Inter-character spacing and ligatures[1]

There was a discussion on the LuaTEX (dev) list about inter character spacing and ligatures. The discussion involved a mechanism inherited from pdfTEX but in ConTEXt we don't use that at all. Actually, support for inter character spacing was added in an early stage of MkIV development as an alternative for the MkII variant, which used parsing at the TEX end. Personally I never use this spacing, unless a design in a project demands it.

In the MkIV method we split ligatures when its components are known. This works quite well. It's anyway a good idea to disable ligatures, so it's more of a fallback. Actually we should create components for hard coded characters like æ but as no one ever complained I leave that for a later moment.

As we already had the mechanisms in place, support for selective spacing of ligatures was a rather trivial extension. If there is ever a real need for it, I will provide control via the normal user interface, but for now using a few hooks will do. The following code shows an example of an implementation.

```
local utfbyte = utf.byte

local keep = {
    [0x0132] = true, [0x0133] = true, -- IJ ij
    [0x00C6] = true, [0x00E6] = true, -- AE ae
    [0x0152] = true, [0x0153] = true, -- OE oe
}

function typesetters.kerns.keepligature(n)
    return keep[n.char]
end

local together = {
    [utfbyte("c")] = { [utfbyte("k")] = true },
    [utfbyte("i")] = { [utfbyte("j")] = true },
    [utfbyte("I")] = { [utfbyte("J")] = true },
}

function typesetters.kerns.keeptogether(n1,n2)
    local k = together[n1.char]
    return k and k[n2.char]
end
```

The following also works:

```
local lpegmatch = lpeg.match
local fontdata  = fonts.identifiers

local keep = -- start of name
        lpeg.P("i_j")
      + lpeg.P("I_J")
      + lpeg.P("aeligature")
```

1. Excerpt from the 'Weird Examples' chapter in hybrid.pdf

```
      + lpeg.P("AEligature")
      + lpeg.P("oeligature")
      + lpeg.P("OEligature")

function typesetters.kerns.keepligature(n)
    local d = fontdata[n.font].descriptions
    local c = d and d[n.char]
    local n = c and c.name
    return n and lpegmatch(keep,n)
end
```

A more generic solution would be to use the `tounicode` information, but it would be overkill as we're dealing with a rather predictable set of characters that have gotten Unicode slots assigned. When using basemode most fonts will work anyway.

So, is this really worth the effort? Take a look at the following example.

```
\definecharacterkerning [KernMe] [factor=0.25]

\start
  \setcharacterkerning[KernMe]
  \definedfont[Serif*default]
  Ach kijk effe, \ae sop draagt een knickerbocker! \par
  \definedfont[Serif*smallcaps]
  Ach kijk effe, \ae sop draagt een knickerbocker! \par
\stop
```

Typeset this (Dutch text) looks like:

Ac h k ij k e f f e, æ s o p d r a a g t ee n k n i cke r bocke r !
Ac h k i j k e f f e , æ s o p d r a a g t e e n k n i c k e r b oc k e r !

You might wonder why I decided to look into it. Right at the moment when it was discussed, I was implementing a style that needed the Calibri font that comes with MS Windows, and I visited the FontShop website to have a look at the font. To my surprise it had quite some ligatures, way more than one would expect.

Hans Hagen
Pragma ADE, Hasselt, The Netherlands
pragma@wxs.nl

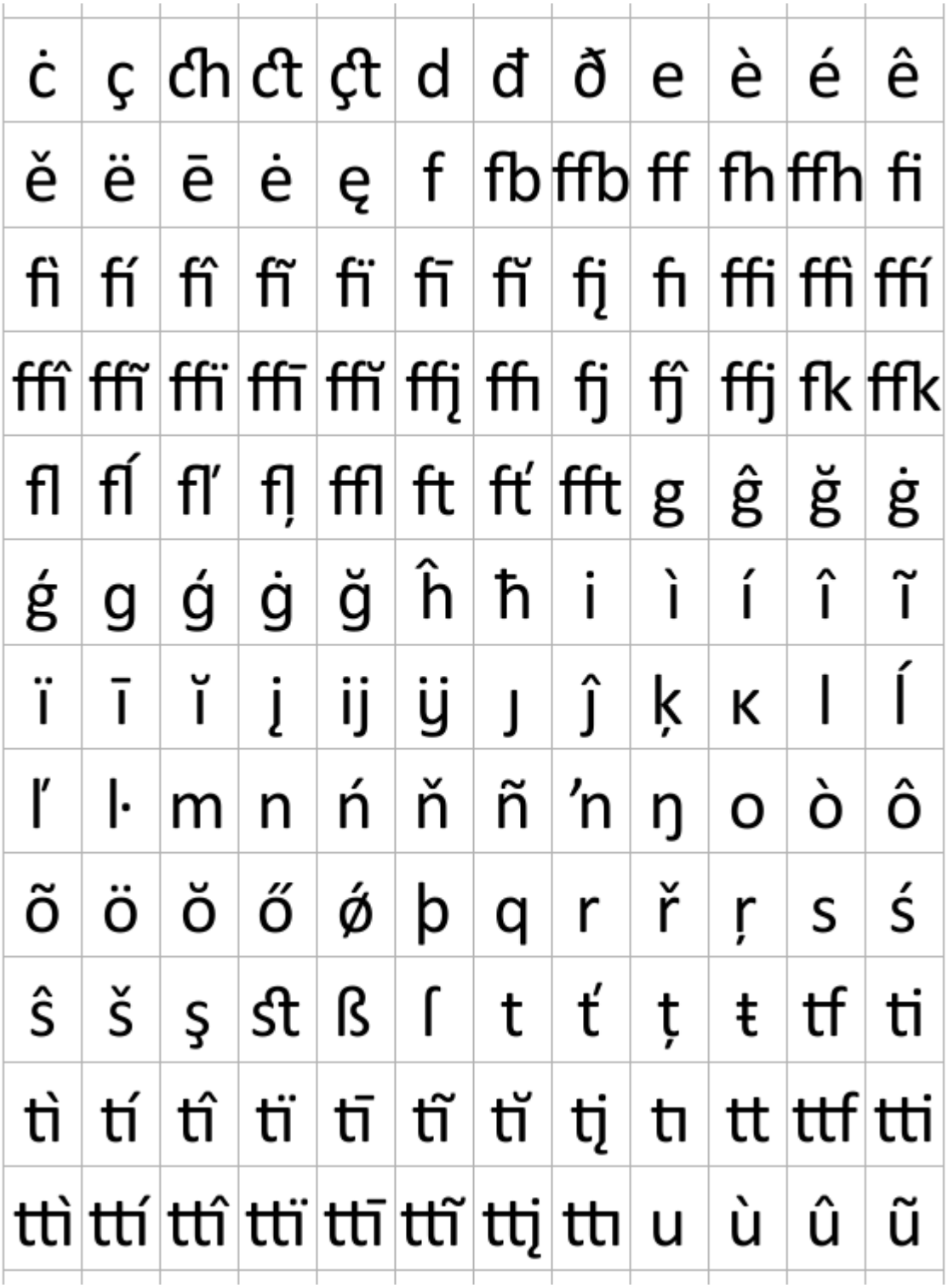

Figure 1. Some of the ligatures in Calibri Regular. Just wonder what intercharacter spacing will do here.