Comparison of Float::NAN in array behaves unexpectedly

It seems that two arrays containing Float::NAN will be considered equal ([Float::NAN] == [Float::NAN]), despite the fact that Float::NAN != Float::NAN.

Tested and reproduced in 1.8.7p371, 1.9.3p362, 2.0.0preview2. (This bug can be reproduced in Ruby 1.8 as well.) Results below.

1.8.7 p371

```
1.8.7 :001 > nan = 0.0/0.0
=> NaN
1.8.7 :002 > nan == nan
=> false
1.8.7 :003 > [nan] == [nan]
=> true
```

1.9.3 p362

```
1.9.3p362 :001 > Float::NAN == Float::NAN
=> false
1.9.3p362 :002 > [Float::NAN] == [Float::NAN]
=> true
```

2.0.0 preview2

```
2.0.0dev :001 > Float::NAN == Float::NAN
=> false
2.0.0dev :002 > [Float::NAN] == [Float::NAN]
=> true
```

Related issues:

Is duplicate of Ruby master - Bug #1720: [NaN] == [NaN] が true になる Closed

History

#1 - 01/09/2013 11:00 PM - charliesome (Charlie Somerville)
- File bug-7676.patch added

Attached a patch fixing this issue - the pointer equality checks in recursive_equal and rb_equal should not be performed as this breaks in the case where a != a.

I'm not committing this straight away because it causes three test failures due to brittle mocks.

#2 - 01/09/2013 11:41 PM - ngoto (Naohisa Goto)
- Status changed from Open to Rejected

duplicate of Bug #1720

See documentation in numeric.c added in r37546
https://bugs.ruby-lang.org/projects/ruby-trunk/repository/revisions/37546/diff/numeric.c
This isn't just Float::NAN, actually; as Charlie's patch shows, it's actually any object that always returns false from ==

```
1.9.3p125 :001 > class X
1.9.3p125 :002>   def ==(other)
1.9.3p125 :003>     false
1.9.3p125 :004>   end
1.9.3p125 :005> end
=> nil
1.9.3p125 :006 > x = X.new
=> #<X:0x0000000ba1648>
1.9.3p125 :007 > x == x
=> false
1.9.3p125 :008 > [x] == [x]
=> true
```

Is this desirable behaviour?

At the very least, the documentation for Array#== should be updated to state that it first does an object identity comparison, then calls == only if the objects aren't the same instance.

I, too, found documentation still lacking. I read #1720, and I understand the rationale for the Float::NAN case.

However, the issue still remains as Simon pointed out above. Please reopen the issue, or update the documentation to reflect the behavior more closely.

I understand that matz wants nan == nan to be undefined, but I think this should remain consistent within a platform, even though it is undefined between platforms.

I would be happy to write a documentation patch for this if Matz can confirm which behavior is correct.

It appears calling eql? on array does not behave in this way:

```
[Float::NAN].eql? [Float::NAN]
=> false
```

Should we aim for consistency between these methods? Does it make sense for one to have an identity check and for the other not to?

I believe it doesn't really make sense for == to have an identity check, as the example in #3 is not how I'd expect Ruby to behave.

I would be happy to write a documentation patch for this if Matz can confirm which behavior is correct.