Subject: Specifications, syntax and semantics for type-bound operators
From: Van Snyder

1 Specifications

1. Bind operators and assignment to types, creating or extending a generic interface.
2. Perform generic resolution using the declared types of the operands.
3. Dispatch to an overriding binding using the dynamic type.

2 Syntax

Bind operators and assignment to a type by using

proc-binding-stmt	**is**	GENERIC [, NON_OVERRIDABLE] :: ■
		■ *binding-generic-spec* => *binding-list*
	or	GENERIC (*proc-interface-name*) ■
		■ [, NON_OVERRIDABLE] :: ■
		■ *binding-generic-spec* => NULL()
binding-generic-spec	**is**	OPERATOR (*defined-operator*)
	or	ASSIGNMENT (=)

3 Semantics

All of the procedures and NULL() bindings for assignment or a particular operator constitute a type-bound generic interface. All of the accessible type-bound generic interfaces, and accessible free generic interfaces, for assignment or for a particular operator, constitute a single generic interface. The bindings and specific procedures of free generic interfaces shall be distinguishable by the rules in subclause 14.1.2.3.

Each binding shall have at least one scalar nonpointer nonallocatable dummy variable of the type being defined. All of the arguments of the type being defined shall be polymorphic if the type is extensible.

All of the polymorphic arguments of a binding shall have the same declared type.

The previous two requirements prevent the following ambiguity: Suppose a procedure P has polymorphic arguments of two different extensible types, say T1 and T2. Suppose it is bound to both of those types by assignment. Suppose those bindings are separately overridden by different procedures, say PA and PB in extensions of the two types, say T1A and T2B. That is, PA has arguments of type T1A and T2, and PB has arguments of types T1 and T2B. Now suppose an object of type T2B is assigned to an object of type T1A. Is PA or PB invoked?

The requirements for arguments shall otherwise be the same as for generic operators or assignment.

A binding overrides one inherited from the parent type if all of the arguments of the type being defined correspond to arguments of the parent type and have the same kind type parameters, and all other arguments have the same type and kind type parameters as corresponding ones. Otherwise it extends the generic interface.

Where a defined operator or assignment appears, all of the accessible generic interfaces for that operator or assignment, both type-bound and free, are considered. A specific procedure is selected, if possible, by using the declared types and kind type parameters of the operands. If the specific procedure thus selected is type-bound, all of the actual arguments that correspond to polymorphic dummy arguments shall have the same dynamic type. If there is an overriding binding for the dynamic type of the operands, the overriding binding is selected.

Requiring the actual arguments to have the same dynamic type also prevents an ambiguity. Suppose a procedure P is bound to a type T by, say, assignment. Suppose T is extended to T1 and T2, and new procedures, say P1 and P2, are bound to T1 and T2, overriding P. Now suppose an object of type T1 is assigned to an object of type T2. Is P1 or P2 invoked?
This ambiguity and the previous one occur because there is no distinguished passed-object dummy argument in the case of assignment or operator binding. It isn't reasonable to specify that dispatching is always dependent on the first argument. Maybe one wants to define assignment from an extensible type to an intrinsic one, for example.

In any case, if a NULL() binding is selected, an error condition occurs.

4 Variation on semantics for generic interfaces

All of the bindings in a type-bound generic interface shall be distinguishable by the rules in subclause 14.1.2.3, and separately, the specific procedures in a free generic interface shall be distinguishable by the rules in subclause 14.1.2.3.

A binding is first sought among accessible type-bound generic interfaces, using the declared type and kind type parameters; if that succeeds, an overriding binding corresponding to the dynamic type is selected; if that fails, a specific procedure is sought from a free generic interface, if any.

Another alternative is similar to the one in the previous paragraph, but free generic interfaces have priority over type-bound ones. This is the way that type-bound user-defined derived-type input/output works. I think that's the wrong way around (see 00-320), but it's what JOR wanted. Maybe we should compound the error for consistency's sake.

5 Variation on semantics for dispatching

If the dummy arguments are polymorphic, their dynamic types are not required to be the same. After a binding is found using the declared types of the operands, if there is an overriding binding for the nearest common ancestor type of the types of the operands, that binding is selected.