

Subject: Partial application in interface bodies
 From: Van Snyder
 Reference: 03-258r1, sections 2.3.1

1 Number

2 TBD

3 Title

4 Partial application in interface bodies.

5 Submitted By

6 J3

7 Status

8 For consideration.

9 Basic Functionality

10 Allow generic identifiers to be partial applications, by putting values to be used as actual arguments for
 11 some dummy arguments, either in interface bodies or [module] procedure statements.

12 Rationale

13 I have a sparse matrix package that includes a MatrixAdd function, with interface

```
14 function MatrixAdd ( A, B, Subtract ) result ( Z )
15   type(Matrix_T), intent(in) :: A, B
16   logical, optional, intent(in) :: Subtract
17   type(Matrix_T) :: Z
18 end function MatrixAdd
```

19 The functionality is that it adds $A + B$ unless the `Subtract` argument is present with the value `.true.`,
 20 in which case it subtracts $A - B$.

21 One cannot access this function with a defined operator. One could wrap it with additional functions
 22 that have only two nonoptional arguments, but this increases code bulk. Numerous studies have shown
 23 that the single most reliable predictor of lifetime cost of software is code bulk.

24 7 Estimated Impact

25 Minor.

26 8 Detailed Specification

27 Allow values to be specified for some arguments in an interface block, either in a [module] procedure
 28 statement, or in an interface body. If a `GENERIC` statement is allowed outside of a type definition
 29 (see that proposal), allow to specify values there, too. Only the remaining arguments are visible, as
 30 arguments, when the procedure is accessed by using the *generic-spec*. After the values of some of the
 31 arguments are specified, the remaining arguments shall satisfy the present requirements.

1 8.1 Example

2 It would be useful to be able to declare something like

```
3 interface operator(+)  
4     module procedure MatrixAdd ! or MatrixAdd(subtract=.false.)  
5 end interface  
6 interface operator(-)  
7     module procedure MatrixAdd(subtract=.true.)  
8 end interface
```

9 with the requirement that after specifying values to use for some arguments, in the interface, there
10 remain one or two nonoptional arguments for which values are not specified, and these arguments meet
11 the present requirements for defined-operator interfaces. In the functional programming community, this
12 is called “partial application” or ”Currying” (after Haskell Curry) of the `MatrixAdd` function.

13 The procedure `MatrixAdd` supports several different representations of sparse matrices, and has a lot
14 of analysis to figure out where the nonzeros of the output will be, and what representation to use.
15 There are only a few places where it looks at the `Subtract` argument. It is undesirable to duplicate the
16 code and specialize the two copies for the `Subtract = .true.` and `Subtract = .false.` cases, because
17 that introduces the opportunity to create incorrect inconsistencies between them as a consequence of
18 maintenance.

19 9 History