L2/98-059

BIDI Reference Code Comments

by Doug Felt (contact Mark Davis for comments), 2/23/98

*** General Comments ***

The author raises several issues in comments in the code. The issues are very pertinent and useful. Many thanks to the author.

This document is divided into two sections. The first lists proposed clarifications and corregenda to the standard. The second describes errors in the proposed reference implementation.

*** Clarifications ***

[Note: someplace in the text leading up to the algorithm, perhaps on page 3-16, the role of embeddings should be described. Here's some possible text:

"Explicit embeddings are introduced by the formatting codes LRE, RLE, LRO, and RLO, and terminated by PDF. These delimit embedding levels of the text, with each nested embedding increasing the level. The various rules in the algorithm always stop at an embedding level transition. Furthermore, they treat transitions differently depending on whether the the transition is encountered as going from a higher level to a lower level, or vice-versa.

"When examining previous text, the algorithm treats a transition to a lower-level embedding like sot of a line that has a base direction of the current embedding. Similarly, when examining succeeding text, the algorithm treats a transition to a lower-level embedding like eot. No tests rely on information from outside the embedding. In particular the characters that delimit the embedding (LRE, PDF, etc.) are not examined, either for character block or directional type. Thus, within an embedding, the algorithm treats the controlled text like it were the entire text.

"The algorithm treats a transition to a higher-level embedding like it would encountering a non-arabic strong directional character of a type corresponding to the higher-level embedding. Thus, from outside an embedding, the algorithm treats the controlled text like it were a (non-arabic) character of the appropriate strong directional type.

"In all the rules, scanning stops when a strong directional character is encountered. Thus a level transition, whether to a higher or lower level, always acts to limit the text to which a rule applies.

"Note that the number of nested embeddings is limited. Excess embeddings are ignored by the algorithm."

*** New corridenda ***

(A)

[Proposed corrigendum A clarifies a remaining ambiguity about the directional type of combining marks.]

An existing corrigendum to page 4-11 adds the phrase:

"Combining marks are given the type of the preceeding letter, and are not called out in this table either."

To this, append:

"Combining marks with no preceeding letter are Other Neutral."

(B)

[Proposed corrigendum B1 corrects a typographical error. Proposed corrigenda B2 and B3 clarify that embedding and override codes and their matching PDFs are not themselves set to the level of the text they enclose.]

(B1)

On page 3-18, rule E2, change:

"...don't change levels with LRE..."

to:

"...don't change levels with RLE...".

(B2)

After the last sentences of rules E2, E3, O2, and O3, add:

"The formatting code itself retains the pushed level, and is not assigned the new level.

(B3)

After the first sentence of rule T4, insert:

"The PDF itself is assigned the popped level (the one in effect before the embedding or override)."

(C)

[Proposed corrigendum C clarifies what it means to 'ignore' an embedding or override code, and additionally removes an ambiguity about the directional types assigned to excess pairs.]

An existing corrigendum rewords the original rule N4 and renames it T6 to replace the original T6, quote:

"T6. In the following rules, an embedding or override code and its matching PDF act as if they were strong characters of the appropriate type. All unmatched PDFs are ignored. If two embeddings with the same level are adjacent, then the PDF terminating the first embedding and the code initiating the next embedding are ignored."

In this corrigendum, replace the first sentence with:

"An embedding or override code and its matching PDF are set to the corresponding strong directional type. An embedding or override code and its matching PDF that are ignored because the maximum level would be exceeded remain Other Neutral."

Additionally, in the last two sentences, amend the phrase "are ignored" to read "remain Other Neutral."

(D)

[Proposed corrigendum D clarifies the role of embedding boundaries in limiting the scope of searching for a previous Arabic numeral when resolving weak types.]

In rule P0, replace the two occurences of

"block boundary"

with:

"block or level boundary"

(E)

[Proposed corrigendum E adjusts the treatment of LRM and RLM so that they are not treated as strong characters when scanning for a previous strong arabic character when resolving weak types.]

In rule P0, replace:

"the first strong character"

with:

"the first strong character that is not LRM or RLM"

(F)

[Proposed corrigendum F clarifies that when resolving implicit levels, the final level is not limited to the range 0-15 as it is when processing embedding and override codes.]

In the paragraph preceeding rule I1, insert the following before the last sentence of the paragraph:

"The resolved level is not limited and in some cases will resolve to 16."

(G)

[Proposed corrigendum G clarifies that a block or lower embedding level boundary acts as sot in determining whether EN goes up zero or two levels when resolving implicit levels.]

To rule I1, append:

"If sot, a block boundary, or a lower embedding level boundary (e.g. a boundary at a preceeding LRO, LRE, RLO, or RLE), is encountered before any other strong character, EN goes up two levels. A higher embedding level boundary (e.g. a boundary at a preceeding PDF) is treated like the corresponding strong character, and so will cause EN to go up two levels if the higher embedding level is odd (the PDF has resolved to become strong right-to-left)."

(H)

[Proposed corrigendum H clarifies that trailing whitespace incudes runs of whitespace before tab separators, as well as before block separators and line breaks.]

Replace rule L1 with:

"Reset the embedding level of segment separators, block separators, and trailing white space to the embedding level. 'Trailing white space' refers to runs of white space before any segment separator, block separator, or the end of the line."

*** Errors in the implementation ***

This list of errors reflects the proposed corrigenda above. Again, thanks to the author for raising issues where the existing standard was ambiguous.

These comments are based on a visual inspection of the code. I have not yet run the code against test data to verify the problems described below, and thus I may be mistaken about some errors. My apologies if this is the case. In order that the code be suitable for expository purposes, though, it may still be worthwhile to rewrite working portions of the code in order to clarify their relationship to the algorithm.

There are a number of places where the proposed code is incorrect. There are three main sources of problems:

1. The code is written to operate over multiple blocks of text in one pass. This introduces errors where information is carried over from one block to the next. The entire algorithm should reset after a block boundary and no information carried forward. The best way to prevent errors, and illustrate this basic principle of the algorithm, is to code the algorithm to operate on a single block terminating with zero or one block separator.

2. The code tends to rely on the presence of embedding codes to stop processes that should stop at embedding level boundaries. The direction class or type of of the character at these boundaries should not be examined when the transition is to a lower level boundary, instead the default for sot/eot should be used. Even when the code does work, the reliance on the resolved value of the directional formatting codes obscures the more fundamental point of the algorithm. The clearest and safest approach is to code routines with explicit tests for the relevant level boundary transitions.

3) The reordering routines miss the essential point that reordering of trailing whitespace occurs on a line or segment basis, not on a block basis. This should be illustrated by calling the reordering routines from code that incorporates external information about line break positions.

`resolveExplicitLevels`

- This incorrectly sets the level of the block separator to the base level of the following block. It should be the base level of the current block. Block separators are logically part of the preceeding block, not the following block.

`changeEuroNumbers`

- Code comments ask whether the back scan to determine the arabic properties are delimited by the block, or by the embedding. The code uses the block, as defined, but per the new corrigenda the code should use the embedding. If an embedding is encountered (higher or lower) before encountering an arabic character, then the european number remains as it is.

- An arabic character with overridden direction is in a different embedding and so is 'not arabic'.

- LRM and RLM should not be treated as strong characters, per the new corrigenda.

`changeTerminators`

- Code comments ask about level boundaries. This should not cross a level boundary. Since the implementation leaves formatting characters in the text, it works, but it might be easier to understand if recoded with explicit tests for level boundary.

`findAdjacentDir`

- the documentation is incorrect, this can return a value other than L or R, namely -1. It will do this if nAdjacentBidiCategory is a separator (B or S). B is treated as eot/sot (although if the algorithm is recoded for single blocks, there never is a preceeding block separator). S should not be treated as eot/sot.

`changeNeutralRun`

- this looks fine as long as it is applied only to runs of neutrals at the same embedding level, and has proper values. As it is, it appears a run of neutrals between tabs would be set to have a bidi catetory of

-1, as findAdjacentDir will return -1 in this circumstance.

`resolveNeutrals`

- The code appears to retain the information about the last strong directional character across block boundaries. This is incorrect. It also treats a preceeding block separator differently from sot, which is also incorrect. I'd have to run the code to make sure it doesn't get the right results by accident, but even if it did I think it would need to be rewritten.

* This is a good reason to rewrite the algorithm to work on single blocks terminated by eot or a block separator, rather than on multiple blocks.

`resolveImplicitLevels`

- The notes ask what happens if the implicit level exceeds [15]. The code doesn't change the level. The new corrigenda say that the level should change.

- This ends up terminating runs of neutrals at level changes because of the presence of explicit formatting codes. This has the effect that the LRE that may preceed a run of neutrals is treated like a 'preceeding L' character, when it should be treated like sot (no preceeding character).

* this is a good reason to explicitly code using level boundary changes, rather than letting the explicit direction codes implicitly stop or reset various scanning operations by virtue of being treated as strong directional characters.

`handleWhitespace`

- The notes ask whether the block separator gets the embedding level of the previous or next block. The code uses the next block, but it should be the previous block.

- This only resets the level of segement separators if they are part of trailing whitespace at the end of the block. This is incorrect, it should reset segment separators wherever they appear in the text.

- The code only resets whitespace before eot or block separators. The new corrigenda includes whitespace before segment separators.

- The code is incorrect in that it assumes 'trailing whitespace' refers to whitespace at the end of a block. It actually refers to whitespace at the end of a line. Thus this code must be invoked upon a portion of a block that is to represent the line, and should not assume the entire block. This interface has to be revised to take the line start and end.

* This is a good reason to recode this second phase of the algorithm (including reorderResolvedLevels) as operating on lines after some external process has determined the line breaks.

`reorderResolvedLevels`

- This should be written to operate on a line, not on a block. Although once linebreaks have forced trailing whitespace to the base direction it will only matter if lines are broken without trailing whitespace, this possibility requires that reordering operate on a single line-- otherwise it can cause text to switch lines, which is not allowed.

`toString`

- (not part of the algorithm) I recommend it not remove the explicit formatting codes but leave them in the string. Clients might want to make these visible to the end user for editing purposes. It also keeps the character count of the input and output text the same, facilitating analysis of the algorithm.