Document number:	P1772R0
Revises:	
Date:	2019-06-17
Project:	ISO JTC1/SC22/WG21: Programming Language C++
Audience:	EWGI, LEWGI
Reply to:	Vincent Reverdy University of Illinois at Urbana-Champaign vince.rev@gmail.com

Variadic overload sets and overload sequences

Abstract

This document constitutes a short remark to P1170: Overload sets as function parameters and P0051: C++ generic overload function. From a library design standpoint we argue that the functionality provided by these papers should be considered together, and not independently. In particular, we feel that the current form of P1170 will not solve the problem of variadic overload sets built from non-inheritable classes that P0051 is facing. Making `std::overload_set` a variadic template class would allow to solve this problem.

Proposal

Before

```
// From P1170
template <typename T>
class overload_set;

// From P0051
template <class ... Fs>
auto overload(Fs&&... fcts);

// From P0051
template <class ... Fs>
auto first_overload(Fs&&... fcts);
```

After

```
// A variadic overload set: calls the best callable
template <class... F>
class overload_set;

// A variadic overload set: calls the first callable that works
template <class... F>
class overload_sequence;
```

First, instead of one class, we introduce two: one that calls the best callable according to overload rules, and the other one that calls the first callable that works. We think that the symmetry of names `overload_set`/`overload_sequence` provides clarity for the user. Second, the suggested implementation of the non-variadic `overload_set` in P1170 is:

Suggested implementation of non-variadic `overload_set` in P1170

```
template <typename T>
class overload_set {
    T f;
public:
    overload_set(/* unspecified */);

    overload_set(overload_set const&) = default;
    overload_set(overload_set&&) = default;
    overload_set& operator=(overload_set const&) = default;
    overload_set& operator=(overload_set&&) = default;
    ~overload_set() = default;

    template <typename... Us>
    invoke_result_t<F&, Us...> operator()(Us&&... us) &
        noexcept(is_nothrow_invocable_v<F&, Us...>);

    template <typename... Us>
    invoke_result_t<F const&, Us...> operator()(Us&&... us) const&
        noexcept(is_nothrow_invocable_v<F const&, Us...>);

    template <typename... Us>
    invoke_result_t<F, Us...> operator()(Us&&... us) &&
        noexcept(is_nothrow_invocable_v<F, Us...>);

    template <typename... Us>
    invoke_result_t<F const, Us...> operator()(Us&&... us) const&&
        noexcept(is_nothrow_invocable_v<F const, Us...>);
};
```

With this definition, a template class inheriting from two `overload_set` to combine two overload sets with different names may not work properly. Introducing an `overload_set` which does not provide a solution to this problem is very likely to be a dealbreaker for users. If `overload_set` is meant to be a "compiler magic" class anyway, it should be variadic and correctly handle the union of several overload sets.

At this point, we do not suggest a specific solution. Our goal was to highlight this potential problem to foster interest. This paper will be updated (potentially with a solution) before the Cologne 2019 Standards Committee Meeting.

References

- P1170: Overload sets as function parameters, Barry Revzin and Andrew Sutton, *ISO/IEC JTC1/SC22/WG21* (October 2018)
- P0051: C++ generic overload function, Vicente J. Botet Escriba, *ISO/IEC JTC1/SC22/WG21* (November 2015)
- Custom Overload Sets and Inline SFINAE for Truly Generic Interfaces, Vincent Reverdy, *CppCon 2018* (September 2018)