Document number:	*P0478R0*
Date:	*2016-10-16*
Audience:	EWG
Authors:	*Bruno Manganelli* <*bruno.manga95@gmail.com*> *Michael Wong* <*michael@codeplay.com*> *Simon Brand* <*simonrbrand@gmail.com*>
Reply-to	*Bruno Manganelli* <*bruno.manga95@gmail.com*>

Template argument deduction for non-terminal function parameter packs

This proposal aims to add support for non-terminal function parameter pack template argument deduction to the C++ language. There is a companion paper P0485 that amends rules for partial ordering of function templates which enables overload syntax, which resolves CWG1825.

Motivation and scope

Function parameter packs are a great tool for generic programming. However, some of their possible applications are constrained by the fact that packs can undergo template argument deduction only if they appear as the last parameter of the function template parameter list (or if every subsequent parameter has a default argument).

This limits the freedom of library interfaces of choosing the order of function parameters, which may lead to inconsistencies if a variadic overload of an already existing function or a variadic overload with an additional parameter is desired to be added, forcing a change of the order of the parameters in order to support the functionality.

```
template<class Variant, class... Callables>
inplace_visit(Variant&& variant, Callable&&... callables);
// Variant(s) last in std::visit, but first in hypothetical std::inplace_visit
```

Sometimes it may be convenient to directly access the last elements of the function parameter list:

```
template <class... Args, class Last>
void signal(Args... args, Last last)
{
    // callback expects 5 arguments, and we only want to pass it the first 5
    if constexpr(sizeof... (Args) > 5) {
        return signal(args...);
    } else if constexpr (sizeof... (Args) == 4) {
        callback(args..., last);
    } else {
```

```
        callback(args...);
    }
}

template <class First, class... Middle, class Last>
auto alternate_tuple(First first, Middle... middle, Last last)
{
    if constexpr (sizeof... (middle) <= 2) {
        return std::tuple(first, last, middle...);
    } else {
        return std::tuple_cat(std::tuple(first, last), alternate_tuple(middle...));
    }
}
```

Proposal

We propose that if exactly one function parameter pack is present in a function template for which explicit template arguments are not provided, such parameter pack would only be deduced to correspond to exactly the number of arguments such that the call to the variadic function is valid.

[Example

```
template <class A, class... B, class C> void foo(A a, B... b, C c);
foo(1, 2, 3, 4); // b is deduced as [2, 3]
```

-End example]

Interaction with the language

Default arguments:

Currently C++ already supports default arguments before and, according to CWG 1609, after a parameter pack. Nothing would change.

Default template arguments:

Default template arguments are already allowed after a parameter pack. There would be no semantic difference.

Overload resolution

This is dealt with by another proposal on partial ordering and transformed templates (P0485) which resolves CWG 1825, and would easily enable overloading by applying the same proposed rules of replacing the pack in the deduced context with N invented types.

This may alter the behavior of a call to the following set of function templates

```
template<class T, class... Args>
void foo(T, int, Args…);
```

```
template<class... Args>
void foo(Args..., int, int);
```

Today foo(1, 2, 3) unambiguously calls the first overload. By allowing deduction for the second version, the latter becomes the more specialized overload.
However, variadic overloads with non-terminal parameter packs do not seem to be common practice, and believe that the utility of this proposal outweighs this obscure breaking change.

Implementation experience

We have found no particular difficulty with creating an experimental implementation of the proposal, which is available at https://github.com/bmanga/clang.

Standarese Wording available on request.

Alternatives

Currently, the workarounds for accessing the last elements of a variadic function template usually involves tuple manipulations and use of std::index_sequence, at the cost of increased complexity.
Future proposals on parameter pack indexing may also help alleviate the problems.

We are still however convinced that it would make code cleaner and more uniform to lift the current arbitrary restriction on the function parameter pack position.

References

CWG 1609: http://www.open-std.org/jtc1/sc22/wg21/docs/cwg_active.html#1609
CWG 1825: http://www.open-std.org/jtc1/sc22/wg21/docs/cwg_active.html#1825
P0485: Amended rules for Partial Ordering of function templates

Acknowledgement

We like to thank Gordon Brown for his review and suggestions.