To: /A subgroup
Subject: Comments on section 7
From: Van Snyder
References: 01-204r1

This paper contains remarks from 01-204r1, with page and line numbers updated to refer to 01-007r1.

1 Curiously inconsistent organization

1.1 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 [111:38] (which is actually two lines), [112:8-9] and [123:13-14]. 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 [109:21-22].

The one at [111:38] 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 [111:38] (which is actually two lines), [112:11] and [123:38-39]. Again, the usual place for such a restriction is in a constraint, but there isn't one associated with syntax rules R712-713 at [109:32-1].

The one at [112:11] 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 [123:13-14].
3. Make the note "two character operands cannot be compared unless they have the same kind type parameter value" at [123:38-39] normative.

The discussion of how operand types and type parameters are converted in numeric intrinsic operations is in 7.1.2, at [112:1-4]. The discussion of how operand types and type parameters are converted in relational operations is in 7.2.3, at [124:22-25].

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 Potential problems (or opportunities) with no edits offered

Do we want to mention WHERE and FORALL in the introduction to this section?	107:2-3

(9) A reference to a specification function where each actual argument associated with a dummy data object is a restricted expression and each procedure associated with a dummy procedure argument is a specification function. 114:44

These uses of "When" are probably correct, but we need to think about them.	118:26,35
Tables 7.6 can be derived (without much trouble) from table 7.5. We don't need both of them.	125:8-24
The entire subclause can be deduced from 7.1.1. It should be a note.	7.4
I suggest: C714$\frac{1}{3}$ In the case of intrinsic assignment, the *variable* and *expr* shall have the same rank or the *expr* shall be a scalar. C714$\frac{2}{3}$ In the case of intrinsic assignment, the types and kind type parameters of *variable* and *expr* shall conform according to the rules in table 7.8. Put table 7.8 here.	129:23+
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 7.8" should be a constraint (see edit proposed for [129:23+] 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." If these changes are not accepted, at least move table 7.8 to [129:23+]. Also see 00-318, 01-103r2 and 01-204r1. **Malcolm doesn't like** these.	130:21-23
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 [130:16-19] 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.	130:3-4
Is it necessary to evaluate *all* expressions within *variable* if, e.g., one of the dimension expressions is zero?	131:4