12.13.3. *Boolean functions of intervals.*

The functions listed in §10.6.3 shall be handled in the same way as those in §12.13.1.

If $\mathbb{T}$-version of the function `isMember`$(m, \boldsymbol{x})$ is provided, the number format $\mathbb{F}$ of the argument m should be compatible with $\mathbb{T}$. An implementation may provide several $\mathbb{T}$-versions, with different formats of m. If $\mathbb{T}$ is a 754-conforming type, versions should be provided with the argument m of any supported 754 format of the same radix as $\mathbb{T}$.

12.13.4. *Extended interval comparisons.*

How the operations in §10.6.4 are handled at Level 2 is implementation-defined.

12.13.5. *Exact reduction operations.* An implementation that provides 754-conforming type for the parent format $\mathbb{F}$ should provide an *accumulator format* datatype $A(\mathbb{F})$ associated with the $\mathbb{F}$, and associated operations. An $A(\mathbb{F})$ datum z is capable to represent exactly dot products of vectors of any reasonable length of arbitrary finite $\mathbb{F}$-numbers.

The following operations should be provided.

- `convert` converts from an accumulator format to a floating-point format, or vice versa, or from one accumulator format to another.
- `exactAdd` and `exactSub` adds or subtracts two accumulator or floating-point format operands, of which at least one is an accumulator, giving an accumulator format result.
- `exactFma` computes $z + x * y$ where z has an accumulator format and x, y are of floating-point format, giving an accumulator format result.
- `exactDotProduct`. Let a and b be vectors of length n holding floating-point numbers of format $\mathbb{F}$. Then `exactDotProduct`(a, b) computes $a \cdot b = \sum_{k=1}^{n} a_k b_k$ exactly, giving an accumulator format result.

The result of all operations may be converted if necessary to a specified result format by application of the `convert` operation.

[*Example. The Complete Arithmetic, specified by Kulisch and Snyder [5],[4], is an example of implementation of accumulator format and exact reduction operations. The recommended accumulator format for the* `binary64` *format in the Complete Arithmetic has 4 bits for sign and status, 2134 bits before the point, and 2150 after the point, for a total of 4288 bits or 536 bytes; this allows for at least* 2^{88} *multiply-adds before overflow can occur.*]