Simplifying unique_copy (Revision 1)

Author: Douglas Gregor, Indiana University
Author: David Abrahams, BoostPro Consulting
Document number: N2786=08-0296
Revises document number: N2742=08-0252
Date: 2008-09-19
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.

When these requirements were written, it was not known that unique_copy could be implemented without either element copiability or an available lvalue referenced by either the InputIterator or OutputIterator arguments, thus the special CopyConstructible and CopyAssignable requirements. We now know that unique_copy can be implemented for move-only value types regardless of iterator category.

This formulation 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 negative requirements above were 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.

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, RvalueOf<InIter::value_type>::type>
        && EqualityComparable<InIter::value_type>
        && HasAssign<InIter::value_type, InIter::reference>
        && Constructible<InIter::value_type, InIter::reference>
  OutIter unique_copy(InIter first, InIter last, OutIter result);

template<InputIterator InIter, typename OutIter,
         EquivalenceRelation<auto, InIter::value_type> Pred>
  requires OutputIterator<OutIter, RvalueOf<InIter::value_type>::type>
        && HasAssign<InIter::value_type, InIter::reference>
        && Constructible<InIter::value_type, InIter::reference>
        && CopyConstructible<Pred>
  OutIter unique_copy(InIter first, InIter last, OutIter result, Pred pred);
```