Mapping to individual characters in expl3

Joseph Wright

It is natural to think that separating a word up into individual characters is an easy operation. It turns out that for the computer this isn't really the case. If we look at a system that natively understands Unicode (like XeTeX or LuaTeX), most of the time one 'character' is stored as one codepoint. A codepoint is a single character entity for a Unicode programme. For example, if we take the input 'café' in a file saved as UTF-8, it is made up of four codepoints:

```
U+0063 (LATIN SMALL LETTER C)
U+0061 (LATIN SMALL LETTER A)
U+0066 (LATIN SMALL LETTER F)
U+00E9 (LATIN SMALL LETTER E WITH ACUTE)
```

So we could, in XeTeX/LuaTeX, use a simple mapping to grab one character at a time from this word and do stuff with it. However, that's not always the case. Take for example 'Spıñal Tap': the dotless-i is a single codepoint, but there is no codepoint for an umlauted-n. Instead, that is represented by two codepoints: a normal n and a combining umlaut. As a user, it's clear that we'd want to get a single 'character' here. So there's clearly more work to do.

Luckily, this is not just a TeX problem and the Unicode Consortium have thought about it for us. They provide a data file and rules that describe how to divide input into *graphemes*: 'user perceived characters'. So 'all' that is needed is to examine the input using these rules, and to divide it up so that 'characters' stay together.

For pdfTeX, there's an additional wrinkle: it uses bytes, not codepoints, and so if we use a naïve TeX mapping, we would divide up any codepoint outside the ASCII range into separate bytes: not good. Luckily, the nature of codepoints is predictable: all that is needed is to examine the first byte and collect the right number of further bytes to re-combine into a valid codepoint.

This work isn't something the average end user wants to do. Luckily, they don't have to as the LaTeX team have worked on this and created a suitable set of expl3 functions to do it: `\text_map_function:nN` and `\text_map_inline:nn`.

For example, we can do (mapping each character to printing itself in parentheses):

```
\ExplSyntaxOn
\text_map_inline:nn
  { Spıñal ~ Tap } { (#1) }
\ExplSyntaxOff
```

and get

(S)(p)(ı)(ñ)(a)(l)( )(T)(a)(p)

in any TeX engine — assuming we are set up to *print* the characters, of course. Getting the right fonts is an independent issue from parsing the input.

Taking a more 'serious' example (and one that is going to use LuaTeX for font reasons), we might want to map over Bangla text: I'm going to use **ন্দ্রকিন্দ্র** as my example. Our `\text_map_inline:nn` function divides up the characters correctly: **(ন্)(দ্)(র)(কি)(ন্)(দ্)(র)**.

In contrast, the generic `expl3` token-list function `\tl_map_inline:nn` gives: **(ন)(◌্)(দ)(◌্)(র)(ক)(◌ি)(ন)(◌্)(দ)(◌্)(র)**, which is a very odd result. In short: Unicode characters are neither bytes nor tokens.

(If you want to try that demo yourself, you'll need a document preamble that can work properly with Bangla: I'm using

```
\usepackage{fontspec}
\newfontface\harfbangla
  {NotoSansBengali-VariableFont_wdth,wght.ttf}
  [Renderer = HarfBuzz, Script = Bengali]
```

then using `\harfbangla` in a brace pair around my demonstrations. Finding a monospaced font that properly renders Bangla is ... tricky.)

So, as you can see, mapping to 'real' text is easy with expl3: you just need to know that the tools are there.

◇ Joseph Wright
Northampton, United Kingdom
joseph dot wright (at)
morningstar2.co.uk

doi.org/10.47397/tb/43-3/tb135wright-grapheme