X3J16/97-0078
W21/N1116
September 29th 1997
J. Stephen Adamczyk, Edison Design Group
John H. Spicer, Edison Design Group

Proposed Revision to Partial Ordering Rules

1. Introduction

A problem has been found in the partial ordering rules for function templates. The problem is caused by the fact that partial ordering is currently done only when several function template specializations are equivalent for overload resolution purposes. The result is that a trivial difference in overload resolution can result in a much worse function (from the point of view of partial ordering) being chosen. The example that follows shows a partial case in which the partial ordering rules are never employed because the `nonconst` function is better match than the `const` function in overload resolution. Had the partial ordering rules been employed the `B<T>&` function would be preferred to the `T&` function because the qualifiers under a reference are ignored.

```
template <class T> struct B {};
template <class T> int f(T&);
template <class T> int f(const B<T>&);
B<int> bi;
int i = f(bi);
```

Similar but perhaps even more counterintuitive behavior occurs with member function templates. In this example, the cv-qualifiers on the implicit `this` parameter take precedence over the partial ordering based on the function parameter types:

```
template <class T> struct B {};
struct A {
      template <class T> int f(T);
      template <class T> int f(B<T>) const;
};

A a;
B<int> bi;
int i = a.f(bi);  // calls A::f(T), should call A::f(B<T>) const;
```

It important to recall that the partial ordering rules are used not only to select a “more specialized” version of a function template from a set provided by a given user, but also to permit the overloading of templates with the same name by different users. For example, if one library declared an operator template such as:

```
template <class T> bool operator ==(T&,T&);
```

That template would make it impossible to ever call another˝template with the signature:

```
template <class T> bool operator ==(const B<T>&, const B<T>&);
```

2. Proposed Solution

The proposed solution is to move the partial ordering comparisons from the point at which a set of better matching functions is found to the earlier point at which a set of viable functions has been established (i.e., from 13.3.3 [over.match.best] to 13.3.2 [over.match.viable].

When the original partial ordering rules were chosen, an important attribute of the rules was that they compared functions, rather than being another factor used to compare one argument/parameter pair with another argument/parameter pair. The proposed solution retains˝this attribute.

3. Working Paper Changes

Add the following to the end of 13.3.2 [over.match.viable]:

> If the set of viable functions contains function template specializations, the partial ordering rules described in 14.5.5.2 shall be applied to the set of function templates of which the functions are specializations. A function template specialization is removed from the set of viable function if the function template of which it is a specialization is less specialized than another function template in the set.

Remove the following text from 13.3.3 [over.match.best]:

> F1 and F2 are template functions, and the function template for F1˝is more specialized than the template for F2 according to the partial ordering rules described˝in 14.5.5.2, or, if not that,