Document number: N4358

Date: 2015-01-20

Project: The C++ Programming Language, Core Working Group

Title: Unary Folds and Empty Parameter Packs

Reply-to: Thibaut Le Jehan ⟨ lejehan.thibaut@gmail.com ⟩

Table of Contents

I Introduction

The purpose of this document is to remove from the standard some of the operators from the table "Value of folding empty sequences" proposed in N4295, Folding expressions [1]. We propose to remove `operator+`, `operator*`, `operator&` and `operator|` from the aforementioned table. The overall goal is to reduce an unexpected and silent behaviour of *unary folds* while keeping the design space open for later additions.

II Motivation and Scope

The purpose of allowing empty parameter packs in *unary folds* is to allow users not to have to write binary folds for the simplest cases. However, whatever is the true intent of the users, there is only one specific type which will always be returned for a given operator when the parameter pack in the unary fold is empty. Let us consider the following `sum` function:

```
template<typename... Args>
auto sum(Args... args)
{
    return (args + ...);
}
```

Writing such a function is easy, and it does what it is expected to do most of the time. However, it will always return the integer 0 when `args` is empty. While generally not a problem, if a function has an overload taking a parameter of the expected return type of `sum` and another overload taking an `int` parameter, it may be a problem. Let us demonstrate it with the following piece of code:

```
VectorType vec = { 1, 2, 3, 4, 5 };
// do things with vec
// ...
vec = sum(some_vecs...);
```

It is common for container classes to overload `operator+` for concatenation. That is for example what `std::string` does. However, some container classes such as Eigen's [2] `Array` may also overload `operator=` to fill container with a given scalar value. With such a class, the piece of code above will do what it is expected to do almost everytime, but will silently fill `vec` with 0 when `some_vecs` is empty instead of assigning an empty vector to it, which would be the expected behaviour.

This unexpected behaviour being silent, finding errors linked to it might be rather difficult. On the other hand, if we decide that the program above is ill-formed when `some_vecs` is empty, the potential problem will be obvious when it arises. Note that, even with that change, simple things remain rather simple:

```
VectorType vec = { 1, 2, 3, 4, 5 };
// do things with vec
// ...
vec = (some_vecs + ... + 0); // Expected behaviour,
                             // four more key strokes
```

Since the fix is *that* simple, we consider that removing the special behaviour of `operator+` with regards to *unary folds* and empty parameter packs may help to catch silent errors while it won't remove any expressive power to fold expressions. We also propose to remove this special behaviour from `operator*`, `operator&` and `operator|` to avoid potential surprises.

That said, we feel that it is worth keeping the special behaviour of `operator&&`, `operator||` and `operator,` with *unary folds* and empty parameter packs: overloading these operators is generally considered bad practice anyway. The only well-known use for an overloaded `operator,` is for assignment of a sequence of values (see Boost.Assign [3] and OpenCV's `Mat` [4]). This kind of assignment should know be achieved with initializer lists anyway. Expression

templates and EDSL may also overload the three aforementioned operators, but care is already required to use these idioms.

III Design Decisions

First of all, we analyzed the rationale behind the default values provided when an empty parameter pack is given to an *unary fold*. It seems that the chosen value for an operation represents the identity element [5] for the groupoid whose set is the most commonly used together with the operation. That's why addition and multiplication return an integer (0 is the identity element for the integer addition and 1 is the identity element for the integer multiplication), the bitwise operations return unsigned integers and logicial operations return boolean values.

Therefore, our first thought was to try to generalize the idea of identity elements to user-defined types for a given operation. However, after haven given it some thought, the whole thing seemed too complex [6] and not really useful outside of mathematical libraries. Therefore, we dropped the idea of creating a generic mechanism to return the identity element of a group when an *unary fold* is given an empty parameter pack. We instead propose to remove some of the valid operators to increase safety, avoid potentially silent errors, and keep the design space open.

While we weren't able to find a generic solution for this specific problem, somebody might be able to analyze it again and to provide an elegant solution with the very same syntax but with different semantics.

IV Wording

14.5.3 Variadic templates [temp.variadic]

Delete the following lines from Table N (~~deleted lines in blue~~):

Table N. Value of folding empty sequences

Operator	Value when parameter pack is empty
~~*~~	~~1~~
~~+~~	~~int()~~
~~&~~	~~-1~~
~~|~~	~~int()~~
&&	true
||	false
,	void()

V Discussion

If N4072, Fixed Size Parameter Packs [7] or an equivalent proposal gets accepted into the standard, we think that giving an empty parameter pack of integers to an *unary fold* should be well-defined:

```
template<std::size_t N>
int sum_ints(int...[N] ints)
{
    return (ints + ...);
}

int a = sum_ints(1, 2, 3); // 6
int b = sum_ints(); // 0, identity element of integers
                    // with addition
```

That would make it possible to reintroduce some default values for `operator+`, `operator*`, `operator&` and `operator|` as proposed in N4295 when the type is already known. While genericity would still not be fully achieved, we would get both syntactic sugar and type safety for the most simple cases.

VI Acknowledgements

Thanks Jens Mauer, Andrew Sutton and Richard Smith for the feedback about the proposal and the helpful advice.

Bibliography

[1] A. Sutton and R. Smith. N4295, folding expressions. [Online]. Available: https://isocpp.org/files/papers/n4295.html

[2] Eigen c++ template library. [Online]. Available: http://eigen.tuxfamily.org/index.php?title=Main_Page

[3] T. Ottosen. Boost assignment library. [Online]. Available: http://www.boost.org/doc/libs/1_57_0/libs/assign/doc/index.html

[4] Opencv mat class documentation. [Online]. Available: http://docs.opencv.org/modules/core/doc/basic_structures.html#Mat

[5] Wikipedia. Identity element. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Identity_element&oldid=626639404

[6] Morwenn. cpp-fold library. [Online]. Available: https://github.com/Morwenn/cpp-fold

[7] B. Maurice. N4072, fixed size parameter packs. [Online]. Available: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n4072.html