ISO/IEC JTC1/SC22 WG11 N488

Disposition of comments on ISO/IEC JTC1/SC22 N3497 (LIA-3 draft CD)

==========

Comments on N3497 (LIA-3 draft CD) by Germany

> Germany recognizes and greatly appreciates the huge time investment
> and effort by the editor to create the present document virtually
> single-handedly. We believe the document is sufficiently mature to
> move to the CD stage.

Noted.

> However, there are still a number of places
> where the text is incomplete or inaccurate, and there still are
> open questions which should be discussed by the committee.

Noted.

> At this point in time, Germany has not done a thorough reading of the
> whole document, but will comment on certain aspects nevertheless.
>
>
> General remarks and questions:
> ----------

> 1. In some cases a bit more (or even any) text to describe the general
> behavior of a group of related operations/functions would be very
> helpful, e.g. in 5.2.5: is one supposed to guess from the
> opeation's name, which is often abbreviated, what it is supposed to do,
> or do we really want everything to be deducible only from the math
> specs? That seems to be a bit too terse.

Accepted.

Short introductory sentences are inserted before each subgroup. The rationale does need to be extended as well, however that may take some time.

> 2. Also in 5.2.5 and maybe in other places, the role of -0 is
> virtually incomprehensible until one discovers the notes on page 7 ---
> should we make this a bit more explicit or at least give a reference?

Accepted. A reference has been added.

> 3. Also in 5.2.5, what is the reason for having re_F ? Only for NaNs?

Noted.

It is included for consistency of operation sets (also consider programming languages with overloading; the re operation should exist for real, imaginary and complex datatypes, just like the im operation).

> 4. The signum fct is often defined with 3 result values in typical
> math books: -1, 0, +1 . Do we want to have a nonzero result for a
> zero argument?

Noted.

Returning (1 and –1) only is consistent with the (0 and –0)-model; it also fits with the use of the signum function in the specifications of complex trigonometric and hyperbolic operations and the cut between (0 and –0) argument values for some of them. The sign operations of part 1 does what is requested in the comment, but those do not fit with how branch cuts and signed 0-es are handled in part 3, so their use must be avoided. Hence, the specifications for complex ascsin, arctan, arccot, arcsec, and arccsc use signum, not sign.) In addition, the way signum is specified in LIA-3 is consistent with the SIGN function in Fortran (which the LIA-1 sign isn't).

> 5. There should be a rationale for the max_error_... parameters ---
> their choice is by no means obvious, and implementors will like to
> know how they were chosen.

They are mainly taken from the Ada95 standard, with some exceptions (for ln_c(F) in particular).

> 6. From my former colleagues' experience in implementing the complex
> functions for IBM's ACRITH (High Accuracy Arithmetic) Library, there
> is no way to implement the general case of the complex power function
> using a fixed, predetermined internal format if you want to guarantee
> a certain error bound in all cases. The real and imaginary parts of
> the result of the complex power fct are basically real functions with
> 4 real parameters, and the trig fcts kill your error estimates because
> you don't know how close their arguments might get to critical values,
> e.g. to certain mulltiples of pi/2. So it is not at all clear how
> and if any implementor can achieve max_error_power <= 7 .

Noted.

This error bound is unspecified in Ada95. The use of a fixed predetermined format for internal calculations are not a part of the LIA requirements. The problem mentioned in the comment is also present for the LIA-2 radix conversion operations (manifested as f.p. read and write operations in e.g. C), but LIA (as well as C) still require fixed error bounds. The number 7 is taken from Ada's complex exponentiation error bound.

> Corrections:

> ----------
> 7. In annex C.4 (Fortran binding), the syntax of the comparison
> operator for equality (eq) should be == and not = throughout.

Accepted.

> 8. In annex C.4 (Fortran binding), the syntax for taking the real
> part (re) is REAL and not REALPART.

Accepted.

> 9. In annex C.4 (Fortran binding), the syntax for taking the
> imaginary part (im) is AIMAG and not IMAGPART (ugly, but that's
> the long tradition: intrinsic functions with a REAL result were
> not allowed to begin with any of the letters I through N because
> that would make their implicit result type INTEGER).

Accepted.

> 10. There are several other misspellings or syntax errors in
> annex C.4 which we will communicate when the need arises.

Noted. However, the editor would appreciate getting them WITH the other comments...

> 11. To complete annex C.4: Fortran provides complex I/O and
> complex literals in the form (Re, Im) .

The part on the numerals have been edited in.
The I/O formats need to be looked into.

==========

Comments on N3497 (LIA-3 draft CD) by Japan

> Japan disapproves CD 10967-3. The major reason is as follows.
> The draft leaves unresolved several technical decisions necessary to
> constitute the final requirements of LIA-3. It contains place-holders
> to be filled in or even clauses that seem to be mere reminders to the
> editor himself,

> - There are many "EDITOR'S NOTE"s which are not intended to be read
> by readers.

There were, and those that remain are, intended to be read by reviewers.

> - Many clauses, notably in Annexes B and C, contain only one line
> which is "...". Obviously, they are incomplete, and are to be
> filled in.

Noted. Some help would be appreciated.

> - Clauses 5.3.2.2 -- 5.3.3.13 have "NOTE - TEMP". They are not notes
> but incomplete version of the definitions. They should be refined
> and moved to the main text.

No, they refer to some relations that hold for the mathematical functions. These will be moved to the rationale, for the benefit of reviewers. Whether they will remain in the final text remains to be seen.

> - We can find many question marks in the text.

There were, and those that remain are, intended to be noticed by reviewers.

> We must judge that the current version is incomplete. Decisions should
> be made on the technical issues, and the draft should be refined accordingly
> before going to further stages of standardization of LIA-3.

Noted. Some help would be appreciated.

> Comments follow item by item.
>
> Foreword
> The word "organization" and "organisation" are used interchangeably.
> They should be unified to "organization".

Accepted.

> Introduction, 3rd line of "The content"
> We think that the word "-real" in "imaginary-real and complex-real
> arithmetic" should be changed to "-floating-point". Its counterpart is
> not "imaginary" nor "complex", but is "integer". 1.1, (c) and (d)

Rejected. The introduction talks about computer approximations to the imaginary-real and complex-real mathematical functions.

> The order of function classes should be changed. In the main text,
> trigonometric functions appear before hyperbolic functions. Here their
> order is different.

Accepted.

> 2., 3rd paragraph, last line
> The word "takes" should be changed to "take" in
> "specifications ... takes precedence".

Accepted.

> 4.1.5
> The definitions of i(X) and c(X) are not precise. For example, the definitional equation
> i(X) = { ^i.y | y \in X }
> does not say, in mathematical sense, that i(x) has the same cardinality with
> that of X when ^i. is an operator. It is not enough to simply say "^i. is
> a constructor". We need to put a definition clearly saying "a constructor
> shall produce distinct values on its distinct argument values".

Accepted. Explanatory text has been added in a note (a constructor is the tag part of a tagged tuple).

> 4.1.5 says "X may include special values" in the definitions of i(X) and
> c(X). According to the interpretation above, we would consider that

> c(X) contains many special values such as sNaN+^i.qNaN, sNaN+^i.sNaN,
> etc., each distinct from other values. Is this the intent?

Yes.

If X contains the values sNaN and qNaN, yes. They are distinct in the mathematical sense. That does not imply that the eq and neq operations will return anything other than invalid(False) when given such arguments. If X does not contain e.g. qNaN, i(X) will not contain i(qNaN). (However, X containing NaNs are not used explicitly in the LIA-3 specifications, but X containing –0 are.)

> 4.1.5, 2nd paragraph, 1st line
> The word "a" should be changed to "an" in "a imaginary".

Accepted.

> 5., 2nd paragraph
> "For each datatype, two of these ... each: qNaN and sNaN." should be
> "For each datatype, two of these abstract values, qNaN and sNaN, may
> represent several actual values."

Accepted.

> 5.1.1, the last sentence of NOTE
> The sentence beginning with "Similarly below ..." is grammatically
> incomplete (does not have a verb).

Accepted. The sentence is reformulated.

> 5.1.2
> The definitions of mul[i(I)] (p.14, 4th definition from the bottom)
> and mul[c(I)] (p.15, 4th definition from the top) are given in
> a different style from other definitions. Other definitions give
> values of real part and imaginary part separately. In other words,
> they use real arithmetic. Only these two definitions use complex arithmetic.

Noted. While all of the specifications could have used expressions based on result_I, result_i(I) or result_c(I), most of them give simpler and equivalent specifications if expressed using already defined operations. However, for mul_i(I) and mul_c(I) specifications for them using result_I is simpler (due to the possibility of internal overflow). Trying to express specifications for mul_i(I) and mul_c(I) using mul_I would be needlessly complicated,and would be more of implementations than specifications. [TEXT SIMILAR TO THIS COULD GO TO THE RATIONALE.]

> The second and the third alternatives of eq[c(I)] (p.16, 1st definition)
> are overlapping. Both contain the case eq[I](x,z) = false and
> eq[I](y,w) = false. Similarly, the first two alternatives of
> neq[c(I)] (p.16, the last definition) are overlapping. Both contain
> the case neq[I](x,z) = true and neq[I](y,w) = true.

Accepted. The cases in the next draft are non-overlapping.

> 5.2.1
> In the specification of errors for approximation helper function h[c(F)],
> expressions like Re(h[c(F)](Re(z)+^i.Im(z))) appear. According to
> this expression, the signature of h[c(F)] should be c(F) *... -> C.

Accepted in principle.

The argument(s) to h_c(F) should be “z, ...” (in C_F), not “Re(z)+î*Im(z), ...”.

> However, in the main text, say in 5.3.2.2 for sin(z), signatures of
> functions are as follows.
> helper function : sin*[c(F)]: C[F] *... -> C
> operation : sin[c(F)]: c(F) *... -> c(F)..
> Both of these does not match to the signature in 5.2.1.

Accepted in principle.

With the correction in the response to the previous comment, the signatures for the helper function matches what is talked about in clause 5.2.1.

> 5.2.1, Note 3
> The word "an" should be changed to "a" in "an ‘rectangular’".

Accepted.

> 5.2.2, 1st line
> The word "requirement" should be changed to plural.
> Apparently, there are two requirements.

Accepted.

> 5.2.3, Note 1
> The phrase "requirement apply" is not grammatically correct.
> Probably, "requirement" should be "requirements".

Accepted in principle, “apply” changed to “applies”.

> 5.2.4
> In an alternative of the definition of no_result[c(F)], the phrase "at
> least one of x and y is a quiet NaN" appears while in another alternative,
> "at least one of x or y is a signalling NaN" appears. The former uses
> "and" while the latter uses "or".

Accepted. Changed to “and”.

> In the definition of no_result[i(f)->c(F)] (p.21, 2nd definition from the
> top), variables "x" and "y" appear in the right-hand side, They have no
> defining occurrences in the left-hand side.

Accepted. The x’, x, and y should all be y.

> In the definition of no_result2[c(F)] (p.21, 3rd definition from the
> top), the second alternative applies when "neither is a signalling NaN".
> We think that "neither" should not be used when there are

> three or more objects involved.

Rejected.

“neither A, nor B, nor C” is good English (as far as the editor is aware).

> 5.2.5
> The signature of many functions, for example of add[i(F)], contains the
> set notation "{ (underflow), overflow }". What is the meaning of these
> parentheses around "underflow"? Since this is a mathematical notation,
> we feel that parentheses do not convey any meaning.

> The definition of sub[F,i(F)] (p.23, 3rd definition from the bottom)
> uses neg[F] function. Therefore, the value -0 may be generated as the
> result. The signature should be changed to "... -> c(F union { -0 }).
> Similarly, the signature of sub[i(F),c(F)] (p.24, second definition from
> the top) should contain { -0 }.

Rejected. Neither of these two will return a –0-subresult for the arguments mentioned by the argument signatures (which do not contain –0). Only if the first argument is/contains a –0 can a –0-subresult be returned.

> The definition of sub[c(F)] (p.24, 4th definition from the top) replaces
> x-z by x+(-z), in order to define subtraction. This is not consistent
> with other definitions of "sub" functions. They directly define subtraction
> without referring to addition. We think that it is not hard to define sub[c(F)] directly.

Accepted.

> The signature of mul[F,i(F)] (p.24, 6th definition from the top) contains
> { -0 } in its result. However, we cannot find in which case this special
> value is generated. -0 would result when the argument is -0, but this
> is not necessary to be mentioned.

Rejected. Arguments in F (and i(F)) can imply the return of a –0 result, e.g. mul_F,i(F)(–1, i*0) returns i*(–0).

> In the signature of div[i(F)] (p.25, 2nd definition from the top) has
> "union { -0 }" in the domain of its second argument. This seems to be
> a simple typo. It should be moved to the range of the result.

Accepted.

> In the definition of div[i(F),c(F)] (p.25, 7th definition from the top)
> contains an expression re[F](y). This is not correct, since y is a real
> value, and thus re[F](y) is y itself. The correct expression is
> re[i(F)](^i.y), meaning appropriately signed zero.

Accepted.

> In the definition of eq[F,i(F)] (p.25, 3rd definition from the bottom),
> two values x+^i.0 and 0+^iw are compared. Shouldn’t these zeros be
> changed to im[F](x) and re[i(F)](^i.w)?

They could, to be absolutely accurate. But for the comparison operations, 0 and –0 compare equal, so no results will change here if the suggested is done.

> The second and the third alternatives of eq[c(F)] (p.26, 5th definition
> from the top) are overlapping. Both contain the case where x = false
> and z = false. Similarly, alternatives of neq[c(F)] (p.27, 2nd definition
> from the top) are overlapping (both x and z are true).

Accepted. Cases made non-overlapping.

> Is it necessary to define signum[F] (p.28, 5th definition from the top)?
> Both the domain and the range of this function is real.

Noted. Yes. Signum is used in the follow-on specifications for inverse trigonometric operations. The sign operation from part 1 is inappropriate.

> 5.2.6
> It says that two parameters box_error_mode_mul and box_error_mode_div
> should exist. But, the interpretation of values of these parameters
> is not specified. Only the statements such as "max_error_mul interpreted
> according to the value of box_error_mode_mul" appear in this section
> (similar statement exists for max_error_div). What happens when
> box_error_mode_mul is true? What if false?

Rejected. This is specified in clause 5.2.1.

> The definition of signum[c(F)] gives result when the value of its argument
> is -infinity, referring to the result when the argument is +infinity.
> However, there seems to be no definition for the +infinity case.

Rejected. It is handled by the “otherwise” case.

> The phrase "Further requirement ... are" appears in the sixth line from
> the bottom of p.29. The word "requirement" should be changed to
> "requirements". The same phrase appears in many other sections.

Accepted.

> 5.3, 1st line
> One of two "of"s should be removed from ".... of of ...".

Accepted.

> 5.3.1.2
> In the requirement on the relationship with real-valued functions, the
> term "library" is used. We feel that such a concrete notion should not
> appear in the context of specific functions. Isn’t it possible to give
> such definitions in Clause 4., or some appropriate place? Sentences
> such as "The requirements implied by the relationships and the requirements
> from part 2 shall hold even if there is no ... operations in any associated
> library for real-valued operations or there is no associated library
> for real-valued operations" are given for many functions. Isn’t it

> possible to give a single meta-requirement for this?

These formulations occur in introductory sentences to such requirements. However, the word “library” will be removed by reformulation [[WHICH?]]

> 5.3.1.3
> In the definition of power[...] functions, there are many incorrect
> usages of "im" function. Subexpressions such as "im[F](y)" and "im[F](w)"
> should be changed to "re[i(F)](^i.y)" and "re[i(F)](^i.w)". The sign
> of zero is different. If the current text is intentionally written,
> some notes would be necessary.
> p.33, 4th from the bottom (both y and w)
> p.33, 3rd from the bottom (y)
> p.33, 2nd from the bottom (w)
> p.34, 3rd from the top (y)
> p.34, 4th from the top (w)

Accepted.

> 5.3.1.4
> In the definition of sqrt[i(F)->c(F)] (p.35, 2nd definition from the top),
> re[i(F)](y) is not correct. It should be changed to re[i(F)](^i.y).
> Since the re[i(F)] function takes purely imaginary value as its argument,
> its argument cannot be "y" (a real value).

Accepted.

> 5.3.1.5
> For the definition of ln[i(F)->c(F)] (p.36, 2nd definition from the top),
> the same comment applies.

Accepted.

> 5.3.1.5, Note 3, 3rd line
> The term "principle value" seems to be a typo. It would be "principal
> value".

Accepted.

> 5.3.1.6
> As in 5.3.1.3, the "im" function seems to be used incorrectly.
> "im[F](y)" and "im[F](w)" should be changed to "re[i(F)](^i.y)"
> and "re[i(F)](^i.w)". Their positions are exactly corresponding to
> 5.3.1.3.

Accepted.

> 5.3.1.6
> The sentence "A further requirement ... is ..." should be changed to
> plural. There are two requirements.

Accepted.

> 5.3.3
> The sentence "Note that the correspondences specified below to other ISO/IEC

> 10967 operations are exact, not approximate." looks strange. We cannot
> understand the exact meaning of this sentence. If this sentence is
> in fact necessary, it should be given as a general statement, not in the
> context of a specific function.

Noted.

The sentence just emphasises that, as always in LIA, that any correspondence between operations is exact, not approximate. This is emphasised for clause 5.3.3 since all of the operations are rather direct correspondences between the operations of 5.3.3 and of 5.3.2. It is not for a single particular function.

> 5.5
> In the first line, the sentence "Rather than ..., imaginary units are
> specified." appears. Is this sentence a requirement? If so, it should
> be phrased using "shall". We cannot understand the status of this
> sentence.

Accepted in principle. The paragraph has been clarified.

==========

Comments on N3497 (LIA-3 draft CD) by United Kingdom

> The document still has comments like "(mockup so far!)" on page 107 and
> "complex I/O !!!" on page 108, and scanning for "EDITOR'S NOTE" will show
> up quite a number of unfinished sections, so we do not feel it is ready to
> go out to the public.

At the time of writing this DoC this occurs for Ada, Fortran, and PL/I. These remaining unfinished sections will be drafted in the next committee draft.

> On page 107 the statement "Arithmetic value conversions
> in Fortran are always explicit, and the conversion function is named
> like the target type, except when converting to/from strings" is doubly
> incorrect. The first part has not been true since the advent of Fortran
> 77 and as for the names of type conversion functions, it really depends
> what is meant by "like".

Accepted. That text was copied from the corresponding text for another programming language.

> We are concerned by the growing size, and there are hints that
> it is set to grow even bigger. We would prefer a small standard sooner
> to an encyclopedia several years later.

Noted.

> There is too much (near) repetition in the draft, so that it is
> difficult to see and understand what is the same, and what is
> different. In a program, clarity and simplicity would result by
> defining and calling a procedure. A similar process here would give
> similar benefits.

Accepted in principle. However, such a process has already been applied.

==========

Comments on N3497 (LIA-3 draft CD) by Fred Tydeman.

> Page 9: round to nearest: 'between two adjacent values' might be
> better as 'between two adjacent finite values'. Assuming X includes
> infinity, then any finite value u that is larger than the maximum
> finite value in X is closest to that maximum finite value, but IEC
> 60559 requires most of them to round to infinity. There also might be
> problems if X includes -0.

Rejected.

X is here a subset of R (real numbers), and thus cannot contain –0 nor infinities or other special values.

> Page 12: 5.1.2: Consider adding just after the header: In general, the
> first operand is x+i*y, and the second operand is z+i*w.

Rejected.

This is not true in general, though it is followed for the c(F) x c(F) argument type.

> Page 17: abs: I believe that abs can overflow, so need to add overflow
> to signature.

Accepted.

abs_I and abs_i(I) can overflow.

> Page 17: signum: What is signum(-0)? +1 or -1? In either case, you
> need to add the -0 case to either the +INF or -INF cases. I assume -0
> should be with -INF to match signumF.

Rejected.

There is rarely any point in having a sign for integer 0. Anyone having an integer datatype with a negative zero, should handle them consistently with how negative zeroes are handled for the floating point case (and signum_F(-0) is -1). The reason infinities are included for integer types, is that such values are useful, especially for unlimited integer types, where hardware is not so impeding.

> Page 21: no_resulti: Do not understand i*y. Where did y come from?

Accepted in principle.

x' and x should be y (editorial renaming mistakes).

> Page 21: no_result2c: 'neither is' -> 'none are'.

Noted. But any change will be deferred to the next edit round. This is

about English phrasing, and should be British English...

> Page 22 (and following): Do not understand why underflow is inside (),
> while overflow is not.

Noted. This is the editor's way of noting in the signature that the operation will NOT signal underflow for an IEEE 754 conforming implementation when it is not trapping underflow (since the result has to be exact for arguments that give so small results). This is not noted in any way in the other parts of LIA, so the bracketing should be removed, possibly with the addition of a note. Alternatively, retain the notation, and explain it.

> Page 23: several of the sub's: In a two's complement representation of
> floating-point (which is rare), I believe that it is possible for an
> overflow to happen for a simple negate of the most negative number.
> So, need to add overflow to the signatures.

Rejected.

Two's complement floating point implementations do not conform to LIA-1, so are out of scope for LIA-3.

> Page 24: muli(F) versus mulF,i(F): Why is -0 treated differently in
> the signature?

Rejected (for consistency with the c(F) case).

For the second signature referred by the comment it could be written
F x i(F) -> i(F) U {î*(-0), underflow, overflow}
While this formulation may be easier for the i(F) case, it does not work so well for the c(F) case.

> Page 27: absi(F): I believe that abs can overflow (2s complement
> representations, which is very rare for floating-point), so need to
> add overflow to signature.

Rejected.

Two's complement floating point implementations do not conform to LIA-1, so are out of scope for LIA-3.

> Page 28: signum: Consider adding: signum maps a vector to the unit
> circle keeping the angle the same.

Maybe accept in principle.

The formulation needs to be changed, since the true and computed angles may be slightly different, due to round-off errors.

> Page 29: signum: I believe that signum(x) = 1+I*0; that is, a complex
> result.

No change needed.

signum_F returns a real result, signum_i(F) returns an imaginary result

(in i(F) really), and signum_c(F) returns a complex result (in c(F) really).

If the comment refers to signum*_c(F), the result of signum*_c(F) is in C (not c(F)), and 1 = 1 + i*0.

> Page 29: signum: Consider adding: signum(x) = -1-I*0, if x < 0

Rejected (or no change needed).

Assuming this is about signum*_c(F): The suggested condition is already implied by signum_c(F)(x) = 1 if x >= 0 and signum*_c(F)(-z) = -signum*_c(F)(z). Note that –0 = 0 in math.

Assuming this is about signum_c(F): The suggested result is already implied by the requirements on signum_c(F) (for x+î*(-0) and negative x). (Consider the third case together with the abovementioned requirement on signum*_c(F).)

> Page 30: 5.2.6: mul: Note 4: 'is not avoided' might be better as "is
> allowed'.

Rejected (though text clarified).

The note is about the result, where NaNs are not avoided. "is allowed" would talk about the arguments, and NaNs are certainly allowed in the arguments, so no note is needed about that. Text clarified as "is not avoided for the result".

> Page 30: 5.2.6: mul & div: Consider adding here, or in the rationale,
> words on how signed infinities and signed zeros are handled. Take in
> to account how C99 treats them: as polar with specific lengths and
> angles.

Rejected.

LIA-3 handles special values as if the normal expansion of (x+iy)(z+iw) was done. That results in a Cartesian (non-polar) interpretation. A polar interpretation is better suited for a polar representation of complex values (which neither LIA-3 nor C specifies; nor does Ada, PL/I, Fortran, Lisp or Haskell).

A specification along the suggested lines would look something like:

mul_c(F)(a, b) = ... ("result" case as now)
= from_polar_p(F)(mul_p(F)(to_polar_p(F)(a), to_polar_p(F)(b)))
if a or b contain IEEE f.p. special values

together with specifications of the conversions and of polar multiplication, mul_p(F), where p(F) is a datatype for polar representation of complex values built on F. So the suggestion would be a large step from the Cartesian interpretation otherwise prevalent in LIA-3.

> Page 36, 5.3.1.4 sqrt: Consider adding to the note: While
> sqrt(x+I*INF) is +INF+I*INF for all x (finite and infinite), and hence
> independent of the x value, sqrt(NaN+I*INF) is NaN+I*NaN rather than

> +INF+I*INF.

Accepted in principle. (Some form of the suggested text may go into the rationale instead.)

Note that if an x being NaN really represents a complex value, possibly with infinities, a result of INF+iINF would be misleading/incorrect, and hence a NaN (in c(F)) is returned.

> Page 37, 5.3.1.5, ln: For the case x==-0 and y<0, it should be
> lnF(-y).

Accepted in principle.

Corrected to ln_F(neg_F(y)); since special values are also covered by that case, -y would be incorrect for those.

> Pages 39...: Please leave in the NOTE-TEMP sections as they make it
> easier to figure out the special cases. Could be relabeled as:
> Mathematical identities.

Noted. These may end up in the rationale instead.

> Page 39, 5.3.2.2, sin: sin i(F) can underflow for a denormal argument,
> as sinh(denormal) underflows.

Rejected.

sinh_F does not underflow for denormals; LIA-2 specifies it to return the argument for denormal/subnormal arguments, without any underflow notification.

> Page 40, 5.3.2.3 cosine: cos(x+I*y): should be -I*sin...

Accepted.

Further, “y=-inf” corrected to “y in {-inf,+inf}”.

> Page 41, 5.3.2.4 tangent: tan i(F) can underflow for a denormal
> argument, since tanh underflows for a denormal.

Rejected.

tanh_F does not underflow for denormals; LIA-2 specifies it to return the argument for denormal/subnormal arguments, without any underflow notification.

> Page 45, 5.3.2.8 arc sine: asin i(F) can underflow for a denormal
> argument, since asinh underflows for a denormal.

Rejected.

asinh_F does not underflow for denormals; LIA-2 specifies it to return the argument for denormal/subnormal arguments, without any underflow notification.

> Page 48, 5.3.2.10 arc tangent: atan i(F) can underflow for denormals
> arguments, since atanh underflows for a denormal.

Rejected.

atanh_F does not underflow for denormals; LIA-2 specifies it to return the argument for denormal/subnormal arguments, without any underflow notification.

> Page 49, 5.3.2.10 arc tangent: Note 2: ’2*pi (even any integer
> multiple of pi)’ -> ’pi’.

Rejected.

The cyclic requirement is basic to all the trigonometric functions, hence the formulation for all the trigonometric functions refer to the 2*pi cycle.

> Page 49, 5.3.2.11 arc cotangent: acot i(F) is -I*acothF

Accepted (also for arccot_i(F)->c(F)).

> Page 50, 5.3.2.11 arc cotangent: Note 2: ’2*pi (even any integer
> multiple of pi)’ -> ’pi’.

Rejected.

The cyclic requirement is basic to all the trigonometric functions, hence the formulation for all the trigonometric functions refer to the 2*pi cycle.

> Page 57, 5.3.3.10 arc hyper tangent: Note 2: ’2*pi (even any integer
> multiple of pi)’ -> ’pi’.

Rejected.

The cyclic requirement is basic to all the trigonometric functions, hence the formulation for all the trigonometric functions refer to the 2*pi cycle.

> Page 58, 5.3.3.11 arc hyper cotangent: Note 2: ’2*pi (even any integer
> multiple of pi)’ -> ’pi’.

Rejected.

The cyclic requirement is basic to all the trigonometric functions, hence the formulation for all the trigonometric functions refer to the 2*pi cycle.

> Page 62, 5.5: Consider moving this part to be before 5.1 as one needs
> the units before they can form general complex numbers.

See below; the comment nearly repeated below.

> Page 67, A.1: I believe that the upper error bound of 2*rnd_errorF for
> expC is too strict. As I recall, for IEC 60559, rnd_errorF is 0.5, so
> that makes the upper bound 1. It seems that expC needs to be less
> strict than mulC.

Accepted.

The bound is changed to 7 ulps, following Ada. However, Ada gives no particular bound for the error that the power operation must stay within. The editor would like to have some advice as to what error bound is reasonable for the power operation.

> Page 69, B.1.1: I believe the PL/I also has integer complex datatypes.

Noted. The editor will check this before the next draft is issued.

> Page 72: B.5.2: Should add a discussion on meaning of NaN+I*INF and
> INF+I*NaN. C99 treats them as if they were a polar number with an
> infinite length and unknown phase angle. Some people find that
> treatment strange. They prefer, that since one part is a NaN, the
> entire complex number be treated as NaN+I*NaN.

Noted.

LIA considers Cartesian complex values as always being decomposable.

> Pages 72 and 73. I believe that it would help if the formulas for the
> complex functions in terms of just real numbers were included either
> here and/or in the NOTE-TEMP section of each respective function. I

Noted.

> believe that they are:
>
> cis(y) = cos(y) + I*sin(y)

cis is not found in math handbooks, just in some programming languages. However, cis(x) is exp(i*x).

> exp(z)=exp(x+I*y)=exp(x)*cis(y)=exp(x)*(cos(y)+I*sin(y))

May be interesting for a note (with approximately equals). Using it directly (replacing * with mul_F) as a basis for a specification of exp would not allow high precision implementations.

> ln(z)=ln(x+I*y)=ln(sqrt(x*x+y*y))+I*phase(x,y)=(1/2)*ln(x*x+y*y)+I*atan2(y,x)
> =ln(r*cis(theta))=ln(r)+ln(cis(theta))=ln(r)+I*theta; polar form

> sqrt(z)=sqrt(x+I*y)=sqrt((sqrt(x*x+y*y)+x)/2 + I*sign(y)*sqrt((sqrt(x*x+y*y)-x)/2)
> =sqrt(r*cis(theta)) = sqrt(r)*cis(theta/2); polar form z**w=exp(w*ln(z))

> sin(z)=sin(x+I*y)=sin(x)*cosh(y)+I*cos(x)*sinh(y)
> sin(I*y)=I*sinh(y)

> cos(z)=cos(x+I*y)=cos(x)*cosh(y)-I*sin(x)*sinh(y)
> cos(I*y)=cosh(y)

> tan(z)=tan(x+I*y)=(sin(2x)+I*sinh(2y))/(cos(2x)+cosh(2y))

> tan(I*y)=I*tanh(y)

> cot(z)=cot(x+I*y)=(sin(2x)-I*sinh(2y))/(cosh(2y)-cos(2x))
> cot(I*y)=-I*coth(y)

> sec(z)=sec(x+I*y)=(coth(y)+I*tan(x))/(sec(x)*sinh(y)+cos(x)*csch(y))
> sec(I*y)=sech(y)

> csc(z)=csc(x+I*y)=(coth(y)-I*cot(x))/(sin(x)*csch(y)+csc(x)*sinh(y))
> csc(I*y)=-I*csch(y)

> asin(z)=asin(x+I*y)=?
> asin(I*y)=I*asinh(y)

> acos(z)=acos(x+I*y)=?
> acos(I*y)=+/-I*acosh(y)

> atan(z)=atan(x+I*y)=?
> atan(I*y)=I*atanh(y)

> acot(z)=acot(x+I*y)=?
> acot(I*y)=-I*acoth(y)

> asec(z)=asec(x+I*y)=?
> asec(I*y)=+/-I*asech(y)

> acsc(z)=acsc(x+I*y)=?
> acsc(I*y)=-I*acsch(y)

> sinh(z)=sinh(x+I*y)=sinh(x)*cos(y)+I*cosh(x)*sin(y)
> sinh(I*y)=i*sin(y)

> cosh(z)=cosh(x+I*y)=cosh(x)*cos(y)+I*sinh(x)*sin(y)
> cosh(I*y)=cos(y)

> tanh(z)=tanh(x+I*y)=(sinh(2x)+I*sin(2y))/(cosh(2x)+cos(2y))
> tanh(I*y)=I*tan(y)

> coth(z)=coth(x+I*y)=(sinh(2x)-I*sin(2y))/(cosh(2x)-cos(2y))
> cosh(I*y)=-I*cot(y)

> sech(z)=sech(x+I*y)=1/cosh(x+I*y)=1/(cosh(x)*cos(y)+I*sinh(x)*sin(y))
> sech(I*y)=sec(y)

> csch(z)=csch(x+I*y)=1/sinh(x+I*y)=1/(sinh(x)*cos(y)+I*cosh(x)*sin(y))
> csch(I*y)=-I*csc(y)

> asinh(z)=asinh(x+I*y)=?
> asinh(I*y)=I*asin(y)

> acosh(z)=acosh(x+I*y)=?
> acosh(I*y)=+/-I*acos(y)

> atanh(z)=atanh(x+I*y)=?
> atanh(I*y)=I*atan(y)

> acoth(z)=acoth(x+I*y)=?
> acoth(I*y)=-I*acot(y)

> asech(z)=asech(x+I*y)=?
> asech(I*y)=+/-I*asec(y)

> acsch(z)=acsch(x+I*y)=?
> acsch(I*y)=-I*acot(y)

Noted for future consideration for the rationale.

> Page 62, 5.5: Consider moving this section to 5.1, so it comes first,
> since other operations are defined in terms of these units.

Rejected.

It is not the case that the other operations are specified in terms of these units. In addition, it is even common for programming languages not to have these units. Note that LIA-3 uses "meta-units" that are similar to the unit operations defined in clause 5.5, it's not the operations in 5.5 that are used in the specifications of other operations.

> Page 72, B.5.3.1: Editor's note: I see no need to include 3 argument
> cycle exp/log functions.

Noted.

> Page 73, B.5.3.2: Editor's note: I see no need to include 3 argument
> cycle trig functions.

Noted.

> Page 73, B.5.3.3: a): 'more rarely' -> 'less commonly'.

Accepted.

> Page 73, B.5.3.3: Editor's note: I see no need to include 3 argument
> cycle hyperbolic functions.

Noted.

> Pages 80-xx, Annexes C.n: It would help if the introduction section to
> each language were to have a paragraph saying what new data types
> would need to be added to each language.

Accepted. [TODO: edit in]

> Page 81, C.1 Ada: power: Do not understand why 'b'. Why not all as
> x**y?

Noted.

b is integer (while y would be real, imaginary, or complex). The formulations will be reviewed again before the next draft is issued.

> Page 85: C.2 C: Need to add a paragraph along the lines of: C99 does
> not currently support any complex integer or imaginary integer types.
> To fully support LIA-3, these would need to be added to C99.

Rejected.

Like for LIA-2, all operations (and datatypes) in LIA-3 are optional. Conforming implementations need only provide a subset (and state which, non-empty, subset) of operations and datatypes from LIA-3 to support. Further, support for LIA-3 operation may be made available via a library that is not part of a programming language's standard libraries.

> Page 85, C.2 C: itimesI->i(I): Need to define II. I assume it is the
> imaginary unit with type of int. Or, add a note that it is defined
> later on page 92 (should those definitions be move here?).

Noted.

It is the imaginary integer unit. This identifier II is suggested at the end of the C binding. A forward reference will be considered.

> Page 86, C.2 C: absi(I): abs(x) should be iabs(x).

Accepted.

> Page 87, C.2 C: maxi(I): max(x,y) should be imax(x,y).

Accepted.

> Page 87, C.2 C: mini(I): min(x,y) should be imin(x,y).

Accepted.

> Page 87, C.2 C: 'type CINT.,.,.,.,.,' should(?) be 'type CINT.'.

Noted.

These unfinished lists of meta-identifier explanations will be reviewed before the next draft is issued.

> Page 87, C.2 C: conji(F): Need to remove 'or conj(x)' as C99 only
> defines conj for complex.

Accepted in principle (conj(x) kept as "not in standard").

> Page 88, C.2 C: eq: Need (*) for F,i(F) case.

Accepted.

> Page 88, C.2 C: neq: Need (*) for F,i(F) case.

Accepted.

> Page 89, C.2 C: 'complex floating point type.,.,.,.,.,' should(?) be:
> 'complex floating point type.'.

Noted.

These unfinished lists of meta-identifier explanations will be reviewed before the next draft is issued.

> Page 89, C.2 C: 'where t ...' should(?) be: where t is "f" for float
> _Complex, the empty string for double _Complex, and "l" for long
> double _Complex.

Noted.

These unfinished lists of meta-identifier explanations will be reviewed before the next draft is issued.

> Page 90, C.2 C: tani(F): tant(x) should be tan(x).

Accepted.

> Page 90, C.2 C: cotc(F), secc(F), cscc(F) should all have the type
> generic function in addition to the type specific one, eg, 'or
> cot(x)', 'or sec(x)', 'or csc(x)'. I assume this based upon coti(F)
> mapping to cot(x) => cot(x) is the type generic one.

Accepted.

> Page 90, C.2 C: arccotc(F), arcsecc(F), arccscc(F) should all have the
> type generic function in addition to the type specific one, eg, 'or
> arccot(x)', 'or arcsec(x)', 'or arccsc(x)'. I assume this based upon
> arccoti(F) mapping to arccot(x) => arccot(x) is the type generic one.

Accepted.

> Page 91, C.2 C: cothc(F), sechc(F), cschc(F) should all have the type
> generic function in addition to the type specific one, eg, 'or
> coth(x)', 'or sech(x)', 'or csch(x)'. I assume this based upon
> cothi(F) mapping to coth(x) => coth(x) is the type generic one.

Accepted.

> Page 91, C.2 C: arccothc(F), arcsechc(F), arccschc(F) should all have
> the type generic function in addition to the type specific one, eg,
> 'or acoth(x)', 'or asech(x)', 'or acsch(x)'. I assume this based upon
> cothi(F) mapping to acoth(x) => acoth(x) is the type generic one.

Accepted.

> Page 91, C.2 C: 'complex floating point type.,.,.,' should(?) be:
> 'complex floating point type,'.

Noted.

These unfinished lists of meta-identifier explanations will be reviewed before the next draft is issued.

> Page 92, C.2 C: complex IO??: Done as a pair of real I/O?? The C99
> committee could not agree on how to do it. Some suggestions were:
> (x,y) or x+yI or x+Iy.

Noted.

These unfinished lists of meta-identifier explanations will be reviewed before the next draft is issued.

> Page 92, C.2 C: 'FXD is a fixed point type' should(?) be changed to
> 'integral type'. Not sure what you mean by fixed point type. Also,
> the use of x as INT, y as FLT, and z as FXD do not match the actual
> usage in the mappings above that paragraph.

Noted.

FXD is used also for I/O formats that do not have an exponent, and C supports that.

> Page 92, C.2 C: 'e is greater than 0.,.,.,.,.,.' should be removed or
> cleaned up.

Noted.

These unfinished lists of meta-identifier explanations will be reviewed before the next draft is issued.

> Page 92, C.2 C: 'Numerals...:' should be on line by itself. Also,
> consider moving this section to the front of this appendix.

Accepted.

> Page 92, C.2 C: ((II)) does not convert II from imaginary to complex.
> Perhaps: (0+II) would work.

Noted.

The parentheses were meta-notation. The lines will be deleted.

> Page 92, C.2 C: ((I)) does not convert I from imaginary to complex.
> Perhaps: (0.f+I) would work.

Noted.

The parentheses were meta-notation. The lines will be deleted.

> Page 95, C.3 C++: plusitimesF: extra ')'.

Accepted.

> Page 95, C.3 C++: add, sub, mul, ...: F,c(F) should go after i(F),F.

Accepted.

> Page 97, C.3 C++: sqrt: How does the user tell the compiler to do F->F
> versus F->C(F)? I believe that using the same funciton name for both
> operations will not work.

Accepted. The latter renamed to sqrtc.

> Page 98, C.3 C++: asin: How does the user tell the compiler to do F->F
> versus F->C(F)? I believe that using the same funciton name for both
> operations will not work. Same problem with acos, atan, acot, asec,
> acsc.

Accepted.

> Page 99, C.3 C++: assech: How does the user tell the compiler to do
> F->F versus F->C(F)? I believe that using the same funciton name for
> both operations will not work.

Accepted.

> Page 100, C.3 C++: complex IO??: Done as a pair of real I/O??

Noted.

Text will be reviewed before the next draft is issued.

> Page 100, C.3 C++: Consider moving the Numerals section to the front
> of this appendix.

Noted.

> Page 100, C.3 C++: ((II)) does not convert II from imaginary to
> complex. Perhaps: (0+II) would work.

Noted.

The parentheses were meta-notation. The lines will be deleted.

> Page 103, C.4 Fortran: IU goes against the Fortran naming convention;
> it implies an integer type, not a floating-point type.

Noted.

The parentheses were meta-notation. The lines will be deleted.

> Page 103, C.4 Fortran: plusitimes: extra ')'.

Accepted.

> Page 105, C.4 Fortran: powerF->c(F): Do you really mean '***'? Is
> this a new operator? If it should be '**', then how does the compiler
> tell F->F from F->c(F)?

Accepted.

***** is used instead, though that may be changed to a function call instead before the next draft is issued.**

> Page 108, C.4 Fortran: Numerals: Is there where 'IU' would be defined?
> Also, consider moving this section to the front of the appendix.

Accepted in part and in principle.

"IU" -> "II". However, the numerals paragraph stays at the end for the time being.

> Page 109, C.5 Haskell: itimes: missing space in 'I *x'.

Accepted.

> Page 111, C.5 Haskell: plusitimes: Is it really ':+'? I do not know
> the language, but it looks so strange.

Noted. Yes, this is the ("standard", i.e. already so) Haskell syntax for constructing complex values from f.p. values. All so-called constructors that are operators have : as the first character of the name. (Further, all so-called constructors that are identifiers have to start with a capital letter.)

> Page 111, C.5 Haskell: add, sub, mul, ...: F,c(F) should go after
> i(F),F.

Accepted.

> Page 114, C.5 Haskell: sini(F): extra space in 's in x'.

Accepted.

> Page 116, C.5 Haskell: Numerals: Consider moving this section to the
> front of the appendix.

The numerals paragraph stays at the end for the time being.
This may be reconsidered before the next draft is issued.

> Page 119, C.6 Java: Appears to be missing the specifications of the
> double arithmetic operations, eg, add, sub, mul, div, ...

Accepted. (A LaTeX source text error prevented their output.)

> Page 119, C.6 Java: sqrt: How does the user tell the compiler to do
> F->F versus F->C(F)? I believe that using the same funciton name for
> both operations will not work.

The names can actually be the same, provided they are in different classes, which they will be.

> Page 119, C.6 Java: Appears to be missing the specifications of the
> double trig functions, eg, sin, cos, tan, ..., sinh, cosh, tanh, ...

Accepted. (Another LaTeX source text error.)

> Page 120, C.6 Java: convert I->c(I): do not understand: ..(x)

Noted.

It means "not finished by the editor". You are welcome to suggest actual text, as a help to the editor.

> Page 120, C.6 Java: convert i(F)->c(F): -0 should be -0.f

Accepted in principle.

0.0 is used (for double). A binding for "float" will be added.

> Page 120, C.6 Java: Remove(?): e is greater than 0.

Noted.

Text will be reviewed before the next draft is issued.

> Page 120, C.6 Java: Numerals: Consider moving this section to start of
> annex.

Noted.

The numerals paragraph stays at the end for the time being.
This may be reconsidered before the next draft is issued.

> Page 120, C.6 Java: Numerals: Last one should be 0.f + I

0.0 is used (for double). A binding for "float" will be added.

> Page 124, C.7 Common Lisp: eq, neq, ...: F,c(F) should go after
> i(F),F.

Accepted.

> Page 126, C.7 Common Lisp: asin: How does the user tell the compiler
> to do F->F versus F->C(F)? I believe that using the same funciton
> name for both operations will not work. Same for acos.

Rejected.

Common Lisp is dynamically typed (with numeric subtyping), and a single function can return differently typed values for different argument values (not just for different argument types).[todo: add a note or something]

> Page 127, C.7 Common Lisp: acosh: How does the user tell the compiler
> to do F->F versus F->C(F)? I believe that using the same funciton
> name for both operations will not work. Same for atanh, acoth, asech.

Rejected.

Common Lisp is dynamically typed (with numeric subtyping), and a single function can return differently typed values for different argument values.

> Page 128, C.7 Common Lisp: Numerals: First '...' should(?) be (i 1)
> Second '...' should(?) be (i 1.0)

Noted.

Text will be reviewed before the next draft is issued.

> Page 132, C.8 ISLisp:eq, neq, ...: F,c(F) should go after i(F),F.

Accepted.

> Page 135, C.8 ISLisp: acosh: How does the user tell the compiler to do
> F->F versus F->C(F)? I believe that using the same funciton name for
> both operations will not work. Same for atanh, acoth, asech.

Rejected.

ISLisp is dynamically typed (with numeric subtyping), and a single function can return differently typed values for different argument values.

> Page 135, C.8 ISLisp: convert i(I)->c(I): How about: (add 0 x)

Accepted in principle.

The function "complex" is used (as for Common Lisp).

> Page 135, C.8 ISLisp: convert i(F)->c(F): How about: (add 0.0 x)

Accepted in principle.

The function "complex" is used (as for Common Lisp).

> Page 135, C.8 ISLisp: Numerals: First '...' should(?) be (i 1) Second
> '...' should(?) be (i 1.0)

Noted.

Text will be reviewed before the next draft is issued.

> Page 137, C.9 Modula-2: ’(and LONGREAL’ -> ’and LONGREAL’.

Accepted.

> Page 142, C.9 Modula-2: arctan i(F) and arctan i(F)->c(F) cannot both
> be arctan(x); need to rename the 2nd to arctanc(x). Same for arccot.

Accepted.

> Page 143, C.9 Modula-2: arcSecHc(x) should(?) be arcsechc(x).

Accepted.

However, case of letters are not significant in Modula-2.

> Page 144, C.9 Modula-2: Numerals...: Should be a new paragraph. Also,
> consider moving this section to that start of this annex.

Accepted in principle.

The numerals paragraph stays at the end for the time being.
This may be reconsidered before the next draft is issued.

> Page 144, C.9 Modula-2: ((II)) does not convert II from imaginary to
> complex. Perhaps: (0+II) would work.

Noted.

It means “not finished by the editor”. You are welcome to suggest actual text, as a help to the editor. Text will be reviewed before the next draft is issued.

> Page 144, C.9 Modula-2: ((I)) does not convert I from imaginary to
> complex. Perhaps: (0.f+I) would work.

Noted.

It means “not finished by the editor”. You are welcome to suggest actual text, as a help to the editor. Text will be reviewed before the next draft is issued.

> Page 145, C.10 PL/I: I believe that PL/I supports both real and
> complex versions of integers and floating-point; no mention is made of
> complex in the introduction. I do not recall any support for
> imaginary.

Noted.

This will be reviewed before the next draft is issued.

> Page 147, C.10 PL/I: a definition of 'xs' is missing; I assume it is
> array of CINTs. It should be moved(?) from page 149 to here and type
> changed from FLT to CINT.

Noted.

This will be reviewed before the next draft is issued.

> Page 149, C.10 PL/I: 'x *** y' ??? is this a new syntax? If not, how
> is x**y for F->F different than F->c(F)?

Noted.

***** is a new operator. However, this may be replaced by a function call before the next draft is issued.**

> Page 150, C.10 PL/I: How to tell arctan i(F) from arctan i(F)->c(F)
> when both are spelled arctan(x)? Perhaps need arctan(x) and
> arctanc(x)? Same for arccot.

Noted.

This will be reviewed before the next draft is issued.

> Page 152, C.10 PL/I: Complex I/O I believe is already supported in
> PL/I, but I forget how.

Noted.

This will be reviewed before the next draft is issued.

> Page 152, C.10 PL/I: 'a is greater than 0.' should(?) be removed.

Noted.

This will be reviewed before the next draft is issued.

> Page 154, C.11 SML: Need definition of xs at bottom of page.

Noted.

This will be reviewed before the next draft is issued.

> Page 154, C.11 SML: add, sub, mul, ...: Move F,c(F) after i(F),F.

Accepted.

> Page 158, C.11 SML: atanc(x) and acotc(s) seem strange compared to the
> others.

Accepted. (The parentheses are harmless, but also unnecessary.)

> Page 159, C.11 SML: ((II)) and ((I)) seem wrong.

Noted.

This will be reviewed before the next draft is issued.

> Page 162, Bibliography: IEC 60559 is missing.

Rejected.

IEC 60559 is a normative reference, and is therefore, per instruction from ISO editor, excluded from the bibliography.

> Page 163, Bibliography: You might consider adding: An Atlas of
> Functions by Jerome Spanier and Keith B. Oldham. Published by
> Hemisphere Published Corp, 1987.

Noted.

This reference may be added before the next draft is issued.

==========