Concepts and Ref-qualifiers

Author: Douglas Gregor, Apple
Document number: N2832=09-0022
Date: 2009-02-08
Project: Programming Language C++, Library Library Working Group
Reply-to: Douglas Gregor <doug.gregor@gmail.com>

Introduction

This proposal updates the Standard Library's assignability concepts to avoid a type-safety hole that concerns associated member function requirements within concepts and their interaction with *ref-qualifier*s [**?**].

The type-safety hole occurs due to the way in which associated member function requirements are type-checked. For example, consider a simple `Assignable` concept:

```
auto concept Assignable<typename T> {
  T& T::operator=(const T&);
}
```

Using this concept, we can assign to both an lvalue and an rvalue of type `T`, as in the following well-formed code:

```
template<typename T>
  requires Assignable<T> && std::DefaultConstructible<T>
  void f(T x) {
    x = T(); // okay
    T() = x;  // okay
  }
```

This code is well-formed since the type-checking of the assignment expressions is done using an archetyp for `T` and the `operator=` requirement is translated into a member function of the archetype.

However, the `Assignable` concept applies to built-in types, e.g., the following concept map is well-formed:

```
concept_map Assignable<int> { }
```

The concept map is well-formed because type-checking for the `operator=` requirement involves type-checking an expression `a = b`, where `a` is an lvalue. Hence, both the concept map `Assignable<int>` and the constrained function template `f` are well-formed, but attempting to instantiate `f<int>` will result in an error because one cannot assign to an rvalue of non-class type.

Closing this type-safety hole in the language will be the subject of a separate proposal. This proposal, in the other hand, modifies all of the assignability concepts by adding a `&` *ref-qualifier* to the `operator=` requirement, so they constrained templates will only be permitted to assign to lvalues. This change avoids the type-safety hole and prepares for the language changes that will close that hole.

20.1.3 Operator concepts [concept.operator]

```
auto concept HasAssign<typename T, typename U> {
  typename result_type;
  result_type T::operator=(U) &;
}
```

28 *Note:* describes types with an assignment operator.

20.1.8 Copy and move [concept.copymove]

`result_type T::operator=(T&& rv) &;` *// inherited from HasAssign<T, T&&>*

7 *Postconditions:* the constructed `T` object is equivalent to the value of `rv` before the assignment. [*Note:* there is no requirement on the value of `rv` after the assignment. — *end note*]