| | |
|---|---|
| Date: | 30 October 1997 |
| To: | J3 |
| From: | Van Snyder |
| Subject: | Edits for procedure pointers |
| References: | 97-147 97-169 97-174r1 97-190 97-218r2 (syntax) |

**Notes to /data – Questions, and things I've done since the meeting:**

The apparent reason to prohibit non-intrinsic elemental procedures to be actual arguments is that some implementors might put the loop at the call, and some might produce *two* procedures for each elemental procedure definition – one that takes a descriptor, and has a loop inside, to be used in elemental circumstances, and one that doesn't take a descriptor, doesn't have a loop inside, and that's used in scalar circumstances. This doesn't seem to affect either pointer association or argument association – the correct procedure could be associated so long as elemental-ness is required to match. This requires a constraint that if an elemental non-intrinsic is a *target* (or actual argument), the *pointer-object* (or called procedure) must have explicit interface (or at least a note explaining that this is implied by the requirment for elemental-ness to match).

Are elemental procedure pointers OK? If so, the third constraint after R519C (in [59:12+]), and the one at [113:5+], need to be deleted, and the rules about pointer assignment should allow assigning an elemental intrinsic procedure to a non-elemental pointer. The same logic applies to non-intrinsic elemental actual arguments, but we shouldn't change those rules now, even if there's not really a technical problem, without a full J3 vote.

Should the two sentences beginning "If *proc-interface* consists..." that immediately precede *Note 5.b* be moved to be after R519B and changed to be constraints?

"type" changed to "type and kind type parameters" immediately before *Note 5.b*.

Deleted two constraints that were earlier added at [204:31+] and [204:33+] because they should have said "disassociated" instead of "undefined," but *variable* references are already covered by "thou shalt not dereference NULL or undefined pointers" rules.

**End of notes to /data.**

Changes are w.r.t. 97-007r1. Page and line numbers are displayed in the margin.

| | | |
|---|---|---|
| Add "(5.1.2.10)"after "EXTERNAL attribute" | | [2:36] |
| [part of R207] | **or** *procedure-declaration-stmt* | [10:9+] |
| [part of R425] | **or** PROCEDURE ( [ *proc-interface* ] ) ■<br>■ , POINTER :: *proc-identity-list* | [38:40+] |

A name is specified to have the **EXTERNAL attribute** if it appears in a type declaration statement having the EXTERNAL attribute specification, an EXTERNAL statement (12.3.2.2), a PROCEDURE declaration statement (5.2), or as a specific procedure name in an interface body [58:28-30]

(12.3.2.1).

If a name that has the EXTERNAL attribute also has an explicit type or appears as a function name in a function reference (12.4) or interface body (12.3.2.1) it is an external function or dummy function.

If a name has the EXTERNAL attribute or is the name of an accessible module procedure it may be used as an actual argument, as a procedure name in a procedure reference (12.4), or as the target of a procedure pointer assignment (7.5.2).

---

## 5.2 Procedure declaration statement [59:12+]

[Note to editor: re-number subsequent sections]

A procedure declaration statement declares a procedure pointer, a dummy procedure, or an external procedure.

[Note to editor: Syntax rule numbers are to be inserted between present rules R519 and R520.]

R519A *procedure-declaration-stmt* **is** PROCEDURE ( [ *proc-interface* ] ) ■
■ [[, *attr-spec* ] ... ::] *proc-identity-list*

R519B *proc-interface* **is** *abstract-interface-name*
**or** *type-spec*

Constraint: *abstract-interface-name* must be the name of an abstract interface (12.3.2.1.3)

R519C *proc-identity* **is** name [ => NULL() ]

Constraint: No attributes other than *access-spec*, INTENT, POINTER, SAVE and OPTIONAL shall be specified for *proc-identity.*

Constraint: If a *proc-identity* has an accessibility attribute, or an INTENT attribute, or the SAVE attribute, the POINTER attribute shall also be specified for that *proc-identity.*

Constraint: If *proc-identity* has the POINTER attribute, *proc-interface* shall not describe an elemental procedure.

Constraint: If => appears in *proc-identity*, name shall have the POINTER attribute.

Constraint: *proc-identity* shall not be the name of an accessible module procedure.

The following table indicates the category of entity named by *proc-identity*:

| Is the POINTER attribute specified for *proc-identity*? | Is *proc-identity* the name of a dummy argument? | Then *proc-identity* is: |
|---|---|---|
| Yes | Yes or No | Procedure pointer |
| No | Yes | Dummy procedure |
| No | No | External procedure |

Appearance of *proc-identity* in a PROCEDURE statement specifies the EXTERNAL attribute (5.1.2.10) for that name.

Appearance of an intrinsic procedure name in a PROCEDURE statement causes that name to become the name of a procedure pointer or external procedure, and thus the intrinsic procedure of the same name is not available in the scoping unit.

If *proc-interface* consists of *abstract-interface-name*, *proc-identity* has an explicit interface, and shall be used only to identify procedures having characteristics given by the named abstract interface.

If *proc-interface* consists of *type-spec*, *proc-identity* shall be used only to identify functions that have the result type and kind type parameters given by *type-spec*.

In contrast to the EXTERNAL statement, it is not possible to use a PROCEDURE statement to identify a BLOCK DATA subprogram. *Note 5.b*

*Note 5.c*

```
! Using abstract procedure definitions in Note 12.x:
!-- Some external or dummy procedures with explicit interface.
PROCEDURE (REAL_FUNC) :: BESSEL, GAMMA
PROCEDURE (SUB) :: PRINT_REAL

!-- Some procedure pointers with explicit interface,
!-- one initialized to NULL().
PROCEDURE (REAL_FUNC), POINTER :: P, R => NULL()
PROCEDURE (REAL_FUNC), POINTER :: PTR_TO_GAMMA

!-- A derived type with a procedure pointer component ...
TYPE STRUCT_TYPE
  PROCEDURE (REAL_FUNC), POINTER :: COMPONENT
END TYPE STRUCT_TYPE

!-- ... and a variable of that type.
TYPE(STRUCT_TYPE) :: STRUCT

!-- An external or dummy function with implicit interface
PROCEDURE (REAL) :: PSI
```

---

[sentence that begins "This also..."] [59:17]
.... This also applies to PROCEDURE, EXTERNAL, and INTRINSIC statements.

---

Constraint: Each *allocate-object* shall be a non-procedure pointer or an allocatable array. [81:36]

---

Constraint: Each *allocate-object* shall be a non-procedure pointer or an allocatable array. [84:23]

---

Constraint: If *pointer-object* is a data object, *variable* shall have the TARGET or POINTER attribute. [113:3-4]

Constraint: If *pointer-object* is a data object or a procedure pointer that has a result type, *target* shall be of the same type, kind type parameters, and rank as the pointer.

Constraint: The *target* shall not be a non-intrinsic elemental procedure. [113:5+]

---

[Note to editor: Delete the first sentence (It's covered by a constraint) and replace it with this stuff, then start a new paragraph with "If the *target* is not ...."] [113:7]

If *pointer-object* is a procedure pointer, *target* shall be the specific name of an external, module, dummy, or intrinsic procedure, a procedure pointer, a reference to a function that returns a procedure pointer, or a reference to the NULL intrinsic function. The only intrinsic procedures permitted are those listed in 13.13 and not marked with a bullet (•). If the specific intrinsic procedure name is also a generic name, only the specific intrinsic procedure is associated with *pointer-object*.

If *pointer-object* is a procedure pointer that has an explicit interface, the characteristics listed in 12.2 shall be the same for *pointer-object* and *target*, except that a *target* having an interface to a pure procedure may be assigned to a *pointer-object* having an interface to a procedure that is not pure, and a *target* that is an elemental intrinsic procedure may be assigned to a *pointer-object*.

---

[inside note 7.46] [113:34+]

```
! P is a procedure pointer and BESSEL is a
! procedure with compatible interface (see note 5b)
P => BESSEL

! Likewise for a structure component
STRUCT % COMPONENT => BESSEL
```

---

Constraint: A variable that is an input item shall not be a procedure pointer. [151:12+]

---

Constraint: An expression that is an output item shall not have a value that is a procedure pointer. [151:14+]

---

Add ", a dummy procedure pointer" after "procedure". [198:8]

---

**12.2.1.2 Characteristics of dummy procedures and dummy procedure pointers.** [198:18]

---

[Remove the word MODULE.] [199:33]

---

[replace second line of R1202] **or** *procedure-stmt* [199:40]

---

[add a line to R1203] **or** INTERFACE PROCEDURE () [199:41+]

Constraint: If *interface-stmt* is INTERFACE PROCEDURE(), each *interface-specification* shall be an *interface-body*.

---

[replace R1206] [200:8]

R1206 *procedure-stmt* **is** [ MODULE ] PROCEDURE *procedure-name-list*

---

[Delete] [200:14-16]

Constraint: A *procedure-name* shall have an explicit interface and shall refer to a procedure pointer, external procedure, dummy procedure, or accessible module procedure. [200:22-24]

Constraint: If MODULE appears, *procedure-name* shall refer to an accessible module procedure.

Constraint: A *procedure-stmt* is allowed only if the interface block has a *generic-spec*.

Constraint: In all interface blocks that have the same generic identifier in any specification part, a *procedure-name* shall not be specified more than once in a *procedure-stmt*, nor be the same as a specific procedure name that appears in a *function-stmt* or *subroutine-stmt*.

---

[after note 12.6] [201:46+]

An interface block introduced by INTERFACE PROCEDURE() is an **abstract interface block**; it defines **abstract interfaces** (12.3.2.1.3).

---

[after note 12.9] [203:18+]

**12.3.2.1.3 Abstract interfaces**

The name given in a *subroutine-stmt* or *function-stmt* in an abstract interface block is the name of an abstract interface. Abstract interface names are in the same class as type names (14.1.2).

*Note 12.x*

```
! Example abstract interfaces.
INTERFACE PROCEDURE()
  ! REAL_FUNC IS ABSTRACT INTERFACE NAME
  FUNCTION REAL_FUNC (X)
    REAL, INTENT(IN) :: X
    REAL :: REAL_FUNC
  END FUNCTION REAL_FUNC
  ! SUB IS ABSTRACT INTERFACE NAME
  SUBROUTINE SUB (X)
    REAL, INTENT(IN) :: X
  END SUBROUTINE SUB
END INTERFACE
```

REAL_FUNC and SUB can be used as *abstract-interface-name* in a procedure declaration statement.

[End note]

---

[Move 203:33-34 here.] [203:28-29]

[Move notes 12.10 and 12.11 (203:39-44) here].

---

It is generally better practice to declare an external or dummy procedure using a PROCEDURE declaration statement, as this allows the interface to be specified in the same place. [203:33-44] *Note 12.y*

---

[Change second “procedure” to “procedure or procedure pointer”.] [204:28]

---

[add a line to R1210] **or** *variable* ( [ *actual-arg-spec-list* ] ) [204:31+]

Constraint: *variable* shall be a procedure pointer, or a structure component that is a procedure pointer.

Constraint: A reference to *variable* shall not appear as a subroutine reference.

---

[add a line to R1211] **or** CALL *variable* ( [ *actual-arg-spec-list* ] ) [204:33+]

Constraint: *variable* shall be a procedure pointer, or a structure component that is a procedure pointer.

---

Examples of procedure reference using procedure pointers. [205:23+]

*Note 12.15a*

```
P => BESSEL
WRITE (*, *) P(2.5)        ! -- BESSEL(2.5)

S => PRINT_REAL
IF (ASSOCIATED(S)) CALL S(3.14)
```

---

### 12.4.1.2 Actual arguments associated with dummy prodedures or dummy procedure pointers [208:16:30]

If the dummy argument is a procedure pointer, the associated actual argument shall be a procedure pointer, a reference to a function that returns a procedure pointer, or a reference to the NULL intrinsic function.

If the dummy argument is a dummy procedure, the associated actual argument shall be the specific name of an external, module, dummy, or intrinsic procedure, a procedure pointer, or a reference to a function that returns a procedure pointer. The only intrinsic procedures permitted are those listed in 13.13 and not marked with a bullet (•). If the specific name is also a generic name, only the specific procedure is associated with the dummy argument.

If an external procedure name or a dummy procedure name is used as an actual argument, it shall have the EXTERNAL attribute.

If the dummy argument has an explicit interface, the characteristics listed in 12.2 shall be the same for the associated actual argument and the corresponding dummy argument, except that an actual argument having an interface to a pure procedure may be associated with a dummy argument having an interface to a procedure that is not pure, and an actual argument having an interface to an elemental intrinsic procedure may be associated with a dummy argument having an interface to a procedure that is not elemental.

If the dummy argument has an implicit interface and either the name of the dummy argument is explicitly typed or the dummy argument is referenced as a function, the dummy argument shall not be referenced as a subroutine, and the actual argument shall have the same result type as the dummy argument.

If the dummy argument has an implicit interface, and a reference to the dummy argument appears as a subroutine reference, the actual argument shall not be a function, or a procedure pointer that has a result type.

| | | |
|---|---|---|
| POINTER | shall be a pointer and may be of any type, or a procedure pointer. Its pointer association status shall not be undefined. | [238:17-21] |
| TARGET (optional) | shall be a pointer or target. If POINTER is a data entity, TARGET shall have the same type, type parameters and rank as POINTER. If POINTER is a procedure pointer, TARGET shall be a procedure, or procedure pointer, for which pointer assignment (7.5.2) to POINTER would be permitted. If TARGET is a pointer then its association status shall not be undefined. | |

| | | |
|---|---|---|
| Case (ii): | If POINTER is a procedure pointer and TARGET is an external procedure, module procedure, intrinsic procedure, or dummy procedure, the result is true if POINTER is associated with TARGET. | [238:25+] |
| Case (iii): | If POINTER is a procedure pointer and TARGET is a procedure pointer, the result is true if POINTER and TARGET are associated with the same procedure. | |

[Note to editor: re-number subsequent cases.]