A simple Arabic typesetting system for mixed Latin/Arabic documents

Yannis Haralambous

▶ **To cite this version:**

Yannis Haralambous. A simple Arabic typesetting system for mixed Latin/Arabic documents. TUGboat: the communications of the TeX users group, 2014, 35 (3), pp.277-283. hal-01185086

HAL Id: hal-01185086
https://hal.science/hal-01185086v1

Submitted on 30 Oct 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution - NonCommercial 4.0 International License

A simple Arabic typesetting system for mixed Latin/Arabic documents: *ḍād*

Yannis Haralambous

Abstract

We describe ض (*ḍād*), a package allowing simple typesetting in Arabic script, intended for mixed Latin-Arabic script usage, in situations where heavy-duty solutions are discouraged. The ض system operates with both Unicode and transliterated input, allowing the user to choose the most appropriate approach.

1 Introduction

As with many TEX projects, this one was started to fulfill an immediate need: the author was writing a paper for an Arabic language Natural Language Processing conference [7] and hence was in need of a straightforward way to introduce Arabic text into his document. In this context, "straightforward" can be subdivided into the following five requirements:

1. it should be compatible both with the IEEE LATEX style [19] (required by the conference) and with the WriteLATEX platform [3], of which the author is an enthusiastic user;
2. it should allow user-friendly and robust input of Arabic text, including when placed inside TEX command arguments;
3. it should typeset in an optimal way all combinations of letters and diacritics that may appear in scholarly text;
4. it should provide some extra features: being able to easily change the letter form, as well as to colorize specific letters without breaking contextual analysis;
5. the font should be easily readable in a context of mixed Latin-Arabic script.

In the following discussion we will see why existing systems did not fulfill the requirements, and how the author solved the problem.

1.1 Existing systems and their pitfalls

As we live in the 21st century, an obvious choice for typesetting Arabic in TEX is XETEX [12], a TEX avatar (in)famous for typesetting in non-Latin scripts by taking advantage of operating system resources.

Requirement 1 XETEX indeed is provided on the WriteLATEX platform. But the LATEX overhead for typesetting Arabic in XETEX is quite heavy (packages `fontspec`, `xunicode`, `arabxetex`, etc.) and hence, not surprisingly, XETEX is incompatible with the IEEE style.

Requirement 2 XETEX is unable to use a transliteration system and hence requires the Arabic text to be input in Arabic script and *only* in Arabic script. Using a transliteration to input Arabic may seem terribly old-fashioned to the reader, but there are cases where it is the best solution. One of these cases is the context of this paper: a mixture of Latin script, Arabic script, and TEX commands.

Indeed, at the GUI level, there are — at least — two drawbacks in combining Arabic and Latin script in the same paragraph:

1. the use of the cursor and of left and right arrow keys is very cumbersome: when you select a location with the cursor you don't know whether you are in right-to-left or left-to-right mode and hence you don't know in which direction to advance, or how to select a given character string;
2. the situation is made even worse by the fact that some punctuation marks (period, exclamation mark, dashes, parentheses, braces, brackets, etc.) are common to the two scripts and hence the — quite sophisticated — bidirectional algorithm is used to determine whether a punctuation mark is to be placed on the left or on the right of an Arabic word. The bidirectional algorithm (or 'bidi' for the insiders) is both a blessing and a curse. It is a blessing because it puts some order in the rendering of mixed right-to-left and left-to-right texts (cf. [6, p. 133–146]) — but this works well only if RLE and LRE are used to indicate the embedding level. Otherwise, in everyday use, bidi is a curse. Fig. 1 shows how the TEX code for writing the word كتب with the middle letter colorized in red is displayed by various programs under various operating systems; not a single one of them really makes sense.

Requirement 4 Sophisticated OpenType fonts [6, § D.9.4] handle relatively well the many letter + diacritic combinations, and most systems (including XETEX) can colorize word parts without breaking contextual analysis through the use of the zero-width joiner character [6, p. 104]. But not a single system is able to colorize single letters inside ﻻ, since this ligature has always been considered as a single glyph by font designers.

Requirement 5 The best way to match Latin and Arabic script is to choose an Arabic font with relatively small differences in height between letters. A quite common choice is the font *Geezah* by Diwan Software Ltd (developed for Apple WorldScript in the early nineties, and still included in Mac OS X, through today). Geezah is a nice font but its diacritics are placed rather suboptimally, and modifying their positions requires a high amount of competence in fiddling around with OpenType features.

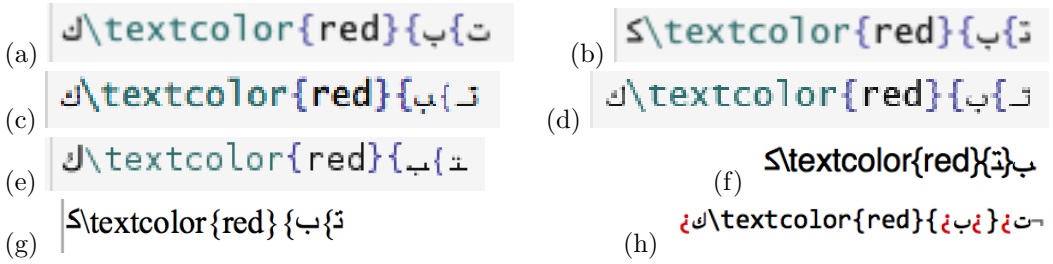

Figure 1: To obtain كتب with letter ـتـ in red, the author has typed the Unicode string "*kāf*, ZWJ (zero-width joiner), `\red{`, ZWJ, *tā'*, ZWJ, `}`, ZWJ, *bā'*". Here is the result, in the following environments: (a) Chrome under Mac OS X, (b) Safari under Mac OS X, (c) Firefox under Mac OS X, (d) Internet Explorer under Windows 8, (e) Iceweasel under Debian, (f) Mellel under Mac OS X, (g) Nisus under Mac OS X, (h) BBEdit under Mac OS X. The reader can see the variety of contextual forms displayed in these examples. It is obviously not straightforward to understand the meaning of the code, and using the cursor to edit it is no less than a Lovecraftian nightmare.

In transliteration this code snippet is simply written `k-\textcolor{red}{-t-}-b`.

1.2 The long and winding road towards a solution

Obviously, Requirement 1 and WriteLaTeX compatibility ruled out Ω [8, 9, 10], despite its powerful machinery for defining transliterations and applying contextual analysis. XeTeX was ruled out since it satisfies none of the requirements (1), (2), (4) or (5). The two remaining choices are pdf(e)TeX [2] (with bidirectional typesetting support) and LuaTeX [11].

Let us make a two-decade jump back in time. On April 12, 1993, the author organized a workshop entitled "TeX and the Arabic Script", at the National Institute of Oriental Languages and Civilizations (INALCO) in Paris. One of his contributions to this workshop was a public domain Arabic TeX font in Geezah design, in which the contextual analysis was entirely done via TeX's smart ligatures and boundary characters [13]. This is indeed the simplest approach: no extra technology was required other than TeX–XeT bidirectional typesetting and TeX 3 smart ligatures. Nevertheless that font had a serious pitfall: because the number of glyphs was limited to 256, it was impossible to handle automatically quadriform letters followed by a vowel. In that case, it was necessary to introduce a vertical bar between the letter and the vowel, so as to obtain its final (or isolated) form (as for example, in كتابٌ = `ktAbuN` which had to be written `ktAb|uN`). The proceedings of this workshop were planned to be published in the *Cahiers GUTenberg*, but this never happened — which is a pity since among the speakers was also the legendary creator of the Moroccan simplified Arabic writing system [15, 17], Ahmed Lakhdar-Ghazal († 2008).

As building a font based entirely on smart ligatures was an interesting approach, the author tried to investigate ways of succeeding now where he had failed in 1993, by the use of modern technologies.

A first idea was to create ligatures between letters and digits 0–3 to obtain contextual forms of the letters, and to have some other mechanism insert the digits. Indeed, LuaTeX provides callbacks for both the token and the node list, which would be natural choices for such a mechanism. Traversal of the node list by LuaTeX is quite efficient, but unfortunately unsuitable for this particular project since at that stage, ligatures have already been applied, so it is too late to insert digits among the nodes. After some attempts, the token list callback was abandoned since it is still in a very rudimentary state. Also, unfortunately, LuaTeX has not yet implemented Lua patterns, which are a kind of regular expressions, and would make programming the contextual analysis much easier.

A second idea was to use large virtual fonts to encode all letter + diacritic combinations, so that the 1993 approach would be applied not to letters alone, but to letter + diacritic combinations. But the attempt to generalize the 1993 approach to large virtual fonts (OVF) has failed as well, because OVFs do not provide support for Knuth's boundary character (indeed, in Ω, ΩTPs provide a much more elegant solution to the word boundary problem, and therefore the boundary character was left out of the Ω system, but again LuaTeX does not provide ΩTPs). In this approach, the default form of a letter would be the medial one, and boundary characters would turn a letter into initial, final or isolated form.

The third attempt proved successful: instead of using boundary characters, the author used smart

ligatures between characters. In this case, the default letter form is the isolated one. The presence of a second letter changes the form of the first from isolated to initial, and the one of the second from isolated to final. A third letter will turn the second letter into middle form and the third letter into final (provided, of course, the second letter is quadriform), and so on. More details are given in § 4.

2 The name

The next question was what to name the package.

Thanks to the Internet, search engines, social media, and the like, people are becoming more and more aware of other languages and writing systems. Why not give this package an Arabic name, be it a single letter?

The author has chosen the letter ض, called *ḍād*, because Arabic is traditionally called the "language of the *ḍād*", since this sound was historically considered as being unique to Arabic.

The reader is probably wondering how to pronounce this letter, technically a "voiced velarized alveolar stop" [18, p. 16]. Here is how [20, p. 10] describes its pronunciation:

> Pronounce the regular sound 'd' and you will find that the tip of your tongue will touch in the region of the upper front teeth/gum. Now pronounce the sound again and at the same time depress the *middle* of the tongue. This has the effect of creating a larger space between the tongue and the roof of the mouth and gives the sound produced a distinctive 'hollow' characteristic, which also affects the surrounding vowels. It is difficult to find a parallel in English, but the difference between 'Sam' and 'psalm' (standard English pronunciation) gives a clue. Tense the tongue muscles in pronouncing 'psalm' and you are nearly there. Now pronounce the a-vowel of 'psalm' before and after 'ḍ', saying 'aḍa', keeping the tongue tense, and that's as near as we can get to describing it in print.

3 How to use ض

The package provides three PostScript Type 1 fonts (plain, bold and typewriter), "real" fonts (regular TFM) and large virtual fonts (OVF and OFM files). There are also rudimentary FD and STY files, a MAP file, Perl scripts for conversion to (and from) UTF-8, the Perl script which builds the font and finally adjustment files, in case the user wants to change kerning and diacritic placement.

It requires LuaTEX for change of direction and OVF/OFM compliance.

To typeset in Arabic, one need only load the `dad` package and use the macro `\arab`, which is a `\long` macro: its argument may have multiple paragraphs.

Arabic text can be input in transliteration as described in Table 1 or in UTF-8. To obtain, for example, الكِتاب one would write `\arab{AlkitAb}` or `\arab{الـكِتـاب}`. By writing `\arabtt{AlkitAb}` one obtains the typewriter version الـكِتـاب (which is less appealing, but fits quite nicely with the Computer Modern Typewriter font).

3.1 Rationale of the transliteration

Here are the rules of the proposed transliteration (see Table 1):

1. pharyngeal ح = `H`, emphatic ص = `S`, ض = `D`, ط = `T`, ظ = `Z` and velar غ = `R` are *uppercased* — do not confuse them with glottal ه = `h`, non-emphatic س = `s`, د = `d`, ت = `t`, ز = `z`, and alveolar ر = `r`;
2. long vowels (ا = `A`, و = `U`, ي = `Y`) and *ʾalif maqṣūra* (ى = `I`) are also *uppercased*;
3. some consonants are modified by adding a character `h` (ذ = `dh`, ث = `th`, ش = `sh`);
4. the stand-alone *hamza* is obtained by a vertical bar `|` and letter *ʿayn* by a grave accent (which, in legacy TEX produces an inverted curly apostrophe, which is sometimes used to transliterate this letter);
5. to avoid confusion between pairs of letters and letters obtained by digraphs, one has to use a dash to separate characters: compare سه = `s-h` and ش = `sh`, or ته = `t-h` and ث = `th`;
6. more generally, the dash plays the rôle of *zero-width joiner*:[1] when writing ب = `-b`, the letter *bāʾ* will be in final form; بـ = `b-` and بـ = `-b-` will produce initial and middle letters, provided of course the letter is quadriform (as is letter *bāʾ* in this example). This is very useful when describing grammar rules, to signify that a letter (or letter group) is an affix;
7. the dash can also be used to reestablish contextual forms when combined with TEX commands, for example, to colorize letters as in Fig. 1. There is only one special case: when we want to colorize a letter of an isolated ligature لا, we add a digit `4` in front of the dash. For the final ligature ـلا it will be a digit `5`. Example: to colorize the *lām*s of تلالا, write

```
\arab{t-\textcolor{red}{-l5-}-A5%
\textcolor{red}{l4-}-A4}
```

[1] Except for the case of letter ذ = `dh` which is biform and hence is not connected with the following letter. By writing ده = `d-h` one obtains letters *dāl* and *hāʾ*, but the *hāʾ* is not in medial form, as it would be in any other case when preceded by a dash.

Table 1: Transliteration of ض system

ء	|	آ	'A	أ	'a	ؤ	'u	إ	'i	ئ	'I
ا	A	ب	b	ة	t*	ت	t	ث	th	ج	j
ح	H	خ	x	د	d	ذ	dh	ر	r	ز	z
س	s	ش	sh	ص	S	ض	D	ط	T	ظ	Z
ع	‘	غ	R	ف	f	ق	q	ك	k	ل	l
م	m	ن	n	ه	h	و	U	ى	I	ي	Y
ٱ	A*	◌ْ	o	◌َ	a	◌ِ	i	◌ُ	u	◌ً	aN
◌ٍ	iN	◌ٌ	uN	◌ّ	+	◌َّ	+a	◌ِّ	+i	◌ُّ	+u
◌ًّ	+aN	◌ٍّ	+iN	◌ٌّ	+uN	◌ٰ	a*	◌ّٰ	+a*	ﷲ	LLh
پ	p	گ	g	چ	C	ژ	J	ۀ	e	ڤ	v
ٮ	'b	ں	'n	ڡ	'f	ٯ	'q	◌ٓ	a**	◌ّٓ	+a**

8. finally, there is yet another use of the dash: when doubled, it produces a kashida stroke: compare ليل = `lYl` and لـيـل = `l--Y--l`. There is also a `\kesh` command for extensible kashida (equivalent to a `\hrulefill` using the default rule thickness font dimension `\fontdimen8`): `l--\kesh--Y--\kesh--l.` produces: لـــــــــــــيــــــــــــــل.
9. some digraphs start with an apostrophe: the *hamza*-carriers أ = `'a`, إ = `'i`, ؤ = `'u`, ئ = `'I`, آ = `'A` and also undotted letters *bāʾ* ٮ = `'b`, *nūn* ں = `'n`, *fāʾ* ڡ = `'f` and *qāf* ٯ = `'q`;
10. other digraphs end with one or more asterisks: the most frequent one is the *tāʾ marbuṭa* ة = `t*` (which can be used also in initial and medial forms, and then becomes a regular *tāʾ*). The asterisk is also used for the *waṣla* (which is only placed on the *ʾalif*) ٱ = `A*` as well as for the vertical *fatḥa* (as in هٰذا = `ha*dhA`) and the *madda*. The latter is normally used only on the *ʾalif* (آ = `'A`) but can be found also in the notorious *muqaṭṭaʿāt* in the Koran, as in عٓسٓقٓ (*Koran* 42:2) or كٓهيٰعٓصٓ (*Koran* 19:1) — sometimes it is even combined with a *šadda* (as in الٓمّٓصٓ, *Koran* 7:1 and [21, p. 111] for the *šadda*);
11. a special transcription is provided for the ligature ﷲ = `LLh` used for the اسم الجلالة "noun of majesty", which is the name of God ﷲ: in this case — and in this case only — an uppercase `L` is used. The reason is that we wish to avoid ambiguity with other uses of the trigram *lām-lām-hāʾ*, for example يُضْلِلْهُ (*Koran* 6:39) where we encounter letters لله but not with the meaning "God". In contrast to other systems, the ﷲ ligature is available also in final form (for فَلِلّٰه which occurs six times in the Koran, for example *Koran* 6:149), and it is possible to add diacritics to its first glyph (as in وَلِلّٰه, *Koran* 2:115 or لِلّٰه, *Koran* 2:165).

3.2 Unicode input

Input can be transliterated or provided directly in Unicode Arabic: `\arab{YAnis}` or `\arab{يـانـِس}` or even `\arab{يـاnis}` or `\arab{YAنـِس}` will produce the same result: يانِس.

All cells of Table 1 can be obtained by the corresponding Unicode characters (mostly via a single character, except for *šadda* + vowel combinations which require two characters). There is a special case, though: the ﷲ ligature (see next section).

For the convenience of the user who wants to write kashida (so that Arabic input is not disrupted) we have defined a command (in Arabic characters) `\تط` (تط are the first two letters of تطويل = *taṭwyl*, the Arabic name of kashida) which is exactly equivalent to `\kesh` and has to be placed between Unicode U+0640 ARABIC TATWEEL characters.

Weak. Weak verbs are those with one or more weak letters (و or ي) as radicals. There are four sub-classes:

- **Assimilated.** Assimilated verbs have *initial* و or (much more rarely) ي, and two sound radicals or middle ء and a sound final radical. Typical doubled roots are يبس, وصل.
- **Hollow.** Hollow verbs have *middle* و or ي and two sound radicals or initial ء and a sound final radical. Typical hollow roots are قول, صير.
- **Defective.** Defective verbs have *final* و or ي and two sound radicals. Typical roots are رجو, رمي.
- **Doubly weak.** Doubly weak verbs have two weak radicals, و, ي or ء. Typical doubly weak roots are ولي, سوى, أتي, رأي, سوء.

In the middle of a verse *hamzah* is merged with the final vowel of the preceding word, e.g., كهلَقَ الْإِنْسَانَ, He created man. Note that the قَ or كهلَقَ has joined the لْ or أَلْإِنْسَانَ and while the *hamzah* sign has disappeared from the text, *'alif* is retained but it is not pronounced.

Practice text 11

1. Man says — قَالَ الْإِنْسَانُ
2. Does man think? — أَيَحْسَبُ الْإِنْسَانُ
3. He said: How long hast thou tarried? — قَالَ كَمْ لَبِثْتَ
4. The truth is out — حَصْحَصَ الْحَقُّ
5. He created man from dry clay — كهلَقَنَا الْإِنْسَانَ مِنْ صَلْصَالٍ

```
\documentclass{article}
\usepackage{dad}
\begin{document}

\textbf{Weak}.  Weak verbs are those with one
or more weak letters (\arab{U} or \arab{Y})
as radicals.  There are four sub-classes:
\begin{itemize}
\item\textbf{Assimilated.} Assimilated verbs
have \emph{initial} \arab{U} or (much more
rarely) \arab{Y}, and two sound radicals or
middle \arab{|} and a sound final radical.
Typical doubled roots are \arab{Ybs},
\arab{USl}.

\item\textbf{Hollow.} Hollow verbs have
\emph{middle} \arab{U} or \arab{Y} and two
sound radicals or initial \arab{|} and a
sound final radical.  Typical hollow roots
are \arab{qUl}, \arab{SYr}.

\item\textbf{Defective.} Defective verbs have
\emph{final} \arab{U} or \arab{Y} and two
sound radicals.  Typical roots are \arab{rjU},
\arab{rmY}.

\item\textbf{Doubly weak.} Doubly weak verbs
have two weak radicals, \arab{U}, \arab{Y}
or \arab{|}.  Typical doubly weak roots
are \arab{UlY}, \arab{sUI}, \arab{'atY},
\arab{r'aY}, \arab{sU|}.

\end{itemize}

In the middle of a verse \emph{hamzah} is
merged with the final vowel of the preceding
word, e.g., \arab{khalaqa Alo'iinosaAna},
He created man.  Note that the \arab{qa}
or \arab{khalaqa} has joined the \arab{lo}
or \arab{'aalo'iinosaAna} and while the
\emph{hamzah} sign has disappeared from the
text, \emph{'alif} is retained but it is not
pronounced.\\[6pt]
\textbf{Practice text 11}\\
1.  Man says --- \arab{qaAla Alo'iinosaAnu}\\
2.  Does man think?  --- \arab{'aaYaHosabu
Alo'iinosaAnu}\\
3.  He said:  How long hast thou tarried?  ---
\arab{qaAla kamo labithota}\\
4.  The truth is out --- \arab{HaSoHaSa
AloHaq+u}\\
5.  He created man from dry clay ---
\arab{khalaqanaA Alo'iinosaAna mino
SaloSaAliN}

\end{document}
```

Figure 2: Sample LaTeX document using ض ([16, p. 5] and [21, p. 54–55])

3.2.1 The ﷲ ligature and Unicode

The ﷲ ligature is traditionally used for writing the name of God: اللهﷲ. It can be found in religious texts, but also in expressions (for example, إن شاء اللهإن شاء الله which means "hopefully" appears even in the French language as *inchallah* and in Portuguese as *oxalá*) and in the very common surname عبد اللهعبد الله Abdullah.

The problem with this ligature is that it contains a rather rare diacritic (a *šadda* combined with a vertical *fatḥa* — the latter is available in the Apple Arabic keyboard layout but not the Microsoft one) and, as a convenience, most standard fonts will replace the character string *lām-lām-hāʾ* (which would normally look like لله) by the complete ligature ﷲ; in other words: the font not only changes the glyphs but, at the same time, also adds the diacritics. This behavior is barely legitimate: a ligature (as in 'fi' or 'لا') is normally limited to a change of glyphs, and should not add new characters (in this case, characters U+0651 ARABIC SHADDA and U+0671 ARABIC LETTER SUPERSCRIPT ALEF) since this means that what is rendered no longer corresponds to the underlying Unicode character string.

Nevertheless, for the user's convenience, we have adopted that behavior also in ض, but only in the case of Unicode input. Therefore when the user types Unicode *lām-lām-hāʾ* (the first *lām* must not be preceded by a quadriform letter), the system will produce the ﷲ ligature.

This method will not work if a diacritic is inserted between the two *lām*s, or if the first *lām* follows a quadriform letter and hence will be medial. For that case, we have defined a macro /لـلـه (the macro name is in Arabic script so that right-to-left direction is not disrupted) which takes an argument: the vowel between the two *lām*s. Hence, to obtain فَلِلّٰهِ the user can choose between one of the following:

فـَ/لـلـه{◌ِ}

faLiLhi

The dotted circle, used to show the combining nature of short vowels and other diacritics, can be obtained by the macros `\arabdottedcircle` or /دائرة with the macro name in Arabic script.

4 TEXnicalities

The ض font is a *tour de force* of smart ligature use: for example, to obtain $k_1t_2b_3$ (كتب) out of `ktb` = $k_0t_0b_0$ (or `"0643"062A"0628`) input, one needs the following smart ligatures:

k_0 `LIG/` $t_0 \to k_1\ t_0$
k_1 `/LIG` $t_0 \to k_1\ t_3$
t_3 `LIG/` $b_0 \to t_2\ b_0$
t_2 `/LIG` $b_0 \to t_2\ b_3$

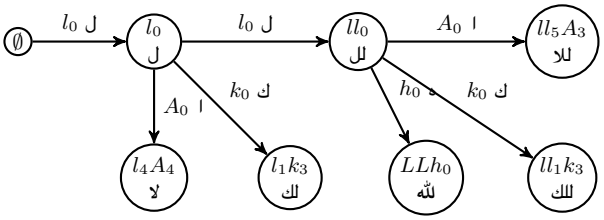

Figure 3: Finite state automaton starting with an isolated *lām* (*ʾalif* ا stands for the set of letter $\mathcal{A} = \{$ ا, أ, إ, آ, ٱ $\}$; ك stands for any Arabic letter besides ه and set $\mathcal{A}$.

as well as the following four for Unicode input:

`"0643` `LIG/` `"062A` $\to k_1\ t_0$
k_1 `/LIG` `"062A` $\to k_1\ t_3$
t_3 `LIG/` `"0628` $\to t_2\ b_0$
t_2 `/LIG` `"0628` $\to t_2\ b_3$

The first ligature of each group leaves t/`"062A` unchanged (isolated) and turns k/`"0643` into initial form. Then the second ligature takes k_1t_0/k_1`"062A`, leaves k unchanged (initial) and turns t/`"062A` into final form. But, because of the following b/`"0628`, the third ligature will turn t into medial form, leaving b/`"0628` unchanged. And, finally, the fourth ligature will leave t unchanged and turn b/`"0628` into final form. It is noteworthy that t changes form thrice: from isolated (default) it turns into final and then into medial form.

All basic Arabic glyphs are placed into the first 8-bit table. Then one 8-bit table (except for table `"06xx` which is used for Unicode input) is added for every letter + diacritic(s) combination, so that we have, in total, 20 tables. The complete font contains 3,514 virtual glyphs, 403,913 ligatures (321,935 of which are smart) and 7,810 kerns.

The most challenging letter is *lām*: the font contains 3 initial *lām*s, 4 medial ones as well as 3 "fake" ligatures *lām-lām* ("fake" in the sense that they are only needed because of TEX's approach of building ligature stepwise and hence needing intermediate steps for all ligatures of length three and more: to obtain the *lām-lām-hāʾ* ligature (see § 3.2.1) one needs an intermediate *lām-lām*, even though this pair of letters does not take any special form. In Fig. 3 the reader can see the finite state automaton starting with an isolated *lām*.

The virtual OVP font is built from the metrics of the PostScript Type 1 font by a Perl script. This script also reads configuration files specifying all kern pairs as well as all horizontal and vertical adjustments of diacritics. By this method, every letter has its diacritics placed at optimal positions. To compile the OVP file produced by the script into

OVF, it is mandatory to use tool *wovp2ovf* of version higher than "1.13 (build 34787)", which will be included in TEX Live 2015.

Names of PostScript glyphs are standard,[2] so that copy-paste from a PDF file results in almost perfect Unicode strings.

4.1 Conversion to and from UTF-8

As a tool for users, we provide two Perl scripts allowing conversion from UTF-8 to our transliteration scheme and back. These scripts can be applied selectively using, for example, the feature of many advanced text editors of applying text filters to selected text areas.

5 Conclusion

There was a period (in the early days of non-Latin-alphabet TEX [4, 5, 14]) where transliteration of input text was the only available method. Then, when Unicode was sufficiently widespread, TEX switched to tools allowing direct non-Latin input. In the case of Arabic, because of the particular characteristics of this script, this is — even today — not always the optimal solution, especially when we are dealing with short extracts of Arabic text combined with Latin-alphabet text and TEX commands. Maybe now is the time to return to methods based on transliteration, as an alternative to direct-input methods. We have implemented this approach, using *only* smart ligatures, as defined by Donald Knuth in 1990 [13], and the large virtual font format introduced by Ω and taken over by LuaTEX.

We hope that this package will be useful to users seeking a straightforward method to introduce short Arabic extracts into Latin-alphabet documents.

References

[1] Adobe Systems. Adobe glyph list. `http://partners.adobe.com/public/developer/en/opentype/glyphlist.txt`, 2002.

[2] Hàn Thế Thành. Micro-typographic extensions to the TEX typesetting system. *TUGboat*, 21(4):317–434, 2000. `http://pdftex.org`.

[3] John Hammersley, John Lees-Miller, et al. The writeLATEX online collaborative LATEX editor. `http://www.writelatex.com`.

[4] Yannis Haralambous. Arabic, Persian and Ottoman TEX for Mac and PC. *TUGboat*, 11:520–522, 1990.

[5] Yannis Haralambous. Towards the revival of traditional Arabic typography through TEX. In *Proceedings of EuroTEX'92*, pages 293–305. CSTUG, 1992.

[6] Yannis Haralambous. *Fonts & Encodings*. O'Reilly, 2007.

[7] Yannis Haralambous, Yassir Elidrissi, and Philippe Lenca. Arabic language text classification using dependency syntax-based feature selection. Submitted to *CITALA 2014*.

[8] Yannis Haralambous and John Plaice. First applications of Ω: Adobe Poetica, Arabic, Greek, Khmer. *TUGboat*, 15:344–352, 1994.

[9] Yannis Haralambous and John Plaice. Multilingual typesetting with Ω, a case study: Arabic. In *Proceedings of the International Symposium on Multilingual Information Processing '97*, pages 137–154. ETL, Tsukuba, Japan, 1997.

[10] Yannis Haralambous and John Plaice. The design and use of a multiple-alphabet font with Ω. In *Electronic Publishing, Artistic Imaging, and Digital Typography*, volume 1375 of *LNCS*. Springer, 1998.

[11] Taco Hoekwater. LuaTEX. *TUGboat*, 28:312–313, 2007. `http://luatex.org`.

[12] Jonathan Kew. XETEX, the multilingual lion: TEX meets Unicode and smart font technologies. *TUGboat*, 26:115–124, 2005. `http://tug.org/xetex`.

[13] Donald E. Knuth. The new versions of TEX and METAFONT. *TUGboat*, 10:325–328, 1989.

[14] Klaus Lagally. ArabTEX — Typesetting Arabic with vowels and ligatures. In *Proceedings of EuroTEX'92*, pages 153–172. CSTUG, 1992.

[15] Ahmed Lakhdar-Ghazal. *Pour apprendre et maîtriser la langue arabe*. Institut d'études et de recherches pour l'arabisation, Rabat, Morocco, 1991.

[16] John Mace. *Arabic verbs and essential grammar*. Teach yourself books, 1999.

[17] Nicole Richert. *Arabisation et technologie*. Institut d'études et de recherches pour l'arabisation, Rabat, Morocco, 1987.

[18] Karin C. Ryding. *Arabic. A linguistic introduction*. Cambridge University Press, Cambridge, 2014.

[19] Michael Shell. IEEEtran LATEX class. `http://ctan.org/pkg/ieeetran`, 2007.

[20] John R. Smart. *Arabic*. Teach yourself books, 1986.

[21] Barakat Ahmad Syed. *Introduction to Quranic script*. Curzon Press, 1984.

◇ Yannis Haralambous
Institut Mines Télécom, Télécom Bretagne, UMR CNRS 6285 Lab-STICC
Technopôle Brest Iroise CS 83818, 29238 Brest Cedex 3, France
`yannis.haralambous (at) telecom-bretagne dot eu`

[2] These names are either taken from the Adobe Glyph List [1], or using the standard convention `uniXXXX.var` where `XXXX` is the Unicode position of the character and `var` ∈ {`ini`, `med`, `fin`, `iso`}. Ligatures are named by the names of their components, concatenated using underscores.