Value constructible type trait.

Document #:	DXXXXR0
Date:	2021-05-02
Project:	Programming Language C++
Audience:	LEWG

Reply-to: Bjørn Reese <breese@stofanet.dk>

1 Introduction

This proposal adds a type trait to check that a type `T` is constructible with given arguments, but where the first argument `V` cannot be the copy and move constructors of the type `T`.

```
template <class T, class V, typename... Args>
struct is_value_constructible;

template <class T, class V, typename... Args>
struct is_trivially_value_constructible;

template <class T, class V, typename... Args>
struct is_nothrow_value_constructible;
```

A templated value constructor taking a single value as a forwarding reference accidentally matches copy and move constructors (see item 26 in [Meyers].) Such a value constructor must add condition to avoid this accidental matching. Several C++ library classes has such a condition, but they are specified in different ways. See the Prior Art section for more information.

2 Examples

With this proposal the value constructor can be specified with `enable_if`:

```
struct alpha
{
  alpha(const alpha&);

  template <typename T,
            typename = std::enable_if_t<is_value_constructible_v<T>>>
  alpha(T&&);
};
```

or with concepts

```
struct alpha
{
```

```
  alpha(const alpha&);

  template <typename T>
    requires (is_value_constructible<alpha, T>)
  alpha(T&&);
};
```

3 Prior Art

The C++ standard and associated proposals has different ways of expressing the above-mentioned exclusion.

`std::any` constrains its templated value constructor **[any.cons]** by

> *Remarks:* This constructor shall not participate in overload resolution unless `VT` is not the same type as `any`, `VT` is not a specialization of `in_place_type_t`, and `is_copy_constructible_v<VT>` is true.

`std::thread` constrains its templated value constructor **[thread.thread.constr]** by

> *Constraints:* `remove_cvref_t<F>` is not the same type as `thread`.

`std::range` uses concepts **[range.common.view]**

```
template<viewable_range R>
  requires (!common_range<R> && constructible_from<V, all_view<R>>)
constexpr explicit common_view(R&& r);
```

`std::not_fn` is missing a similar constraint [GCC70564].

`std::any_invocable` [P0288] constrains its templated value constructor **[inv.con]** by

> *Contraints:*
>
> - `remove_cvref_t<F>` is not the same type as `any_invocable`, and
> - `remove_cvref_t<F>` is not a specialization of `in_place_type_t`, and
> - `is_constructible_v<VT, F>` is `true`, and
> - `is-callable-from<VT>` is `true`.

4 Proposed Wording

The following changes are proposed to the working draft [N4727].

Add to section **[meta.type.synop]**

```
template<class T> struct is_move_constructible;
template <typename T, class... Args> struct is_value_constructible;
```

```
template<class T>
  inline constexpr bool is_move_constructible_v = is_move_constructible<T>::value;
template <typename T, class... Args>
 inline constexpr bool is_value_constructible = is_value_constructible<T>::value
```

Add entry to table 47 in section [**meta.unary.prop**] where the template is

```
template<class T, class V, class... Args>
struct is_value_constructible;
```

the condition is

is_class_v<T> is true, and
is_same_v<T, remove_cvref_t<V> > is false, and
is_constructible_v<V, Args...> is true

the preconditions are

T and V shall a complete types

Add new section [**concept.valueconstructible**].

18.4.15 Concept value_constructible

The value_constructible concept constrains the initialization of a variable of a given type with a particular set of argument types that excludes matching copy and move constructors of the given type.

```
template<class T, class V, class... Args>
 concept value_constructible = is_value_constructible_v<T, V, Args...>;
```

1 If T is an object type, then T models value_constructible only if

(1.1) – constructible_from<T, V, Args...> is true.

(1.2) – remove_cvref_t<V> is not the same type as T.

Change [**any.cons**] paragraph 8 into:

Remarks: This constructor shall not participate in overload resolution unless ~~**VT** is not the same type as **any**~~ **is**_value_constructible_v<any, T> is true, **VT** is not a specialization of **in_place_type_t**, and **is_copy_constructible_v<VT>** is true.

Change [**thread.thread.constr**] paragraph 3 into:

Constraints: ~~remove_cvref_t<F> is not the same type as thread~~
is_value_constructible<thread, F, Args..> is true

Add a constraint to [**func.not.fn**]:

Contraints: is_value_constructible<F> is true.

5 References

[N4727] Richard Smith, *Working Draft, Standard for Programming Language C++*
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/n4727.pdf

[P0288] Matt Calabrese and Ryan McDougall, *P0288: any_invocable*
http://wg21.links/P0288

[Meyers] Scott Meyers, *Effective Modern C++*, O'Reilly Media, 2015

[GCC70564] "Problem with std::experimental::not_fn"
https://gcc.gnu.org/bugzilla/show_bug.cgi?id=70564