Feedback on the Proposed Update for UAX#9 (unidode.org/reports/tr9-28.html)

Asmus Freytag

April 2o, 2o13

Fixed limit for levels

The rationale for fixed limit had been to make sure all implementations "top out" at the same place. Keep fixed limit. Make so high as to make "topping out" rare, but still testable. Could tie it to support or presence of isolates? Otherwise old limits?

Overly "algorithmic" definition BD16

I am a bit concerned with the current wording for BD 16. Notionally, in the spec, it's a definition (the subsection is called "Definitions") but it is written as a rule (i.e. it describes an algorithm, and, in particular, a rather detailed one.)

The problem with that is that, while on can translate the words into code, one gains no higher level understanding of what that code is supposed to achieve, and has to take on trust that the few examples represent all possible cases.

I also found the original description of the algorithm with a "stack" somewhat misleading, since it works only with those types of stack that let you get access to elements in the stack. That makes it impossible, for example, to use simple recursion to get the effect of the "stack".

I believe this approach to BD 16 is unnecessary and the language could be replaced by something like this:

> BD 16. A bracket pair is a pair of characters consisting of an opening paired bracket and the nearest following closing paired bracket such that the Bidi_Paired_Bracket property value of the former equals the latter, subject to the following constraints:
>
> - each character of the pair can belong only to one pair,
> - pairs may nest properly, but their spans may not overlap otherwise, and
> - if two potential pairs overlap, only the one starting at a lower text position is considered.
>
> Note that paired brackets can only occur in an isolating run sequence because they are processed in rule N0 after explicit level resolution. See Section 3.3.2 Explicit Levels and Directions.

If, after consideration the UTC were to conclude that this particular part of the algorithm (i.e. the part implementing the first bullet of N0) is tricky enough to warrant some additional guidance in text, I would not object if a more detailed description of an algorithm was included, for example as part of the implementation notes. However, the reference implementation should normally address the need for giving such an algorithmic example.

The term “opening paired bracket” is really awkward, because there is need to also talk about such characters that are actually “paired” so you get “paired paired brackets” which is unnecessarily cumbersome.

Corresponding small change to rule N0

Currently, rule N0 says:

- Identify the bracket pairs in the current isolating run sequence according to BD16.
- For each bracket-pair element in the list of pairs of text positions...

Instead it should probably say:

- Starting from the beginning of the run, locate all bracket pairs as defined in BD16 and record their starting and ending text positions.
- Process the bracket pairs in order of increasing starting text position.
- For each pair...

Using this wording, there is no need to introduce an explicit concept of a "list" or "stack" in the description of the logical algorithm - these are artifacts of a particular implementation and therefore do not belong in the specification of the generic algorithm.

Making this change also avoids the undefined term "bracket-pair element".

The biggest benefit in my view is that it collects all the “procedural language” inside rule N0 and keeps it out of the definitions BD16.

Optionally, the first bullet could become its own rule, but that would work only if the UTC were to follow my suggestion to not use the “cute” zero- based numbering scheme for N0. (As I mentioned at the top, I see some usability issues with that, so I would be in favor of a new series, e.g. “P1”, “P2”.).

Minor

1. The subsection “Definitions” could use subheaders to allow locating specific definitions by topic.
2. The number N0 can easily be misread as the word NO.
3. The wording in N0, item c is confusing. Existing:

   > “*Otherwise, if there is a strong type, it is opposite the embedding direction, so test for adjacent strong types as follows:*”

   The problem is the way the “if” is used: by English syntax it applies to the parenthetical corollary, and no longer works as a “flow control” in the algorithm. The attempt to rescue this by the word “so” makes the text confusing (“Otherwise…so test” ??). Here’s a better way:

   > *Otherwise, if any strong type (opposite the embedding direction) is found, test for adjacent strong types as follows:*