Thoughts on Perl6 eqv

Curt Tilmes

Curt.Tilmes@nasa.gov

The Perl Conference 2017 in DC

2017-06-20

http://design.perl6.org/S03.html

- Canonical equivalence

```
$obj1 eqv $obj2
```

  Compares two objects for canonical equivalence. For value types compares the values. For object types, compares current contents according to some scheme of canonicalization.

http://design.perl6.org/S03.html

- Canonical equivalence

```
$obj1 eqv $obj2
```

  Compares two objects for canonical equivalence. For value types compares the values. For object types, compares current contents according to some scheme of canonicalization.

```
proto sub infix:<eqv>(Any $?, Any $?) is pure { * }
multi sub infix:<eqv>($?)             { Bool::True }

# Last ditch snapshot semantics.  We shouldn't come here too often, so
# please do not change this to be faster but wronger.  (Instead, add
# specialized multis for datatypes that can be tested piecemeal.)
multi sub infix:<eqv>(Any:U \a, Any:U \b) {
    nqp::p6bool(nqp::eqaddr(nqp::decont(a),nqp::decont(b)))
}
multi sub infix:<eqv>(Any:D \a, Any:U \b) { False }
multi sub infix:<eqv>(Any:U \a, Any:D \b) { False }
multi sub infix:<eqv>(Any:D \a, Any:D \b) {
    nqp::p6bool(
      nqp::eqaddr(a,b)
        || (nqp::eqaddr(a.WHAT,b.WHAT) && nqp::iseq_s(a.perl,b.perl))
    )
}
```

Consider camparing sets:

```
my $s1 = set 1..10;
my $s2 = set 10...1;
say $s1.perl;
say $s2.perl;
say $s1 eqv $s2 ?? 'Set eqv' !! 'Set not eqv';
```

```
set(1,9,5,4,10,3,7,6,8,2)
set(1,9,4,5,3,10,6,7,2,8)
Set eqv
```

Consider camparing sets:

```
my $s1 = set 1..10;
my $s2 = set 10...1;
say $s1.perl;
say $s2.perl;
say $s1 eqv $s2 ?? 'Set eqv' !! 'Set not eqv';
```

```
set(1,9,5,4,10,3,7,6,8,2)
set(1,9,4,5,3,10,6,7,2,8)
Set eqv
```

- These have different `.perl`, but are still `eqv` to one another due to:

```
multi sub infix:<eqv>(Setty:D \a, Setty:D \b) {...}
```

Now put those sets into a class as attributes:

```
my $s1 = set 1..10;
my $s2 = set 10...1;

my class A { has $.s; }

my $a1 = A.new(s => $s1);
my $a2 = A.new(s => $s2);

say $a1.perl;
say $a2.perl;

say $a1 eqv $a2 ?? 'Class eqv' !! 'Class not eqv';
```

```
A.new(s => set(1,9,5,4,10,3,7,6,8,2))
A.new(s => set(1,9,4,5,3,10,6,7,2,8))
Class not eqv
```

- The fallback `Mu eqv .perl` says they are not `eqv`

- Fine, just define your own eqv

- Fine, just define your own eqv

```
my $s1 = set 1..10;
my $s2 = set 10...1;

my class A { has $.s; }

my $a1 = A.new(s => $s1);
my $a2 = A.new(s => $s2);

say $a1.perl;
say $a2.perl;

multi sub infix:<eqv>(A \l, A \r) { l.s eqv r.s }

say $a1 eqv $a2 ?? 'Class eqv' !! 'Class not eqv';
```

- Fine, just define your own eqv

```
my $s1 = set 1..10;
my $s2 = set 10...1;

my class A { has $.s; }

my $a1 = A.new(s => $s1);
my $a2 = A.new(s => $s2);

say $a1.perl;
say $a2.perl;

multi sub infix:<eqv>(A \l, A \r) { l.s eqv r.s }

say $a1 eqv $a2 ?? 'Class eqv' !! 'Class not eqv';
```

```
A.new(s => set(1,9,5,4,10,3,7,6,8,2))
A.new(s => set(1,9,4,5,3,10,6,7,2,8))
Class eqv
```

Problem solved?

Try it with Test:

```
use Test;

my $s1 = set 1..10; my $s2 = set 10...1;

my class A { has $.s; }

my $a1 = A.new(s => $s1); my $a2 = A.new(s => $s2);

say $a1.perl; say $a2.perl;

multi sub infix:<eqv>(A \l, A \r) { l.s eqv r.s }

is-deeply $a1, $a2, 'Objects Match';
```

Problem solved?

Try it with Test:

```
use Test;

my $s1 = set 1..10; my $s2 = set 10...1;

my class A { has $.s; }

my $a1 = A.new(s => $s1); my $a2 = A.new(s => $s2);

say $a1.perl; say $a2.perl;

multi sub infix:<eqv>(A \l, A \r) { l.s eqv r.s }

is-deeply $a1, $a2, 'Objects Match';
```

```
A.new(s => set(1,9,5,4,10,3,7,6,8,2))
A.new(s => set(1,9,4,5,3,10,6,7,2,8))
not ok 1 - Objects Match

# Failed test 'Objects Match'
# at eqv4.pl line 18
# expected: A.new(s => set(1,9,4,5,3,10,6,7,2,8))
#      got: A.new(s => set(1,9,5,4,10,3,7,6,8,2))
```

I've defined my own `eqv` operator for my class, but it is only in my lexical scope. Others who operate on my class are unaware of it.

I've defined my own `eqv` operator for my class, but it is only in my lexical scope. Others who operate on my class are unaware of it.

Proposal:

1) Instead of string comparing `.perl` for the canonical representation of an object, instead iterate over the attributes and `eqv` each one.

2) Have the Fallback object `eqv` check for a `.eqv()` method on the class, and call that if present.