What is “Partial Ordering of Template Functions”

This document is X3J16/97-0092 WG21/N1130. Author: Erwin Unruh

John Spicer has asked to change the rules of how partial ordering interacts with overload resolution. When I was trying to find a better solution John Wiegley showed me a bad example. I also think I do have a better understanding of what it is we want to do.

The bad example is quite simple:

```
template <class T> void f( T* );                    // #1
template <class T> void f( const T*);

f( (int*) 0 );
```

The second function is more specialized than the first. Both are viable. Under the current rule the first is preferred because it does not involve a conversion, whereas the second has to undergo a qualification conversion. Under the rules proposed by John Spicer the second would be called because it is more specialized than the first, which would eliminate the first from the set of viable functions.

This is highly counterintuitive. Even worse: Function 1 would never be called since there is always the possibility of adding a const in the call.

After realizing this problem I asked the question: Why are we trying to choose the second function in

```
template <class T> class B;

template <class T> void f(T*);
template <class T> void f( const B<T>* );           // #4

f( (B<int>*) 0 );
```

After some thought (and discussion with John Wiegley) I came to the conclusion that we have the concept of *being more deduced* in mind. Function 4 is more deduced because it is T=int instead of T=B<int>. The same applies to the first example when called with const int*. The deduction T=int is more deduced than T=const int.

When having the concept of more deduced in mind, we are going to think of categories of being more deduced. The differences of (T=int vs. T=B<int>) and (T=int vs. T=const int) are not the same. The latter is a qualification difference and the former is a *specialization* difference.

Now it becomes clear why we want to have function 4: The specialization difference has more weight than the qualification conversion. It even explains the uneasy feeling I have regarding the following example:

```
template <class T> struct B{};
template <class T> struct D : public B<T> {};

template <class T> void f(T*);
template <class T> void f(B<T>*);

f( (D<int>*) 0 );
```

The problem in here is weighting the specialization difference against a derived-to-base conversion. I don’t know how to weight them.

Thinking even further there is a set of different deduction kinds. Each of them may cause a function to be more specialized than another. I am just not ready to give weights to those differences and compare those weights with conversion weights.

Trying to do the right thing is going to invent a new structure which is somehow similar to overloading. If we had time, I would gladly think about doing something, but we don't have any time left. I think we need to do the best we can do now. Integrating partial ordering into overload resolution is something which I cannot do at the moment! It will require too much work.

Having seen this idea come up, I think we should think a little bit of how to best prepare for some future work in this area.

Erwin Unruh