|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| US |  | Entire draft |  | ge | Address existing open issues in core and library issues lists | Make technical and editorial changes as appropriate for each issue, or resolve as NAD |  |
| US |  | 16.8 | ¶ 1 | te | The definition of the macro \_\_cplusplus refers to C++14, not C++17 | Update definition to reflect the expected ratification month |  |
| US |  | 20.14.2 | ¶ 2 | te | The distinction between *INVOKE*(f, t1, t2, … tN) and *INVOKE*(f, t1, t2, … tN, *R*) is too subtle. If the last argument is an expression, it represents tN, if it’s a type, then it represents R. Very clumsy. | Rename *INVOKE*(f, t1, t2, … tN, *R*) to *INVOKE\_R*(*R*, f, t1, t2, … tN) and adjust all uses of this form.  (Approximately 10 occurrences of invoke would need to change.) |  |
| US |  | 20.15.2 and 20.15.6 |  | te | The trick of encoding a functor and argument types as a function signature for is\_callable and result\_of loses cv information on argument types, fails for non-decayed function types, and is confusing. E.g.,  typedef int MyClass::\*mp;  result\_of\_t<mp(const MyClass)>;  // should be const, but isn’t  typedef int F(double);  is\_callable<F(float)>; *// ill-formed* | **Minimal change**: Replace is\_callable<Fn(ArgTypes...)> with is\_callable<Fn, ArgTypes...> and replace is\_callable<Fn(ArgTypes...), R> with is\_callable\_r<R, Fn, ArgTypes...>. Do the same for is\_nothrow\_callable  **Preferred change:** All of the above, plus deprecate result\_of<Fn(ArgTypes...)> and replace it with result\_of\_invoke<Fn, ArgTypes...> |  |
| US |  | 20.15.2 and 20.15.6 |  | te | “is\_callable” is not a good name because it implies F(A…) instead of *INVOKE*(F, A…) | Rename “is\_callable” to “is\_invocable” and rename “is\_nothrow\_callable” to “is\_nothrow\_invocable” |  |
| US |  | 1.10.2 | ¶ 14 | ed | The term “block with forward progress guarantee delegation” is cumbersome. “Forward” is redundant and “guarantee” is implicit. | Replace the term “block with forward progress guarantee delegation” with “block with progress delegation” throughout the standard. |  |
| US |  | 20.19.4 | Section heading | ed | “Sequential” should be “Sequenced” (per P0336r1, which was adopted 2016-06) | Change “Sequential” to “Sequenced” in section heading |  |
| US |  | 20.19.6 | Section heading | ed | “Parallel+Vector” should be “Parallel+Unsequenced” (per P0336r1, which was adopted 2016-06) | Change “Parallel+Vector” to “Parallel+Unsequenced” in section heading and change section label from “[execpol.vec]” to “[execpol.parunseq]” |  |
| US |  | 25.2.3 | ¶ 1 | ed | Need a cross-reference directing readers to execution policies [execpol] section | Add a cross-reference link to section 20.19, somewhere within the paragraph. |  |
| US |  | 25.3, 25.4, 25.5 |  | ed | Presentation of parallel algorithms is confusing. Despite having parallel overload prototypes in section 25.1 <algorithm> synopsis and blanket wording 25.2.5, it is still confusing to figure out which algorithms have parallel overloads. | Copy the prototypes for the parallel algorithm overloads alongside their serial versions in the per-algorithm description. The common description of a serial and parallel overload will reinforce that they exist and have the same semantics. In the cases where they do not have the same semantics, their separate descriptions will make that clear, too. |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |