Subject: Comments on Section 7, Unresolved issues 334 and 335
From: Van Snyder
References: 02-123, 02-133

1 Edits that make a technical change

Edits refer to 02-007. 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.

A pointer assignment statement in which *data-pointer-object* is nonpolymorphic and *data-target* is polymorphic is currently allowed, so long as the dynamic type of *data-target* is the same as the type of *data-pointer-object*. This implies (at least in good implementations) that there will be a run-time check and error message if it fails. Now that we have SELECT TYPE, it would be better to prohibit this at the constraint level.

C714$\frac{1}{2}$ (R735) If *data-target* is polymorphic (5.1.1.8), *data-pointer-object* shall be polymorphic. 136:1-

---

If *data-pointer-object* is polymorphic (5.1.1.8), it assumes the dynamic type of *data-object*. 136:33-34

2 Edits having potential technical content

Surely we don't allow just any kind type parameter for the result in the case of integer operands having different kind type parameters but the same decimal exponent range, real operands having different kind type parameters but the same decimal precision, or logical operands that happen to have different kind type parameters.

[Editor: "or ... range" ⇒ "if the decimal exponent ranges are different; if the decimal exponent ranges are the same, the kind type parameter of the expression is the same as that of one of the operands, but it is processor dependent which one".] 117:30-31

[Editor: "or ... precision" ⇒ "if the decimal precisions are different; if the decimal precisions are the same, the kind type parameter of the expression is the same as that of one of the operands, but it is processor dependent which one".] 117:33-34

[Editor: "processor dependent" ⇒ "the same as that of one of the operands, but it is processor dependent which one" after "expression".] 117:37

3 Edits – Hopefully just editorial

[Editor: Insert ", pointer assignment" after "defined assignment".] 111:3

[Editor: Move first paragraph of 7.1.1 to be last paragraph of 7.1, because it has nothing to do with form of expressions.] 111:10-11

C701$\frac{1}{2}$ (R701) The *designator* shall not be a whole assumed-size array. 111:27

[Editor: Indent the last line of Note 7.3.] 112:19-

[Editor: Indent the last line in each of of Notes 7.4 and 7.5.] 113:3-,18-

---

[Editor: Indent the last line of Note 7.6.] 114:1-

[Editor: Indent the last five lines of Note 7.7.] 114:8-

[Upside down.] 115:1-3
A **numeric intrinsic operator** is one of +, -, *, / or **. A **numeric intrinsic operation** is an intrinsic operation for which *intrinsic-operator* is a numeric intrinsic operator.

[Editor: After making changes specified below for [129:5-9], move [129:2-14] to here, and delete [129:1] (because the moved stuff is not about interpretation).] 115:23+

[Editor: After making changes specified below for [129:19-23] and [129:28ff], move [129:16-28] to here, and delete [129:115] (because the moved stuff is not about interpretation).] 115:27+

[Editor: Insert "5.1.1.8" after "12.4.2".] 116:32

[Too wordy. Editor: "In ... case" ⇒ "If the function reference is generic".] 116:32-33

[Editor: "The ... current" ⇒ "The shape, dynamic type and type parameters of a pointer that is associated with a target are those of the". 117:7

[Editor: "," "it has no shape, and its type and type parameters are its declared type and type parameters;".] 117:8

[Editor: Insert ", type parameters, and shape" after the second "type"; Exchange "The type of the result of a defined ... (7.3)" and "The shape ... otherwise".] 117:14-17

[No chance to take a breath. In also sounds like expressions of nonintrinsic type don't have type parameters. Editor: "An ... length parameter" ⇒ "The type parameters of the result of an intrinsic operation are as follows:" at [117:18-19]. Then make a list, starting each list item with "For an expression..." at lines 22, 24, 26, 34 and 37.] 117:18-39

[Editor: "," ⇒ "and".] 119:20

[Editor: Insert "of" before the first "the" in the third line of Note 7.14.] 122:4+4

[Editor: "`X`" ⇒ "`A`" twice (because it's presumably likely to be a different variable from in the preceding example). 122:9+8-9

122:12+1-3

NOTE 7.16

In the examples in Note 7.15, if `L` or `W` defines its argument, evaluation of the expressions under the specified conditions causes `Z` to become undefined, no matter whether `L`(Z) or `W`(Z) is evaluated.

[Editor: Replace "Nonallowable" in the second heading in Note 7.19 with "Forbidden", for consistency with the text two lines above it.] 123:7+a bunch

[Editor: Insert "(7.1.3)" after "operations".] 128:17

129:5-9

(2) Either

(a) A generic interface (12.3.2.1) provides the function with a *generic-spec* of OPERATOR (*op*), or

(b) There is a type-bound generic binding (4.5.1.5) in the declared type of x_2 with a *generic-spec* of OPERATOR (*op*) and there is a corresponding binding to the function in the dynamic type of x_2,

(3) The type of d_2 is compatible with the dynamic type of x_2,

(2) Either 129:19-23

(a) A generic interface (12.3.2.1) provides the function with a *generic-spec* of OPERATOR (*op*), or

(b) There is a type-bound generic binding (4.5.1.5) in the declared type of x_1 or x_2 with a *generic-spec* of OPERATOR (*op*) and there is a corresponding binding to the function in the dynamic type of x_1 or x_2, respectively,

(3) The types of d_1 and d_2 are compatible with the dynamic types of x_1 and x_2, respectively,

---

[The specifications in 7.1.1 seem more to be specifications than implications. Editor: "implied ... form" ⇒ "corresponding to the form of expressions specified in"; "which ⇒ "that"; Delete the comma on line 31.] 129:30-31

[This subclause so far has specified precedence of operations. Editor: Insert "operations defined by" before "operators".] 130:2

[Editor: Indent the fourth nonblank line of the continuation of Note 7.32.] 131:0+5

[Now that we have a syntax term index, Note 7.35 is unnecessary. I don't think there are other notes of the same form anyway. Editor: Delete Note 7.35.] 131:12+1-2

[Editor: "A variable" ⇒ "The *variable*".] 132:1

[Editor: Move to [136:0+]. 136:8-9

[Editor: Move to [136:0+]. 136:13-18

4 Exposition of assignment needs work

The description of intrinsic assignment is quite tangled. The first problem is that it eventually gets around to saying "intrinsic assignment is an assignment that isn't defined assignment," but it does that wrong. It also has stuff about interpretation in the section in conformance, and stuff about definition in the section on interpretation. Hopefully, all we do here is rearrange stuff, and get it right, not introduce technical differences.

An *assignment-stmt* shall meet the requirements of either a defined assignment statement or an intrinsic assignment statement. 132:2

[Editor: "12.3.2.1" ⇒ "12.3.2.1.2".] 132:5

[The description of defined assignment at [132:12-13] is inadequate. This is an example of "say it twice, get it wrong at least once." We could duplicate the description in 7.5.1.6, but the description of defined assignment will below be made complete in the subclause entitled **Defined assignment statement**, which was 7.5.1.2 but is to be moved and will have the definition (but not interpretation) parts of 7.5.1.6 put into it. Therefore, we now can simply say "not defined assignment":] 132:8-13

An **intrinsic assignment statement** is an assignment statement that is not a defined assignment statement (7.5.1.2), and in which

(1) If *variable* is an array then *expr* shall also be an array,

(2) The shape of *variable* and *expr* shall conform, and

(3) Either

(a) The types of *variable* and *expr* are intrinsic and conform as specified in Table 7.8, or

(b) The dynamic types of *variable* and *expr* are the same derived type with the same type parameter values and *variable* is not polymorphic.

[Editor: Move table 7.8 to here.]

[Editor: Delete "and have the same kind type parameter" because it's already said in table 7.8.] 132:16

[Editor: "is of ... parameters" ⇒ "and *expr* are of derived type".] 132:19-22

[Editor: Delete (because conformance is now more spread out into 7.5.1.3).] 132:25

[Editor: Move (including Note 7.37) to [134:4-].] 133:2-13+2

[Editor: After making changes noted above for [132:5], move [132:3-6] to here. Then, after making changes noted below for [135:7-11] and in section 6 for [135:15], move [135:4-19] to here (because the moved stuff is not about interpretation).] 135:1+

(2) Either 135:7-11

(a) A generic interface (12.3.2.1) provides the subroutine with a *generic-spec* of ASSIGNMENT (=), or

(b) There is a type-bound generic binding (4.5.1.5) in the declared type of x_1 or x_2 with a *generic-spec* of ASSIGNMENT (=) and there is a corresponding binding to the subroutine in the dynamic type of x_1 or x_2, respectively,

(3) The types of d_1 and d_2 are compatible with the dynamic types of x_1 and x_2, respectively,

5 Unresolved issue 334

[Editor: Delete unresolved issue 334 note.] 137:15+1ff

[Editor: "and type parameters" ⇒ "; corresponding type parameters shall either both be deferred or both have the same value"]. 137:24

6 Unresolved issue 335

[Editor: Insert ", x_1 and x_2 are conformable," after "elemental".] 129:27

[Editor: Delete unresolved issue 335 note.] 129:28+1ff

[Editor: Insert ", x_1 and x_2 are conformable," after "elemental".] 135:15