Overload Set Types
Bengt Gustafsson
Str Louis – June 2024
Synopsis

std::vector<float> in = getInputValues();

std::vector<float> out;
std::transform(in.begin(), in.end(), std::back_inserter(out), std::sin);
Presentation contents

• Introduction
• History
• Proposed solution
• Examples
• Issues with single overloads
• Overloaded functions don’t have types.
• Unoverloaded functions have types.
• Giving overloaded functions type allows more usage.
• In template code you may not know how many functions you have due to ADL.
• Operators don’t have a syntax representing all overloads.
History

• P0119  Andrew Sutton proposes the basics.
• P0382  A rebuttal of P0119 due to ADL issues.
• P3312R0  This paper.
• P3312Rx  Updates including discussion of P0382.
• An overloaded function has an overload-set-type.
• Overloaded member functions, all constructors, destructors and operators have overload-set-types.
• Like lambdas, different uses of the same function name may create different overload sets.
• A quoting mechanism can force a function or operator name to have its overload set type.
• A quoted function name allows ADL to find overloads.
• A quoted function name can check contracts.
std::vector<float> in = { 1, 2, 3 };  
auto out = std::ranges::transform(in, std::sin);  
auto inv = std::ranges::transform(in, `-`);

class MyClass {
public:
    MyClass(float (*fp)(float));
};

auto ptr = std::make_unique<MyClass>(std::sin);

CHECK(!check_preconditions<myOverloadedFunction>(1, "Hello");  // P3183
Features

• An **overload-set-type** is compile time only.
• It’s similar to a synthesized lambda.
• can be converted to its overloaded function pointers.
• Includes defaulted parameters.
• Includes function templates.
• Brings specifiers such as constexpr and noexcept with it.
• Works for free and member functions, constructors, destructors and operators.
• For operators includes overloads for fundamental and pointer types (just as a lambda would).
• Includes contracts.
Function template instantiations

• For function templates each use of the name is new.
• Unique addresses are however not guaranteed.

```cpp
void f(float);
void use(auto fun) { fun(1); }

use(f); // f is a function pointer, for which use is instantiated
void f(const char*);
use(f); // f is an overload set type, a new use is instantiated
use(f); // This *may* cause another instantiation. (think lambda).
void f(std::string);
use(f); // As the same overload is selected instantiation *may* be reused.
void f(int);
use(f); // As another overload is selected by use a new instantiation is needed.
```
• For class templates the overload set contents is for when the instantiation is first done.
• If this differs between TUs it is an ODR violation.
• There is only one class template instance per fully qualified function name.
• To ensure that an overload set type is used even if 0 or 1 overloads is visible quoting may be required.
What’s up with P0382?

• The example in P0382 does not show the problem.
• However, after adjustment there could be behavioral change.
• The rules can be set to avoid this.

```cpp
class C1 {
};
bool empty(const C1&);  // Exactly one declaration above remove_empty definition.
template<typename I>
I remove_empty(I first, I last)
{
    return std::remove_if(first, last, empty);  // P: point of checking of visible overloads of empty.
}
class C2 : public C1 {  // Note inheritance!
};
bool empty(const C2&);
std::vector<cont::C2> vc2(10);
auto end = cont::remove_empty(vc2.begin(), vc2.end());  // Which empty overload does remove_if call?
```
Proposed solution to P0382 issue

- Mentioning a function name does not automatically add ADL lookup.
- Back-quotes add ADL lookup to the \textit{overload-set-type}.

```cpp
class C1 {
};
bool empty(const C1&); // Exactly one declaration above remove_empty definition.

template<typename I>
I remove_empty(I first, I last)
{
    return std::remove_if(first, last, `empty`); // P: ADL enabled lookup for empty inside remove_if
}
class C2 : public C1 { // Note inheritance!
};
bool empty(const C2&);
std::vector<cont::C2> vc2(10);
auto end = cont::remove_empty(vc2.begin(), vc2.end()); // remove_if calls empty(const C2&).
```
Mental model for unquoted/quoted case.

```
struct __empty_overload_set_type_1 {
    bool operator()(const C1& c) {
        return empty(c);
    }

    operator bool &(const C1& c) const { return empty; }
};

magical set of conversion functions depending on overload set including ADL.
```
Complete back-quoting rules

• Mentioning a function name does not automatically add ADL lookup even if the function is overloaded.
• Back-quotes add ADL lookup to the overload-set-type.
• Back-quotes required if function name is not declared.
• Back-quotes required around operator token to get both free and member declarations.
• Back-quotes forces overload-set-type even if exactly one function overload exists.
• Back-quotes never allowed with qualified function name.