IEEE Standards Interpretations for IEEE Std 1003.2™-1992 IEEE Standard for Information Technology--Portable Operating System Interfaces (POSIX®)--Part 2: Shell and Utilities

Interpretation Request #96
Topic: bc - precision **Relevant Clauses:** 4.3.7.1

I would like to disagree with the following "binding interpretation". ... I would like to an request official, binding interpretation from the IEEE concerning the following point in IEEE Std 1003.2-1992 (POSIX.2). POSIX.2 Subclause 4.3 specifies the semantics of the "bc" utility. In subclause 4.3.7.1, lines 1570-1571 state: The scale of an invocation of each of these functions shall be the scale of the value of the scale register when the function is invoked. The functions referred to are the math functions s(), c(), a(), l(), e() and j(). Suppose that bc is invoked with the -l flag, and that the scale is 20. What is the value of the expression scale(a(0)) On the one hand, the sentence quoted above implies that it should be 20. On the other hand, the value of a(0) is exactly 0, and the scale of 0 is 0.

The specification on lines 1570-1573, page 196 is talking about the scale used while computing the value of the arctangent, not about the computed arctangent, since arctangent (0) is zero, scale(a(0)) is equivalent to scale(0) which is zero. The standard clearly states the behaviour for bc and scale(), and conforming implementations must conform to this. The standard does not give any algorithms on how to compute these functions.

Why should the time of invocation be mentioned IF *ALL* computations must be done at the same scale. *IF* it is talking about all computations in the function, then the *RESULT* would have to have the same scale as all the other computations. therefore scale(a(0)) should be the same scale as when the function is invoked. scale(x) when (x == 0) IS NOT always zero! Consider the following code: x = 0.00000 scale(x) 5 (This is the result of /usr/bin/bc on a SUN 4.1 system. Identical results from GNU bc which was

implemented directly from the IEEE 1003.2-1992 POSIX draft standard.) I say that this IS the correct behavior.

From lines 1405-1407
A numeric constant shall be an expression. The scale shall be the number of digits that follow the radix point in the input representing the constant, or zero if no radix point appears. By forcing the scale to remain the same value while computing the functions can produce errors in the results. For example, from the GNU bc implementation, the following were computed. a is the original GNU bc arctan function and xa is the same EXCEPT all computations are done in the scale at the time of call (as per the interpretation) (Note: this is using a nonstandard feature of GNU bc of multi-character names. This does not change the computation results.) a(10000) 1.57069632679522995256 xa(10000) 1.57069632679522995258 scale = 25 a(10000) 1.5706963267952299525626550 xa(10000) 1.5706963267952299525626544 NoteÒthe original "a" is more accurate in the digits returned because it can change the scale during computation. The effect is even worse for the ln function (l(x)) as follows. (Again "l" is the original version and "xl" is the version that does not adjust scale during computation.) scale=20 l(1000000) 13.815510557964274104 xl(1000000) 13.815510557964274098 scale=30 l(1000000) 13.815510557964274104107948728106 xl(1000000) 13.815510557964274104107948727296

Notice that the original computation returns the exact 20 digits that are the first 20 of the longer computation. The "new version" that computes at the original scale is wrong in the last 5 digits of the 20. That is a big error in my opinion. Quoting again: The specification on lines 1570-1573, page 196 is talking about the scale used while computing the value of the arctangent, not about the computed arctangent, since arctangent (0) is zero, scale(a(0)) is equivalent to scale(0) which is zero. As I understand the above, the function can not change scale during the computation of the value and be conforming. For the sine function, this extreme is even worse. In one place in the GNU bc library code, sine sets scale to zero for a computation.

Changing that to compute at the same scale as everything else yields the following results: (again "s" is the original, "xs" is without changing scale) s(1) .841470984807896506665 xs(1) -1.000000000000000000002 s(.5) .479425538604203000027 xs(.5) 1.000000000000000000002

Here is my interpretation of lines 1570-1571
The scale of an invocation of each of these functions shall be the scale of the value of the scale register when the function is invoked. "an invocation" refers only to the result, not how it was computed. "the value of the scale register when the function is invoked" was used to allow the algorithm to use a larger scale to produce more accurate results and then return the result with the scale defined at the start of the algorithm. scale(0) is not always 0. This depends on how the 0 was produced. For example: - scale(0.000) should be three. - scale = 20; scale(0/1) is 20 by definition of / and is the value 0.00000000000000000000. (lines 1443- 1445) Thank you for considering this.

In working on the above disagreement, I discovered something that I would like to point out to the committee about output of numbers as defined in 1003.2 subclause 4.3.7.1. Lines 1353 - 1387 define the format and content of output numbers in bc. It appears to me that there is a major difference between this specification and traditional bc. That is in the output of the number zero. (0) Lines 1355 - 1373 define 3 "paragraphs" on how to output a number. talks about the minus sign talks about the integer portion of the digits (3) talks about the fractional digits. It appears that (3) is *always* considered. This changes the output of the number 0. Traditional output of the number zero was away the character "0" regardless of the scale. For example, "scale(0.00000)" prints "5" and "0.00000" prints "0". In my reading of lines 1335-1373, the line (1365) "If the numeric value is zero, bc shall write the character 0." is ONLY part of (2) and does not stop applying part (3) to the number zero. Therefore, "0.00000" should print as "0.00000". Is this non-traditional behavior wanted?

Interpretation Response
Upon further consideration, we agree that the response to PASC Interpretation #77 is incorrect. The interpretation on #77 should have been: the first sentence of the paragraph on page 200 l1570-1573 is poorly worded. the scale of the result of calling one of the math functions provided when the -l option is specified, could be interpreted to be the value of the scale register at the time the function is invoked or the scale of 0 (which is 0). The standard is unclear on this issue, and no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

Rationale for Interpretation
None.