Subject: Comments on section 7
From: Van Snyder

1 Edits

Edits refer to 01-007r1. 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 immediately following text, while a page and line number followed by + (-) indicates that immediately following text is to be inserted after (before) the indicated line. Remarks are noted in the margin, or appear between [and] in the text.

[There are two kinds of assignment statements – "value" assignment and pointer assignment – and they are both discussed in this section. Editor: "statement" ⇒ "statements".] 111:3

[The constraint appears to say that one can define "==" for ones own derived type objects, but not ".EQ.". I understand that this is really saying that an overloading of an intrisic operator won't have the same precedence as a defined operator, but that's a subtle point that may be lost on a reader who is not experienced with the idioms used to construct the standard. It would be helpful to have a note to explain this constraint.] 112:28+

NOTE 7.4$\frac{1}{2}$

It is possible to define the interpretation of an intrinsic operator for derived types. Such definition does not, however, change the precedence of the intrinsic operator to be the same as the precedence of a defined operator.

---

(4) An actual argument if the associated dummy argument is not a pointer. 117:23+

[Editor: "when" ⇒ "if" (or "where").] 118:1

[Editor: "is" ⇒ "are".] 118:3

[Editor: There appears to be an extra blank before "is".] 119:5

[Editor: Insert "or dummy procedure pointer" after "procedure".] 119:43

[The sentence "Raising a negative-valued primary to a real power is prohibited" could be interpreted to allow `(-1.5) ** 2.0`, since the exponent is mathematically an integer. Editor: "real power" ⇒ "power of real type".] 122:18

[Editor: "A" ⇒ "An accessible".] 130:35

[Editor: "A" ⇒ "An accessible".] 131:5

[Editor: "which" ⇒ "that".] 131:16

2 Curiously inconsistent organization

2.1 Types and type parameters of operands

The requirements on the types and kind type parameters of operands are discussed in a curiously inconsistent way.

The requirement that the kind type parameters of character concatenation shall be the same is described at least three times – at [115:38] (which is actually two lines), [116:8-9] and [128:13-

14]. Interestingly, most other restrictions on kind type parameters are constraints, but there isn't one concerning the kinds of characters involved in a concatenation operation associated with syntax rules R710-711 at [113:22].

The one at [115:38] is in a note, but the other two are normative.

The requirement that the kind type parameters of characters compared by the intrinsic relational operations shall be the same are also described at least three times – at [115:38] (which is actually two lines), [116:11] and [128:38-39]. Again, the usual place for such a restriction is in a constraint, but there isn't one associated with syntax rules R712-713 at [113:32-1].

The one at [116:11] is normative, but the other two are in notes.

Notice that the requirements are both expressed normatively in 7.1.2, but one is normative and one is informative in 7.2.[23].

My preferences are

1. Express both of these requirements as constraints,
2. Remove or make a note of "with the same kind type parameters" at [128:13-14].
3. Make the note "two character operands cannot be compared unless they have the same kind type parameter value" at [128:38-39] normative.

The discussion of how operand types and type parameters are converted in numeric intrinsic operations is in 7.1.2, at [116:1-4]. The discussion of how operand types and type parameters are converted in relational operations is in 7.2.3, at [129:22-25].

It would be better to discuss both of these conversions in 7.1.2, or to move the one that is in 7.1.2 to be in 7.2.1.

2.2 Restricted expression

The term "restricted expression" is introduced for the purpose of defining "specification expression" in 7.1.6. It is not used anywhere other than for this purpose. Such a device is not needed to define "initialization expression" in 7.1.7.

[Editor: Move to [119:34+].]	119:7
[Editor: "restricted" ⇒ "specification" and move the constraint to [119:34+] (after the syntax rule moved by the previous edit).]	119:8
[Editor: Combine with the paragraph ending at [119:6]; "A restricted expression" ⇒ "it".]	119:9
[Editor: "restricted" ⇒ "specification" thrice.]	119:17,18,20
[Editor: "restricted" ⇒ "specification" thrice.]	119:25,27,28
[Editor: "restricted" ⇒ "specification" thrice.]	119:31,32,34

3 Potential problems with no edits offered

Do we want to mention WHERE and FORALL in the introduction to this section?	111:2-3
What if the dummy argument has assumed or deferred type parameters? At [99:40-41] and [309:34-36], type parameter inquiry is only proscribed for deferred parameters of disassociated	117:40

pointers or unallocated allocatables, so inquiring about assumed type parameters of dummy arguments is presumably OK. In order to do this, it seems the optional argument of NULL ought to be required. This is also the topic of interpretation request 19, at least in the context of assumed character length. Make sure this is updated if interppretation request 19 changes this. One way is to replace "in" by "either in" at [118:2] and add "or corresponding to a dummy argument that has assumed or deferred type parameters" at the end of the sentence at [118:3].

The term "base object" appears to be defined only for structures. 119:11-14

Is the prohibition against dummy procedures too strong? Couldn't it be against nonoptional dummy procedures, together with a prohibition against having actual arguments associated with them when the function is used in a specification expression? 119:43

[78:33-38] and [85:10-13] are similar but not identical. The difference isn't entirely accounted for by the fact that one is in an attribute description, and the other is in a statement description. This is perhaps another illustration that saying something twice is an opportunity to get it wrong, or at least incomplete (twice). Consolidate them, removing the duplication, and move them to 7.1.7. 7.1.7 120-122

These uses of "When" are probably correct, but we need to think about them. 123:12,21

Tables 7.6 and 7.7 are redundant. We don't need both of them. 130:9-24

Does this intentionally contradict 14.1.2.4.1, which says that an interface with an argument of the correct type, type parameters and rank takes precedence over an elemental interface? If so, what happens if both (a) and (b) are satisfied? 130:39-41

Does this intentionally contradict 14.1.2.4.1, which says that an interface with arguments of the correct type, type parameters and rank takes precedence over an elemental interface? If so, what happens if both (a) and (b) are satisfied? 131:10-13

The entire subclause can be deduced from 7.1.1. It should be a note. 7.4

I suggest: 133:23+

Constraint: In the case of intrinsic assignment, the *variable* and *expr* shall have the same rank or the *expr* shall be a scalar.

Constraint: In the case of intrinsic assignment, the types and kind type parameters of *variable* and *expr* shall conform according to the rules in table 7.9.

Put table 7.9 here.

This paragraph's title is "Intrinsic assignment conformance rules" so we don't need "for an intrinsic assignment statement" again. The part about "rules of Table 7.9" should be a constraint (see edit proposed for [133:23+] above). Replace by "The *variable* and *expr* shall conform in shape. If *variable* is of derived type, corresponding type parameters of *variable* and *expr* shall have the same values." 134:20-22

If these changes are not accepted, at least move table 7.9 to [134:22+]. Also see 00-318 and 01-103r2. **Malcolm doesn't like** these.

The description of the absence of defined assignment is incorrect. It would be possible to enumerate the conditions here (e.g. having to do with rank), but they are already spelled out in 7.5.1.6. This could be corrected by, for example, "generic ... parameters" ⇒ "defined assignment, as specified in 7.5.1.6". This results, however, in a circular definition if [134:15-18] isn't adjusted as well. It would be a bit of surgery, but it would work to specify first what is defined assignment, and then specify that intrinsic assignment isn't defined assignment. 134:2-3

Is it necessary to evaluate all expressions within *variable* if, e.g., one of the dimension expressions 135:4

is zero?

Does this intentionally contradict 14.1.2.4.1, which says that an interface with arguments of the correct type, type parameters and rank takes precedence over an elemental interface? If so, what happens if both (a) and (b) are satisfied?	136:40-44
It would be simpler to require that *target* not be a pointer with undefined association status. We don't allow the RHS of an intrinsic assignment to be undefined; why should we allow it in the case of pointer assignment?	138:6-7,16-17
Either we should mention both procedure references and defined assignment statements, or neither of them.	138:35