ArabTEX — Typesetting Arabic with Vowels and Ligatures

Klaus LAGALLY

Universität Stuttgart
Institut für Informatik
Breitwiesenstraße 20-22
D-7000 Stuttgart 80, Germany

lagally@informatik.uni-stuttgart.de

Abstract

We present a TEX macro package for generating the arabic writing from a standardized ASCII input notation. It can handle partial or full vocalization, and generates automatically most of the common ligatures. There is limited support for Farsi, Urdu, and Pashto. ArabTEX is compatible with Plain TEX and also most LATEX environments; arabic and other material can be mixed freely. For special purposes the standard transliteration can be additionally generated.

ArabTEX uses no preprocessor and thus should be compatible with any TEX implementation that allows dynamic loading of additional macro files and fonts.

Key words: Arabic, transliteration, vocalization, ligatures.

1 Introduction

This is a personal story. The author, interested in the arabic language since he was a young boy, some time ago by accident found out about an evening course on Arabic at a local school, and decided to join in. The course was designed for people wishing to visit an arabic country with some knowledge of Arabic, and as the teacher would not recommend any suitable and easily affordable textbook for that purpose, he handed out his own handwritten notes. This intrigued the author, and so he bought an arabic grammar book from a renowned publisher [Fischer87]. Upon closer inspection the arabic examples looked somewhat strange, and after contacting the author of the book it turned out that the latter had added the vowel signs to the arabic examples on the printing plates by hand!

This came as a great surprise, especially when considering the fact that the underlying printed arabic text looked beautiful. Apparently there remained some unsolved problems in the printers' business, and knowing the power of TEX[Knuth84], the author decided to try doing something about it.

The result of that effort is now called ArabTEX, a system consisting of a large macro package and several fonts.

2 Design goals

The typical user of ArabTEX, as we imagine her/him,

- knows some Arabic,
- is interested in high quality writing,
- has little money to spare,
- cannot afford specialized equipment,
- is willing to learn some simple rules, but:
- is not, and is not willing to become, a TEX expert.

This description fits well onto several linguists we know. Alas, not every one of them can even afford a simple PC.

From this projected user profile follow some requirements for the system:

- it should be inexpensive,
- it should not require specialized equipment,
- it should be easily portable,
- it should be sufficiently powerful to generate any reasonable arabic text with high quality,
- it should, after some training, be usable by a person who is not a computer expert.

However,

- it need not be extremely efficient,

- it need not support everyday office use,
- it need not be interactive.

As it happens, our starting point was TEX (in fact, LATEX[Lamport86]), and we noticed that there are two quite different populations of TEX users:

- the experts, in full control of all specialized features, constantly finding new applications, and
- the everyday users, getting their work done by filling in some forms designed by a expert, and letting TEX do the rest.

Our hypothetical user definitely belongs to the second category. Therefore, for him it is extremely important to have a convenient user interface. Devising such an interface turned out to be a major task.

3 Characteristics of the Arabic script

The arabic script, like the scripts for all semitic languages, runs from right to left. This fact, whereas leading to some complications in connection with line-breaking whenever we want to mix arabic and non-arabic texts, turned out to be an absolutely minor problem in comparison with the fact that the arabic script is a cursive style, extremely well adapted to hand-writing. As far as we know, this has always been so [Endress82b], and contrary to common belief the script is very easy to write; even a motivated beginner can acquire a fair hand-writing style within a few weeks. Calligraphic excellence, of course, is a different matter [Schimmel70].

In a cursive hand, we do not assemble character after character on a common baseline, but try to join adjacent letters into a softly flowing curve. This makes for ease of writing, and also for aesthetic beauty, but has the consequence that the script, although still arranging the individual words in a horizontal sequence, is essentially two-dimensional. Another consequence is that the form of a letter depends on the context, and if adjacent letters are combined into ligatures a surprising manifold of different forms may emerge. Most of these are not mandatory, but their omission will lead to a serious loss of quality that can easily be noticed even by an outsider, and quality has always been considered very important.

A script of that characteristic is not very convenient to print, and indeed the arabic script has resisted mechanization for a long time [Endress82a].

The first attempts to print Arabic with movable type were undertaken about 1500 A.D., surprisingly in central Europe, but the printing tradition of Arabic seriously started in 1727 when the "Ottoman printing agency" in Istanbul was founded. It had the types made in the Netherlands where the technology existed, and for several decades only official documents and scientific works were allowed to be printed. Religious works like the Qur'an and its commentaries still were reproduced by hand-writing, and later by lithography from hand-written originals; thus the risk of misprints in the Holy Scriptures was avoided. A second official printing agency was founded 1821 in Cairo; others followed, and in 1906 a new typeface standard was adopted, with remarkably good results, that is still in use today.

Of the several different writing styles that exist, Naskhi was adopted for printing as it is very easily readable, and mostly adheres to the baseline. Still, even printing Naskhi is a formidable task; whereas a european printer's box contains less than 100 different letter forms including capitals, digits, and special characters, you need far more than 500 different forms for good quality arabic printing.

The situation improved in the 1970's when photo-typesetting equipment became available and the first computer programs to typeset Arabic were developped [MacKay77]. Now also other writing styles like Nasta'liq, as used mainly in Iran and the adjacent countries, could be handled, and many new typefaces, e.g. for newspapers, were developped. But you can still find headlines which have obviously been reproduced from a hand-written original. The calligrapher's profession is still alive (see, e.g., [Hāšim80]).

Even if the technology for printing arabic texts nowadays exists, some problems remain. In the Arabic language, as in all semitic languages, the main information resides in the consonants and the long vowels, and usually only these are written explicitly. Short vowels, the doubling of a consonant, and the like are either not indicated at all or expressed by diacritical marks placed above or below the characters. A native speaker generally does not need this additional information as he can deduce it from the context; it is only required when introducing new words, for resolving ambiguities, and in religious texts where the exact pronunciation is considered important. Considering the already very large number of different letter forms in a printer's box, also storing all the possible combinations would be prohibitively expensive, and thus manual corrections are necessary. This is awkward and expensive, so it is avoided whenever possible, and thus the religious texts we have seen all have been reproduced from manuscripts.

3.1 Transcription and Transliteration

If we want to generate the arabic writing of a given text automatically, we have to denote the text in a way that can readily be processed by our computer. There exists no standard suitable for our purpose, so we have to invent one; and since linguists always had related problems and also are among our prospective users, we try to imitate their solutions as closely as possible. In this context there exist two concepts that are closely related (and therefore frequently confused): transcription and transliteration.

"Transcription" means: representing the *sounds* of the given language as closely as possible. This can even be done in the language itself, e.g., transcribing the sound of the english word "enough" as "enuff"; on the other hand there exists a language independent standard, the International Phonetic Alphabet.

"Transliteration" on the other hand means: representing the *writing* of the given language by using a different set of characters. In theory, just a unique representation is needed; in practice it is also required that the transliteration be easily readable, and also give some indication of the sounds. Therefore some compromises are usually made, with the consequence that deducing the writing from the transliteration requires some knowledge of the language in question.

For Arabic and some other languages using the arabic script, there exist two nearly identical international standards [DIN31635, ISO/R233] for transliteration in the given loose sense. As there are more arabic letters than in the Latin alphabet, these conventions make heavy use of diacritical marks, and so we cannot use them directly for our purpose.

3.2 Input notation

If we want to typeset arabic texts with TEX, we have two possibilities:

- either have a preprocessor transform our input text into some intermediate notation that can be processed by TEX,
- or enhancing the power of TEX by adding suitable macros so that it can process our input text directly.

The first possibility is extremely flexible, as far as the possible input codings are concerned, and can be made very efficient. It has been used

in some existing systems, e.g. *Scholar*TEX [Haralambous91]. However, every user now needs a version of the preprocessor tailored to her/his computer system and cooperating well with the local TEX implementation. Thus we may run into portability and maintenance problems, and possibly a complicated installation procedure.

The second possibility, which we adopted, by itself is as portable as TEX itself is; but, writing the needed algorithms in TEX macro language is no easy task, and the macros might not run as efficiently as a preprocessor system. Like everywhere, here also is a tradeoff between generality and speed.

If, as we did, we choose the macro solution then TEX must be able to read our input notation directly, therefore we should better use only the standard 7-bit ASCII characters (there are extensions to TEX using 8-bit characters but these are in no way standardised so we could run into severe compatibility problems). We want to keep the input notation easily readable, but we have the problem that we need about 30 different letters, and some of them sound very much alike. Even when also using the capital letters for coding (Arabic needs no capitals), we could not find a one-to-one correspondence between ASCII characters and arabic sounds that is easy to read and remember.

The solution we finally found was to use both one-character and two-character encodings, and to adhere closely to the standard transliteration. The rules are simple:

- whenever the transliteration uses just a single letter, we also use that letter;
- whenever the transliteration uses a letter with a diacritical mark, we use the same letter and *precede* it with the punctuation mark most closely resembling the diacritic.

This is easily remembered, fairly readable, and works well because punctuation marks (except hyphen) never occur within a word.

Using this coding scheme we get an additional bonus: if, for some reason, we want to also typeset the standard transliteration of an arabic word, we have to code the diacritical marks used; and whereas this can be done in TEX using existing commands, these look awkward and are not easy to learn and remember. On the other hand it turned out not to be too difficult to derive the transliteration from our coding scheme, and so we can use it for both

purposes, thereby avoiding the danger of constantly confusing two closely related, but different, notations.

In fact, the description we gave is somewhat oversimplified. There are some (fortunately rare) exceptions to the transliteration rules, and sometimes words written differently are transcribed identically, so in these cases we have to code additional information.

4 Processing Arabic Text

In the following we give a general overview of the tasks our system has to perform when typesetting Arabic. We discuss this in the context of a simplified model: viz., that a text as seen by TEX is a sequence of paragraphs, each of which is a sequence of words. TEX will transform each word into an internal representation and will arrange these word images into lines. The sequence of lines thus generated will be broken up into pages which will be sent to a device-independent output file, later to be viewed or printed by a device-dependent driver program. There is indeed much more to it but the details are not relevant to our exposition.

4.1 Overall structure: Quotations, Paragraphs

If we want to typeset a document containing arabic text, we will distinguish two different cases:

- short arabic quotations inside a line of text in some european language,
- longer arabic passages consisting of one or several paragraphs.

An in-line quotation is handled as a whole. We process the arabic words in reverse order, one word at a time, and insert the results into the normal output. This could lead to problems if a quotation would be split across a line boundary, because in that case the two parts should be individually reversed. We ought to do the line-breaking first and the reversal afterwards, but we know of no easy way of doing that with TEX. To handle this problem, an extension of TEX, TEX–XET, has been proposed [Knuth and MacKay87], but it is not generally available, and also not compatible with the standard printer driver programs. So we have to forbid line-breaking within a quotation, and for technical reasons quotations have to be very short anyway.

Longer arabic passages are handled differently. Here we process the individual words in their natural order, arrange the results in reverse order, and do the line-breaking ourselves. Inside an arabic paragraph we can again have insertions, e.g., short quotations (now of non-arabic text), or even in-line mathematical formulas. For the same reasons as above, we have again to forbid line-breaks inside an insertion.

In both cases we have to take care of the fact that numbers in Arabic are written like in the european languages, i.e., the sequence of digits is not reversed. We could have put the responsibility for indicating what is considered to be a number on the user; however we decided just to define a number as a sequence of characters starting with a digit and ending with a space, and to typeset this sequence in the natural order.

4.2 Numbers, Words, Subwords

As we saw, every arabic word or number is processed individually, and the result is a description of its graphical representation given in terms of symbols from a given font arranged in a two-dimensional pattern. There is no unique correspondence between these symbols and arabic characters; a character image might be built up from several symbols, and it also sometimes happens that a symbol represents more than one character. The reason behind this is that the arabic characters may be collected into several classes whose members are closely related and differ only in a few features that can be separated out. Fortunately the same is true for the ligatures, and we can also handle the vocalization by the same mechanism, so that a single font of less than 256 characters is sufficient for expressing a much larger set of graphical symbols and combinations.

When we want to typeset a number in the arabic script, we just arrange the isolated graphical symbols corresponding to the digits from left to right and we are done.

Typesetting a word of text is more involved. Logically, a written word consists of a sequence of character images connected to each other as far as possible, and possibly changing their shape depending on the context. In addition, these character images may carry diacritical marks. Not all characters can be joined to their successors (probably because the writing would become ambiguous otherwise), and thus we can consider a word being a sequence of subwords, whose characters are all connected. To each subword corresponds a graphical representation, and these are arranged side by side.

In this step they are possibly displaced vertically such that their last (i.e. leftmost) character has its normal position on the baseline, and horizontally such that their spacing looks pleasant.

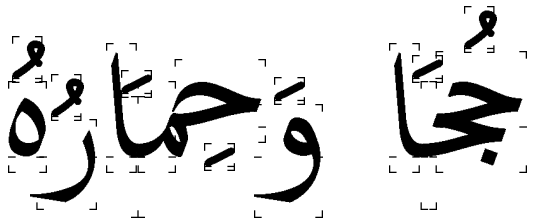

Figure 1: Character assembly with components shown.

4.3 Characters, ties, diacritics

Depending on its position in a subword, a character might take on one of several shapes: the isolated, initial, medial, and final shape. This forms might still be modified if the character enters into a ligature. Fortunately, as far as ligatures are concerned all characters of a class perform alike, thus the number of different cases, although large, remains manageable.

When we process a word, we perform the following steps:

- we sequentially process the input representation to break it up into a sequence of individual characters, each with accompanying diacritical information;
- we process this sequence in reverse order to determine the shape of each character depending on its position in the subword and on the surrounding characters;
- starting on the baseline, we position these character shapes so that they join smoothly, either directly or by means of connecting strokes. To each character, we add the appropriate diacritical marks (there may be none or even more than one per character). For an example, see Figure 1.
- Whenever the next character considered (this is the preceding one, when writing by hand!) cannot be joined to its logical successor, we have reached a subword boundary; we reposition this character so it will again sit on the baseline, and add suitable spacing.

The resulting graphical representation of the word is passed back to the caller to be inserted into the output.

5 User Interface

In the following we shall only describe the main features; for more details, see the ArabTEX documentation [Lagally92].

5.1 Activating ArabTEX

To use the ArabTEX package with a file to be processed by Plain TEX, load it via `\input arabtex`; with LATEX, include `arabtex` as a document style option. In both cases, several additional files and the default font will be installed.

5.2 Mode control

As there are several language-dependent writing conventions, you have to select a language by one of the commands `\setarab`, `\setfarsi`, `\seturdu`, `\setpashto`, or `\setverb` (no special processing in this case).

There are three different modes of handling short vowels:

- `\vocalize`: short vowels written in the input will be indicated in the output by diacritical marks;
- `\fullvocalize`: also the absence of a short vowel will be indicated;
- `\novocalize`: short vowels will show up in the transliteration, but will be omitted in the arabic writing. You can locally override this feature.

By `\arabtrue`, `\arabfalse`, `\transtrue`, `\transfalse` you can switch on and off the generation of the arabic writing and/or the standard transliteration. By default, the arabic writing is on, and the transliteration is off.

Bold-face can be selected by `\setbold`; `\setnormal` will revert to normal.

5.3 Arabic text

Short arabic quotations in normal text are included in angle brackets. These thus have a special significance (outside of mathematical mode) and can no

more be used for other purposes, e.g., for normal text or in local macros. This special behaviour is switched on by language selection, and can be switched off again by `\setnormal`.

An arabic paragraph is started by the command`\begin{arabtext}` and ends with `\end{arabtext}`. This looks like, and nearly operates like, a LaTeX environment even when working with Plain TeX. However, neither displayed mathematical text nor other LaTeX environments may be nested in an arabic paragraph.

Inside an arabic paragraph we can have non-arabic quotations delimited by angle brackets, and in-line mathematical formulas delimited by single dollar signs. These insertions must fit on one output line.

5.4 Input coding

Table 1: Coding of arabic characters

a	ا	*a*	b	ب	*b*	p	پ	*p*	t	ت	*t*
_t	ث	*t̲*	^g	ج	*ǧ*	.h	ح	*ḥ*	_h	خ	*ḫ*
c	ﻋ	*c*	^c	چ	*č*	,c	ﺣ	*ć*	d	د	*d*
_d	ذ	*d̲*	r	ر	*r*	z	ز	*z*	^z	ژ	*ž*
s	س	*s*	^s	ش	*š*	.s	ص	*ṣ*	.d	ض	*ḍ*
.t	ط	*ṭ*	.z	ظ	*ẓ*	‘	ع	‘	.g	غ	*ġ*
f	ف	*f*	q	ق	*q*	v	ڤ	*v*	k	ك	*k*
g	گ	*g*	l	ل	*l*	m	م	*m*	n	ن	*n*
h	ه	*h*	w	و	*w*	y	ي	*y*	T	ة	*t*

The input notation, the arabic writing in the isolated form, and the transliteration of the characters used for Arabic and Persian are given in Table 1. For Urdu, Pashto, and for special purposes there are some additional codings. Note also the following:

- `<T>` is *tah marbouta*, `<N>` is *tanwin*, `<Y>` is *alif maqsoura*.
- `<A>`, `<I>`, `<U>` denote the long vowels, `<a>`, `<i>`, `<u>` the short vowels if required.

- `<'>` (right quote) is *hamza* (glottal stop). After `\setarab`, its carrier will be determined by the context according to the full *hamza* rules, otherwise by a following short vowel.
- `<'A>` generates *madda*.
- Doubled consonants are written twice (*shadda*).
- `<|>` will break unwanted ligatures, `<->` joins two words and will only show up in the transliteration, and `<-->` will elongate the connection between two adjacent letters (*kashida*).
- The definite article is always written `<al->` (with hyphen), even if it precedes a (double) "sun letter".

5.5 Special features

For Farsi, Urdu, Pashto and some other languages using the arabic script, the coding conventions are slightly different, and not described here. Furthermore, the language-specific processing may be locally overridden, and there is also a verbatim mode capable of representing unusual or archaic ways of writing. Mode-changing commands may also occur inside an arabic paragraph thus allowing local mode changes.

6 Implementation

The ArabTEX system consists of a large number of macros, and their interaction is surprisingly complex. They are grouped into several packages, each devoted to a separate task. As ArabTEX can be considered a translator, we imitate the usual modularization of a compiler. In that view, ArabTEX consists of a Driver Module calling a number of auxiliary modules for specialized tasks, and finally passing the output back to the normal TEX paragraph mechanism. Thus arabic text can also appear inside most LATEX environments, including moving arguments. However, LATEX is no prerequisite for running ArabTEX.

6.1 The Driver Module

The Driver Module, `arabtex.sty`, is loaded by LATEX or by a small Loader Module, `arabtex.tex`, when using Plain TEX. The latter module simulates the (few) LATEX features used by ArabTEX.

The Driver Module, when executed, defines and initializes some common variables and loads the remaining files constituting ArabTEX. It also implements the mode-changing commands, and contains several local submodules:

- the Insertion Processor for arabic quotations,
- the Paragraph Processor for arabic paragraphs,
- the Output Processor,
- the Word Processor.

Both the Insertion Processor and the Paragraph Processor pass single arabic words to the Word Processor to generate the graphical representation (and/or possibly the transliteration) and process the resulting output further.

The Insertion Processor breaks up short quotations into individual words and feeds both the resulting arabic representation and the transliteration into the normal output stream.

The Paragraph Processor also breaks up the input into individual words; the output of the Word Processor, however, is now handled differently. The transliteration, if generated, is fed into the normal output stream; the arabic representation is passed to the Output Processor.

The Output Processor lines up the arabic representations from right to left in a local buffer. Whenever a line is completed, it is interleaved with the normal output, if any. At the end of an arabic paragraph, the buffer is flushed, and the paragraph is finished by the normal TEX paragraphing mechanism. For an example, see Figure 4.

The Word Processor passes the input to the Scanner Module, `ascan.sty`, to generate a standardized internal representation independent of the external coding. This internal representation is then passed to the Transliteration Module, `atrans.sty`, if the transliteration is wanted. Otherwise, or additionally, it is passed to the Parser Module, `aparse.sty`, to isolate the individual graphical components. The output of the Parser Module is further processed by the Assembly Module, `awrite.sty`, to generate the arabic representation.

6.2 The Scanner Module

The main task of the Scanner Module is to break up the input stream into tokens denoting individual arabic characters; should the input notation be changed, then only the Scanner Module would have to be adapted accordingly. There is one case handled in a special way: for *hamza* the character preceding it is repeated after it to ease further processing.

6.3 The Transliteration Module

This module has to transform the sequence of tokens into the external representation of the standard transliteration. As the transliteration does not always follow the arabic writing closely, some special cases have to be considered, e.g., in connection with endings and with the definite article whose spelling depends on the first consonant of the following word. Also sometimes an initial vowel has to be suppressed (*wasla*).

ǧuḥā wa-ḥimāruhu

›atā ṣadīqun ›ilā ǧuḥā yaṭlubu minhu ḥimārahu li-yarkabahu fī safratin qaṣīratin wa-qāla lahu: sawfa ›u‹īduhu ›ilayka fī 'l-masā›i , wa-›adfahu laka ›uǧratan.fa-qāla ǧuḥā: ›anā ›āsifun ǧiddan ›annī lā ›astaṭī‹u ›an ›uḥaqqiqa laka raġbataka, fa-'lḥimāru laysa hunā 'l-yawma.wa-qabla ›an yutimmu ǧuḥā kalāmahu bada›a 'l-ḥimāru yanhaqu fī 'ṣṭablihi.fa-qāla lahu ṣadīquhu: ›innī ›asma‹u ḥimāraka yā ǧuḥā yanhaqu.fa-qāla lahu ǧuḥā: ġarībun ›amruka yā ṣadīqī! ›atuṣaddiqu 'l-ḥimāra wa-tukaḏḏibunī?

Figure 2: Arabic transliteration.

6.4 The Parser Module

The Parser Module has to break up the token sequence into a backward sequence of "writing syllables". A "writing syllable" is not to be confused with a syllable in the usual sense, but consists of a single consonant or long vowel with additional diacritical information denoting e.g., a short vowel, consonant doubling, *tanwin* and *hamza*. Whereas the basic algorithm is straightforward, there is a surprisingly large number of special cases since the various languages supported by ArabTEX have different notational conventions, and there are also some options (not described here) to locally modify the writing. A typical example is the handling of *hamza*, the glottal stop. Whereas denoting a distinctive sound, it is not considered a letter, and

thus a carrier for it has to be determined which depends on the context in a rather complicated way.

جُحَا وَحِمَارُهُ

أَتَى صَدِيقٌ إِلَى جُحَا يَطلُبُ مِنهُ حِمَارَهُ لِيَركَبَهُ فِي سَفرَةٍ قَصِيرَةٍ وَقَالَ لَهُ :

سَوفَ أُعِيدُهُ إِلَيكَ فِي المَسَاءِ ، وَأَدفَعُ لَكَ أُجرَةً .

فَقَالَ جُحَا :

أَنَا آسِفٌ جِدًّا أَنِّي لَا أَستَطِيعُ أَن أُحَقِّقَ لَكَ رَغبَتَكَ ، فَالحِمَارُ لَيسَ هُنَا اليَومَ .

وَقَبلَ أَن يُتِمَّ جُحَا كَلَامَهُ بَدَأَ الحِمَارُ يَنهَقُ فِي اصطَبلِهِ .

فَقَالَ لَهُ صَدِيقُهُ :

إِنِّي أَسمَعُ حِمَارَكَ يَا جُحَا يَنهَقُ .

فَقَالَ لَهُ جُحَا :

غَرِيبٌ أَمرُكَ يَا صَدِيقِي ! أَتُصَدِّقُ الحِمَارَ وَتُكَذِّبُنِي ؟

Figure 3: Vocalized Arabic text.

6.5 The Assembly Module

Finally, from the reversed sequence of "writing syllables" produced by the Parser Module, the graphical representation is determined. Every "writing syllable" consists of a basic character and diacritical information. Every character belongs to a character class, represented by a "skeleton", and is locally identified by a "modifier" (usually a pattern of dots).

The further processing of a "writing syllable" proceeds in several steps:

- The skeleton and the modifier are determined.
- Depending on context, the appropriate joining form of the skeleton (isolated, initial, medial, final) is determined.
- Also depending on the context, the skeleton may take part in a ligature and thus get a different shape. Generally, and with very few exceptions, ligature generation is optional; and since it is also complicated (though not difficult), it has been delegated to a separate Ligature Module, `aligs.sty`.

- After the definite form of the skeleton has been determined, it is positioned in the output. If it is an isolated or final shape, it is generally put on the baseline with suitable spacing to its left neighbour, if any. Otherwise it is joined to its left neighbour, either directly or by means of a connecting stroke whose form depends on the partners. As the connection point of its left neighbour need not be on the baseline, the skeleton possibly must be vertically adjusted, and a new connection point for its right neighbour, if that exists, will be determined.
- After positioning the skeleton, the modifier will be added to identify the character in question.
- Finally, the diacritical information is added.

6.6 The Ligature Module

This module is called by the Assembly Module for each character. It will receive as input information a description of a skeleton shape and the shape of its right neighbour, and will return a possibly changed skeleton shape, a possibly changed shape of the right neighbour, and frequently also a connecting stroke. With the exception of very few, but important, cases where ligatures are mandatory, the Ligature Module might return its input information unchanged, and indeed there is an option to switch most ligatures off. However, the art of forming ligatures evolved gradually during many centuries of writing, and their inclusion will greatly improve the quality of the result; and whereas a good many cases are handled already, there is still room for improvement.

7 Experiences

One of the reasons for implementing ArabTEX this way was to test the power of TEX on a large example. We found that it could be done, but we drastically underestimated the amount of work involved. The techniques used in the described modules are comparatively straightforward; even the full power of context-free language analysis is rarely needed. However, due to the great number of special cases the complexity is considerable, and the macro technique used is extremely vulnerable to trivial coding errors whose effects will propagate throughout the system very quickly, and frequently will lead to very puzzling results. Thus systematic structuring is a must, and

جُحَا وَحِمَارُهُ *ğuḥā wa-ḥimāruhu*

›atā ṣadīqun ›ilā ğuḥā yaṭlubu minhu ḥimārahu li-yarkabahu fī safratin qaṣīratin wa-qāla lahu:

أَتَى صَدِيقٌ إِلَى جُحَا يَطلُبُ مِنهُ حِمَارَهُ لِيَركَبَهُ فِي سَفرَةٍ قَصِيرَةٍ وَقَالَ لَهُ :

sawfa ›u‹īduhu ›ilayka fī 'l-masā›i , wa-›adfahu laka ›uğratan.

سَوفَ أُعِيدُهُ إِلَيكَ فِي المَسَاءِ ، وَأَدفَهُ لَكَ أُجرَةً .

fa-qāla ğuḥā:

فَقَالَ جُحَا :

›anā ›āsifun ğiddan ›annī lā ›astaṭī‹u ›an ›uḥaqqiqa laka raġbataka, fa-'lḥimāru laysa hunā 'l-yawma.

أَنَا آسِفٌ جِدًّا أَنِّي لَا أَستَطِيعُ أَن أُحَقِّقَ لَكَ رَغبَتَكَ ، فَالحِمَارُ لَيسَ هُنَا اليَومَ .

wa-qabla ›an yutimmu ğuḥā kalāmahu bada›a 'l-ḥimāru yanhaqu fī 'ṣtablihi.

وَقَبلَ أَن يُتِمَّ جُحَا كَلَامَهُ بَدَأَ الحِمَارُ يَنهَقُ فِي اصطَبلِهِ .

fa-qāla lahu ṣadīquhu:

فَقَالَ لَهُ صَدِيقُهُ :

›innī ›asma‹u ḥimāraka yā ğuḥā yanhaqu.

إِنِّي أَسمَعُ حِمَارَكَ يَا جُحَا يَنهَقُ .

fa-qāla lahu ğuḥā:

فَقَالَ لَهُ جُحَا :

ġarībun ›amruka yā ṣadīqī! ›atuṣaddiqu 'l-ḥimāra wa-tukaḏḏibunī?

غَرِيبٌ أَمرُكَ يَا صَدِيقِي ! أَتُصَدِّقُ الحِمَارَ وَتُكَذِّبُنِي ؟

Figure 4: Arabic text with transliteration.

a complete redesign after having a working prototype payed off very well and led to a considerable increase of stability. There are still some errors in the system, but they seem to be well hidden, and show up at a surprisingly low rate.

Furter plans, besides correcting errors, are: designing a Nasta‘liq font that looks better for Persian, and generally improving on the still very rudimentary support for non-arabic languages using the same script.

Acknowledgments

The development of ArabTeX would not have been possible without the assistance of many people. Apart from my local team, helpful advice came among others from Ivan Derzhansky, Wolfdietrich Fischer, Ahmed El-Hadi, Abdelsalam Heddaya, Iqbal Khan, Tom Koornwinder, Eberhard Krueger, Asif Lakehsar, Jan Lodder, Richard Lorch, Eberhard Mattes, and Bernd Raichle. I also have to thank the many users who sent bug reports and comments.

References

[DIN31635] DIN 31 635: *Umschrift des Arabischen Alphabets*, Deutsches Institut für Normung e.V., 1982.

[Endress82a] Gerhard ENDRESS, *Die Arabische Schrift*, in [Fischer82], p. 165 ff.

[Endress82b] Gerhard ENDRESS, *Handschriftenkunde*, in [Fischer82], p. 271 ff.

[Fischer82] Wolfdietrich FISCHER (ed.), *Grundriß der Arabischen Philologie*, Band 1: Sprachwissenschaft, Dr. Ludwig Reichert Verlag, Wiesbaden 1982.

[Fischer87] Wolfdietrich FISCHER, *Grammatik des Klassischen Arabisch*, 2. Auflage, Verlag Otto Harrassowitz, Wiesbaden 1987.

[Haralambous91] Yannis HARALAMBOUS, "TEX and Those Other Languages", *TUGboat*, Volume 12 (1991), pp. 539–548.

[Hāšim80] هاشم محمّد الخطّات ، قواعد الخطّ العربيّ (HĀŠIM MUḤAMMAD AL-ḤAṬṬĀṬ, *Qawā'id al-Ḫaṭṭi al-'Arabī*), Maktaba an-Nahḍa, Baghdad; Dār al-Qalam, Beirut, 1400/1980.

[ISO/R233] ISO/R 233 - 1961: *International System for the Transliteration of Arabic Characters*, International Standards Institution, 1961.

[Knuth84] Donald E. KNUTH, *The TEXbook*, Volume A of *Computers & Typesetting*, Addison-Wesley, Reading, Mass., 1984.

[Knuth and MacKay87] Donald E. KNUTH and Pierre A. MACKAY, "Mixing right-to-left texts with left-to-right texts", *TUGboat*, Volume 8 (1987), pp. 14–25.

[Lagally92] Klaus LAGALLY, *ArabTEX, a System for Typesetting Arabic*, User manual. Report 6/92, Fakultät Informatik, Universität Stuttgart, 1992.

[Lamport86] Leslie LAMPORT, LaTeX, *a Document Preparation System*, Addison-Wesley, Reading, Mass., 1986.

[MacKay77] Pierre MACKAY, *The KATIB System, a revolutionary advancement in Arabic Script Typesetting by means of the Computer*, in *Scholarly Publishing* **8,2** (Toronto 1977) pp. 142–150.

[Schimmel70] Annemarie SCHIMMEL, *Islamic Calligraphy*, E.J.Brill, Leiden, Netherlands 1970.

Appendix

Installing ArabTEX

ArabTEX uses no preprocessor and thus should be compatible with any TEX implementation that allows dynamic loading of additional macro files and fonts.

The ArabTEX distribution consists of the following components:

- TEX macro files with extensions `.sty` and `.tex`: these files are installed on the TEX input path for source files.
- Font metric files (extension `.tfm`) and compressed pixel files (extension `.pk`) for the fonts `nash14` and `nash14bf` at several common magnification steps. Installation of these files is strongly system dependent; in case that they cannot be used, the METAFONT sources are also available (extension `.mf`) to rebuild the fonts locally.
- installation notes, user manual, answers to questions, demos, and the like: ASCII and/or TEX files for local printing.

The system is available from the author's institution (anonymous FTP from `ifi.informatik.uni-stuttgart.de`, directory `pub/arabtex`) and from many other common servers. At the time of this writing, version 2.02 is current. The old version 1 should no more be used.

ArabTEX is copyrighted, but free use for scientific, experimental and other strictly private, noncommercial purposes is granted.

Space and time requirements are not negligible; however, ArabTEX has been used frequently and successfully even on a PC XT standard configuration.