Subject: Issue 211, more work on derived types and constructors
From: Van Snyder
References: 00-276 00-251 00-275

1 Introduction

Subclause **4.5.6 Construction of derived-type values** doesn't work for extended types. This paper uses the term "component order" defined in paper 00-275 to eliminate the terms "flattened form" and "nested form" that are the subject of issue 211.

The paragraph at [57:16-21] explains that a *type-param-value* is "assigned to the ... type parameter." This is true only in the case that the *type-param-value* is an expression. It is not true in the case that the *type-param-value* is an asterisk or a colon – at least not without a lot of wriggling that simply isn't there. It's better to say here only how the *type-param-value*s correspond to the type parameters.

The explanation of how component values are constructed from the values in the constructor uses the phrase "converted according to the rules of intrinsic assignment to a value that agrees in type and type parameters with the corresponding component ... the shape of the expression shall conform with the shape of the component." Since this doesn't say it does intrinsic assignment, or that it works as if by intrinsic assignment, there's some question how it handles pointer and allocatable components of a derived type component value. This paper explicitly says "as if by intrinsic assignment."

The paragraph on constructing a value for an allocatable component is silent concerning type parameters, especially deferred parameters, and includes the phrase "any other expression that evaluates to an array" at [58:31] – a holdover from the days when the only allocatable entities were arrays.

This paper revises the discussion of construction of values for allocatable components to include deferred type parameters, and so as not to depend on "arrayness" to get "allocatability."

2 Edits

Edits refer to 00-007r2. 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 for the editor are noted in the margin, or appear between [and] in the text.

An **ancestor component** is the parent component or an ancestor component of the parent component. 54:15+ Same ¶

R451 *component-spec* **is** [*keyword* =] *component-constructor* 56:42-57:4
R451a *component-constructor* **is** *expr*
or *target*

Constraint: Every *type-param-value* within the *derived-type-spec* shall be a *scalar-int-expr*.

Constraint: No component of the type shall have more than one corresponding *component-spec*.

Constraint: There shall be a *component-spec* that corresponds to each component that does not have default initialization.

Constraint: If a *component-constructor* corresponds to a nonpointer nonallocatable component, its type, kind type parameters and rank shall conform to those of the corresponding component in the same way that the *expr* shall conform to the *variable* in an intrinsic assignment (7.5.1.4).

Constraint: If a *component-constructor* corresponds to an allocatable component, the *target* shall be a reference to the intrinsic function NULL() with no arguments, or the type and type parameters of the *component-constructor* shall conform to those of the corresponding component in the same way that the *expr* shall conform to the *variable* in an intrinsic assignment (7.5.1.4), the rank of the *component-constructor* shall be the same as the rank of the component, and every deferred type parameter of the *component-constructor* shall correspond to a deferred type parameter of the component. 57:16-21

J3 internal note

Unresolved integration issue xxx The last two of the above constraints refer to intrinsic assignment, wherein the referenced requirements on type, kind type parameters and rank are not constraints.

Constraint: If a *component-constructor* corresponds to a component that has the POINTER attribute, the *target* shall satisfy the same constraints as required of a *target* in a pointer assignment statement (7.5.2) in which the component is the *pointer-object*.

Each *type-param-value* within the *derived-type-spec* (4.5.5) is assigned to the corresponding type parameter after being converted according to the rules for an intrinsic assignment statement (7.5.1.5) in which the type parameter is the *variable* and the *component-constructor* is the *expr*.

If the first *component-spec* does not include a component name keyword and has the same type as an ancestor component, it corresponds directly to the ancestor component having the same type, and therefore indirectly to the components of that type. Each succeeding *component-constructor* that does not include a keyword corresponds to a succeeding component after the last component of that ancestor component, in component order (4.5.4).

Otherwise if the first *component-spec* does not include a component name keyword it corresponds to the first component in component order, and each succeeding *component-constructor* that does not include a keyword corresponds to a succeeding component, in component order (4.5.4).

If a component name keyword is present, the *component-constructor* corresponds directly to the component named by the keyword.

If a *component-constructor* corresponds to a nonpointer nonallocatable component, its array bounds if any and its type parameters shall conform to those of the corresponding component in the same way that the *expr* shall conform to the *variable* in an intrinsic assignment. The value of the *component-constructor* is assigned to the component after being converted according to the rules for an intrinsic assignment statement (7.5.1.5) in which the *component-constructor* is the *expr* and the component is the *variable*.

If a *component-constructor* corresponds to a pointer component, the *target* shall satisfy the same requirements as the *target* in a pointer assignment in which the component is the *pointer-object*, and the *target* is assigned to the component as if the *target* and component were the *target* and *pointer-object* in a pointer assignment (7.5.2).

[Editor: "*expr*" ⇒ "*component-constructor*".] 57:22

[Editor: Delete. This also deletes unresolved issue note 211.] 57:24-32

[Remove the term "nested form" from the example.] 57:39-44

```
COLOR_POINT( PV, 3 )                    ! Available even if TYPE(POINT)
                                        ! has private components.
COLOR_POINT( POINT(1.0, 2.0), 3 )       ! Requires the components of
                                        ! TYPE(POINT) to be accessible.
COLOR_POINT( 1.0, 2.0, 3.0 )            ! Requires the components of
                                        ! TYPE(POINT) to be accessible.
```

[Editor: Delete – superceded by last paragraph of edit for [57:16-21] above.] 58:12-14

If a *component-constructor* corresponds to an allocatable component, it shall 58:25

[Editor: "expression" ⇒ "*component-constructor*".] 58:27

[Editor: Delete "of the constructor".] 58:28

the *component-constructor* is an allocatable entity, the corresponding component has the same allocation status as that allocatable entity and, if it is allocated, the same bounds (if any) and value. Otherwise the component corresponding to the *component-constructor* is allocated, with the same bounds if the *component-constructor* is an array, and it has the same value. 58:29-33

ancestor component (4.5.3.1): the parent component or an ancestor component of the parent component. 403:29+