04/20/2021 12:11 AM - jeremyevans0 (Jeremy Evans)

I have submitted a pull request to fix multiple assignment evaluation order: https://github.com/ruby/ruby/pull/4390

#20 - 04/21/2021 05:49 PM - jeremyevans (Jeremy Evans)

- Status changed from Assigned to Closed

Applied in changeset git|50c54d40a81bb2a4794a6be5f1861152900b4fed.

Evaluate multiple assignment left hand side before right hand side

In regular assignment, Ruby evaluates the left hand side before
the right hand side. For example:

```
foo[0] = bar
```

Calls foo, then bar, then []= on the result of foo.

Previously, multiple assignment didn't work this way. If you did:

```
abc.def, foo[0] = bar, baz
```

Ruby would previously call bar, then baz, then abc, then
def= on the result of abc, then foo, then []= on the
result of foo.

This change makes multiple assignment similar to single assignment,
changing the evaluation order of the above multiple assignment code
to calling abc, then foo, then bar, then baz, then def= on
the result of abc, then []= on the result of foo.

Implementing this is challenging with the stack-based virtual machine.
We need to keep track of all of the left hand side attribute setter
receivers and setter arguments, and then keep track of the stack level
while handling the assignment processing, so we can issue the
appropriate topn instructions to get the receiver. Here's an example
of how the multiple assignment is executed, showing the stack and
instructions:

```
self                                    # putself
abc                                     # send
abc, self                               # putself
abc, foo                                # send
abc, foo, 0                             # putobject 0
abc, foo, 0, [bar, baz]                 # evaluate RHS
abc, foo, 0, [bar, baz], baz, bar       # expandarray
abc, foo, 0, [bar, baz], baz, bar, abc  # topn 5
abc, foo, 0, [bar, baz], baz, abc, bar  # swap
abc, foo, 0, [bar, baz], baz, def=      # send
abc, foo, 0, [bar, baz], baz            # pop
abc, foo, 0, [bar, baz], baz, foo       # topn 3
abc, foo, 0, [bar, baz], baz, foo, 0    # topn 3
abc, foo, 0, [bar, baz], baz, foo, 0, baz # topn 2
abc, foo, 0, [bar, baz], baz, []=       # send
abc, foo, 0, [bar, baz], baz            # pop
abc, foo, 0, [bar, baz]                 # pop
[bar, baz], foo, 0, [bar, baz]          # setn 3
[bar, baz], foo, 0                      # pop
[bar, baz], foo                         # pop
[bar, baz]                              # pop
```

As multiple assignment must deal with splats, post args, and any level
of nesting, it gets quite a bit more complex than this in non-trivial
cases. To handle this, struct masgn_state is added to keep
track of the overall state of the mass assignment, which stores a linked
list of struct masgn_attrasgn, one for each assigned attribute.

This adds a new optimization that replaces a topn 1/pop instruction
combination with a single swap instruction for multiple assignment
to non-aref attributes.

This new approach isn't compatible with one of the optimizations
previously used, in the case where the multiple assignment return value
was not needed, there was no lhs splat, and one of the left hand side
used an attribute setter. This removes that optimization. Removing
the optimization allowed for removing the POP_ELEMENT and adjust_stack
functions.

This adds a benchmark to measure how much slower multiple
assignment is with the correct evaluation order.

This benchmark shows:

- 4-9% decrease for attribute sets
- 14-23% decrease for array member sets
- Basically same speed for local variable sets

Importantly, it shows no significant difference between the popped (where return value of the multiple assignment is not needed) and !popped (where return value of the multiple assignment is needed) cases for attribute and array member sets. This indicates the previous optimization, which was dropped in the evaluation order fix and only affected the popped case, is not important to performance.

Fixes [Bug #4443]

#21 - 05/05/2021 03:24 PM - Eregon (Benoit Daloze)

- Related to Bug #15928: Constant declaration does not conform to JIS 3017:2013 added

#22 - 05/05/2021 03:46 PM - Eregon (Benoit Daloze)

I wrote some concerns over this change in https://bugs.ruby-lang.org/issues/15928#note-10.
I think the previous semantics of multiple assignments are better for various reasons.
We could change single assignment order, always evaluate RHS first, like MRuby behaves, if consistency is wanted.