Subject: Comments on Clause 7
From: Van Snyder

1 Edits

Edits refer to 06-007r1. Page and line numbers are displayed in the margin. Absent other instructions, a page and line number or line number range implies all of the indicated text is to be replaced by associated text, while a page and line number followed by + (-) indicates that associated text is to be inserted after (before) the indicated line. Remarks are noted in the margin, or appear between [and] in the text.

1.1 General

[Editor: Delete the last sentence of the note in Table 7.1, since it duplicates [138:3] and [138:20].] 137:7+27-28

[The type of the expression is defined by Table 7.1, and it says so at [140:35]. Editor: Delete "type ... and the".] 141:33

[Editor: If we don't allow general expressions in declarations in the specification part of BLOCK constructs, insert "or construct" after "subprogram".] 142:4

[Editor: Delete to allow general expressions in declarations in the specification part of BLOCK constructs.] 142:14-17

[The paragraph is irrelevant to initialization expressions, since they can't invoke specification functions. Editor: Delete it.] 144:41-145:1

[A not-very-difficult reading of 7.1.4.1 and 12.5.3 suggests that the concern of this paragraph is groundless. Editor: Delete it.] 145:11-12

[A not-very-difficult reading of 7.1.4.1 and 5.2 suggests that the concern of the sentence that begins "The type..." is groundless. Editor: Delete it.] 145:13-15

[A not-very-difficult reading of 7.1.4.1 and 5.2 suggests that the concern of the sentence that begins "The type..." is groundless. Editor: Delete it.] 145:16-17

[A not-very-difficult reading of 7.1.4.1 and 5.2 suggests that the concern of the sentence that begins "The type..." is groundless. Editor: Delete it.] 146:1-2

[A not-very-difficult reading of 7.1.4.1 and 4.7 suggests that the concern of the sentence that begins "The type..." is groundless. Editor: Delete it.] 146:6-7

[This paragraph follows from [147:9-11]. As such, it ought to be in a note. Editor: Move it to be at the beginning of Note 7.21, with the first phrase there ending up at the end of the moved paragraph.] 148:3-6

[This paragraph follows from [147:9-11]. As such, it ought to be in a note. Editor: Move it to be at the beginning of Note 7.22, with the first phrase there ending up at the end of the moved paragraph.] 148:7-8

[The part of the second sentence of this paragraph that deals with type and type parameters belongs at [140:39+]. Editor: Copy it there, making it a new paragraph, then "type, type parameters and interpretation" ⇒ "type and type parameters" in the copy. Then delete "type, type parameters and" here. Then delete the reference to 7.2 at [140:39].] 150:20-22

[Editor: "are" ⇒ ", is".] 154:12

[The paragraph at [155:2-4] is essentially identical to the one at [154:14-16]. Editor: ", which ... combined" ⇒ ". This precedence determines the order in which the operands are to be combined in determining the interpretation of the expression". Then delete [155:2-4].] 154:14-16,155:2-4

[The double negative makes item (4) confusing. It's also the only item that doesn't begin with "if". Editor: Replace it:] 157:12-13

(4) if the variable is not an allocatable array with the same rank as *expr*, is a co-array, or is a co-indexed object, the shapes of the variable and *expr* shall conform,

[Editor: To prevent automatic reallocation of co-arrays insert ", is not a co-array," after "allocatable".] 157:21

[Editor: To further prevent automatic reallocation of co-arrays insert "non-co-array" after "allocated" twice.] 158:21,24

[The paragraph is misleading by being only a third of a description. Editor: Either append a sentence "A pointer may also become disassociated by execution of a NULLIFY or DEALLOCATE statement, or may become undefined if an event described in 16.5.2.2.3 occurs." and then put the whole paragraph in a note, or (better yet) delete the paragraph altogether since it doesn't describe pointer assignment.] 162:25

[Editor: Move "(4.3.1.3)" from [164:11] to [164:4].] 164:4,11

[Editor: Insert a note:] 164:13+

NOTE 7.48a

Given that sequence types are not extensible, that an extension cannot have the SEQUENCE attribute, and that a *data-pointer-object* shall be type compatible with its *data-target*, the only way the dynamic type of the *data-target* can be different from the declared type of the *data-pointer-object* is if the *data-target* is unlimited polymorphic and is associated with a target.

1.2 Problem with intrinsic assignment

[Putting "each co-array component" ahead of "each other ... component" at [161:4-6] gives the impression that "other" refers to "co-array", when in fact it refers to nonpointer nonallocatable components for which defined assignment is not accessible. An improvement is "co-array ... applied" ⇒ "other nonpointer nonallocatable component or each co-array component, and using the following sequence of operations for each non-co-array allocatable component". Even if that is done, however, part of the description of intrinsic assignment is incomplete in that it doesn't completely specify what happens to allocatable co-array components, and another part is redundant in that it repeats the description of how defined assignment is applied.] 161:1-15

In a derived-type intrinsic assignment, an allocated allocatable co-array component of the variable shall correspond to an allocated component of the value of *expr* that has the same type parameters and shape, and the same dynamic type if the component is polymorphic. An unallocated allocatable co-array component of the variable shall correspond to an unallocated component of the value of *expr*. A derived-type intrinsic assignment is performed as if each component of the variable were assigned from the corresponding component of the value of *expr* according to the following process.

[The above introduces new semantics for allocatable co-array assignment by not requiring the components to be allocated. This could easily be removed. It restates the requirements for type, type parameter and shape conformance to avoid saying "non-co-array" in items (2-4) in the following list. It could be done the other way.] *Remark to J3*

(1) If the component has the pointer attribute, pointer assignment (7.4.2) is used.

(2) If the component of the variable is allocatable and allocated, and the component of the value of *expr* is not allocated or has different dynamic type, type parameters or shape, or the dynamic type of the component of the variable has a finalizer, the component of the variable is deallocated.
[Item (2) explicitly allows an obvious optimization. The phrase "or the dynamic type of the component of the variable has a finalizer," absent from [158:21-26], retains f2003 semantics.] *Remark to J3*

(2a) [Alternative to (2)] If the component of the variable is allocatable and allocated, it is deallocated.

(3) If the component of the value of *expr* is allocatable and not allocated, no assignment is performed and this process is completed.

(4) If the component of the variable is allocatable and not allocated, it is allocated with the same dynamic type, type parameters, and bounds as the corresponding component of the

value of *expr*.

(5) The component of the variable is assigned from the value of the component of *expr* as if by an assignment statement (7.4.1).

[Item (5) avoids restating the conditions leading to defined assignment. Since Item (4) ensures the component is allocated, the semantics of the assignment statement will not deallocate and allocate it again.] *Remark to J3*

2 Correct a defect in intrinsic assignment

[Suppose one has

```
real, allocatable :: A(:)
integer, allocatable :: B(:)

allocate ( a(10), b(5) )
b = 42
a = b
```

According to [158:21], A is deallocated before the assignment because the shapes differ, and then according to [158:26] it is allocated with type integer. This is clearly absurd. At least in 06-007r1 the qualifier "the variable is polymorphic and" was inserted before "the dynamic" at [158:29], but this doesn't solve the problem of trying to allocate A with type integer.]

[Editor: Insert ", if the variable is polymorphic," before "with".] 158:26

3 Technical change from 2003

For allocatable entities, intrinsic and defined assignment that arise directly from assignment statements, and assignments of derived-type components, are slightly (and needlessly) different:

(1) In assignment of derived-type components, allocatable components of the variable are unconditionally deallocated, and then not allocated if the corresponding component of *expr* is not allocated. In intrinsic assignments that arise directly from assignment statements, an allocatable variable is deallocated only if its type, type parameters or shape are different from *expr*, and then it is unconditionally allocated, which implies that *expr* has to be allocated.

(2) In assignment of derived-type components, allocatable components of the variable are unconditionally deallocated even if the assignment ultimately gets handled by defined assignment. In assignment that arises directly from an assignment statement, the variable is not deallocated and reallocated if the assignment is a defined assignment.

These could be made more nearly or completely parallel in several ways.

(1) The conditions for deallocating allocatable components of derived-type objects could be made the same as for intrinsic assignments that arise directly from assignment statements, with components for which defined assignment is ultimately done not deallocated. This would make intrinsic assignment that arises directly from assignment statements parallel to assignments of derived-type components, at the expense of a change in the semantics of derived-type assignment (finalizers would only run if the component is deallocated). This leaves defined assignments that arise directly from assignment statements different from the other two.

(2) Intrinsic assignments that arise directly from assignment statements could allow deallocated-to-deallocated assignment, and add a caveat "except if the variable has a finalizer" to the conditions for deallocation. This would make intrinsic assignment that arises directly from assignment statements parallel to assignments of derived-type components, at the expense of

a change in the semantics of intrinsic assignments that arise directly from assignment statements (variables with finalizers would not be deallocated). This leaves defined assignments that arise directly from assignment statements different from the other two.

(3) An allocatable variable in any assignment, intrinsic or defined, that arises directly from an assignment statement, could be deallocated under the same conditions as for intrinsic assignment that arises directly from an assignment statement, along with a caveat "except if the variable has a finalizer." This would make assignment that arises directly from an assignment statement and assignment of derived-type components exactly parallel, at the expense of a change in the semantics of both intrinsic and defined assignments that arise directly from assignment statements. This makes all three assignments parallel.

3.1 Change assignment of allocatable derived-type components

This can be done by omitting "or the dynamic type of the component of the variable has a finalizer" from item (2) in the edits in section 1.

3.2 Change intrinsic assignments arising directly from assignment statements

[This also allows to simplify the edits in section (1) above:]

[Editor: Replace "*expr*" by "it has a finalizer, *expr* is allocatable and allocated or".] 158:21

[Editor: Insert "*expr* is allocated and" before the second "the variable" 158:23

[Editor: Replace "it" by "the variable".] 158:24

[Editor: Append a new sentence "If *expr* is allocatable and not allocated, no further action takes place."] 158:26

In a derived-type intrinsic assignment, an allocated allocatable co-array component of the variable shall correspond to an allocated component of the value of *expr* that has the same type, type parameters, and shape. A derived-type intrinsic assignment is performed as if each pointer component of the variable were assigned from the corresponding component of the value of *expr* using pointer assignment (7.4.2), and each nonpointer component were assigned as if by an assignment statement. 161:1-15

3.3 Change all assignments arising directly from assignment statements

There are two ways to do this: Sneak up a little closer by putting complicated rules in place to deallocate the variable in a defined assignment, or always deallocate a polymorphic allocatable variable if the conditions in paragraph three of 7.4.1.3, as modified by the four edits for page 158 in section 3.2 above, are met.

3.3.1 Sneaking up a little closer

[Do the edits in section 3.2 above, and also the following.]

[Append the following within the same paragraph.] 162:11

If the variable is allocatable, the dummy arguments d_1 and d_2 of the subroutine that defines the assignment have the same type, kind type parameters and rank, d_1 is not allocatable and not polymorphic, d_2 is not polymorphic, and *expr* is allocatable and not allocated or the variable and *expr* have different dynamic type, type parameters or shape, the variable is deallocated. Then, if *expr* is not deallocated, the variable is allocated with the same dynamic type, type parameters and bounds as *expr*. Then, if the variable is allocatable and not allocated, no further action takes place.

3.3.2 Exact parallelism

[Do the edits in section 3.2 above. Then move paragraph three of 7.4.1.3 to the end of 7.4.1.1.]

3.4 Exact parallelism changing all assignments

[Do the edits in section 3.2 above, omitting "it has a finalizer,". Then move paragraph three of 7.4.1.3 to the end of 7.4.1.1.]

171 ### 3.5 In any case...

172 [Editor: Delete "(7.4.1.3)" and insert "for which the conditions specified in 7.4.1.3 are satisfied" before 131:12-13
173 "is deallocated".]

174 ## 4 Comments without edits

175	Items (1), (2), (3), (5), (6) and (7) could reasonably be constraints.	157:8-22
176	What if the first argument has INTENT(INOUT)?	171:26-27
177	Shouldn't the first sentence be a constraint on R755?	174:7-8