# Ruby master - Bug #12671

# Hash#to_proc result is not a lambda, but enforces arity

08/12/2016 01:25 AM - headius (Charles Nutter)

| | | | |
|---|---|---|---|
| **Status:** | Open | | |
| **Priority:** | Normal | | |
| **Assignee:** | | | |
| **Target version:** | | | |
| **ruby -v:** | ruby 2.3.0p0 (2015-12-25 revision 53290) [x86_64-darwin14] | **Backport:** | 2.1: UNKNOWN, 2.2: UNKNOWN, 2.3: UNKNOWN |

**Description**

```
 $ ruby23 -e 'pr = {foo:1}.to_proc; puts pr.lambda?; pr.call rescue puts $!; pr.call(1, 2) rescue puts $!'
false
wrong number of arguments (given 0, expected 1)
wrong number of arguments (given 2, expected 1)
```

I believe it should be marked as a lambda, since it enforces arity.

## History

**#1 - 08/12/2016 05:47 AM - nobu (Nobuyoshi Nakada)**

None-lambda doesn't mean that it never checks its arguments.
And if it's a lambda, it doesn't match the arity value.

As for the implementation detail, there is no room for arity in ifuncs.

**#2 - 08/12/2016 06:46 AM - headius (Charles Nutter)**

JRuby implements Hash#to_proc as:

```
class Hash
  def to_proc
    method(:[]).to_proc
  end
end
```

This allows us to present the proc as a lambda with correct arity:

```
$ jruby -e "pr = {}.to_proc; puts pr.arity; puts pr.lambda?"
1
true
```

It works for MRI too:

```
$ ruby23 -e "class Hash; def to_proc; method(:[]).to_proc; end; end; pr = {}.to_proc; puts pr.arity; puts pr.lambda?"
1
true
```

I think this is more representative of this proc's behavior. Can MRI do it this way?