Recommendations for the Development of Interval Arithmetic in Fortran 2000

1. Add **square brackets [** and] to the Fortran character set. (There are many reasons, most of them not connected with intervals, why this may be desirable.) This will enable a convenient, mathematically established notation for interval constants.

It is recommended that this extension be mandatory for ALL future implementations of Fortran regardless of whether or not they support intervals.

2. Introduce "interval types" in a way that is **orthogonal** to the existing way of specifying numeric intrinsic types. One way of doing this is to introduce a new type parameter (which could be termed FORM type parameter) in addition to the kind type parameter for REAL - in such a way that it can be extended to other intrinsic types, e.g. COMPLEX and INTEGER. (Please also refer to document WG5 V16.)

This will allow building a mirror image of the specification structure of the existing numeric types for the new interval types (in the long term). Implementation restrictions on the set of admissible kind-parameter values for interval types might be allowed initially, and might be disallowed in a future standard.

An example of the intended kind of syntax is:

REAL (KIND = KIND(O.ODO), [FORM =] INTERVAL)

(This new type parameter could also be used for other purposes, e.g. to specify a type for fuzzy logic.)

The introduction of completely new intrinsic type statements for interval types is viewed as being more controversial, a bigger burden on the language, and not easily extensible.

3. Add the new operators required for interval arithmetic in such a way that they have their natural **precedence. In** particular, the new relational operators should have the same precedence as the existing relationals, and the operators for "intersection" and "interval hull" should have higher precedence than these.

Since this constitutes a change of the language that will potentially change the interpretation of existing programs, it is proposed that specific measures, which could be in the spirit of the following two, be taken to greatly reduce or eliminate the impact of this incompatibility:

a) reduce the likelihood of collisions with operator names in existing code by choosing more explicit, interval-specific names, and/or

b) make new interval operators (and other interval features) available only via special action on the part of the programmer, e.g. when a use statement for an intrinsic interval module is specified.

4. In the hope of resolving the discussions about the two seemingly incompatible **semantic models** of interval arithmetic, we would like to suggest another viewpoint.

Let us define the two models as

a) the **"set-theoretic"** model where all operands/arguments are interpreted as independent sets,

and

b) the **"functional"** model, where operands/arguments which are formally the same variable are interpreted as being dependent sets.

We strongly believe that the fundamental definition of the elementary operators and intrinsic functions should follow the set-theoretic model and thus the runtime semantics of these individual operations should be unambiguously defined. However, we also believe that tighter interval enclosures produced by applying the functional model to whole expressions or sections of code are highly desirable in many cases.

We propose that optimizations (e.g. algebraic simplifications) which are possible under the functional model be explicitly allowed - conceptually during a preprocessing phase - and that the stricter set-theoretic model be applied to the code resulting from this phase.

We also believe that the user should be given the power to require strict interpretation of his/her code according to the set-theoretic model, i.e. it should be possible to prohibit optimizations which are possible under the functional model, at least globally, but possibly also locally by other selective means.