NAME

MooseX::Types::VariantTable − Type constraint based variant table

SYNOPSIS

```
# see also MooseX::Types::VariantTable::Declare for a way to
# declare variant table based methods

    use MooseX::Types::VariantTable;

my $dispatch_table = MooseX::Types::VariantTable−>new(
    variants => [
        { type => "Foo", value => \&foo_handler },
        { type => "Bar", value => \&bar_handler },
        { type => "Item", value => \&fallback },
    ],
);

# look up the correct handler for $thingy based on the type constraints it pa
my $entry = $dispatch_table−>find_variant($thingy);

# or use the 'dispatch' convenience method if the entries are code refs
$dispatch_table−>dispatch( $thingy, @args );
```

DESCRIPTION

This object implements a simple dispatch table based on Moose type constraints.

Subtypes will be checked before their parents, meaning that the order of the declaration does not matter.

This object is used internally by Moose::Meta::Method::VariantTable and MooseX::Types::VariantTable::Declare to provide primitive multi sub support.

ATTRIBUTES

ambigious_match_callback

A code reference that'll be executed when find_variant found more than one matching variant for a value. It defaults to something that simply croaks with an error message like this:

```
Ambiguous match %s
```

where `%s` contains a list of stringified types that matched.

METHODS

new

add_variant `$type`, `$value`

Registers `$type`, such that `$value` will be returned by `find_variant` for items passing `$type`.

Subtyping is respected in the table.

find_variant `$value`

Returns the registered value for the most specific type that `$value` passes.

dispatch `$value`, `@args`

A convenience method for when the registered values are code references.

Calls `find_variant` and if the result is a code reference, it will `goto` this code reference with the value and any additional arguments.

has_type `$type`

Returns true if an entry for `$type` is registered.

has_parent `$type`

Returns true if a parent type of `$type` is registered.

TODO

The meta method composes in multiple inheritence but not yet with roles due to extensibility issues with the role application code.

When Moose::Meta::Role can pluggably merge methods variant table methods can gain role composition.

AUTHOR

Yuval Kogman <nothingmuch AT woobling DOT org>

Florian Ragwitz <rafl AT debian DOT org>

COPYRIGHT