L2/14-040

Matitiahu Allouche and Aharon Lanin

Subject: Issues with the current version of the UBA as expressed in UAX#9 version 29

Issues with the current version of the UBA as expressed in UAX#9 version 29

GENERALITIES

As we all know, bidi matters are often challenging. The UBA is a major element of bidi support, and because of its complexity, its formulation in UAX#9 may not be always perfectly clear, and its strict application may produce results unexpected by the authors or incorrect results in some cases. We describe below four such issues.

The question is: what do we do with the known unexpected or incorrect results?

Personally, we think that when UAX#9, and therefore the reference implementations, depart from its authors' intention in a non-debatable way, it should be corrected, and so should the reference implementations afterwards. Others may argue that the cases found until now are very unlikely to happen in real life. We reply: all the better! Fixing UAX#9 will not break any existing documents, but it will make sure that the infinity of future documents will not suffer from the current imperfections.

Moreover, we have to worry about not only future documents, but future implementers and future editors of UAX#9. Leaving it with internal inconsistencies and clear bugs is not healthy. Implementers will be hard pressed to implement in their code what they know to be bugs. Editors could be misled into making unjustified assumptions or copy/paste our bugs into new changes that have worse consequences.

We simply do not see a real downside to cleaning up the spec in ways that are not likely to affect any present documents, soon after the original spec came out. Don't forget that currently, the only implementation in wide-spread use is Microsoft's. It does not follow UAX#9 in the problematic cases described below, it does the intended thing (except when related to isolates, since it does not implement isolates at all). If it is changed to follow the current UAX#9, we have instability. If it isn't, and neither is UAX#9, we have lack of interoperability with other implementations (unless they all choose to ignore UAX#9's bugs). The least instability / lack of interoperability will result if UAX#9 is changed, and soon.

Anyway, We have appended to this document argumentations in favor of leaving alone the UBA and UAX#9 as much as possible, so that the reader can see the different points of view and form one's opinion.

ISSUE #1: NSM following a paired bracket

Consider the following string: a(b)'
where the apostrophe represents an NSM.
When presented within a RTL paragraph, all characters receive bidi type L (leading to embedding level 2) by application of rule N0C1, except the NSM which has been transformed to an ON by application of W1 and receives embedding level 1 by application of N2.
The consequence is that the NSM will be displayed on the left side of a(b), or maybe on the a itself, far away from the right parenthesis that it was supposed to affect.
IMHO, this result is incorrect and was not intended by the authors of the BPA definition.
It may seem that the occurrence of an NSM after a bracket should be very unlikely. We can see a use case not too farfetched. In statistics formulae, it is common to represent the average of an expression by adding a bar over it. In

plain text, the bar could be obtained by adding U+0304 Combining Macron or U+0305 Combining Overline after each character of the expression. If the expression is something like a(b), we get exactly the case in question.
Note that Windows (tested on version 8.1) in Notepad and in IE (tested on version 11) implements the BPA but assigns to the NSM the same embedding level as to the parenthesis it follows.

Our suggestion: change UAX#9 to make NSM affecting paired brackets receive the same level as the brackets. For instance, add a note to the text following N0 saying "when the type of paired brackets is changed to L or R as the result of this rule, original NSM characters (characters classified as NSM before applying rule W1) immediately following the brackets must have their type changed similarly".

ISSUE #2: Brackets within the scope of LRO or RLO: should they be considered brackets for the BPA?

Andrew and Laurentiu have stated that it was never their intent that the BPA should be applied to them.
Nevertheless, careful exegesis of the current UAX#9 shows that brackets whose bidi type has been changed to L or R as the effect of X6 should still be considered as brackets while applying N0.
In most cases, when everything from the opening bracket to the closing bracket and in between is part of the same override, it makes no difference whether the brackets are paired or not, all the characters are changed to L or R according to the override status. A difference may exist when the opening and closing brackets are not both in the scope of an override, like in the following string:

LRO { PDF LRE R } R

If the opening bracket is treated as a bracket, the brackets are paired and both are changed to L by applying N0c2. If the opening bracket is not considered as a bracket for the BPA, the closing bracket will be treated as R by applying N1. This modifies the presentation substantially.
Our suggestion: brackets within an override should be changed to L or R and ***NOT*** be subject to the BPA. This can be achieved by adding the words "and whose bidirectional character type (property Bidi_Class) is not L, R or AL" to the definitions of opening and closing paired brackets in BD14 and BD15.
In N0, a note will be added saying that this requirement is applied to the current character type after X6, not the original one.
Reasons:
1) This seems easier to understand: either the bracket is affected by the override for all its characteristics, or it is not. If it becomes an L or an R, it is no longer a bracket.
2) There are 2 use cases that we know of for overrides:
2a. Part numbers: in this case, the UBA is disabled because there is no meaning to the characters, at least no meaning that a computer algorithm may guess. Even if there is a bracket somewhere in the part number, there is very little chance that it is the logical match of a bracket outside the part number.
2b. Legacy Hebrew (maybe also Arabic) text in visual order: the weird case would happen if some text in visual order is concatenated with some text in logical order. Most likely the 2 pieces of text have different origins. It seems very unlikely that the opening bracket in the visually ordered text be logically related to the closing bracket in the logically ordered text.
Admittedly, we are dealing with heuristics, not certainties. However, the heuristics should handle properly the most likely possibility. In this case, it is that the 2 brackets are not related.
Note: neither Notepad or IE11 treat the brackets as a pair in this case.

ISSUE #3: Isolate initiators and terminators within the scope of an override

Consider an isolate (or a series of isolates with nothing between them, or an unmatched PDI) inside an override, surrounded on both sides by embeddings, with nothing in between it and them. For example take the following (the paragraph direction does not affect the results):

[RLO]a[LRE]b[PDF][LRI]c[PDI][LRE]d[PDF]e[PDF]

Currently the spec and the reference implementations resolve a and e to level 1, and b, c, and d to level 2, which is all fine. The issue is that they also resolve the LRI and PDI to level 2, not to level 1.
Resolving to level 2 gives the following visual order:

ebcda

Resolving to level 1 gives a different visual order:

edcba

If we take the same example, but replace the [LRI]c[PDI] with a neutral, e.g. an asterisk, it is changed to R by the RLO according to X6 and gets resolved to level 1.

This is a problem because as stated in various non-binding passages, an isolate is supposed to have the same effect on the ordering of the text surrounding it as a neutral character.

Now, how is it that we wind up resolving to level 2?

The LRI and PDI's explicit embedding level is 1. The explicit embedding level of the character before the LRI (b, after X9) is 2, as is that of the character after the PDI (d). Thus, the LRI and PDI are a separate isolating run sequence (as is c, but that is irrelevant). This sequence's sos and eos are L, since the explicit levels of the characters preceding and following it are both 2 (which is higher than its 1). The problem is that X1-X8 leave the bidirectional character type of the LRI and PDI unchanged. X6 does not change it to R, as it would for a neutral character, because X6 states that it does not apply to B, BN, RLE, LRE, RLO, LRO, PDF, RLI, LRI, FSI, and PDI. As a result, N1 still applies to the LRI and PDI, and being surrounded by L's (sos and eos), they become L. Thus, they wind up resolving to level 2. This outcome was unforeseen (at least by Aharon), and is basically a bug in the algorithm spec.

Our suggestion:

The following bullet has to be added to X5a, X5b, and X6a:

- If 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.

In X5a and X5b, the new bullet would be inserted between the current first and second bullets. In X6a, it would be appended as the last bullet.

Rationale: the general idea of isolates is that they define a microcosm between the initiator and the terminator, while the initiator and terminator themselves are part of the outside isolating run sequence. As such, they should be treated as regular characters of this sequence, and if the current directional override status is not neutral, the isolate initiator and terminator will become L or R just like any non-formatting character.

Issue #4: nested bracket pairs behave surprisingly

Let me start with a rather weird case:

```
Case 1 (paragraph direction LTR)
Raw data:  202D.202E.0062.202C.007B.202C.202A.05D1.007D.05D2
           LRO  RLO  b    PDF  {    PDF  LRE  H1   }    H2
Levels  :  x    x    3    x    3    x    x    3    3    3
```

The only char within the brackets is a Hebrew letter, whose direction is opposed to the embedding direction (set by the LRO and LRE), so that N0c applies. The context is set by the RLO/b/PDF preceding the opening bracket, whose effect is like a R (by setting the sos), so that N0c1 applies and the brackets receive level 3.

Now another case, which is identical to the first one except that a pair of matching parentheses was added around the curly brackets.

```
Case 2 (paragraph direction LTR)
Raw data:  202D.202E.0062.202C.0028.007B.202C.202A.05D1.007D.0029.05D2
           LRO  RLO  b    PDF  (    {    PDF  LRE  H1   }    )    H2
Levels  :  x    x    3    x    2    2    x    x    3    2    2    3
```

We would expect that the added pair of parentheses should be treated exactly like the pair of curly brackets in the previous case. In fact, we expected both the parentheses and the curly brackets to receive level 3, but they don't. It is caused by the opening curly bracket getting character type L in X6 due to the LRO, and making N0b apply to the parenthesis preceding it.

The result conforms to the letter of UAX#9 but is quite surprising.

Suggestion: if our suggestion for resolving issue #2 (brackets within the scope of overrides) is accepted, issue #4 will also go away.

CONCLUSION

All the 3 changes that we suggest tend to decrease the number of surprising manifestations of the UBA: NSMs following bracket (issue #1) are not dissociated from the bracket; brackets within override (issue #2) are overridden from being paired brackets; isolate initiators and terminators (issue #3) always affect their surroundings like neutral characters.
Since it is all about simplification, we expect that the changes to the implementations should be quite easy. We can only vouch for the ICU implementation, for which we can say that the changes would involve no more than 2 hours work. After that come the changes to the test sets, and the extensive regression testing, but this is mostly an investment of computer time, not manpower.

We copy below opinions expressed by various experts while discussing the issues mentioned above. It goes without saying, but we say it all the same, that our own opinion differs substantially from part of what appears below, mainly in that we are in favor of fixing what needs to be fixed, after following the proper change process, rather than sticking with the status quo out of what looks to me like timidity. Anyway, the enlightened reader will have the elements to form his own stance on the subject matter.

Asmus Freytag about issue #1 (November 1st, 2013)

My vote is for
2. Do nothing because the case in question is not a real life case and there is no known example of an NSM affecting a bracket. If this is the option chosen, the only concern is to avoid generating such cases in the tests.
Plus this consideration:
2A: Fiddling with the algorithm, except in cases where it's buggy for real input (and esp. common input) is to be avoided at all costs.

Ken Whistler about issue #1 (November 1st, 2013)

I agree with Asmus that #3 Modify the UBA ought to be off the table – at least for a non-useful edge case like this one.
Discovering this problem at this point is yet another example of the basic axiom of bidi: the algorithm is never perfect. (As in never, never, never, ever, ever, never.) Tweaking UBA 6.3 by reordering rules, ostensibly to fix just *this* wee problem is absolutely guaranteed to break something else, which in turn would generate a need for another tweak. And I'd be willing to stake a $100 bet on that right now. ;-)
I don't think we should quite "do nothing", however. I think the behavior of an NSM directly following a matched right bracket ought to be documented, so people aren't surprised.
However, I am not in favor of avoiding the generation of such cases in the tests. Such cases *are* to be covered by the algorithm, by definition, and the outcome is what it is. So in my opinion, the current behavior of Cref and Jref is correct, whatever one might think about the "best outcome" a priori. Note that as Laurentiu indicated, these cases were encountered in the testing phase, because both Cref and Jref were tested exhaustively out to n=8. What that means is that they encountered many 1000's of such cases – nay, likely millions of them. The fact that the particular issue doesn't show up in the existing BidiTest.txt is presumably the result of it only enumerating permutations out to 4 and not dealing specifically with bracket pairs.
So my favored option is:
2B. Do not change the UBA algorithm, but in the next revision, explicitly *add* the problematical cases to BidiCharacterTest.txt to ensure that people see and test the behavior. Additionally, document the issue in UAX #9, 7.0 (and earlier in FAQ material). And correct the ICU4C and ICU4J behavior at the earliest possible opportunity, because as written now, they claim conformance to UBA 6.3 but in fact are *not* conformant to the algorithm. And

finally, document what the author's workaround is in the extreme edge case when they absolutely *do* want the bracket matching *and* insist on having a combining mark apply to the right bracket.

Asmus Freytag about issue #2 (November 5th, 2013)

This is a question as to what is desirable behavior.
If text runs display correctly for a human reader after being marked up with non-isolating embeddings and overrides then brackets should match to the reader. Moreover, if text was created using an editor with PBA and is viewed using PBA then the results should match.
Both of these considerations would mean that allowing brackets to match shouldn't matter.
The case we would be looking for is one where the matching produces something that doesn't work in the text, and where that is a result of very natural (and/or common operations during editing, not because someone who can run the bidi algorithm in their head set up a very complex corner case).
So, is it common, today, that texts have unmatched brackets, and that these unmatched brackets are controlled using overrides or embeddings...?
Am I missing a common case where ignoring embeddings is going to make matters look worse? (Given that the PBA is not necessarily perfect, only a pretty good heuristic).
Or do we have any legacy implementations of the PBA that match what the ICU code does?
If neither, I think this is another one of the cases that should be documented but otherwise the PBA should remain stable.

Aharon Lanin about issue #2 (November 5th, 2013)

Mati's case has two interesting qualities. One is that the opening bracket is in an override. The other, even more interesting thing, is that the closing brackets is *in a separate embedding* that - because it is immediately adjacent and has the same embedding direction - forms a single level run and thus are in a single isolating sequence with the first embedding. Thus, despite the brackets being in two separate embeddings, N0 still applies to them. And, since the second embedding is *not* an override, N0 actually has an effect on the ordering (something that cannot happen when the paired brackets are within a single override).
Now, the question that Mati and Asmus basically pose is whether this is a good thing or a bad thing. If it is a very bad thing, we could conceivably use the unintended ambiguity of the term "character type" in X6 in order to prevent N0 from applying when either bracket is in an override.
I think that this question is irrelevant, and would be so even if it were still April and we had not yet published 6.3.
The reason I think so is that the characters in one embedding affecting the ordering of characters in an immediately adjacent embedding, while being basically a very bad thing, is in no way new to 6.3. Consider the following (the paragraph direction does not matter):

[RLE]IN 1999, I READ a.[PDF][RLE]1999 WAS A GOOD YEAR.[PDF]

The visual ordering of this, in both 6.2 and 6.3, is:

.RAEY DOOG A SAW a .1999 DAER I ,1999 NI

The fact that N0 applies when some of the characters it looks at are in one embedding, and the rest are in an adjacent one is certainly no worse than the fact that this is the case in N1 or W1 or W2 etc., which affect the ordering a lot more frequently than N0.
Because of backward compatibility, we could not change that in 6.3, and instead added isolates, where the characters inside an isolate never have any effect on the ordering outside the isolate.
Thus, I am against changing the reference implementation for the current UAX#9 formulation. However, I do think that the
UAX#9 bug that applies N0 when brackets appear inside an override should be fixed even (and even especially!) if it *never* makes any difference (as it does not, except in the special case of immediately adjacent embeddings), for the sake of cleanliness.
Furthermore, relevant test cases need to be added to BidiCharacterTest.txt in the next Unicode release. Besides the special case above, they should include simple ones where the bracket pair is within an override (and N0 does not actually have any effect), e.g. [RLO]a(b)[PDF]. BidiCharacterTest.txt does not currently seem to have any such cases.

Mark Davis about issue #3 (November 11th, 2013)

My personal opinion is that because (a) it will effect a very small set of cases, and (b) because it will take a little while anyway for programs to switch to the new UBA, that we should be willing to make the change.
We could gather consensus to do it, and if reached, then issue a corrigendum.

Laurentiu Iancu about issue #3 ((November 12th, 2013)

I think that a corrigendum, if issued, (or a revision of UAX #9, for that matter) has the overhead that it should be accompanied by the corresponding changes made to both reference implementations, and accompanied by another round of consistency testing to ensure that the two updated implementations are again in synch.
I realize that not fixing the spec means that an isolating run sequence does not behave like a neutral in all possible contexts, which in principle is contrary to the role of isolates. However, given the special conditions for that violation (isolates tightly flanked by embeddings and enclosed within overrides), and the side effects described by Aharon for the suggested fix, and the overhead of implementing it (in spec, code, and testing) especially during the loaded Unicode 7.0 release cycle, it would seem more practical to live with the imperfections due to the complexity (the interactions between the parts) of the algorithm and just document them.
So an alternative would be to add test cases to BidiCharacterTest.txt and/or BidiTest.txt which exhibit the problem, and to add text to UAX #9 for implementers to be aware of it – as was suggested for the parentheses issue that Mati reported previously.

Ken Whistler about issue #3 (November 12, 2013)

In my opinion, formally addressing this by issuing a Corrigendum might be the worst of several bad options, particularly because this is not the *only* wart that has now been discovered. A Corrigendum is likely to lead to proliferations of implementations that do not implement the Corrigendum and those that do, which is worse than if everybody implements with the flaw as is.
I think the better course is to accumulate and document all of the flaws identified during these early days. These can be accumulated in the proposed update for UAX #9, as soon as folks would get off their collective bu..., err kiesters to prepare it. In that context, with *all* issues spelled out and documented, and with implications for various proposed changes spelled out with (explicit) pros and cons, we would then have the appropriate context to decide which (if any) of the flaws need to be formally addressed by a change in the algorithm (and accompanying updates to bidi reference implementations and test cases) and in what time frame.
Just jumping to issue an immediate Corrigendum for a single issue discovered like this would be a major mistake, IMO.

Asmus Freytag about issue #3 (November 12, 2013)

I'm of the firm opinion that the entire bidi algorithm is a heuristic.
That is, even if designed and implemented correctly it will occasionally do something that is counter-intuitive to authors, or that requires explicit work-arounds to achieve the intended layout.
Given that, the most important feature the algorithm can have is a *single* definition, so that authors can have the confidence that their work-arounds are supported on every platform.
Minor nits on the design or the algorithm, esp. if they require large textual updates should by definition be ruled out for fixes.
The most important goal at this point is to have a stable basis on which people can migrate.
This goes for the paired brackets as well as the isolates - the goal is to get people to implement them consistently, not to give them a receding target to chase.
I am opposed to presenting a proposed update that treats these things as suggested fixes. The proper approach is to add them

to the implementation notes as "gotcha's" in the way the spec works (as well as the test cases, of course).
If - and only if - there is, at some future time, a major hidden flaw that is discovered, I would support a UTC effort at piggybacking some other cleanup.

Mark Davis in answer to Asmus Freytag (November 11th, 2013)

I think we understand your position. Mine is somewhat different, as I articulated above. It is up to the UTC to review any problems we turn up as implementations work on this and tests are refined, and figure out the severity and expected frequency. We have a small window when implementations are working on support of UBA 6.3.0. During this window, after carefully weighing the impact, the UTC could decide to make a change. For anything that didn't have sufficient impact, we'd simply document as gotchas and be done with them.
And as noted above, whatever we do we need to supply test cases that show how the UBA handles them.

Asmus Freytag in answer to Mark Davis (November 13th, 2013)

Effectively, I believe the window is closed.
While it may take some time for 6.3 to filter down to implementers, it will take the same amount of time for 7.x to filter down to implementers. And if the changes are less obvious/less drastic, the rate that they will filter down to implementers will tend to be slower, not faster.
There's a clever book out there "The half-life of facts", which is worth reading.
The inevitable result of making *any* changes to the algorithm will be that there are multiple, co-existing versions. This begins with the reference implementation. For me to be able to implement the PBA took a certain threshold of criticality. That threshold is unlikely to be generated by a small tweak. Possibly, this can be overcome for the reference implementation, but the same issue is going to play out in every shop working on the UBA.

Laurentiu Iancu about issue #4 (December 29th, 2013)

I agree that the two sequences (case 1 and 2) should ideally resolve the same way, whether the brackets are single or double. (A match in resolved levels between a sequence with single brackets and a corresponding sequence with doubled brackets is precisely one of the invariants that I had on my list for intent-based validation testing of the spec, had there been time for that.)
Perhaps a higher-level question is whether the brackets which are overridden by LRO, in both case 1 and 2, whose updated types are L instead of ON, may arguably not participate in bracket matching, because they are not both ON (by the time N0 is reached), and the N rules fundamentally deal with neutral types. In particular, in case 1, the result of N0 changes the type of the opening brace from L (previously set by the LRO) to R, which is contrary to the L override. However, I think that the way the spec is written today, the brackets are subject to bracket processing regardless of the override, and the reference implementations follow the current spec, as far as I can tell.

Andrew Glass about issue #4 (December 31st, 2013)

The intent of the N0 rules is to handle neutral brackets. “In the next phase, neutral and isolate formatting (i.e. NI) characters are resolved one isolating run sequence at a time.” So, pace Laurentiu, I would say that a bracket should not be treated by N0 if it is not neutral. And therefore a bracket pair should not be discovered when either or both are subject to an override such that they are not of type ON.
However, given that this is the way the reference implementations have been written, and given the nature of these problematic cases, I’m not sure if it makes sense to change the reference implementations at this time. Perhaps it is better to clarify or annotate the rules to make it clear that N0 *does* apply to brackets regardless of whether or not the brackets themselves are subject to an override. Any user who wishes to accomplish split behavior for their brackets would be able to make use of isolates.

Ken Whistler about issue #4 (January 2nd, 2014)

While I agree that Mati’s second case is puzzling on the face of it, I find myself

agreeing with Laurentiu here about interpretation of the spec as currently written.
Although it may have been everybody's *intent* that N0 rules only handle "neutral brackets", as I read it (and implemented it), and apparently as Asmus also read it (and implemented it), I see:
N0. Process bracket pairs in an isolating run sequence sequentially…
* Identify the bracket pairs in the current isolating run sequence according to BD16.
And then referring back to BD16 for the definition:
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 … etc., etc.

I read that as a clear mandate to scan the isolating run sequence based on the Bidi_Paired_Bracket property value identifying pairs by the further qualifications in BD16, which nowhere indicate that there are any constraints in that matching based either on original Bidi_Class values or on the current state of the resolved Bidi_Class values at this point in the rule application. Obviously Asmus read it the same way for his implementation.
In the C reference implementation, that test is implemented with the following code fragment (removed by Mati).
. . .
Now, I clearly also have accessible the current Bidi_Class of the character in question, and it would be reasonably trivial to extend this checking here to guarantee that matching brackets would only be identified when their Bidi_Class was still ON. But I will not make that change at this point for a number of reasons:
1. A reasonable interpretation of the current spec would claim that Asmus and I are correct.
2. Any change to one reference implementation must be coordinated to ensure that a matching change would be made in the other (and both be properly versioned.)
3. Neither reference implementation should be substantively changed *between* versions of the UAX itself, IMO.
4. Most importantly, this odd case, like each of the previously identified odd cases that Mati (or others) have turned up need to first be documented and discussed by the UTC, to determine whether any of them require either a clarification of the language of the spec and/or a change in behavior of the algorithm. And separately, depending on outcome, how this should be handled for the reference implementations.
With shipping implementations, and with reference implementations that have been publicly posted for months now, we are well beyond the point where we can just fix bugs like this without going through a full process of deliberation in the UTC to air out all the implications of changing anything.
I would like to see this particular case added to a document submitted to the February UTC for consideration of each of the anomalies identified to date, so we can decide whether to proceed simply with clarifying documentation about odd behavior in edge cases in the spec or need to make further actual changes to either the reference implementations and/or the UBA itself – and if so, exactly which ones.

Asmus Freytag about issue #4, but also in general (January 6th, 2014)

Making the UTC aware of the issues discovered is only proper, and, formally, the UTC has the power to either maintain the status quo or make any amendment it chooses.
If I were arguing this before the UTC, I would argue strongly for maintaining the status quo as the least disruptive. Yes, it is disappointing to have the result differ from an "ideal" outcome, but my argument in the past has been, and would be in this case, that the bidi algorithm is special in that a) it is required, and that b) authors are explicitly intended to rely on the fact that the implementation in their authoring environment matches that in the reading

environment.
In essence, only universally compatible implementations can guarantee portability of bidi text.
In that context, "fiddling" with the algorithm will inevitably lead to incompatible implementations; it is a sad fact that many derivative standards as well as implementations do not automatically (or correctly) upgrade to the latest version, but instead the selection of a reference version, frozen in time, is common.
Now, the addition of the PBA itself was an incompatible break; as was the handling of isolates. The UTC did the right thing by insisting that both be done at the same time, so as to not make the UBA a shifting target over several versions (or years).
However, it establishes the precedent that, given sufficient advantages in usability for authors or end users, the UBA can be upgraded. At the same time, there's another precedent, that of the broken (but frozen) mirroring property for the decorative brackets. That property was kept stable, because the small increase in usability far outweighed the disruption of having divergent implementations (and therefore uncertainty to authors which character codes to use), or, worse, the existence of data created to different expectation of mirroring behavior.
In essence, this second precedent establishes the principle that "data does not get upgraded".
A final consideration is that the UBA, even with its recent amendments is not "perfect" -- the UBA is essentially a pretty good heuristic, but there remain complex cases where an author is expected to use bidi marks of overrides to achieve correct layout.
In essence, the benefit of the UBA is not in being perfect, but in being consistent.
When the UTC considers this issue, it would be helpful if the paper described not only the test cases, but reminded the UTC of these precedents and the importance of them to the UTCs cost-benefit analysis.
At the moment, all we have is test cases (unless I missed something). That is, we don't have any compelling real-world examples that are expected to occur frequently enough to where UTC could conclude that *practical* usability of the algorithm is sufficiently degraded so that a fix, even if it leads to the coexistence of both data and implementations based on different versions would still, on balance, far improve the user experience.
In other words, the lack of perfection would have to outweigh a lack of consistency introduced by the fix.
I would look for, in particular, examples where the test cases could be (and are likely) to be created in automatic text generation, such as report generators. A scenario that is easily handled by an author, by use of marks and over, may a trap for report generators and not allow easy work arounds. If that were to be the case, if the case could reasonably be expected to occur at least as frequently as other issues requiring work arounds, then, and only then, would I counsel the UTC to consider of even putting the option of making an amendment on the table.
That's a pretty high bar, but to me, it follows from the special nature of the UBA and the existing precedents.

<end of quotations>