L2/23-014

Proposal for amendments to UAX #9

To: PAG, UTC
From: मनीष गोरेगांवकर (Manish Goregaokar)
Date: 2023-01-06

The Unicode bidi algorithm is a rather complicated beast, and the spec is such that it somewhat requires an understanding of the full spec to understand any part. There's a fair amount of *spukhafte Fernwirkungen* where the meaning of one line of an algorithm in the spec is affected by spec text that is pretty far away in a non-obvious manner.

Excising all of this is not a short-term task, but in the short term, there are a bunch of things that can be cleaned up, including fixing one spec bug. I've also included suggestions for the long term that are *not* being proposed right now but are intended as further context as to what *could* be improved given time.

Each top level section of this document addresses a particular section of UAX #9, and ends with a visual diff of that section with all proposed changes.

Table of Contents

Spec bugs around Retaining Explicit Formatting Characters

There are two specification bugs in the *Retaining BNs and Explicit Formatting Characters* section. While it is not the authoritative text on the algorithm (it proposes changes to the algorithm that, taken as a unit, are supposed to be idempotent in terms of the final output), any cases where it is not idempotent should be considered a bug and updated accordingly.

Do more precise surgery on X6

In *Retaining BNs and Explicit Formatting Characters*:

> In rule X6, remove the exclusion of BN characters. In other words, apply the rule to all types except B, RLE, LRE, RLO, LRO, PDF, RLI, LRI, FSI, and PDI.

X6 has two steps:

> - Set the current character's embedding level to the embedding level of the last entry on the directional status stack.
> - Whenever the directional override status of the last entry on the directional status stack is not neutral, reset the current character type according to the directional override status of the last entry on the directional status stack.

Applying the first one to BNs is all good. The second, one, however, causes a deviation from the algorithm.

In essence, *Retaining BNs and Explicit Formatting Characters* modifications all rely on being able to skip over BNs and explicit formatting characters in *most* of the algorithms. However, the second bullet point in this rule, if allowed to apply to BNs, materializes a new "real" entry (directionality L or R), which does not get skipped over in subsequent algorithms and can have real effects that would not occur with the unmodified algorithm.

There's a worked out example using the testcase `0605 208B 202E 2060` (`AN ES RLO BN`) in the appendix for those interested in further information. The basic idea is that the `RLO BN` in the normal bidi algorithm would simply disappear, however with the modifications instead you get an `AN ES ... R`, which eventually affects the application of N1/N2.

The proposed wording makes the X6 entry in *Retaining BNs and Explicit Formatting Characters* only apply to the first bullet point of X6.

In the long term, we should retain this section but move each of its modifications into the spec text of the relevant steps, as a conditional "if retaining explicit formatting characters, do ...".

Include X10's isolate initiator check in the fixups

X10 states, for the computation of eos:

> For eos, compare the level of the last character in the sequence with the level of the character following it in the paragraph (not counting characters removed by X9), and if there is none or the last character of the sequence is an isolate initiator (lacking a matching PDI), with the paragraph embedding level.

The check for "the last character of the sequence is an isolate initiator" should also be skipping over removed-by-X9 characters since, well, when applying the actual algorithm, they would have been removed.

This turns up in the conformance testcase `0605 2067 202C 0590` (`AN RLI PDF R`), where the first isolating run sequence ends in an RLI (and should get `sos`/`eos` as `L`), but if the PDF is retained, it will end up comparing the last level in the run (0) with the level of the next character in the paragraph (1), resulting with an odd (`R`) level.

The proposed fix adds a sentence in the X10 fixup to also skip BNs for the isolate initiator check.

Proposed spec text: bugs around Retaining Explicit Formatting Characters

Changes marked in blue

5.2 Retaining BNs and Explicit Formatting Characters

Some implementations may wish to retain the explicit directional embedding and override formatting characters and BNs when running the algorithm. In fact, retention of these formatting characters and BNs is important to users who need to display a graphic representation of hidden characters, and who thus need to obtain their visual positions for display.

The following describes how this may be done by implementations that do retain these characters through the steps of the algorithm. Note that this description is an informative implementation guideline; it should provide the same results as the explicit algorithm above, but in case of any deviation the explicit algorithm is the normative statement for conformance.

- In rules X2 through X5, insert an initial step setting the explicit embedding or override character's embedding level to the embedding level of the last entry on the directional status stack. This applies to RLE, LRE, RLO, and LRO.
- In rule X6, remove the exclusion of BN characters for the purposes of setting embedding levels. Continue not updating the character types of these characters. In other words, apply the first bullet point of the rule to all types except B, RLE, LRE, RLO, LRO, PDF, RLI, LRI, FSI, and PDI.
- In rule X7, add a final step setting the embedding level of the PDF to the embedding level of the last entry on the directional status stack, in all cases.
- In rule X9, do not remove any characters, but turn all RLE, LRE, RLO, LRO, and PDF characters into BN.
- In rule X10, when determining the sos and eos for an isolating run sequence, skip over any BNs when looking for the character preceding the isolating run sequence's first character and following its last character. Do the same when determining if the last character of the sequence is an isolate initiator.
- In rule W1, search backward from each NSM to the first character in the isolating run sequence whose bidirectional type is not BN, and set the NSM to ON if it is an isolate initiator or PDI, and to its type otherwise. If the NSM is the first non-BN character, change the NSM to the type of *sos*.
- In rule W4, scan past BN types that are adjacent to ES or CS.
- In rule W5, change all appropriate sequences of ET and BN, not just ET.
- In rule W6, change all BN types adjacent to ET, ES, or CS to ON as well.
- In rule W7, scan past BN.
- In rules N0–N2, treat BNs that adjoin neutrals the same as those neutrals.
- In rules I1 and I2, ignore BN.
- In rule L1, include the embedding and override formatting characters and BNs together with whitespace characters and isolate formatting characters in the sequences whose level gets reset before a separator or line break. Resolve any LRE, RLE, LRO, RLO, PDF, or BN to the level of the preceding character if there is one, and otherwise to the base level.

Editorial fixes to N0

N0 has a couple places where it could be clearer.

Explicitly mention BD14/BD15's effects on N0

N0 says the following:

> Note that this rule is applied based on the current bidirectional character type of each paired bracket and not the original type, as this could have changed under X6. The current bidirectional character type may also have changed under a previous iteration of the for loop in N0 in the case of nested bracket pairs.

This is in reference to the fact that BD14 and BD15 care about current bidi type, used when running BD16 to identify bracket pairs.

This is not really clear, and the second sentence is just inapplicable since each bracket pair will be processed exactly once by N0.

The proposed text below removes the quoted note and replaces it with a more targeted note on the line where N0 invokes BD16.

In the long term, this can probably be improved further by having BD16 take its arguments explicitly.

Move mention of EN/AN closer to where they are needed in N0

N0 mentions

> Process bracket pairs in an isolating run sequence sequentially in the logical order of the text positions of the opening paired brackets using the logic given below. Within this scope, bidirectional types EN and AN are treated as R.

This is relevant in the two spots where N0 is checking the contained and surrounding context of bidi characters. It's easy to miss the mentioned line.

The proposed text below retains this line of the spec but adds notes to the two spots it is useful, reminding the reader of it.

In the long term, we can turn the current line in the spec into an informational note, and change the two spots in N0 where it is needed to explicitly, normatively reference `EN`/`AN` as one of the classes being looked for, and what they are treated as.

Be explicit about the mention of `sos`

> The spec inconsistently refers to sos as a strong type in its own right as well as a useful marker for the start-of-sequence position.

Therefore, test for an established context with a preceding strong type by checking backwards before the opening paired bracket until the first strong type (`L`, `R`, or `sos`) is found.

This is a wider distinction that the spec could be making more, but in this case it affects the control flow of the algorithm and ought to be more explicit.

The proposed text below explicitly writes that the control flow of the algorithm is to look for a strong type, and, if not found, use the type of `sos`.

In the long term, we can look for all mentions of sos and eos in the spec text and replace them with "the `sos/eos` of *sequence*", adding control flow where necessary.

Clarify that the preceding strong type is always used

N0 has a step that states:

> If the preceding strong type is also opposite the embedding direction, context is established, so set the type for both brackets in the pair to that direction.
>
> ...
>
> Otherwise set the type for both brackets in the pair to the embedding direction.

There are only two possible values for the preceding strong type and the embedding direction (`L` or `R`). With that information, this rule has the effect of unconditionally setting the type for the brackets to the preceding strong type.

It seems useful to spell it out this way so that the concept of "established context" is, well, better established, but as stated the redundancy gives off the impression that there might be something the reader may have missed, perhaps a secret third type. This compounds with the previously mentioned problem with "(`L`, `R`, or `sos`)".

The proposed text below retains this spec text but adds a note specifying what the eventual effect is.

In the long term, we can investigate collapsing these points

Clarify that all searches are within the isolating run sequence only

3.3.4 *Resolving Weak Types* and 3.3.5 *Resolving Neutral and Isolate Formatting Types* both operate within isolating run sequences. They are not supposed to perceive the existence of characters outside the isolating run sequence. Even if the isolating run sequence has "gaps", those gaps are to be skipped over when searching for sets of "adjacent" characters.

This is a concept that the spec could be more clear about in general, but particularly in the case of N0, it has a step that states:

> Inspect the bidirectional types of the characters enclosed within the bracket pair.

Most of the spec uses text like "search forwards" and "search backwards" which can be more easily seen to be applying to an isolating run sequence, but "characters enclosed in the bracket pair" more strongly evokes a sense of operating on the *paragraph*.

The proposed text below adds a note reminding implementers to only look within isolating run sequences.

In the long term, we can add named concepts for searching within isolating run sequences, and use them everywhere. Also we can mention *somewhere prominent* that isolating run sequences can have gaps that contain meaningful characters covered by other sequences.

Proposed spec text: editorial fixes to N0

Changes marked in blue

3.3.5 Resolving Neutral and Isolate Formatting Types

In the next phase, neutral and isolate formatting (i.e. NI) characters are resolved one isolating run sequence at a time. Its results are that all NIs become either **R** or **L**. Generally, NIs take on the direction of the surrounding text. In case of a conflict, they take on the embedding direction. At isolating run sequence boundaries where the type of the character on the other side of the boundary is required, the type assigned to *sos* or *eos* is used.

Bracket pairs within an isolating run sequence are processed as units so that both the opening and the closing paired bracket in a pair resolve to the same direction. ~~Note that this rule is applied based on the current bidirectional character type of each paired bracket and not the original type, as this could have changed under X6. The current bidirectional character type may also have changed under a previous iteration of the *for* loop in N0 in the case of nested bracket pairs.~~

N0. *Process bracket pairs in an isolating run sequence sequentially in the logical order of the text positions of the opening paired brackets using the logic given below. Within this scope, bidirectional types EN and AN are treated as R.*

- *Identify the bracket pairs in the current isolating run sequence according to BD16.*
  Note that BD14 and BD15 identify bracket characters based on the current bidirectional character type of each paired bracket and not the original type, as this could have changed under X6.
- *For each bracket-pair element in the list of pairs of text positions*
  a. *Inspect the bidirectional types of the characters enclosed within the bracket pair.*
  b. *If any strong type (either L or R) matching the embedding direction is found, set the type for both brackets in the pair to match the embedding direction.*

Note: EN and AN should be treated as a strong R type when searching within the brackets.
Note: Like all other operations where we are scanning text whilst processing neutral and weak characters, implementations should take care to only scan for characters that are contained in the isolating run sequence, which may have gaps covered by other sequences.

o **[** e **]** o → o **e** e **e** o

o **[** o e **]** → o **e** o e **e**

o **[** NI e **]** → o **e** NI e **e**

c. *Otherwise, if there is a strong type it must be opposite the embedding direction. Therefore, test for an established context with a preceding strong type by checking backwards before the opening paired bracket until the first strong type (L, R~~, or sos~~) is found, using the value of sos if there is none.*
Note: EN and AN should be treated as a strong R type when searching for established context.

1. *If the preceding strong type is also opposite the embedding direction, context is established, so set the type for both brackets in the pair to that direction.*

   o **[** o **]** e → o **o** o **o** e

   o **[** o NI **]** o → o **o** o NI **o** o

2. *Otherwise set the type for both brackets in the pair to the embedding direction.*

   e **[** o **]** o → e **e** o **e** o

   e **[** o **]** e → e **e** o **e** e

   Note: These two steps put together will unconditionally set the type for both brackets to the preceding strong type, as there are only two possible values (L and R).

d. *Otherwise, there are no strong types within the bracket pair. Therefore, do not set the type for that bracket pair.*
e **(** NI **)** o → e **(** NI **)** o
Note that if the enclosed text contains no strong types the bracket pairs will both resolve to the same level when resolved individually using rules N1 and N2.

- *Any number of characters that had original bidirectional character type NSM prior to the application of W1 that immediately follow a paired bracket which changed to L or R under N0 should change to match the type of their preceding bracket.*

Editorial fixes to BD16

BD16 has a couple places where it could be clearer.

Explicitly return an empty list when the stack is full

BD16 states

> If an opening paired bracket is found and there is no room in the stack, stop processing BD16 for the remainder of the isolating run sequence

however, the algorithm is expected to have a result, and it's not clear what that result should be. From the reference implementation it seems like the expected behavior is to return an empty list.

The proposed text below explicitly returns an empty list here.

Empty stack case

BD16 states

> Declare a variable that holds a reference to the current stack element and initialize it with the top element of the stack.

This is in a context where the stack may be empty: when you encounter a bare closing bracket that has no opener. It is unclear what to do in this case, though there is one rather obvious answer, since Step 5 already handles unpaired closing brackets in cases where the stack started out nonempty.

The proposed text below explicitly suggests jumping to Step 5 when the stack is empty.

Name the result list

In BD16, there are two mutated variables: "a stack" and "a list of elements". They're named according to their types rather than what they do, and as rather similar collections the terms can be slightly confusing.

The proposed text below explicitly names the list according to its purpose, as "list of resulting bracket pairs".

Proposed spec text: editorial fixes to BD16

Changes marked in blue

BD16. A *bracket pair* is a pair of characters consisting of an opening paired bracket and a closing paired bracket such that the Bidi_Paired_Bracket property value of the former or its canonical equivalent equals the latter or its canonical equivalent and which are algorithmically identified at specific text positions within an isolating run sequence. The following algorithm identifies all of the bracket pairs in a given isolating run sequence:

- Create a fixed-size stack for exactly 63 elements each consisting of a bracket character and a text position. Initialize it to empty.
- Create a list of resulting bracket pairs for elements each consisting of two text positions, one for an opening paired bracket and the other for a corresponding closing paired bracket. Initialize it to empty.
- Inspect each character in the isolating run sequence in logical order.
    - If an opening paired bracket is found and there is room in the stack, push its Bidi_Paired_Bracket property value and its text position onto the stack.
    - If an opening paired bracket is found and there is no room in the stack, stop processing BD16 for the remainder of the isolating run sequence and return an empty list.
    - If a closing paired bracket is found, do the following:
        1. Declare a variable that holds a reference to the current stack element and initialize it with the top element of the stack. If the stack is empty, skip to step 5.
        2. Compare the closing paired bracket being inspected or its canonical equivalent to the bracket in the current stack element.
        3. If the values match, meaning the two characters form a bracket pair, then
            - Append the text position in the current stack element together with the text position of the closing paired bracket to the list of resulting bracket pairs.
            - Pop the stack through the current stack element inclusively.
        4. Else, if the current stack element is not at the bottom of the stack, advance it to the next element deeper in the stack and go back to step 2.
        5. Else, continue with inspecting the next character without popping the stack.
- Sort the list ~~of pairs of text positions~~of resulting bracket pairs in ascending order based on the text position of the opening paired bracket.

Note that bracket pairs 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*.

Editorial fix to W6

Predicates straddling loops considered harmful

W4-W6 have the following contents:

> W4. A single European separator between two European numbers changes to a European number. A single common separator between two numbers of the same type changes to that type.
>
> W5. A sequence of European terminators adjacent to European numbers changes to all European numbers.
>
> W6. Otherwise, separators and terminators change to Other Neutral.

The W, X, and N rules are logically expected to be run one at a time on the entire isolating run sequence, i.e. each W step is logically a loop on the characters in the sequence. Implementations may choose to collapse these loops for performance provided they do the appropriate level of bookkeeping, of course, but they're still *logically* individual loops.

Given that, having an "otherwise" in a *separate loop* from its predicate is ..... highly confusing. Trying to compare this with an implementation gave me the same mental experience as the first time I learned about Duff's device.

There's a pretty clear answer as to what this rule is trying to say, "Any separators and terminators left over (after applying W4 & W5) change to `ON`.". However it's a bit less clear when comparing the spec with implementations that attempt to collapse these loops, and there's a plausible interpretation which is "apply the W4 predicate *again* and if it doesn't apply, change to `ON`". Which doesn't work, since an `ET CS EN` (no W4 application) can change to an `EN CS EN` via W5 and then the W4 predicate suddenly applies again.

The proposed text below changes W6 to not use the word "otherwise" and uses "All remaining" instead.

Proposed spec text: editorial fix to W6

Changes marked in blue

W6. *~~Otherwise,~~All remaining separators and terminators (after the application of W4 and W5) change to Other Neutral.*

AN ET → AN ON

L ES EN → L ON EN

EN CS AN → EN ON AN

ET AN → ON AN

Appendix: Workthrough of spec bug in X6

This is a workthrough of the bug explained in Do more precise surgery on X6.

An example from the testcases is `0605 208B 202E 2060` (bidic). They start out with classes `AN ES RLO BN`.

According to the regular algorithm, after running X9, this should really just be `AN ES` since the `RLO BN` will get removed. During the weak rules, this becomes `AN ON`, and during N2 resolution it becomes `AN L` since the run sequence is LTR on both ends. We should end with `AN L`, and we see that happening in the bidic output (which chooses to include the "removed" characters, but does nothing with them).

According to the fixed up algorithm, here's what happens:

- After applying X6, the `BN` becomes `R` due to the override being on the stack (`AN ES RLO R`)
- After X9, the `RLO` becomes `BN` (`AN ES BN R`)
- After W6, it the `ES` becomes `ON` as usual, and so do the `BN`s after it (`AN ON ON R`)
- N1 applies since the `ON`s are sandwiched between `AN` (=`R`, in this context) and `R` (`AN R R R`)

This gives us `AN R R R`

What *should* happen is:

- After applying X6, the `BN`s stay `BN`s and thus ignorable (`AN ES RLO BN`)
- After X9, the `RLO` becomes `BN` (`AN ES BN BN`)
- After W6, it the `ES` becomes `ON` as usual, and so do the `BN`s after it (`AN ON ON ON`)
- N1 does not apply, the `ON`s are sandwiched between an `AN` (=`R`) and the embedding direction (`L`)
- N2 sets to the embedding direction, giving us `AN L L L`

Conceptually, given that there's so much of the fixed up algorithm that relies on the `BN`s being "the skippable ones", X6 changing `BN`s to the override direction seems to throw a wrench in the works of a lot of the algorithm. It's worth noting that there is only one other place in the "fixed up" algorithm where BNs are modified (W6, N0-N2), and it is for absorbing the class of adjacent characters, so that sequence properties can be maintained. Whereas, the current text for X6 is able to modify `BN`s in a way that can materialize a strong class in a context where there previously was none (just overrides that would have, in the original algorithm, dissipated entirely with no effect since they would have no meaningful characters to actually paint with their overrides).