L2/23-008

UTC #174 properties feedback & recommendations

Markus Scherer / Unicode properties & algorithms group, 2023-jan-18

Participants

The following people have contributed to this document:

Markus Scherer (chair), Josh Hadley (vice chair), Asmus Freytag, Elango Cheran, Ken Whistler, Manish Goregaokar, Mark Davis, Ned Holbrook, Peter Constable, Rick McGowan, Robin Leroy

1. UCD

1.1 Stabilize toNFKC_Casefold on XID_Continue

Recommended UTC actions

1. **Consensus:** The UTC recommends to the officers a new stability policy: “For each string S containing only code points with the property XID_Continue at a given Unicode version, toNFKC_Casefold(S) under that version is identical to toNFKC_Casefold(S) under any later version of Unicode.”
2. **Action Item** for Mark Davis, PAG: Relay the recommendation for the additional stability policy for XID_Continue toNFKC_Casefold to the officers and update the policy page. See L2/23-008 item 1.1.
3. **Action Item** for Ken Whistler, PAG: In DerivedCoreProperties.txt, document the implications of changing Default_Ignorable_Code_Point, for Unicode Version 15.1. See L2/23-008 item 1.1.
4. **Action Item** for Mark Davis, PAG: In UAX #31, document for NFKC_Casefold that if the Default_Ignorable_Code_Point value changes for an existing XID_Continue that its NFKC_Casefold mapping must not change. For Unicode 15.1. See L2/23-008 item 1.1.
5. **Action Item** for Mark Davis, PAG: In the core spec, document for NFKC_Casefold that if the Default_Ignorable_Code_Point value changes for an existing XID_Continue that its NFKC_Casefold mapping must not change. Point to related text in UAX #31; consider related edits to UAX #31. For Unicode 16.0. See L2/23-008 item 1.1.

(Note: A GitHub issue has been filed to request an “invariant test” for such changes.)

Feedback

From the Source Code Working Group:

> The stability policy guarantees that toCasefold◦toNFKC is stable on encoded characters.
>
> However, the recommended operation for case-insensitive identifier comparison since Unicode 6 (or maybe 5.2?) is not toCasefold◦toNFKC, but toNFKC_Casefold; toNFKC_Casefold is not stable.

Making toNFKC_Casefold stable would require stabilizing the assignment of Default_Ignorable_Code_Point on encoded characters; this seems too constraining, as it changed as recently as 2013: https://www.unicode.org/L2/L2013/13011.htm#134-C16.

However, the Default_Ignorable_Code_Points in XID_Continue is more stable: https://util.unicode.org/UnicodeJsps/list-unicodeset.jsp?a=%5B%5B%3ADefault_Ignorable_Code_Point%3A%5D%26%5B%3AXID_Continue%3A%5D%5D&g=age&i=Gc.

Background information / discussion

- Basically, we provide stable XID_Continue, and it is recommended for use with case-insensitive identifiers, for which we use NFKC_Casefold, and so it makes sense to stabilize that, too, for those characters.
- We could still change the DI value of XID_Continue characters as long as we kept the NFKC_Casefold behavior of those characters stable (via an override).

1.2 We should add toNFKC_SimpleCaseFold and simple counterparts to the full stability policies

Recommended UTC actions

1. **Consensus:** Create a new derived property NFKC_SimpleCasefold (NFKC_SCF), derived as its non-Simple counterparts except for the use of the Simple_Case_Folding instead of the Case_Folding, for Unicode Version 15.1.
2. **Action Item** for Mark Davis, PAG: Add NFKC_SimpleCasefold to DerivedNormalizationProps.txt and PropertyAliases.txt, for Unicode Version 15.1. See L2/23-008 item 1.2.
3. **Action Item** for Mark Davis, PAG: Add NFKC_SimpleCasefold to the list of Full Properties in Section 2.7 of UTS #18, Unicode Regular Expressions, for a future revision of that UTS. See L2/23-008 item 1.2.
4. **Action Item** for Ken Whistler, PAG: Add NFKC_SimpleCasefold to the Property Table of Unicode Standard Annex #44, Unicode Character Database, for Unicode Version 15.1. See L2/23-008 item 1.2.
5. **Consensus:** The UTC recommends to the officers that the Case Folding and Case Pair stability policies be extended to simple as well as full case transformations. See L2/23-008 item 1.2.
6. **Action Item** for Mark Davis, PAG: Relay to the officers the recommendation for extending stability policies for case mappings to simple mappings, and update the policy page. See L2/23-008 item 1.2.

Feedback

From the Source Code Working Group:

While we recommend the use of full case folding for identifier comparison, some implementations need simple case folding for compatibility; for instance, Ada is a case-insensitive language which added support for Latin-1 in 1995, defining things so that ß was inequivalent to ss, and thus has had to use the simple case mappings and folding when adding Unicode support (in fact they briefly referred to full case folding in Ada 2005, but reverted it in the next Technical Corrigendum noticing the incompatibility; this was likely never implemented with full case folding).

We recommend the use of toNFKC_Casefold, rather than just toCasefold or toCasefold∘toNFKC, as the identifier equivalence; we should provide a corresponding solution for those who are stuck with simple case folding.

A formal definition of the operation in the core spec should not be needed (we don’t have one for the simple case mappings and case folding), but a property is needed.

Likewise we may not need the Changes_When, we don’t have those for the simple mappings.

Since these properties are meant for identifier comparison, the same stability policies should apply to them as apply to full case folding.

1.3 change UAX 44 multi-value example away from kCantonese

PRI #465 Proposed Update UAX #44, Unicode Character Database

Recommended UTC actions

1. The PAG reviewed this feedback, but the EDC is recording actions for it.

Feedback (verbatim)

Date/Time: Thu Jan 5 23:44:42 CST 2023
Name: Ben Yang
Report Type: Public Review Issue
Opt Subject: 465

The following text is found in UAX#44:

Most properties have a single value associated with each code point.
However, some properties may instead associate a set of multiple different values with each code point. For example, the provisional kCantonese property, which lists Cantonese pronunciations for unified CJK ideographs, has values which consist of a set of zero or more romanized pronunciation strings. Thus, the Unihan Database contains an entry:

U+342B kCantonese gun3 hung1 zung1

This line is to be interpreted as associating a set of three string values, {“gun3”, “hung1", “zung1”} with the kCantonese property for U+342B.

However, since I believe Unicode 14.0, "kCantonese" has been modified to only allow a single entry, so this text is no longer accurate.

If we'd like to stick with using a Unihan property to demonstrate, how about "kVietnamese"? Here's a suggestion for an edit:

Most properties have a single value associated with each code point. However, some properties may instead associate a set of multiple different values with each code point. For example, the provisional kVietnamese property, which lists Vietnamese pronunciations for unified CJK ideographs, has values which consist of a set of zero or more pronunciation strings. Thus, the Unihan Database contains an entry:

U+6258 kVietnamese thác thách thốc thước thướt

This line is to be interpreted as associating a set of three string values, {"thác", "thách", "thốc", "thước", "thướt"} with the kVietnamese property for U+6258.

Background information / discussion

https://www.unicode.org/reports/tr44/#Property_Values_As_Sets

https://www.unicode.org/reports/tr38/#kCantonese

Since Unicode 14, kCantonese is no longer a normal multi-value property, and does not currently map any code point to more than one value.

1.4 Incorrect applicable version for XID_Continue stability

Recommended UTC actions

1. **Consensus:** The UTC recommends to the officers that the applicable version be updated to 4.1+ on the policy "Once a character is XID_Continue, it must continue to be so in all future versions".
2. **Action Item** for Mark Davis, PAG: Relay to the officers the recommendation for correcting the application version of the XID_Continue stability policy, and update the policy page. See L2/23-008 item 1.4.

Feedback

From Robin Leroy:

https://www.unicode.org/policies/stability_policy.html#Property_Value:

Applicable Properties: XID_Continue
Constraints: Once a character is XID_Continue, it must continue to be so in all future versions.

Applicable Unicode Versions: 3.0.1+

https://unicode.org/Public/4.0-Update/DerivedCoreProperties-4.0.0.txt:

```
30FB          ; XID_Continue # Pc       KATAKANA MIDDLE DOT
```

https://util.unicode.org/UnicodeJsps/character.jsp?a=30FB&B1=Show:

XID_Continue: No
Unicode/Emoji version: 15.0

The reason is

[101-C16] Consensus: Change the General Category of U+30FB KATAKANA MIDDLE DOT and U+FF65 HALFWIDTH KATAKANA MIDDLE DOT from "Pc" to "Po". [L2/04-391]

and it was not added to Other_ID_Continue.

Background information / discussion

Mark Davis verified programmatically that the last time characters were removed from XID_Continue was in Unicode 4.1.

1.5 Character classes of U+2126 OHM SIGN and U+00B5 MICRO SIGN

Recommended UTC actions

1. **Action Item** for Rick McGowan: Respond to Steven Pemberton with the information in the background section of L2/23-008 item 1.5.

Feedback (verbatim)

Date/Time: Wed Dec 14 10:54:46 CST 2022
Name: Steven Pemberton
Report Type: Error Report
Opt Subject:
Character classes of U+2126 OHM SIGN and U+00B5 MICRO SIGN

Programming languages and other software systems using Unicode generate functions directly from the Unicode database, for instance functions to convert strings to uppercase, lowercase and titlecase.

For instance, CSS, which has a text-transform property. If I take the text "Resistance is 950µΩ" and apply the different transforms, I get displayed:

text-transform: uppercase:
RESISTANCE IS 950ΜΩ

text-transform: lowercase:
resistance is 950μω

text-transform: capitalize:
Resistance Is 950μΩ

This is clearly unacceptable, since it changes the meaning of the text.

To fix this, U+2126 OHM SIGN and U+00B5 MICRO SIGN should be classified as "Symbol, Other", and not be assigned case equivalents.

Respectfully,

Steven Pemberton

Background information / discussion

- We do not guarantee that transforming text (other than normalization) will not affect its meaning.
- More generally, applying case mappings to technical text rather than “normal language” is a mistake, and cannot be fixed in the encoding nor via properties.
- Case mappings are lossy even on normal text (lowercasing iPod or McGowan; anything German; uppercasing Irish). Matters are much worse in technical text.
- Under case folding stability, no change is possible.
- Under normalization, µ and Ω normalize to their standard Greek counterparts, so treating them differently is not possible.
- ASCII letters used for SI units are also not exempt from casing, and also change meaning with case: 1 ms = 1 millisecond, whereas 1MS = 1 megasiemens.
- Not all letters for SI units have duplicates, this is the reason why the few that were introduced separately have been made canonically equivalent to standard letters. This way, all SI units are treated the same.

2. New Scripts & Characters

PAG members reviewed the following proposals, provided feedback to SAH, and the feedback has been addressed.
No further recommended actions from our side.

- L2/22-289 Final Proposal to encode the Tai Yo Script.
- L2/22-236 Proposal to encode KAWI SIGN NUKTA
- L2/22-250 Canonical combining class for nukta characters
- L2/22-260 Proposal to encode three characters in Tulu-Tigalari / [173-C27] approving two of them
- L2/22-281 Proposal to encode Two Quranic Arabic Characters
- L2/22-218 Proposal to Encode Chisoi

3. Bidi

3.1 UAX #9: Feedback on text for N0 and BD16

PRI #460 Proposed Update UAX #9, Unicode Bidirectional Algorithm

Manish Goregaokar submitted the feedback below, and then found more issues and iterated with PAG members, resulting in

L2/23-014 Proposal for amendments to UAX #9

Recommended UTC actions

1. **Action Item** for Manish Goregaokar, PAG: In Proposed Update UAX #9, Unicode Bidirectional Algorithm, apply the changes proposed in L2/23-014, for Unicode Version 15.1. See L2/23-008 item 3.1.
2. **Action Item** for Manish Goregaokar, PAG: Prepare a proposal with additional revisions of UAX #9 clarifying the flow of control and data, for a future version of Unicode. See L2/23-008 item 3.1.

Feedback (verbatim)

Date/Time: Sun Dec 18 00:13:30 CST 2022
Name: Manish Goregaokar
Report Type: Public Review Issue
Opt Subject: 460
I was recently implementing N0 and I have a couple largely (but not entirely)
editorial suggestions:

1. BD16 variable naming

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 they're somewhat confusable, I think it's worth giving them names
(probably `stack` or `open_brackets` and `result`)

2. BD16 bailout

In BD16 it says "stop processing BD16 for the remainder of the isolating run
sequence". Is this a degenerate case, or is behavior expected to be defined
here? It's unclear *where* this bails out to: BD16 is expected to produce a
result so you have to end up with something. A couple options are to return
an empty list, return the current list, or return the current list, but
sorted.

3. BD16 Empty stack case

"Declare a variable that holds a reference to the current stack element"
 does not handle the empty stack case (i.e. a string that starts with a
 closing bracket). It's kind of obvious from context (just go straight to
 step 5) but it probably could be spelled out explicitly.

4. N0 step c sos handling

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. I suspect this could be disambiguated here by saying something like "until the first strong type (L or R) is found. If none, use the value assigned to _sos_"

5. N0 step c complexity

Step c basically does the following:

- (c) look for the first strong type s preceding the bracket, falling back to the value of sos if necessary
- (c 1) If s != e, assign brackets to s
- (c 2) Else (when s == e) assign brackets to e

(c 1) and (c 2) could be much more succinctly written as just "assign brackets to s". I'll admit that being overly Gricean is probably not the best when it comes to reading specifications, but the fact that this was written out as two steps gave me pause and I had to reevaluate if I had missed something important.

I think we should do one of:

- Collapsing (c 1) and (c 2) to a single step that always assigns to the strong type found, and add some non normative text about how this finds established context but falls back to the embedding direction.
- Add some non normative text saying "this has the effect of assigning to the strong type found" or something so it's clear that the spec is being verbose here.

The main value I see in keeping (c 1) separate is that it is an implicit definition of the "established context" concept, which is not used in the spec but might be generally useful.

4. Text Segmentation

4.1 Chapter 3 D58 Grapheme base & D61b Graphical application

Recommended UTC actions

1. **Action Item** for Mark Davis, Robin Leroy, PAG: Review the core spec for "grapheme base" and Grapheme_Base and propose changes, as appropriate. See L2/23-008 item 4.1.

Feedback (verbatim)

Date/Time: Thu Sep 15 05:52:10 CDT 2022
Name: Rossen Mikhov
Report Type: Error Report
Opt Subject: Unicode Chapter 3 Conformance

https://www.unicode.org/versions/Unicode15.0.0/ch03.pdf
Version 15.0.0

Location:
D62b Graphical Application

Problematic text:

A nonspacing mark in a defective combining character sequence is not part of
a grapheme cluster and is subject to the same kinds of fallback processing
as for any defective combining character sequence.

Explanation:

"Grapheme cluster" is defined in D60 as "The text between grapheme cluster
boundaries". So, formally, any character is part of some grapheme cluster,
be it a degenerate one.

What is more troubling with this definition D62b is that it states that
nonspacing marks apply to grapheme bases, with "Grapheme base" being
defined in D58 as based on Grapheme_Base. But Grapheme_Base is no longer
used by UAX29. It isn't clear if nonspacing marks should "graphically
apply" to things other than Grapheme_Base characters and Korean syllables,
for example what about emoji ZWJ sequences.

Background information / discussion

This was originally filed for consideration for UTC #173 but was overlooked at the time. The group agrees that we should consider deprecating the Grapheme_Base property but not before fixing the spec where it refers to it.

4.2 LineBreakTest.txt bug for unassigned U+1F02C

Recommended UTC actions

1. Discussed; no action for UTC

Feedback (verbatim)

> Date/Time: Thu Nov 10 06:14:10 CST 2022
> Name: Tomasz Gucio
> Report Type: Error Report
> Opt Subject: LineBreakTest.txt
>
> Hello,
>
> I've found what might be an error in the latest line break test file. In the
> case below, the expected result is no break between the two code points based
> on the first code's property (Other). However, this and other codes in the
> range are Ideographic (ID) and so line break should be allowed.
>
> Property data: LineBreak-15.0.0.txt, Date: 2022-07-28, 09:20:42 GMT [KW, LI]
>
> 1F02C..1F02F;ID   # Cn   [4] <reserved-1F02C>..<reserved-1F02F>
> Test case: LineBreakTest-15.0.0.txt, Date: 2022-02-26, 00:38:39 GMT
>
> × 1F02C × 1F3FF ÷   # × [0.3] <reserved-1F02C> (Other) × [30.22] EMOJI MODIFIER FITZPATRICK TYPE-6 (EM) ÷ [0.3]
> Best regards,
> Tomasz

Background information / discussion

It is not obvious what the LineBreakTest.txt file comment <reserved-1F02C> (Other) means because "Other" is not a Line_Break value alias.

In any case, U+1F02C is Extended_Pictographic and unassigned, and rule LB30b keeps these together with a following EM: “[\p{Extended_Pictographic}&\p{Cn}] × EM”

So this seems to be working properly.

However, it might be a good idea to better document the intended behavior of “Other” for this test, and possibly update instances of “(Other)” to “(ExtPicUnassigned)”. An issue for this has been created in the unicodetools repo.

5. Collation

5.1 Collation of Hebrew punctuation Geresh & Gershayim

L2/23-016 DUCET: Sort quotation marks+Geresh+Gershayim like their ASCII fallbacks

Recommended UTC actions

1. **Consensus:** Change the Unicode default sort order of certain punctuation marks according to L2/23-016.
2. **Action Item** for Markus Scherer, PAG: Change the Unicode default sort order of certain punctuation marks according to L2/23-016, for Unicode Version 15.1.

Summary

In the Unicode default sort order, several punctuation marks that look similar to ASCII apostrophe and double quote, or for which those ASCII characters are common fallbacks, sort primary-differently from them. This yields poor sorting, and makes it hard to search for text. CLDR collation tailorings already change the behavior of some of these characters. Also, a prominent text search implementation works around this issue. The proposal is to make these characters primary-equal to their ASCII fallbacks.

6. Security

6.1 Properties for UAX #31 Standard Profiles

Recommended UTC actions

1. **Consensus:** Create two properties, ID_Compat_Math_Start and ID_Compat_Math_Continue, as documented in section 7.1, Mathematical Compatibility Notation Profile, of the proposed update of UAX #31 (revision 38, draft 6), for Unicode Version 15.1.
2. **Action Item** for Robin Leroy, PAG: Create two properties, ID_Compat_Math_Start and ID_Compat_Math_Continue, as documented in section 7.1, Mathematical Compatibility Notation Profile, of the proposed update of UAX #31 (revision 38, draft 6), for Unicode Version 15.1. Update PropertyAliases.txt and PropList.txt as appropriate. See L2/23-008 item 6.1.
3. **Action Item** for Ken Whistler, PAG: Document the new properties ID_Compat_Math_Start and ID_Compat_Math_Continue in UAX #44, for Unicode Version 15.1. See L2/23-008 item 6.1.
4. **Action Item** for Mark Davis, PAG: Add ID_Compat_Math_Start and ID_Compat_Math_Continue to the list of Full Properties in Section 2.7 of UTS #18, Unicode Regular Expressions, for a future revision of that UTS. See L2/23-008 item 6.1.

Feedback

From Robin Leroy:

The standard profiles in the proposed Section 7 of UAX #31 rely on the following sets:

[∂𝛛𝜕𝝏𝞉𝟃∇𝛁𝛻𝜵𝝯𝞩∞]
[∂𝛛𝜕𝝏𝞉𝟃∇𝛁𝛻𝜵𝝯𝞩∞⁽⁾₍₎⁺₊⁼₌⁻₋⁰₀¹₁²₂³₃⁴₄⁵₅⁶₆⁷₇⁸₈⁹₉]
[[:Pattern_Syntax:]&[:Emoji_Presentation:]]
Since these sets are meant to be used in the definition of the lexical structure of computer languages, it makes sense if they can be referred to, e.g., in regular expressions that implement UTS #18.

This means we would need properties for sets 1 and 2. It was suggested at UTC #173 that 3. be made a derived property, but it may be less clear how necessary this is, given that UTS #18, UnicodeSet, etc. allow for set intersection already.

I am using the names ID_Compat_Math_Start and ID_Compat_Math_Continue for the first two in the draft for now. Ken suggested Emoji_PSEP for the third.

Question: how many properties do we need, and what are their names?

Background information / discussion

PAG discussed and concluded that there should be 2 new properties for the first 2 sets, but not for the 3rd which is trivially computable from the intersection of existing properties.

6.2 Suspected endless loop in UTS #55 `zero()` example

Recommended UTC actions

1. **Action Item** for Rick McGowan: Thank David Starner for their feedback of [Mon Dec 5 17:34:03 CST 2022] and relay the information in the background section of L2/23-008 item 6.2.

Feedback (verbatim)

Date/Time: Mon Dec 5 17:34:03 CST 2022
Name: David Eugene Starner
Report Type: Public Review Issue
Opt Subject: 466

```
void zero(double** matrix, int rows, int columns) {
     for (int i = 0; i < rows; ++i) {
          double* row = matrix[i];
          for (int i = 0; i < columns; ++i) {
               row[i] = 0.0;
          }
     }
}
```
This program looks like it zeros a rows by columns rectangle, but it actually only zeros a diagonal, because the identifier i on line 4 is a Cyrillic letter, whereas i is the Latin letter everywhere else.

That program looks like it goes into an infinite loop if rows > columns, as the inner loop index overwrites the outer one. It's bad code whether or not the identifier on line 4 is Latin or Cyrillic.

Background information / discussion

See https://www.unicode.org/reports/tr55/tr55-1.html#Spoofing-confusables

The example looks right. David seems to be overlooking the comment that the inner loop head uses Cyrillic "і" not Latin "i". If it did use the same variable name, then the compiler *might* warn about one ‘i’ shadowing the other, but there would be no infinite loop: the program would zero a `rows` by `columns` rectangle, as described.

What the program looks like: https://gcc.godbolt.org/z/dfdG1vbGc
What the program really is: https://gcc.godbolt.org/z/v7bb858EP

6.3 PD-UTS 55 lacks information on spoofing and usability issues / Brahmic

Recommended UTC actions

1. PAG recommends no action, except to record in the minutes that the feedback from Norbert Lindenberg [Thu Dec 15 00:05:11 CST 2022] should be considered by PAG as part of prior action 172-A82.

Feedback (verbatim)

Date/Time: Thu Dec 15 00:05:11 CST 2022
Name: Norbert Lindenberg
Report Type: Public Review Issue
Opt Subject: 466

The proposed draft UTS #55, Unicode Source Code Handling, lacks information on spoofing and usability issues arising from lookalike syllables in Brahmic scripts.

Most Brahmic scripts have been encoded in Unicode according to principles that differ from those used for most other scripts. For most non-Brahmic scripts, spacing characters are encoded in visual order, with nonspacing marks following the spacing characters they attach to. If multiple nonspacing marks attach to the same base, marks that interact typographically are encoded from innermost (closest to the base) to outermost, while Unicode normalization handles ambiguities caused by nonspacing marks that don’t interact typographically. For most Brahmic scripts, the intent is that characters are encoded in phonetic order, independent of visual placement relative to each other, and Unicode normalization is largely disabled by using the canonical combining class 0 for most combining marks.

To ensure interoperability between smart keyboards, predictive input systems, spelling checkers, font rendering systems, fonts, systems for searching and sorting text, optical character recognition systems, speech input and output systems, text normalization, and other text processing software, the Unicode Standard would have to define the encoding order of orthographic syllable components precisely and unambiguously for each Brahmic script. However, the Unicode Standard fails to do so. Fonts and font rendering systems to some extent try to impose order by inserting dotted circles into character sequences that their designers find inappropriate, but do so incompletely and inconsistently, with a tendency to relax rules over time.

The result is that in a number of Brahmic scripts a given orthographic syllable can be encoded in multiple ways with the same rendering. This is well documented, for example, for Khmer – see Horton et al. 2017, Lindenberg 2019, Hosken 2021. For example, the word ស្ត្រី (woman) can be encoded with three different character sequences with identical rendering in all major rendering systems: ស្ត្រី, ស្រ្តី, ស្ត្រី – even after eliminating ambiguities introduced by the intentional confusable subjoined consonants ្ដ (coeng da) and ្ត (coeng ta). See Hosken 2021 pages 34-36 for more examples.

The issues could be documented in UTS 55 as follows.

Spoofing using lookalike orthographic syllables

The Unicode Standard uses phonetic encoding order for most Brahmic scripts, but does not define the encoding order of orthographic syllable components for most such scripts. As a consequence, syllables can often be encoded in multiple character sequences that render identically.
This can be used for spoofing, for instance, by constructing identifiers that look like they are the same, but are actually different.

Example: Consider the following Python program:

```
ស្ត្រី = True
ស្រ្តី = False
if ស្ត្រី:
print("True!")
else:
print("False?”)
```

The program looks like it would print “False?”, but it actually prints “True!” because the ស្រ្តី assigned False is a different variable than the ស្ត្រី assigned True, and the ស្ត្រី tested in the if-statement is the one assigned True.

Usability issues arising from lookalike orthographic syllables

When working with Brahmic scripts, there is a common usability issue whereby one accidentally types an orthographic syllable using the wrong character sequence, with no difference in the resulting rendering. For example, the code shown in “Spoofing using lookalike orthographic syllables” may be the result of one engineer typing ស្ត្រី, another typing ស្រ្តី, which look identical but are in fact different variables.

To address these problems, the Unicode Standard would have to specify the encoding order of orthographic syllable components for all Brahmic scripts. A proposal for Khmer is currently under discussion.

References:

Joshua Horton, Makara Sok, Marc Durdin, Rasmey Ty: Spoof-Vulnerable Rendering in Khmer Unicode Implementations. 2017.
https://lt4all.elra.info/proceedings/lt4all2019/pdf/2019.lt4all-1.35.pdf

Norbert Lindenberg: Issues in Khmer syllable validation. 2019.

https://lindenbergsoftware.com/en/notes/issues-in-khmer-syllable-validation/

Martin Hosken: Khmer Encoding Structure. 2021.
https://www.unicode.org/L2/L2021/21241-khmer-structure.pdf

Background information / discussion

The issues described in the feedback fall under the much broader umbrella of the dismal state of our confusability detection for those scripts. Confusable detection (which is defined in #39, not in #55) should be significantly improved, in particular for those scripts that have a profusion of « do not use » sequences.

Work is ongoing to improve this aspect of confusable detection in #39.

Script-specific considerations do not belong in PDUTS #55, whose audiences cannot be expected to dive into those subtleties; once the suitable mechanisms are defined in #39, they will be picked up by the recommendations in #55 (either automatically or with a small update to #55, depending on what exactly we do).

6.4 Review SCWG proposals for UTC #174

L2/23-017 Recommendations of the source code working group for UTC #174

Recommended UTC actions

1. PAG recommends that the UTC review PDU-UAX31. Note the further technical changes in the evolution of this update.
2. PAG endorses the recommendations in the doc.

Summary

Further technical changes to the proposed update for #31 and proposed draft to #55.

6.5 Errors in UTR#36 Unicode Security Considerations

Recommended UTC actions

1. **Action Item** for Mark Davis, Robin Leroy, PAG: Address the editorial issues described in the feedback of [Thu Jan 5 20:53:39 CST 2023], as described in Section 6.5 of L2/23-008.

Feedback (verbatim)

Date/Time: Thu Jan 5 20:53:39 CST 2023
Name: NAKAUCHI Tomohiro
Report Type: Error Report
Opt Subject: UTR#36 Unicode Security Considerations

I've read UTR#36-rev.15 interestedly. At that time, I found some errors.

So, I would like to report those:

1. I have a question that in line 619 to 625 why such structure(html) is used? (which invokes that the tool tip shows 'U+0906 DEVANAGARI LETTER AA' in my web browser.)
   I think that description of 'title' attribute used in span element is inappropriate.
2. In addition, at the same parahraph, description says that 'look identical (ẋ̣ and ẋ̣̇)', but my web browser shows that the former is <x, dot_below, dot_above> and the latter is <x, dot_below, dot_above, dot_above>. So, I think that these are not identical.
3. In line 1393, word 'such as' is duplicated.
4. In section 2.7 Numeric Spoofs, I think that description is slightly incorrect.
   Although string appeared like "89" is represented by Bengali and Oriya, this description says that is only 'the Bengali string "৪৭"'.
5. In line 1628, text written is 'Final_Sigma as provided in Table 3-15'.
   I guess that this is 'Table 3-17'.
6. In line 3044, word 'from' is duplicated.

Background information / discussion

PAG agrees with the feedback and recommends the editorial changes suggested, except we should use a different example for #2. Specifically, a sequence where the CCA stays put, such as x̣̄ vs. x̣̄.