Subject: FINAL procedures as type-bound procedures
From: Van Snyder
References: 99-108, 00-138 00-170 00-186

1 Introduction

This paper is based on 00-138, which was available but not discussed at meeting 152. The syntax is slightly different from what was adopted for final procedures in 99-108, and slightly different from what was proposed in 00-138. There is more work to be done for final procedures, especially specifying the order in which objects cease to exist, and therefore the order in which their final procedures are executed.

This paper depends on paper 00-186. It should be processed, if at all, after that paper passes, or another one that specifies how kind type parameters interact with type-bound procedure invocation passes.

2 Edits

Edits refer to 00-007. 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 immediately following text, while a page and line number followed by + indicates that immediately following text is to be inserted after the indicated line. Remarks for the editor are noted in the margin, or appear between [and] in the text.for finalization

---

or PROCEDURE (*proc-interface-name*), ■ 44:18+
■ FINAL => NULL()
or PROCEDURE, FINAL => *final-binding-list*

[Editor: Add to the constraint:] 44:25
If *proc-interface-name* and FINAL are both specified, the interface shall specify a subroutine that has one dummy argument with a declared type of *type-name* and that is polymorphic if and only if *type-name* is extensible. This argument shall not have INTENT(OUT), nor have the ALLOCATABLE, ASYNCHRONOUS, OPTIONAL, POINTER, VALUE or VOLATILE attribute. If the dummy argument is an array it shall have assumed shape. All nonkind parameters of the dummy argument shall be assumed.

Change "VALUE" to "INTENT(VALUE)" and put it before OPTIONAL if paper 00-170 passes. *Editor*

---

final-binding	**is** *procedure-name*	44:31+
	or NULL(*procedure-name*)	
	or NULL(*procedure-pointer-name*)	

Constraint: The *procedure-name* shall be the name of an accessible module procedure or external procedure. The *procedure-pointer-name* shall be the name of an accessible procedure pointer. The procedure or procedure pointer shall be a subroutine with an explicit interface having one dummy argument with a declared type of *type-name* and that is polymorphic if and only if *type-name* is extensible. This argument shall not have the ALLOCATABLE, ASYNCHRONOUS, OPTIONAL, INTENT(OUT), POINTER, VALUE or VOLATILE attribute. If the dummy argument is an array it shall have assumed shape. All nonkind parameters of the dummy argument shall be assumed.

Change "VALUE" to "INTENT(VALUE)" and put it before OPTIONAL if paper 00-170 passes. *Editor*

Constraint: If several subroutines are bound to the type with *binding-attr* FINAL, they shall be distinguished according to the rules for unambiguous procedure references (14.1.2.3).

4.5.1.5.1 Final subroutine

49:30+

A procedure binding that specifies FINAL is a **final subroutine** for objects of the type. The set of final subroutines that are bound to the type or that are inherited (4.5.3.1) from the parent type and not overriden (4.5.3.2) is a generic interface.

If any final subroutines are specified for a type and set of kind type parameters, at least one of them shall have a scalar dummy argument.

A final subroutine may be elemental.

When any object is deallocated (6.3.3, 6.3.3.1) or becomes undefined by the events specified by items (3) or (13)(c) in 14.7.6, if a final subroutine is selected as specified in 14.1.2.4.2$\frac{2}{3}$, it is invoked with the object as its actual argument. If the subroutine causes other objects of the same type and kind type parameters to be deallocated or to become undefined by the events specified by items (3) or (13)(c) in 14.7.6, it shall be recursive.

Immediately following execution of a final subroutine, if it overrides (4.5.3.2) one, the overridden final subroutine is invoked, with the object as its actual argument. This process is repeated until no further final subroutine is available.

Immediately following this process, the object becomes deallocated or undefined.

If a procedure binding that specifies FINAL (4.5.1.6) cannot be distinguished from one inherited (4.5.3.1) from the parent type according to the rules for unambiguous procedure references (14.1.2.3), it overrides that binding. 55:0-

---

346:22++ (after material inserted at this point by 00-186)

14.1.2.4.2$\frac{2}{3}$ Resolving final procedure references

The *effective set of final subroutines* for a type is the set of final subroutines inherited from the parent of the type, minus the overridden ones, plus the ones declared in the type. Each subroutine in an effective set has a corresponding one in each effective set for each extension type – either the same subroutine or one that overrides it. Each effective set of final subroutines is a generic interface.

A final subroutine for an object is selected by:

(1) At most one subroutine is selected from the effective set of final subroutines for the declared type of the object, according to the generic resolution rules (14.1.2.4.1).

(2) If a subroutine is selected in step (1), the reference is to the subroutine from the effective set, for the dynamic type of the object, that corresponds to the subroutine selected in step (1).

If the reference is to a deferred binding, an error condition occurs.