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

1 Number

TBD

2 Title

Partial application in interface bodies.

3 Submitted By

J3

4 Status

For consideration.

5 Basic Functionality

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

6 Rationale

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

```
function MatrixAdd ( A, B, Subtract ) result ( Z )
  type(Matrix_T), intent(in) :: A, B
  logical, optional, intent(in) :: Subtract
  type(Matrix_T) :: Z
end function MatrixAdd
```

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

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

7 Estimated Impact

Minor.

8 Detailed Specification

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

8.1 Example

It would be useful to be able to declare something like

```
  interface operator(+)
    module procedure MatrixAdd ! or MatrixAdd(subtract=.false.)
  end interface
  interface operator(-)
    module procedure MatrixAdd(subtract=.true.)
  end interface
```

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

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

9 History