Lifting Restrictions on `requires`-Expressions

Document #: WG21 P0266R1
Date: 2016-07-10
Revises: P0266R0
Audience: CWG
Reply to: Walter E. Brown <webrown.cpp@gmail.com>

Contents

Abstract

This paper proposes to lift restrictions, currently imposed by the Concepts-Lite Working Draft [N4553], on the contexts in which a *requires-expression* is allowed to appear. The proposal directly addresses Concepts Issue #3, "Allow requires-expressions in more contexts" [Issue03], which was opened pursuant to a National Body comment, and may also address Issue #29, "Allow concepts to be evaluated in any context" [Issue29].

1 Introduction

I want freedom for the full expression of my personality.
— MAHATMA GANDHI

The Concepts-Lite Working Draft, [N4553], provides wording for several new C++ language features. In our opinion, chief in importance among them are the *requires-clause* and *requires-expression*, each introduced by the new `requires` keyword. This paper seeks to lift certain restrictions imposed by the current wording on the use of a *requires-expression*.

Our proposal is similar to one made in our earlier paper [N4434]. Among other "tweaks" to the then-current draft of Concepts-Lite, we had proposed "to allow a concept name plus appropriate arguments . . . in any context where a `bool` value may reasonably appear." In the time since, we have continued to conduct very extensive experimentation with all the Concept-Lite features as implemented for the recently-released gcc 6.1 and the future gcc 7. Based on our application of these language features, we now believe it appropriate (and possibly even more important) to allow the analogous relaxation for a *requires-expression*.

2 Proposal

According to [N4553], "A *requires-expression* provides a concise way to express requirements on template arguments" [expr.prim.req]/1. We agree, but also believe there is even greater utility to such an expression, which is currently limited (by [expr.prim.req]/4) as to the contexts in which it is allowed to appear:

> A *requires-expression* shall appear only within a concept definition (7.1.7), or within the *requires-clause* of a *template-declaration* (Clause 14) or function declaration (8.3.5).

We propose to lift this restriction and thereby to allow such a construct to appear in any context that permits a **`bool`**-valued expression.

In particular, easing the above-cited limitations will avoid such boilerplate circumlocutions as:[1]

```
template< class T, class U >
constexpr bool
  is_assignable_v = false;

template< class T, class U >
  requires requires( T&& t, U&& u )
                   { std::forward<T>(t) = std::forward<U>(u); }
constexpr bool
  is_assignable_v<T, U> = true;
```

in favor of the far more straightforward:

```
template< class T, class U >
constexpr bool
  is_assignable_v = requires( T&& t, U&& u )
                             { std::forward<T>(t) = std::forward<U>(u); };
```

While similar Concepts-Lite code could be written today, it would need to be phrased as a (variable- or function-style) **`concept`** to do so. We believe that's not good enough, for a concept can't (yet) be evaluated outside a *requires-clause* or equivalent environment. We urge the adoption of this proposal to obtain the maximum possible utility from a *requires-expression*.

3 Open issues addressed

Two open Concepts Issues would be addressed by adoption of this proposal.

Issue #3, "Allow requires-expressions in more contexts," asks to "Eliminate the [above-cited] requirement, thereby permitting other uses for this new kind of expression of type bool. (For example, requires-expressions might replace many or all of the Boolean type traits.)" [Issue03]. Adoption of the present proposal would resolve this issue, which was opened via a National Body comment.

Issue #29, "Allow concepts to be evaluated in any context," is also addressed by the present proposal, effectively rendering the issue moot. The issue cites the following currently-invalid example [Issue29]:

```
static_assert( C<X>(), "" ); // for some concrete X and concept C
```

Adoption of the present proposal would make the following equivalent code well-formed:

```
static_assert( requires C<X>(), "" ); // for some concrete X and concept C
```

[1] The apparent reduplication of the **`requires`** keyword in the example code is not an error: the first occurrence introduces a *requires-clause*, while the second introduces a *requires-expression*. The present proposal seems likely to reduce the need for such stuttering.

Such a minor rewrite of the desired example would accomplish the Issue's objective: by broadening the contexts in which a *requires-expression* may be evaluated, we implicitly broaden the contexts in which a concept may be evaluated.

4 Proposed wording[2]

Modify subclause 5.1.4 [expr.prim.req] as indicated below. Note that the proposed excision in ¶4 reflects the principal purpose of this paper; the changes proposed for ¶3 and ¶7 constitute other adjustments requested by LWG during its review (in Jacksonville) of this paper's R0.

3 ~~A *requires-expression* has type bool and is an unevaluated expression (5). Note: A *requires-expression* is transformed into a constraint in order to determine if it is satisfied (14.10.2). — end note]~~ A *requires-expression* is a prvalue of type **`bool`** whose value is **`true`** when its corresponding constraint is satisfied (14.10.2) and whose value is **`false`** otherwise. [*Note:* A *requires-expression* is transformed into a constraint in order to determine whether it is satisfied. — *end note*] Expressions appearing within a *requirement-body* are unevaluated operands (5).

4 ~~A *requires-expression* shall appear only within a concept definition (7.1.7), or within the *requires-clause* of a *template-declaration* (Clause 14) or function declaration (8.3.5).~~

[*Example:* . . . — *end example*]

[*Note:* . . . — *end note*]

7 ~~The substitution of template arguments into a *requires-expression* may result in the formation of invalid types or expressions in its requirements. In such cases, the constraints corresponding to those requirements are not satisfied; it does not cause the program to be ill-formed. If the substitution of template arguments into a *requirement* would always result in a substitution failure, the program is ill-formed; no diagnostic required.~~ Invalid types or expressions may appear among the requirements of a *requires-expression*. Such requirements do not cause the program to be ill-formed. Rather, the constraints corresponding to such requirements are not satisfied.

[*Example:*

```
template<typename T> concept bool C =
  requires {
    new int[-(int)sizeof(T)]; // ill-formed, no diagnostic required
      // invalid expression formed via template argument substitution;
      // corresponding constraint will be not satisfied
  };
```

— *end example*]

5 Bibliography

[Issue03] US: "Allow requires-expressions in more contexts." Issue #3 in "C++ Concepts Active Issues List (Revision 4)." Revised 2016-06-19. http://cplusplus.github.io/concepts-ts/ts-active.html#3.

[Issue29] Andrew Sutton: "Allow concepts to be evaluated in any context." Issue #29 in "C++ Concepts Active Issues List (Revision 4)." Revised 2016-06-19. http://cplusplus.github.io/concepts-ts/ts-active.html#29.

[2] All proposed wording is relative to the Concepts-Lite Working Draft [N4553], the precursor to the Concepts-Lite Technical Specification [TS19217]. Text like this is to be added, text ~~like this~~ is to be deleted, and editorial notes are displayed like this .

[N4434] Walter E. Brown: "Tweaks to Streamline Concepts Lite Syntax." ISO/IEC JTC1/SC22/WG21 document N4434 (pre-Lenexa mailing), 2015-04-10.
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4434.pdf.

[N4553] Andrew Sutton: "Working Draft, C++ Extension for Concepts." ISO/IEC JTC1/SC22/WG21 document N4553 (post-Kona mailing), 2015-10-02. A pre-publication draft of [TS19217].
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4553.pdf.

[TS19217] International Standards Organization: "Information technology — Programming languages — C++ Extensions for concepts." Technical Specification ISO/IEC TS 19217:2015, 2015-11-15.
http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=64031.

6 Document history

Version	Date	Changes
0	2016-02-12	• Published as P0266R0.
1	2016-07-10	• Augmented §4 (proposed wording) with drive-by fixes per CWG guidance at Jacksonville. • Further updated §4 per guidance from the Project Editor. • Added §3 (open issues) and corresponding references. • Applied editorial tweaks. • Published as P0266R1.