Ruby master - Feature #15477

Proc#arity returns -1 for composed lambda Procs of known arguments

12/28/2018 05:27 AM - robb (Robb Shecter)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

```
f = -> x { x + 2 }
g = -> x { x * 2 }
h = f << g

f.arity # => 1
g.arity # => 1
h.arity # => -1  THIS SHOULD BE 1 because h "knows" that it takes exactly 1 argument:
h.call  # => ArgumentError (given 0, expected 1)
```

Lambda Procs which are composed using << seem to partially lose knowledge of their arity. I don't know if this affects other procs, or the >> operator as well. The Proc#arity docs state that -1 is returned only when a variable or unknown number of arguments are expected by the Proc. But here, that's not the case.

History

#1 - 01/01/2019 11:09 AM - mame (Yusuke Endoh)

- Backport deleted (2.4: UNKNOWN, 2.5: UNKNOWN, 2.6: UNKNOWN)

- ruby -v deleted (ruby 2.6.0p0 (2018-12-25 revision 66547) [x86_64-linux])

- Tracker changed from Bug to Feature

Looks not a bug to me. Moving to the feature tracker.

A patch is attached.

```
diff --git a/proc.c b/proc.c
index c09e845ec0..45e2a21551 100644
--- a/proc.c
+++ b/proc.c
@@ -3063,6 +3063,16 @@ compose(VALUE dummy, VALUE args, int argc, VALUE *argv, VALUE passed_proc)
         return rb_funcallv(f, idCall, 1, &fargs);
 }

+static VALUE
+compose_proc_new(VALUE procs)
+{
+    VALUE first_proc = RARRAY_AREF(procs, 1);
+    int max_arity, min_arity = rb_proc_min_max_arity(first_proc, &max_arity);
+    int lambda_p = rb_proc_lambda_p(first_proc);
+    struct vm_ifunc *ifunc = rb_vm_ifunc_new((rb_block_call_func_t) compose, (void *)procs, min_arity, max_ar
ity);
+    return cfunc_proc_new(rb_cProc, (VALUE)ifunc, lambda_p);
+}
+
 /*
  *  call-seq:
  *     prc << g -> a_proc
@@ -3089,7 +3099,7 @@ proc_compose_to_left(VALUE self, VALUE g)
     GetProcPtr(self, procp);
     is_lambda = procp->is_lambda;

-    proc = rb_proc_new(compose, args);
+    proc = compose_proc_new(args);
     GetProcPtr(proc, procp);
     procp->is_lambda = is_lambda;

@@ -3122,7 +3132,7 @@ proc_compose_to_right(VALUE self, VALUE g)
```

```
     GetProcPtr(self, procp);
     is_lambda = procp->is_lambda;

-    proc = rb_proc_new(compose, args);
+    proc = compose_proc_new(args);
     GetProcPtr(proc, procp);
     procp->is_lambda = is_lambda;
```

#2 - 01/10/2019 08:04 AM - matz (Yukihiro Matsumoto)

Yes, please.

Matz.

#3 - 03/15/2019 12:39 PM - majjoha (Mathias Jean Johansen)

For what it is worth, this appears to be an issue when dealing with curried procs as well.

```
curried_proc = ->(a, b) { a + b }.curry # => <Proc:0x00007fa7698e7700 (lambda)>
first = curried_proc.(1) # => <Proc:0x00007fa76991a0d8 (lambda)>
curried_proc.arity # => -1
first.arity # => -1
```