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 #43
Topic: Regular Expressions imprecise specification **Relevant Clauses:** 2.8

Issue D: These questions address areas of imprecision in the specifications of REs. With regard to question [13], 1003.2 uses various terms roughly synonymously (leftmost, first, earliest) (subpattern, subexpression) for describing a left-to-right order across a line of text. It would be well if the text (and rationale) used one term consistently. [12] Can a duplicated subexpression match the null string? If so, will the duplication be repeated until the expression does match the null string? Proposed solution A subexpression that can match a null string shall not be duplicated. Rationale Although adjacent duplication symbols are illegal (for no apparent reason, particularly for EREs), \(x*\)* is a legal expression, in which the *s are not adjacent, that raises the question: how many times is the null string matched? Suppose that \(x*\)* were allowed. Does matching it to xxx yield \1 = xxx or \1 = null? The latter alternative is consistent with the rule that a null string is longer than no match. By extending xxx with a null string instead of nothing, one gets a longer match. More null strings make even longer matches.

To avert metaphysical questions about the last element of an infinite sequence, or an element following an infinite sequence, one could forbid null iterations. But this has an unsatisfying corollary that \(x*\)* wouldn't match the null string. Compromise positions might forbid null iterations after the first iteration or after the first null iteration. Such special pleading and the resultant implementation complexity is not worth the return. Lord and McIlroy have implemented a no-null-unless- first policy; Spencer has implemented repeat-until- null. Spencer's interpretation seems perhaps less strained, until you realize that it makes references (\1, \2, etc) to contents of duplications useless because they will always be null or undefined. The proposed solution also outlaws EREs like (a?|b)*, (a?)\{2,2\}, and . ([a- z]*(|^))* The set of regular languages is unchanged,

however. There would still be legal equivalents to these and all other outlawed expressions.

An alternative, to leave the meaning undefined, is unacceptable. Users will be unaware of the exact bounds of portability, and their biggest intellectual investments - the hardest expressions and sed scripts - are liable to be unportable. [13] What is the meaning of the BRE \(\)? [2.8.5.2] Proposed solution Forbid this expression. It would be also acceptable, but rather less so, to define that the pattern \(\) matches the null string.

Rationale: The pattern has no analog in EREs, no utility, and to our knowledge no basis in history. It should thus be banned. Otherwise, at least define what it means. [14] On line 2792, what is left-to-right order in a match?

Proposed solution: Insert the following text at the end of line 2792: A nested subpattern is to the left of the subpattern that contains it. An earlier iteration of a duplicated subpattern is to the left of a later iteration. Neither side of an alternation is to the left of the other. Length ties in an alternaion are broken in favor of the left side.

Rationale: Consider the pattern \(\(...\)*\(.....\)*.\).* matched against the string xxxxxx. Is the leftmost subpattern the first complete subpattern, namely ..., or the pattern that lexically starts first, namely \1? Call these two cases "first-finished" and "first-begun". The proposed solution adopts first- finished. A first-finished match yields \1 = xxxx, \2 = xxx, and \3 unmatched. A first- begun match yields \1 = xxxxxx, \2 unmatched, and \3 = xxxxx. Lord and Spencer both do first- finished matching, although Spencer argues for first-begun. In many cases, including the present example, first- finished matching can be done with less backtracking. McIlroy implemented both and found first- begun much more awkward than first- finished. Historical practice favors first-finished. The left-associativity of concatenation (line 3256) and the transparency of parentheses (line 2977) together imply that \(...\)*\(.....\)*.* and \(\(...\)*\(.....\)*\).* should match the same way. Under first-begun matching to xxxxx, in the former case subpattern \(...\)* matches xxx; in the latter it matches the null string. Thus first-begun matching entails a contradiction. [15] To which match is a backreference to a duplicated subexpression bound? [2.8.3.3(4)]

Proposed solution: A backreference to a subexpression contained in a duplication is bound to the (possibly nonexistent) match to the subexpression in the most recent iteration of the duplication. Thus \(\(.\)\2\)* matches xxyyzz, with \2 referring to x, y, and z respectively in the three iterations of the outer subexpression. However, \(\(.\)*\2#\)* matches only the first six characters of xx#yy##, because in a third iteration of the outer subexpression, . would match nothing (as distinct from matching a null string) and hence \2 would match nothing.

Rationale: Current implementations agree on this interpretation, which is a natural generalization of binding in regmatch structures by regexec(). [B.5.2]

Interpretation Response
The standard does not require the successful matching of a null string after a successful match on a non null string for duplication symbols. However, the standard does not clearly say that you are not allowed to match null strings after a successful match. Therefore the standard in ambiguous.

The definition of a back reference expression in subclause 2.8.3.3 does not specify which match of a subexpression followed by a duplication symbol is to be returned. We note however that the definition regcomp() defined in clause B5.1 pg 728 344-346 indicate that the back reference expression will match the last string matched by the sub expression. 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. Q13 Pg 89 line 3263 requires this form to be accepted. But, the text 2.8.3 does not describe its meaning. Concerns about the wording of this part of the standard have been forwarded to the sponsor.

This section New to this revision Part 14: The text on page 82, lines 2975-2979, and page 77, lines 2791-2796 are in conflict and as such 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. Part 15: The standard does not speak to this issue, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

Rationale for Interpretation
None.