Improving Completeness Requirements for Type Traits

Document #: WG21 P1285R0
Date: 2018–10–05
Audience: LWG
Reply to: Walter E. Brown <webrown.cpp@gmail.com>

Contents

Abstract

This paper discusses Library Issues 3022, 2939, 3099, and 2797, each involving type traits' completeness requirements. We propose wording to resolve two of these issues.

And the heaven and the earth were complete. . . .
— GENESIS 2:1

1 Introduction

Four currently-open LWG issues share a common theme: that completeness requirements for library type traits' template arguments are in need of review and likely adjustment:

- LWG 3022: **`is_convertible<derived*, base*>`** may lead to ODR

  Given two incomplete types, **`base`** and **`derived`**, that will have the expected base/derived relationship when complete, the trait **`is_convertible`** claims to support instantiation with pointers to these types (as pointers to incomplete types are, themselves, complete), yet will give a different answer when the types are complete vs. when they are incomplete.

  We should require pointers (and pointers to pointers etc.) point to a complete type, unless one is a pointer to cv-void. We may also want some weasel-wording to permit pointers to arrays-of-unknown-bound, and pointers to cv-qualified variants of the same incomplete type.

- LWG 2939: Some type-completeness constraints of traits are overspecified

  . . . . Unfortunately, there exists [*sic*] some [type traits] cases, where we currently overspecify imposing complete type requirements where they are not actually required. For example, for the following situation the answer of the trait could be given without ever needing the complete type of **`X`**:

  ```
  struct X; //Never defined
  static_assert(std::is_convertible_v<X, const X&>);
  ```

  Unfortunately we cannot always allow incomplete types, because most type constructions or conversions indeed require a complete type, so generally relaxing the current restrictions is also not an option. . . . .

- LWG 3099: **`is_assignable<Incomplete&, Incomplete&>`**

  LWG 2939 suggests that the the [*sic*] preconditions of the type traits need reevaluation. This issue focuses specifically on **`is_assignable`** and, by extension, its variants:

  . . . .

  We note a discrepancy: **`is_copy_assignable<T>`** requires **`T`** to be a complete type, but the equivalent form **`is_assignable<T&, const T&>`** does not. The requirement for **`is_copy_assignable<T>`** seems sensible, since there's no way to determine whether or not the assignment **`declval<T&>() = declval<const T&>()`** is well-formed when **`T`** is incomplete. It seems that the same argument should apply to all of the above "assignable" traits, and that they must require that the referent type is complete when given a reference type parameter to be implementable.

- LWG 2797: Trait precondition violations

  Failed prerequirement for the type trait must result in ill-formed program. Otherwise hard detectable errors will happen: . . . .

We had hoped, in this paper, to resolve all these issues by rewording the requirements re the completeness of the type traits' template arguments. However, this proved substantially more challenging than expected.

2 Discussion

During the recent Batavia LWG meeting[1] and during subsequent email discussions, it became clear that a correct formulation of requirements for each of the various type traits was a decidedly non-trivial proposition.

Even for only a single trait, namely **`is_assignable`**, we were unable to reformulate the current precondition[2] in such a way as to take comprehensive account of all such cases as:

- **`is_assignable_v<Base*&, CompleteBase*>`**.
- with **`T`** an (rvalue reference to) incomplete enumeration type and with arbitrary **`U`**, **`is_assignable<T, U>`** is always false because you can never assign to an enumeration rvalue.
- with **`T`** a (reference to) complete class type having an **`operator=(const U&)`** and with **`U`** incomplete, **`is_assignable<T&, const U&>`** is true.
- opaque pointers on the LHS of the assignment.
- **`is_assignable<int&, Incomplete&>`** must be banned, as **`Incomplete`** might have a conversion function when completed, which would cause an ODR violation.

Accordingly, with regret, this paper proposes no resolution for LWG issues 2939 and 3099. Additional insights are welcomed.

[1] Held the week of 2018–08–04.

[2] "**`T`** and **`U`** shall be complete types, *cv* **`void`**, or arrays of unknown bound."

3 Proposed wording[3]

3.1 Relocate paragraphs 1 and 2 from the end of [meta.type.synop] to the end of [meta.rqmts], renumbering paragraphs, editing existing text, and appending new text as shown.

The new wording in paragraph 5 is intended to resolve LWG 3022; it seems also to resolve LWG 2797, since "undefined behavior" permits issuing a diagnostic as the issue requests.

~~1~~4 Unless otherwise specified, ~~T~~the behavior of a program that adds specializations for any of the templates ~~defined~~specified in this subclause [meta] is undefined ~~unless otherwise specified~~.

~~2~~5 Unless otherwise specified, an incomplete type may be used to instantiate a template specified in this subclause. The behavior of a program is undefined if:

(5.1) — an instantiation of a template specified in this subclause directly or indirectly depends on an incompletely-defined object type `T`, and

(5.2) — that instantiation could yield a different result were `T` hypothetically completed.

4 Acknowledgments

Many thanks to the readers of early drafts of this paper for their thoughtful comments. Special thanks to Mike Spertus for suggesting the direction of the Proposed Wording, and to Tim Song for producing several of the cited examples/counter-examples.

5 Bibliography

[N4762] Richard Smith: "Working Draft, Standard for Programming Language C++." ISO/IEC JTC1/SC22/WG21 document N4762 (post-Rappersville/pre-San Diego), 2018–07–07. http://wg21.link/n4762.

6 Document history

Rev.	Date	Changes
0	2018–10–05	• Published as P1285R0, pre-San Diego.

[3]All proposed additions and ~~deletions~~ are relative to [N4762]. Editorial instructions and drafting notes are displayed against a gray background.