External Tools Not Only for ArabTEX Documents

Karel Mokrý
Department of Probability and
Mathematical Statistics
Faculty of Mathematics and Physics
Charles University in Prague
karel@ucw.cz

Otakar Smrž
Institute of Formal and
Applied Linguistics
Faculty of Mathematics and Physics
Charles University in Prague
smrz@ufal.mff.cuni.cz

Abstract

The tools presented in this paper have been designed to facilitate dealing with Arabic documents which are supposed to be processed by ArabTEX, or the ones which take advantage of the ArabTEX ASCII transliteration encoding.

We will explain the exceptional role this encoding plays among other standards, and provide a series of non-trivial conversion scripts.

Following exclusively some formal rules of Arabic morphology and orthography, one can be successful enough to eliminate certain typing errors, and prevent misuse of notation. We therefore created ArabSpell as a first-aid spelling checker for vocalized literary Arabic, and have developed it into a general and independent rule-driven spelling system.

For educational purposes, Arabic script might feature color distinction between the on-the-line string and its vocalization marks. Since no option of such kind is available in the current version of ArabTEX, we would also like to inform of the `acolor.sty` package which meets the above requirement.

1 ArabTEX Encoding Concept

As described elsewhere in detail (Lagally, 1999), the typesetting system of ArabTEX defines its own Arabic script transliteration which covers both contemporary and historical orthography in an excellent way. Moreover, the notation is human-readable, and thus suited for use wherever the original were too difficult to render.

Unlike other transliteration concepts based on one-to-one mapping of graphemes, ArabTEX evaluates the context of each ASCII character to generate the corresponding representation in the Arabic script. The input sequence `<iqra' h_a_dA an-na.s.sa bi-intibAhiN>` will produce اِقْرَأْ هٰذَا ٱلنَّصَّ بِٱنْتِبَاهٍ *iqraʾ hāḏā ʾn-naṣṣa bi-ʾntibāhin* implying one should "read this text carefully".

Observation can be made that the input is quite close to its phonetic transcription, yet extra features are introduced to enable unique derivation of the resulting equivalent in the Arabic script. Forms of the letters (initial, medial, final, isolated) and use of ligatures (combinations of graphemes), definite article assimilation, *hamza* carrier and silent *ʾalif* get determined algorithmically. Note how vowels including auxiliary ones are dealt with, and see that no distinction is made in the input.

As a token of flexibility, ArabTEX notation is interpretable with several options. The rewrite rules are subject to the language (Arabic, Persian, Urdu etc.) and the degree of vocalization (`\fullvocalize` set in our example). However, it is to stress that these only control the quality of the output, while no restriction is imposed on the input. The notation does not prescribe any syntax rules or forbidden character sequences. If we mixed the characters in our example up arbitrarily, ArabTEX would consider it equally correct.

2 ArabCode Conversion Package

The complex ArabTEX transliteration cannot be efficiently used in systems working with the original Arabic script in real-time. Therein, the more natural approach of representing individual graphemes is taken. The encoding standards in question include Unicode or UTF, Windows CP 1256, ISO 8859-6, ASMO 449, Buckwalter Transliteration or even others.

The system of ArabTEX has been equipped with alternate input reading modules, so that re-coding documents to the ArabTEX notation

proper be not a prerequisite to system's applicability. Nonetheless, the problem of conversion has stayed unsolved.

ArabCode package of conversion scripts provides the missing foot-bridge between the two transliteration concepts, and links the separate encodings with each other. ArabCode has been implemented in Perl and PHP in order to enable platform independent processing of Arabic, especially within Web applications.

2.1 ArabTEX versus UTF

Unfortunately, the ArabTEX notation interpretation algorithm has not been documented by its author. Built-in in overall code, it cannot be used as is but for typesetting in ArabTEX.

Therefore, the way ArabCode works can only simulate the expected behavior. Results of both systems tally for true Arabic input, still, identical output is not guaranteed for somehow corrupted or extraordinary data. Improvement of the method in view of extensibility, fidelity and elegance is in progress.

Unicode Transformation Format (UTF) has been chosen as the target encoding because it covers the nuances of the Arabic script most completely. Consecutive conversions may be applied. The inverse process is a simpler task, yet similar limitations hold.

Optional reduction of diacritics to pre-defined levels is supported by ArabCode, which is rather useful in the upward direction.

2.2 Unicode, Windows, Buckwalter etc.

The descriptive scope of UTF is equivalent to that of Unicode, but there is a practical aspect speaking for the transformation. Unlike Unicode, UTF does not require the lower ASCII characters to be mapped to two bytes, making documents in lower ASCII inherently compatible with it.

The encodings of Windows CP 1256, ISO 8859-6 and ASMO 449 are not as rich in graphemes as needed indeed. Surprisingly, dagger *ʾalif* or *waṣla* are absent, not to mention historical writing.

Buckwalter transliteration maps contemporary Arabic to lower ASCII without such loss of information.

ArabCode offers more or less trivial conversion scripts for all of these standards.

3 ArabSpell Rule-Driven Spelling System

ArabSpell and its spelling algorithms were originally devised for checking the entries of an arising human-edited lexical database, and that is why ArabSpell does not call for vocabulary at all. Its power rests in verification of some formal rules of Arabic morphology and in respect of the ArabTEX notation.

In contrast with previous versions, flexibility of the system and the programming techniques in use has been enhanced considerably. Extended finite-state networks have been utilized as to resolve the input data. Definition of the language in question as well as the very response of the spelling system are fully under user's control. The respective information is comprised in an external grammar, for which convenient syntax has been designed.

Here, the novel concept of ArabSpell shall be outlined in a nutshell.

3.1 RLG and NFA as system's core

We suppose that the theory of grammars and automata is generally understood. Reference to Rozenberg and Salomaa (1997), Chytil (1984) or essential university courses may be given.

Right linear grammar (RLG) denotes the regular grammar rules of which conform to either of the patterns

```
source :<>: <word of terminals>target
source :<>: <word of terminals>
```

where `source` and `target` represent one nonterminal symbol each, `:<>:` being the operator of derivation. Unescaped angle brackets delimit a word of terminal symbols, not excluding an empty word (which may be omitted before a nonterminal). We have just applied the notation of ArabSpell.

Nondeterministic finite automaton (NFA) with incidental empty-word transitions is an abstract engine to which RLG can be transformed most easily, requiring the automaton to accept exactly the same language as the grammar generates. It is to assign each nonterminal a single state, and create one more state as the final one. The initial symbol of the grammar shall become the initial state of the engine. In accordance with the rules, relevant states must be linked, the missing nonterminal mapping to the

```
# Nonterm Generative Rules ##########################################################

syllable  :< "Unruly input!" >:  [C][V][C+empty]syllable       # iterate deriving
                                 [C][V][C+empty]  [C][ending]  # stop it now

# Cluster Definition Rules ##########################################################

[C]  :<>:  <'> <b> <t> <_t> <^g> <.h> <_h> <d> <_d> <r> <z> <s> <^s> <.s> <.d>
           <.t> <.z> <‘> <.g> <f> <q> <k> <l> <m> <n> <h> <w> <y>

[V]  :<>:  <a> <i> <u> <A> <I> <U>  :<>:  <_a>  :< "Dagger 'alif occurred." >:
           <aa>  :< "Deprecated ... use <A> instead!" >:
           <iy>  :< "Deprecated ... use <I> instead!" >:
           <uw>  :< "Deprecated ... use <U> instead!" >:

[ending]  :< "Invalid ending?" >:  <uN> <iN> <aN> <aNY> <Y>  :<>:  <aNA> <UA>
                                   <aW> <aWA>  :< "Silent 'alif enforced." >:

[empty]  :<>:  <>                                              # see [C+empty] above
```

Figure 1: Example of ArabSpell grammar syntax. The generated language fits the structure of literary Arabic syllable. Simple error messages can be reported even upon acceptance of the input.

final state. Auxiliary states are generated and transitions between them marked sequentially by the terminal symbols of the word of the particular rule.

It is advisable to remove any empty-word edges, carefully of course. Then, computation with the NFA gets easier and more efficient.

Limitations on the format of RLG rules are very strict. However, we may alleviate the inconvenience of only one right-hand-side nonterminal per rule by introducing clusters of literals, which we define as sets of interchangeable words of terminals. Clusters will behave very much like nonterminals in the grammar, yet they shall not be associated with any particular states when it comes to their compilation into the automaton. Else, language equivalence would be violated.

Thanks to the idea of clusters, eminent optimization of the number of nodes and edges in the NFA network is achieved.

For the purpose of spell-checking, the system need be supplemented with an appropriate report mechanism. Not only will the engine tell which input tokens do, or do not, belong to the language of the grammar. If so, token derivation will be tracked rule by rule and measures taken for correct but suspicious input. Milestones in the history of states might give some clues to the errors otherwise.

In fact, the implementation of the reporting scheme enables ArabSpell to operate even far beyond the class of regular languages.

3.2 Grammar of Arabic syllable

Let us examine two phonetical principles which affect Arabic morphology, namely that *syllables of literary Arabic start with one and only one consonant*, and that they *do not end with more than one consonant.*

Formalization of this structure will help us reveal some cases of letter omission or insertion anywhere in the word stem, which is crucial. Further in our approximation, reserved orthographical sequences will be allowed to pass as word endings.

Figure 1 shows such a model. Only the initial nonterminal is utilized here while taking great advantage of `[clusters of literals]`. Elements of a cluster are specified by predefined sets on which operators of union `+` and difference `-` can be applied.

Blank lines or semicolons separate individual rules from each other, `#` marks comments. The right-hand-sides of the rules gather into groups unless delimited by the operator of derivation,

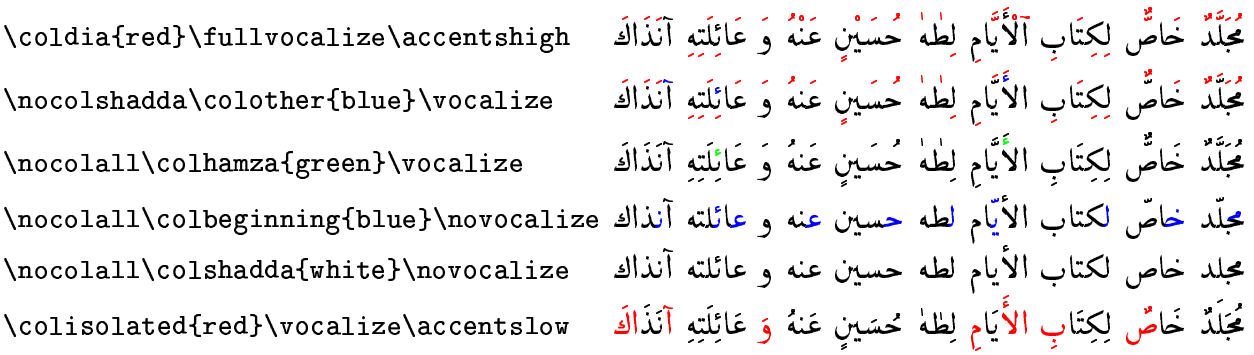

Figure 2: Example of `acolor.sty` effects combined with ArabTEX vocalization control. The input text itself has not been modified.

whose multi-functionality seems, with the reporting scheme in mind, intuitive. Still, it deserves an extra treatment since...

3.3 Spelling `:< Perl subroutines >:`

... are what provides the extension beyond regularity.

As indicated above, derivational operators do not label states and groups of rules with mere string constants, but they declare and refer to Perl subroutines which get executed upon request. Their return value, if there is any, then reports as expected.

Imagine that the number of syllables in a word is our new criterion of correctness. Writing a regular grammar would be very tedious, inflexible and inefficient. Once we attach

```
:< $myToken{"syllables"}++; return >:
```

to the nonterm generative rules of Figure 1, syllables will count silently (this subroutine `returns` an empty list of values). When completed, the evaluation code

```
:< if ($myToken{"syllables"}>5)
      { "Quite many syllables." } >:
```

for which the grammar syntax reserves a place, will alternatively confirm our concern.

Various relations within or among tokens of a document can be supervised like this. Bracket matching, word repetition or sentence length are the evergreen problems.

In Arabic, definite and indefinite article (or some other ending) could easily get into conflict as they are affixed at distinct positions in the word. The rules of the grammar will become clearer if some memory of the definite article (going first) is kept throughout the derivation, and invoked if the offending morpheme crops up. The respective code reads

```
:< $myToken{"definite"}=1; return >:

:< if (exists($myToken{"definite"}))
      { "Conflict of morphemes!" } >:
```

These have only been elementary examples. Reserved variables can be utilized to supply different language versions of the messages, or to interact with the inner code of ArabSpell. In deterministic grammars, subroutines might serve for data parsing.

3.4 More language models

ArabSpell is distributed freely with an elaborate grammar of vocalized literary Arabic in ArabTEX notation. Phenomena as formal feminine ending, definite article assimilation, presence or absence of initial hamza, consistence of prefixed particles and endings etc. have been analyzed in detail.

Several model grammars are also at hand, and do not confine themselves to Arabic. Feedback from linguists is encouraged and welcome.

4 Arabic Script in Color

Arabic does usually not indicate short vowels and other diacritics in writing, yet vocalization is necessary to ensure proper interpretation.

That is why diacritical marks are typeset in color in primers or textbooks of this language. No information is missing, while through a transparent foil of that color, vocalization turns invisible in order for the reader to train.

The `acolor.sty` package for ArabTEX and LATEX offers comfortable control over coloring. There are some pre-defined sets of symbols you may apply the color to, but types can be handled separately as well.

Activate ArabTEX and load the package with `\usepackage[`*option*`]{acolor}`, the option telling which set of commands to redefine. Afterwards, `\col`*option*`{`*color*`}` will do the color change. To cancel coloring, use `\nocol`*option*.

Consult the manual for technicalities. We believe that Figure 2 shall provide enough insight.

Acknowledgement

Arabic script displays in this paper were typeset using the ArabTEX package for TEX and LATEX by Prof. Dr. Klaus Lagally of the University of Stuttgart. Existence of this system has inspired our work principally.

References

ALECSO, editor. 1988. اَلْمُعْجَمُ اَلْعَرَبِيُّ اَلْأَسَاسِيُّ *(Essential Arabic Lexicon)*. Larousse, Tunis. In Arabic.

Michal Chytil. 1984. *Automaty a gramatiky (Automata and Grammars)*. SNTL, Prague. In Czech.

Jiří Fleissig and Charif Bahbouh. 1992. *Základy moderní spisovné arabštiny, I. díl (Essentials of Modern Standard Arabic, Book I)*. Dar Ibn Rushd, Prague. In Czech.

Jiří Fleissig and Charif Bahbouh. 1995. *Základy moderní spisovné arabštiny, II. díl (Essentials of Modern Standard Arabic, Book II)*. Dar Ibn Rushd, Prague. In Czech.

Jeffrey E. F. Friedl. 1997. *Mastering Regular Expressions*. O'Reilly & Associates, Inc., Cambridge Farnham Paris Tokyo etc.

al-Majani, editor. 1996. *Arabic-English and English-Arabic Pocket Dictionary*. Dar al--Majani, Beirut.

Klaus Lagally. 1999. ArabTEX: a System for Typesetting Arabic, User Manual Version 3.09. Technical Report 1998/09, Institut für Informatik, Universität Stuttgart.

Grzegorz Rozenberg and Arto Salomaa, editors. 1997. *Handbook of Formal Languages*, volume 1 Word, Language, Grammar. Springer--Verlag, Berlin Heidelberg New York etc.

Sriram Srinivasan. 1997. *Advanced Perl Programming*. O'Reilly & Associates, Inc., Cambridge Farnham Paris Tokyo etc.

Distribution Comment

The tools presented in this paper are freely available at `http://www.ucw.cz/~karel/` and `http://ckl.mff.cuni.cz/smrz/`.