Document number: Nxxxx

Date: November 19, 2014

Project: The C++ Programming Language

Title: Unary Fold Expressions 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.

II Motivation and Scope

The purpose of allowing empty parameter packs in unary fold expressions is to allow users not to have to write binary fold expressions 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 is empty. Let's say that the following `sum` function exists:

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

It is easy to write such a function, and it does what we want it to do most of the time. However, it will always return the integer 0 if the parameter pack is empty. While it is generally not a problem, if a function ihas an overload ofr the expected return type of `sum` and another overload for `int`, it may be a problem. Consider 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's for example what `std::string` does. However, some container classes, such as Eigen's [2] `Array` also overload `operator=` to fill container with the 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 if `some_vecs` is empty instead of assigning an empty vector to `vec`, which would be the expected behaviour.

This unexpected behaviour being silent, finding it might be rather difficult. On the other hand, if we decide to remove the special return value of `operator+` for an empty parameter pack in a fold expression, the line `vec = sum(some_vecs...);` will make the program ill-formed instead, clearly highlighting the error. Moreover, the fix is really simple:

```
VectorType vec = { 1, 2, 3, 4, 5 };
// do things with vec
// ...
vec = (some_vecs + ... + 0);
```

Since the fix is *that* simple, we consider that removing the special behaviour of `operator+` with regards to fld expressions with empty parameter packs may help to catch subtle and 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 surprises.

However, we feel that it is worth keeping the special behaviour of `operator&&`, `operator||` and `operator,` with fold expressions over empty parameter packs: overloading these operators is generally considered bad practice. The only well-known libray known to use an overloaded `operator,` is Boost.Assignment [3], whose usefulness is somewhat superseded by initializer lists. 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 expression. It seems that the chosen values for an operation represent the identity elements [4] for mathematical group 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.

Therefore, our first thought was to try to generalize the idea of identity elements to user-defined types for a given operations. However, after haven given it some thought, the whole thing was too complex 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 expression is given an empty parameter pack. We instead propose to remove some of the valid operators to increase safety an avoid some potential silent errors.

IV Acknowledgements

Thanks Jens Mauer for the helpful advice you provided.

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] W. T. F. Encyclopedia. Identity element. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Identity_element&oldid=626639404