# Simplifying unique_copy

Author: Douglas Gregor, Indiana University
Document number: N2742=08-0252
Date: 2008-08-25
Project: Programming Language C++, Library Working Group
Reply-to: Douglas Gregor <dgregor@osl.iu.edu>

## 1 Introduction

This proposal simplifies unique_copy, by removing a mandated optimization (in the form of iterator-category—dependent requirements) in favor of a more direct specification, while retaining implementor's freedom to optimize these cases.

### 1.1 The Problem

The unique_copy algorithm has by far the most complicated concepts specification of any algorithm, to the point of being embarrassing. The fundamental problem is the following requirement in [alg.unique]p5:

> If neither InputIterator nor OutputIterator meets the requirements of forward iterator then the value type of InputIterator shall be CopyConstructible (34) and CopyAssignable (table 36). Otherwise CopyConstructible is not required.

This requirement actually mandates three different implementations of unique_copy: one for (input, output), one for (forward, output), and one for (input, forward). With the predicate/operator== distinction, we end up with six implementations hidden behind the two unique_copy signatures shown in the specification. With concepts, however, we need to show each signature because the requirements differ from one signature to another, leading to the current concepts specification:

```
template<InputIterator InIter, typename OutIter>
  requires OutputIterator<OutIter, InIter::reference>
        && OutputIterator<OutIter, const InIter::value_type&>
        && EqualityComparable<InIter::value_type>
        && CopyAssignable<InIter::value_type>
        && CopyConstructible<InIter::value_type>
        && !ForwardIterator<InIter>
        && !ForwardIterator<OutIter>
  OutIter unique_copy(InIter first, InIter last, OutIter result);
template<ForwardIterator InIter, OutputIterator<auto, InIter::reference> OutIter>
  requires EqualityComparable<InIter::value_type>
  OutIter unique_copy(InIter first, InIter last, OutIter result);
template<InputIterator InIter, ForwardIterator OutIter>
  requires OutputIterator<OutIter, InIter::reference>
        && HasEqualTo<OutIter::value_type, InIter::value_type>
        && !ForwardIterator<InIter>
  OutIter unique_copy(InIter first, InIter last, OutIter result);
template<InputIterator InIter, typename OutIter,
         EquivalenceRelation<auto, InIter::value_type> Pred>
  requires OutputIterator<OutIter, InIter::reference>
        && OutputIterator<OutIter, const InIter::value_type&>
        && CopyAssignable<InIter::value_type>
        && CopyConstructible<InIter::value_type>
        && CopyConstructible<Pred>
        && !ForwardIterator<InIter>
        && !ForwardIterator<OutIter>
  OutIter unique_copy(InIter first, InIter last, OutIter result, Pred pred);
template<ForwardIterator InIter, OutputIterator<auto, InIter::reference> OutIter,
         EquivalenceRelation<auto, InIter::value_type> Pred>
  requires CopyConstructible<Pred>
  OutIter unique_copy(InIter first, InIter last, OutIter result, Pred pred);
template<InputIterator InIter, ForwardIterator OutIter,
         Predicate<auto, OutIter::value_type, InIter::value_type> Pred>
  requires OutputIterator<OutIter, InIter::reference>
        && CopyConstructible<Pred>
        && !ForwardIterator<InIter>
  OutIter unique_copy(InIter first, InIter last, OutIter result, Pred pred);
```

The concept requirements specified here state the actual operations needed to implement the various forms of the unique_copy algorithm. The negative requirements are needed to direct overload resolution, since there is no natural ordering among these overloads.

### 1.2 A Brief History

In C++98, the unique_copy algorithm was underspecified (it did not mention CopyAssignable or CopyConstructible), but the common practice was to provide all six implementations. The resolution to DR 241 introduced the language that mandated six implementations.

### 1.3 The Solution

This proposal eliminates the requirement for the ForwardIterator variants of this algorithm. Instead, the algorithm requires CopyConstructible and CopyAssignable value types (always). Implementers are, of course, free to add more-specialized overloads that optimize away the copy assignment and copy constructions when a forward iterator is available.

### 1.4 Move-Only Types

The side effect of the simpler specification is that it no longer permits the use of move-only types in unique_copy, since unique_copy always requires CopyConstructible and CopyAssignable. I believe this is a reasonable trade-off for several reasons:

1. Minimizing specification complexity is extremely important, especially with the introduction of concepts. Users will look to the standard for advice on how to use concepts, and we do not want them following the lead of unique_copy as it is currently written.
2. It's not a backward-compatibility problem: we didn't have move-only types in C++98, so no conforming C++98 or C++03 code will be broken by this change.
3. unique_copy doesn't make sense for move-only types. The algorithm requires EqualityComparable, which itself implies that you can have multiple copies of a single value. Move-only types, on the other hand, generally represent handles to resources that are uniquely held, and hence will not have particularly meaningful operator==. The prototypical example of a move-only type, unique_ptr, has an operator== that is only true when both pointers are NULL. Thus, unique_copy on unique_ptrs merely removes duplicate NULL pointers: this isn't a strong case when weighed against the specification complexity.

## 2 Proposed Resolution

In the concepts-based standard library, replace the six overloads of unique_copy with the following two signatures:

```
template<InputIterator InIter, typename OutIter>
  requires OutputIterator<OutIter, InIter::reference>
        && OutputIterator<OutIter, const InIter::value_type&>
        && EqualityComparable<InIter::value_type>
        && CopyAssignable<InIter::value_type>
        && CopyConstructible<InIter::value_type>
  OutIter unique_copy(InIter first, InIter last, OutIter result);
template<InputIterator InIter, typename OutIter,
         EquivalenceRelation<auto, InIter::value_type> Pred>
  requires OutputIterator<OutIter, InIter::reference>
        && OutputIterator<OutIter, const InIter::value_type&>
        && CopyAssignable<InIter::value_type>
        && CopyConstructible<InIter::value_type>
        && CopyConstructible<Pred>
  OutIter unique_copy(InIter first, InIter last, OutIter result, Pred pred);
```

In the pre-concepts standard library, modify [alg.unique]p5 as follows:

> *Requires*: The ranges [first,last) and [result,result+(last-first)) shall not overlap. The expression ∗result = ∗first shall be valid. ~~If neither InputIterator nor OutputIterator meets the requirements of forward iterator then the~~The value type of InputIterator shall be CopyConstructible (34) and CopyAssignable (table 36). ~~Otherwise CopyConstructible is not required.~~