Subject: Comments about SELECT TYPE
From: Van Snyder

1 Background

Should type parameters participate in type selection? It seems more reasonable that they should than that they shouldn't. It also seems more reasonable that only kind parameters participate than that both kind and nonkind parameters do. If we do specify kind parameters, we will need a place-holder for nonkind parameters. I suggest either asterisk or colon. Do we want to go so far as to allow ranges or lists of kind parameter values? Ranges could be represented by [low]:[high] (colon alone means "any value"); lists could be represented by array constructors. If type parameters participate in type selection, and we allow ranges or lists of kind parameters in TYPE IN and TYPE IS, we need an entirely separate set of syntax rules parallel to *type-spec* that allows ranges and lists. This would be simpler if we used a van Wijngarten grammar instead of BNF, but now is not the time to switch. 176:31-32

If a type guard is TYPE IS, and the type (sans parameters) is the same as the type of the *type-selector*, all the parameters are available for testing. Similarly, If a type guard is TYPE IN, and the type of *type-selector* is a descendant type of the declared type of the *extensible-type-name*, all the parameters of *extensible-type-name* are available for testing. Furthermore, allowing arbitrary combinations of type parameters in the type guard statements gives a form of conditional compilation, viz.:

```
SELECT TYPE ( FOO )
TYPE IS ( DOUBLE PRECISION )
! special double precision code -- compiling not needed if FOO is REAL
TYPE IS ( REAL )
! special default real code -- compiling not needed if FOO is
! DOUBLE PRECISION
END SELECT
```

In order to work for intrinsic types, the syntax of TYPE IS needs to be changed to something similar to TYPE IS (*type-spec*). If we want to allow ranges and lists of parameters, we can't use *type-spec* as-is. 176:31-32 again

---

It is necessary to say something about type aliases. The syntax terms that are mentioned in the discussion will depend on whether type parameters participate in type selection. 176:31-32 again

---

It is necessary to specify the type parameters of the associate name. 177:31+

2 Edits

Edits refer to 99-007r1. 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.

R819 *type-guard-stmt* **is** TYPE IS (*type-guard-list*) [*select-construct-name*] 176:31-37

or TYPE IN (*type-guard*) [*select-construct-name*]
or TYPE DEFAULT [*select-construct-name*]

R819a *type-guard* **is** INTEGER [(*type-guard-parameter-list*)]
or REAL [(*type-guard-parameter-list*)]
or DOUBLE PRECISION
or COMPLEX [(*type-guard-parameter-list*)]
or CHARACTER [(*type-guard-parameter-list*)]
or LOGICAL [(*type-guard-parameter-list*)]
or TYPE (*derived-type-guard*)
or TYPE (*type-alias-guard*)

R819b *derived-type-guard* **is** *derived-type-name* [(*type-guard-parameter-list*)]

R819c *type-alias-guard* **is** *type-alias-name* [(*type-guard-parameter-list*)]

Constraint: The *derived-type-name* shall be the name of an accessible derived type.

Constraint: In a TYPE IN *type-guard-stmt*, the *derived-type-name* shall be the name of an extensible type.

R819c *type-guard-parameter* **is** [*keyword* =] [*scalar-int-initialization-expr*] ■
■ [: [*scalar-int-initialization-expr*]]
or [*keyword* =] (/ *scalar-int-initialization-expr-list* /)

Constraint: The *type-guard-parameter* for a nonkind type shall be a colon.

Constraint: Within a given *select-type-construct*, if a type indicated by a *type-name* or by a *type-alias-name* appears more than once in all of the TYPE IS *type-guard-stmts*, the sets of kind parameter values in one appearance shall not all have a nonempty intersection with the sets of kind parameter values in another appearance.

Constraint: Within a given *select-type-construct*, if a type indicated by a *type-name* or by a *type-alias-name* appears more than once in all of the TYPE IN *type-guard-stmts*, the sets of kind parameter values in one appearance shall not all have a nonempty intersection with the sets of kind parameter values in another appearance.

The above two constraints are another example wherein things would be simpler if type aliases denoted new types.

Note to J3

Note 8.9$\frac{1}{2}$

For example, suppose the type T appears in two TYPE IS or TYPE IN *type-guard-stmts*. It is permitted if in one case it has type parameters T(1:2, 3:4), and in the other case it has type parameters T(1:2, 5:6). A case having type parameters T(2:3, 4:5) would conflict with the first case, because the first and second sets of type parameters both have a nonempty intersection.

A *type-guard-parameter* denotes a set of parameter values. The following table shows the set of parameter values denoted by each form of *type-guard-parameter*. 176:39+

Form of *type-guard-parameter*	Set of values
E	the single value given by E
colon	all possible parameter values
: E	all possible parameter values v such that $v \leq E$
E :	all possible parameter values v such that $E \leq v$
E_1 : E_2	all possible parameter values v such that $E_1 \leq v \leq E_2$
(/ *expr-list* /)	the values given by the expressions in *expr-list*

In the absence of a type parameter keyword, a *type-guard-parameter* denotes the set of values for the type parameter occupying the corresponding position of the type definition; that is the first *type-guard-parameter* denotes the set of values for the first parameter of the type, the second *type-guard-parameter* denotes the set of values for the second parameter of the type, etc. If a type parameter keyword is specified, the *type-guard-parameter* denotes the set of values for the type parameter named by the keyword. The values of expressions in a *type-guard-parameter* are not required to be of the same integer kind as the corresponding type parameter.

It is not necessary to specify a *type-guard-parameter* for every kind type parameter. If a *type-guard-parameter* is not specified, the effect is different for intrinsic types and derived types. For intrinsic types the effect is the same as if the default kind were specified. For derived types the effect is the same as if the unspecified parameter were specified by a colon. It is not possible, however, to omit a parameter within a *type-guard-parameter-list* by using two consecutive commas.

If no *type-guard-parameter-list* is specified, the effect is the same as if an empty *type-guard-parameter-list* were specified.

---

1. If the dynamic type of the type selector is the same as one of the types named in a TYPE IS type guard statement, and every kind type parameter of the type selector is within the set of parameter values specified for the corresponding type parameter in the *type-guard-parameter-list* for that type, the block following that statement is executed. 177:11-19

2. Otherwise, if there is exactly one case in which the dynamic type of the type selector is an extension of a type named in a TYPE IN type guard statement, and every kind type parameter of the type selector is within the set of parameter values specified for the corresponding type parameter in the *type-guard-parameter-list* for that type, the block following that statement is executed.

3. Otherwise, if there are several cases in which the dynamic type of the type selector is an extension of a type named in a TYPE IN type guard statement, and every kind type parameter of the type selector is within the set of parameter values specified for the corresponding type parameter in the *type-guard-parameter-list* for that type, one of those cases must specify a type that is an extension of the types specified in the others; the block following that statement is executed.

Step 3 doesn't work if *type-guard-list* is allowed in TYPE IN type guard statements, except by the use of extremely complicated constraints. For example, if TYPE IN statements had types (A,B), (C) and (D,E), and E is an extension of D is an extension of C is an extension of A, then B cannot be allowed to be an extension of E. Ick! For symmetry, maybe it should also be simply *type-guard* in TYPE IS type guard statements.	*Note to J3*

[Editor: A new paragraph.] 177:31+
Within any *type-guard-stmt-block*, the type parameters of the *associate-name* are the same as the type parameters of the *type-selector*.