Ruby master - Bug #12809
passing a proc to Kernel#lambda does not create a lambda

10/05/2016 03:48 PM - sylvain.joyeux (Sylvain Joyeux)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td>akr (Akira Tanaka)</td>
</tr>
<tr>
<td>Target version:</td>
<td></td>
</tr>
<tr>
<td>ruby -v:</td>
<td>2.0.0-p643, 2.1.9, 2.2.5, 2.3.1p112, 2.4.0preview2</td>
</tr>
<tr>
<td>Backport:</td>
<td>2.1: UNKNOWN, 2.2: UNKNOWN, 2.3: UNKNOWN</td>
</tr>
</tbody>
</table>

Description
I would expect the following:

```ruby
p = proc { |a| a * 2 }
l = lambda(&p)
puts l.lambda? # => true
l.call(1, 2) # => ArgumentError
```

But it does not, basically I there looks very much like a non-lambda proc (another test is that a return in p would cause l.call to raise LocalJumpError)

History

#1 - 12/21/2016 06:00 AM - shyouhei (Shyouhei Urabe)
- Status changed from Open to Assigned
- Assignee set to akr (Akira Tanaka)

#2 - 12/21/2016 06:05 AM - akr (Akira Tanaka)
- Status changed from Assigned to Rejected

Current behavior is intentional design.

If "lambda" change the lambda-ness of the given block, the author of "proc { |a| a * 2 }" can not know how the block is interpreted. This is what the design try to avoid.

Rather, it may be possible to raise error on "lambda" with non-lambda proc object. This is different issue, though.