# On the use of symbolic markup in the Production of a Multi-Lingual Dictionary [1]

Klaus Lagally [2]
Institut für Informatik, Universität Stuttgart
Breitwiesenstraße 20-22, 70565 Stuttgart, GERMANY
EMail: lagally@informatik.uni-stuttgart.de

**Abstract:** [symbolic markup, dictionary, TEX]

*We propose some simple design rules for encoding multi-lingual texts for flexibility of further automatic processing. Our main recommendation is to include the available descriptive information with the data, and to use symbolic markup conventions. We report on the successful application of these rules in the first steps of compiling a multi-lingual dictionary.*

## 1 Introduction

The recent availability of relatively inexpensive powerful computer systems opens up a host of new possibilities for many fields, among them e.g., research on Oriental languages. Due to industrious collecting activities a wealth of written material has been accumulated whose evaluation by traditional means might, given the available human resources, take decades or even centuries. Much of the necessary work is of clerical nature, and could well be automated, once the material were in machine-readable form. But the necessary software is usually not available, or not affordable, and will probably have to be developed from scratch, preferably in a cooperation between Orientalists and computer experts. Also the encoding of the data is a manual process that should have to be performed only once, and some prior consideration is advisable to avoid duplication of effort.

As an example, imagine the task of building a catalog for a large number of Arabic manuscripts. This could possibly be handled by using one of the available bilingual word processors. But the data format used will probably be private and not easily accessible, and since these tools are geared towards generating a printed version only, there is no easy way to include additional descriptive information which could otherwise be used for further evaluations.

In the sequel we present some recommendations which we believe can be helpful, and report on our first results of their application.

## 2 On data encoding

We believe there is a basic distinction between data and text, if the latter is viewed as a pattern of ink on paper, or some other physical representation. If the text can be understood at all we can derive from the pattern individual words that are connected into sentences and, hopefully, convey some meaning. This activity is commonly called *reading*, and extracts structural and semantic information from the pattern itself. When we encode the text as data to be processed and evaluated further we frequently are not only interested in the pattern itself but also in this additional information now available; the pattern itself may even be of little interest depending on the application, if some equivalent external representation can be reconstructed.

Reading and encoding the text is only a first, sometimes laborious step, and is often done at a point of time where not all further evaluation steps are known. Thus it is advisable to encode the information in a way that can be processed by, and transmitted between, various different computers and software systems. Our choice is obviously influenced by the rapidly evolving state of technology and emerging standards, but we may expect future developments not to invalidate current solutions.

At the time of this writing the main limitation is the inability of many electronic transmitting systems to reliably transfer anything but plain 7-bit ASCII data [ISO646], which on the other hand can

be processed by virtually any computer system now available. Thus this code is an obvious starting point, and fortunately nearly all more powerful encodings that have been proposed since contain it as a genuine subset, with unchanged meaning.

ASCII is primarily intended for encoding English texts, but it can equally well be used for transliterating other languages by a suitable re-interpretation (as in, e.g., [ASMO449]) and, if necessary, using more than one code byte for a character of the language in question. This can be done in a multitude of ways, and standards for switching the character mapping [ISO2022] have already existed for a long time.

Should the restriction to 7 bits disappear soon we may also use the ISO 8859-x family of extensions to 8 bits per character catering individually for the needs of various European languages, plus Arabic [ISO8859-6, ASMO708] and Hebrew [ISO8859-8]; but as these codes overlap we still have to indicate the coding used locally within multi-lingual documents, as also in the case of an ASCII transliteration.

Switching to longer code words of 16 or more bits as proposed, e.g., in [UNICODE] or [ISO10646] will not solve all of these problems, but might introduce a considerable overhead. With the exceptions of Far Eastern languages the alphabets needed are of moderate size, and the benefits of not having to indicate the encoding will probably not offset an increase in size of the data files by a factor of 2, especially since, as we shall show, we usually want to add other descriptive information anyway.

We thus advocate to stay, for quite some time from now, with a rather primitive encoding, supplemented by a sufficient amount of descriptive information.

## 3 Symbolic markup

Up to now we were only concerned with the encoding of the text proper. Devising a notation for the additional structural and semantic knowledge looks hopeless at first and seems to require clairvoyance, since the future processing needs cannot even be guessed. But indeed some progress is possible.

Once we consider the coded text as a linear sequence of code symbols, any additional knowledge about it can be described by a set of attributes assigned to the individual symbols, or to ranges of symbols. We might not yet for every attribute know how to process it further, nor even its exact meaning; but we certainly know whenever attributes are different, and this is all we need now. The main issue when encoding the data is to preserve all the information then available; exploiting it can come later.

A sufficiently powerful mechanism that does not require the a priori knowledge of a taxonomy of features, consists of some means of denoting *ranges* of code symbols, and a mechanism of associating the *name* of an attribute or a set of attributes to such a range. We need a sufficiently rich repertoire of names such that differing attributes or sets of attributes can be denoted differently. The names are arbitrary, and their interpretation needs only to be fixed much later whenever the data will be evaluated, and for different evaluations we may well use different interpretations as required. We only have to agree on the basic format of the markup to distinguish it from the text proper. This basic idea is called *symbolic markup*.

Symbolic markup is not a new idea but has been used in several contexts for some time, and we shall briefly review two of its special applications. In doing so we shall skim over many details, simplify grossly, and also deviate from the customary terminology.

### 3.1 SGML

The idea of SGML, for “Standard Generalized Markup Language” [ISO8879], originated within the printing industry with the goal of helping to separate the logical structure of a document from the details of its external printed representation, and thus ease the production process. It soon turned out that its possible scope is much wider, and one of its variants, HTML, presently has important applications in the distributed Hypertext system called the World-Wide Web [BL+94].

The basic markup mechanism in SGML works roughly as follows: a range of characters carrying an attribute A is delimited by a start tag <A> and an end tag </A>. Instead of a single attribute A may also denote an attribute class, and in this case the start tag also carries an indication of the actual member of the class, and/or additional descriptive information. The set of possible tag identifiers is fixed for any document type by some formal definition not described here, but due to the class mechanism the set of possible attributes is virtually unbounded.

[Smi92] stresses the usability of symbolic markup for capturing arbitrary information also outside of the production of documents. The main difference to our approach seems to be that for a SGML

document the complete syntax of the markup used must be put down beforehand in a Document Type Definition (DTD), whereas we propose to postpone this step until the actual processing.

### 3.2 TEX

TEX [Knu84] is a program written by D. E. Knuth to support high quality computer type-setting of text and mathematics. It is in the public domain, and compatible implementations exist for a large range of computing systems. TEX will take care of all the visual formatting including line-breaking, hyphenation, formatting of formulas, page layout etc. The output produced is completely device independent and may be viewed on a computer screen display or also directed to a large range of printing devices, provided that appropriate device driver programs are available.

TEX provides a large number of markup commands for controlling the typesetting process, and also a powerful macro extension mechanism that enables the user to introduce new markup tags and define their meanings arbitrarily, so that symbolic markup is easily possible. TEX can also be (mis)used as a portable general purpose data processor.

Due to the extensibility of TEX a number of macro packages have been developed to cater for special applications, among them:

- $\mathcal{AMS}$-TEX (see [Bee85]), supplying an additional set of mathematical symbols;
- LATEX [Lam94], providing styles for several common document classes and supporting the logical structuring into chapters and sections, building a title block, positioning figures and tables, and managing cross references, index information, a table of contents etc.;
- Ml-TEX [Fer85], some multi-lingual extensions for European languages;
- Babel [Bra91], a package supporting language-specific processing for more than 20 languages;
- ArabTEX [Lag92a, Lag92c, Lag93a, Lag94b, Lag94a], catering for right-to-left languages such as Arabic, Persian, Urdu, Pashto, Hebrew etc. with full support of diacritics and vowels, ligatures, and also the common standard transcriptions [3].
- A number of further packages, e.g., for including graphics, are described in [GMS84].

Most of these packages may be combined to make use of all the additional features provided, and further extensions may be defined freely. [Lam94] strongly advocates using symbolic markup in document design.

### 3.3 Abstract Data Bases

In some respects our approach is related to using a data base system but there are some marked differences. In a data base system, the information is stored as a collection of records consisting of a fixed number of fields; for every field the meaning and the format is determined a priori by a data base scheme. In contrast to this we advocate having an undetermined number of ranges of symbols with some attributes assigned to them, and we may introduce new attributes at any time. Also we do not require the data to be a collection of subunits with basically the same structure, even if this may frequently be the case. So we could simulate a classical data base system easily, but our approach is much more general, and could be called an “abstract data base”.

Of course, because we leave most the structure and the interpretation of the text unspecified, we cannot expect our data to be usable directly for any specific evaluation, and to process them by any given application program we will have to do some preprocessing first to get the data into the format our application expects. Fortunately, the preprocessing task will be rather well-defined, consisting mainly of omitting information presently of no interest, and reformatting the remaining data according to the needs of the application program. Whenever the format of the input data required, as well as the relevant structure of our abstract data base, can be described by a formal grammar, we can automatically generate the preprocessing program by any of several existing generator systems, e.g., Lex [Les75], YACC [Joh75], or WRG [Lag90]; and in many cases the reformatting task will be fairly trivial so we might as well write the preprocessor from scratch.

## 4 Recommendations and Guidelines

From the considerations given above we derive the following recommendations on how to devise a coding scheme suitable for capturing a structured text while also preserving the known associated information.

- Decide on the basic encoding of the text.
- Decide on the method and the format of the markup.
- Assign markup tags arbitrarily, and document their meaning. Take care to mark up portions of text with different meanings differently.
- Try to capture all the available structure information about the text. Concentrate on the logical structure and do not worry about the layout, except if it carries essential information.
- Do not omit any available information that has no apparent use. It might become important and useful later, if it is preserved now.
- Rely on the computer to perform clerical tasks efficiently when given enough information; but remember that it is not intelligent, and that you will have to do the thinking.
- Do not worry about efficiency of processing. Computers can be expected to continue getting faster.

Some of these recommendations may sound quite obvious and trivial. According to our experience they are not, or at least not always observed.

## 5 An example of an application

We have tested the viability of our approach within an ongoing project [Ser93, Ser94] of compiling a dictionary of Greek loan-words within Arabic. A central requirement is the ability to print Arabic, Greek, Syriac or Hebrew, and Latin script, and we decided to use and, if required, extend the author's ArabTEX system.

In addition to printing we wanted to automatically generate several indices sorted according to the collating sequences of the various languages used, and this proved feasible. We found that the necessary preprocessing could easily be handled by TEX itself plus some existing system routines.

### 5.1 Input encoding

As we decided to use TEX for all processing, we will use the basic TEX conventions [Knu84]. This means the coding used will be 7-bit ASCII [ISO646] both for text and for markup. In TEX a markup command is distinguished by a name consisting of Latin letters and preceded by an inverse slash, and, if required, followed by parameter strings included in curly braces; one of them might well be the range of symbols the markup command applies to. In addition to the standard TEX commands we shall define additional symbolic commands as required.

We next take the intrinsic structure of the available input data into account. Presently they reside on a large number of index cards, each of which carries the information available about a specific Arabic lemma. There are main entries describing words derived from Greek directly or via some intermediate steps, and secondary entries that describe writing variants and refer to some main entry.

We represent these data as a possibly unordered sequence of text blocks in free format. Every text block starts with a markup command of the form `\alemma {the lemma}` followed by the descriptive information and terminated by an empty line (for ease of editing only). The descriptive information may contain components in several languages that are marked up by `\ar {Arabic text}`, `\gr {Greek text}`, `\sy {Syriac text}`, `\he {Hebrew text}` as required; other languages, e.g., Coptic could be added. This also applies to some of the European languages occurring as their encoding conventions are slightly different. In addition there are a few more symbolic tags like `\see` for pointers to other entries, `\var` for denoting variants, `\cod` for referring to sources, and a few more. Note that we distinguish between `\alemma` and `\ar` as their rôles are different, and for the same reason we denote, e.g., a Greek lemma (`\glemma`) and an explanation in Greek (`\gr`) differently.

Greek text is mapped to 7-bit ASCII using the encoding proposed by Silvio Levy [Lev88] and supported by GREEKTEX, another extension to TEX freely available [Dry94]. For Arabic, Syriac, and Hebrew we use the standard encoding implemented in the ArabTEX system; it is a linearised variant of the ZDMG transliteration [DIN31635, ISO/R233] that, however, uses no diacritical marks and can easily be handled using a standard computer keyboard.

The following example is typical; we made liberal use of white space to keep the input data, which might have to be edited, human readable:

```
\alemma {qAbUs}
JA 1886 (1) 460.
\see \ar {qwA_tUs} (ib.)

\alemma {qAbIl}
\glemma {k'aphloc} \from \slemma {qpIl'}
ZDMG 1897 (51) 470.
\de {der Kleinh"andler, Speisewirth:}
\ar {mi_tl insAn _dAhib fI al-sUq 'inda al-qAbIl
ya^sum al-^siwA'| wa-al.tabI_h}
\de {"'Wie ein Mensch welcher auf dem Markte
bei [dem] Speisewirth vorbeigeht und den Duft
der gekochten und gebratenen Speisen riecht."'}

\alemma {qAtismA}
\glemma {k'ajisma} pl. \ar {qAtismAt}
GRAF VERZ. 86
\de {"'Kathisma in der Psalmeneinleitung"'.}
\var \ar {qA.tsmA} (pl. \ar {qA.ssmAt}), \ar {kAtsmA}.

\alemma {qAtsmAt}
GRAF VERZ. 86
\see \ar {qAtsmA} (ib.)
```

### 5.2 Printing the text

If we want to print a listing of the data in dictionary format we have to write a small driver program in the TEX macro language that will determine the general output format, and that will assign to all yet undefined tags, as their meaning, the required external representation by calling some TEX or ArabTEX routines. Then it will read the input data file and let TEX process it to do the formatting. As presently no Syriac font is available we substituted Hebrew temporarily.

The resulting output for a sample page is given in the appendix. The correspondence with the encoding example should be obvious.

### 5.3 Sorting

Up to now we have assumed that our input data are sorted according to the Arabic lemma, obeying the standard Arabic collating sequence (on the basic glyphs only, without respect to vowel indicators and other diacritical marks). In the long run this will not remain true as a consequence of editing and introduction of new entries, and we shall have to re-sort the data from time to time. We cannot expect that a sorting routine obeying the Arabic collating sequence, and understanding our input notation is available, and therefore we exploit the fact that any Operating System known to us provides a sorting routine that can at least sort the lines of a text file according to the standard ASCII collating sequence; and we transform our input file into another one that, when sorted mechanically, will contain the entries in the required order.

It was not too difficult, even though nontrivial, to write another TEX macro program that interprets the same data in a different way: instead of producing formatted output, it will read the data one complete entry at a time, filter out the Arabic lemma, and compute an alphanumeric sorting key from its internal Arabic glyph representation that is available within ArabTEX. Now we copy the entry to

an output file and prepend to every line a new tag of the form `\key {the key}`; and this new file can now be processed by the standard sorting routine. The additional tag will not interfere with the printing process if we define its meaning as to produce no output at all. Thus we can use our sorted file as a new version of our input data, and whenever sufficiently many new entries have been added, we can reprocess the file, compute sorting keys for the new entries, keep the already existing ones, and re-sort again.

The sorting keys are comparatively long and thus add some space overhead even if they do no harm otherwise. But they are really required only during the sorting process itself and can be recreated whenever needed. Getting rid of them again is easy: process the data again after redefining the meaning of the `\key` command so that it will copy the rest of the entry (without the key) to an output file.

### 5.4 Compiling indices

For compiling an index, e.g., on the Greek terms, from the same data set some more processing is required, but this task is simpler. We again process the data one entry at a time but now only keep those entries that contain a Greek lemma (these are the main entries), and build a new output file containing for each main entry just the following items: a sorting key (again hidden within the argument of a tag, but this time computed from the Greek lemma so as to respect the Greek collating sequence), the Greek lemma itself, and the Arabic lemma. This file can again be sorted by the standard sorting program and be printed by an obvious variant of the printing program described above.

For indices on other languages we proceed analogously, and we could also easily build a retrograde index by processing the internal representation of the Arabic lemma in reverse order. The technical details are given in [Lag95].

### 5.5 Re-encoding texts

We can imagine applications where data derived from various sources have to be combined within a processing task, and where different encodings have been used from the outset. According to our general markup philosophy these parts have to be made sufficiently self-describing by adding suitable tags, and thus several encodings can coexist. This does not happen in the application at hand, but nonetheless we performed a few experiments in order to find out whether transforming e.g. the Arabic lemmata to a different encoding could be done easily, while leaving the rest of the data unchanged. In fact nothing more was needed than reading the entries one at a time, get at the internal representation, compute the new external representation, and output the entry with the newly encoded lemma (and a suitable markup tag) in place of the old version.

Of course this mechanism is not limited to our special application. It can as well be used for converting any file between our private encoding, ASMO449 (ISO9036) [ISO9036, ASMO449], and ASMO708 (ISO8859-6) [ISO8859-6, ASMO708] encodings back and forth; and extending it to other encodings as, e.g., the one used inside Arabic Windows into this scheme is easy. See also [Lag95].

### 5.6 Further processing

Among the lines given we could rather easily open up the way for various other evaluations of the same data. We could, e.g., search for lemmata in several languages, build concordances, collate versions of the same basic text for identifying variants, or derive a differently formatted file that is suitable for loading it into a sufficiently powerful data base system.

None of this has yet been done, but we also see no basic difficulties apart from the work to be expended in writing the necessary programs. We found TEX, as it is geared towards text processing from the outset, especially suitable for comparable tasks, but we can also not deny the fact that using the TEX macro language for programming purposes requires special skills not widely available, and other programming techniques could be used as well.

### 5.7 Discussion

The mechanism we presented proved usable but has some apparent drawbacks. One of them is that before starting any new kind of processing task we have to do some non-trivial programming; this, as we believe, is inherent. Using TEX macros for programming was locally convenient in our case

as we already had ample experience, but is not mandatory; other programming techniques could have been used as well. The fact that the parts given in Oriental languages are presently coded in a transliteration, eases the task of manual editing when using a very simple plain text editor, but is otherwise not essential. The encodings for the various languages are logically independent of each other, and could easily be changed, even automatically, if some multi-language editor were available. We might even use different encodings in different parts of our data at the same time, as long as we keep them distinguishable by different markup tags.

## 6 Conclusion

Our experience has shown that encoding quite heterogeneous data in a way that preserves the available meta-information, enabled us to perform a variety of related but quite diverse automated processing tasks on the same abstract data base, without having to perform any manual re-encoding. The programming effort required and also the processing load invested were not trivial, but we believe that the costs incurred were reasonable given the fact that some of the tasks had, to our knowledge, never been attempted successfully before.

We generally believe in the benefits of cooperation, also between fields as diverse as Orientalistics and Computer Science; and we expect the cost of computing power to continue to decrease rapidly. Our main concern has been to reduce, as far as possible, the amount of labour that cannot be delegated to a machine, in order to liberate humans from purely mechanical chores and to enable them to concentrate on tasks where they can better exploit their specific abilities.

## References


[ASMO449] Arab Standards and Metrology Organization.
*7-bit coded Arabic character set for information interchange.*
ASMO 449 Tech. Rep., Amman, Jordan, 1982.

[ASMO708] Arab Standards and Metrology Organization.
*8-bit Coded Arabic/Latin Character Set for Information Interchange.*
ASMO DS 708 Tech. Rep., Amman, Jordan, 1985.

[Bee85] Barbara Beeton.
Mathematical Symbols and Cyrillic Fonts Ready for Distribution.
*TUGboat,* 6(2):59–66, 1985.

[BL+94] Tim Berners-Lee et al.
The World-Wide Web.
*Communications of the ACM,* 37(8):76–82, 1994.

[Bra91] Johannes Braams.
Babel, a Multilingual Style Option for Use with LaTeX's Standard Document Styles.
*TUGboat,* 12(2):291–301, 1991.

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

[Dry94] K. J. Dryllerakis.
GreekTeX.
Available electronically via the InterNet from the Comprehensive TeX Archive Network (CTAN), 1994.
A set of Greek fonts, associated macros and documentation; based on fonts devised by Silvio Levy and Yannis Haralambous.

[Fer85] M.J. Ferguson.
A Multilingual TeX.
*TUGboat,* 6(2):57–58, 1985.

[GMS84] Michael Goossens, Frank Mittelbach, and Alexander Samarin.
*The LaTeX Companion*.
Addison-Wesley, Reading, Mass., etc., 1984.

[ISO10646] International Organization for Standardization.
*Universal Coded Character Set*.
Technical report ISO DIS 10646 (draft international standard), ISO Geneva, 1992.

[ISO2022] International Organization for Standardization.
*Information processing – ISO 7-bit and 8-bit coded character sets – Code extension techniques.*
ISO 2022.

[ISO646] International Organization for Standardization.
*Information processing – ISO 7-bit coded character set for information interchange.*
ISO 646.

[ISO8859-6] International Organization for Standardization.
*Information processing – 8-bit single-byte coded graphics character sets – Part 6: Latin/Arabic alphabet*.
ISO 8859-6, 1987.

[ISO8859-8] International Organization for Standardization.
*Information Processing – 8-bit single-byte coded graphics character sets – Part 8: Latin/Hebrew alphabet*.
ISO 8859-8, 1987.

[ISO8879] International Organization for Standardization.
*Information Processing – Text and Office Systems – Standard Generalized Markup Language (SGML)*.
Technical Report ISO 8879, ISO Geneva, 15 October 1986; Amendment 1, 1 July 1988.

[ISO9036] International Organization for Standardization.
*Information processing – Arabic 7-bit coded character set for information interchange.*
ISO/TC 97 - ISO 9036, 1987.

[ISO/R233] International Organization for Standardization.
*International System for the Transliteration of Arabic Characters*.
ISO/R 233 - 1961.

[Joh75] S.C. Johnson.
Yacc — Yet Another Compiler Compiler.
Computing Science Technical Report 32, AT&T Bell Laboratories, Murray Hill, N.J., 1975.

[Knu84] Donald E. Knuth.
*The TEXbook*, volume A of “Computers & Typesetting”.
Addison-Wesley, Reading, Mass., 1984.

[Lag90] Klaus Lagally.
WRG — ein neuer Generator für Top-Down-Parser mit automatischer Fehlerbehandlung.
Report 1990/01, Universität Stuttgart, Fakultät Informatik, 1990.

[Lag92a] Klaus Lagally.
ArabTEX, a System for Typesetting Arabic.
In *ICEMCO92, Proc. 3rd International Conference and Exhibition on Multi-lingual Computing (Arabic and Roman Script)*, pages 9.4.1–9.4.8, University of Durham, UK, December 10–12, 1992.
See also [Lag92b].

[Lag92b] Klaus Lagally.
ArabTEX, a System for Typesetting Arabic.
Report 1992/11, Universität Stuttgart, Fakultät Informatik, 1992.

[Lag92c] Klaus Lagally.
ArabTEX — Typesetting Arabic with Vowels and Ligatures.
In *EuroTEX ’92, Proc. 7th European TEX Conference*, pages 153–172, Prague, Czechoslovakia, September 14–18, 1992.
See also [Lag92d].

[Lag92d] Klaus Lagally.
ArabTEX — Typesetting Arabic with Vowels and Ligatures.
Report 1992/07, Universität Stuttgart, Fakultät Informatik, 1992.

[Lag93a] Klaus Lagally.
ArabTEX, a System for Typesetting Arabic. User Manual Version 3.00.
Report 1993/11, Universität Stuttgart, Fakultät Informatik, 1993.

[Lag93b] Klaus Lagally.
Some Problems in Arabizing LaTeX.
Report 1993/15, Universität Stuttgart, Fakultät Informatik, 1993.

[Lag94a] Klaus Lagally.
How to extend ArabTEX to handle Hebrew, 1994.
Unpublished internal Notes.

[Lag94b] Klaus Lagally.
Some Problems in Arabizing LaTeX.
In *ICEMCO94, Proc. 4th International Conference and Exhibition on Multi-lingual Computing (Arabic and Roman Script)*, pages 9.10.1–9.10.8, London, April 7–9, 1994.
See also [Lag93b].

[Lag94c] Klaus Lagally.
Using TEX as a Tool in the Production of a Multi-Lingual Dictionary.
Report 1994/15, Universität Stuttgart, Fakultät Informatik, 1994.

[Lag95] Klaus Lagally.
A Non-standard Application of ArabTEX: Generating Sorted Indices.
Report 1995/02, Universität Stuttgart, Fakultät Informatik, 1995.

[Lam94] Leslie Lamport.
*LaTeX, a Document Preparation System. User's Guide and Reference Manual*.
Addison-Wesley, Reading, Mass., second edition, 1994.

[Les75] M.E. Lesk.
Lex — a Lexical Analyser Generator.
Computing Science Technical Report 39, AT&T Bell Laboratories, Murray Hill, N.J., 1975.

[Lev88] Silvio Levy.
Using Greek Fonts with TEX.
*TUGboat*, 9(1):20–24, 1988.

[Ser93] Nikolaj Serikoff.
A Dictionary of Greek Borrowings and Loan Words in Arabic [Tasks, Methods, Preliminary Results].
*Graeco-Arabica*, 5:267–273, 1993.

[Ser94] Nikolaj Serikoff.
A Project to Build a Multi-Lingual Dictionary of Greek Loan-Words within the Arabic Language, 1994.
Personal communication.

[Smi92] Joan M. Smith.
*SGML and Related Standards*.
Ellis Horwood Ltd., New York, 1992.

[UNICODE] The Unicode Consortium.
*The Unicode Standard. Worldwide Character Encoding. Version 1.0, Volume 1*.
Addison-Wesley, Reading, Mass., 1991.

---

[1] A preliminary version of this paper [Lag94c] has been presented to the XVIIth Congress of the Union Européenne des Arabisants et Islamisants, St. Petersburg, GUS, 20-26 August, 1994, and is to appear in *Manuscripta Orientalia*. The present version is published here for the first time.

[2] Klaus Lagally, born in Munich (GERMANY) in 1937. University studies in Mathematics and Physics, Ph.D. in Theoretical Physics 1967. Work on Operating Systems and Programming Languages. Professor of Computer Science 1976, Universität Stuttgart, Germany. Current research interests: Arabic text processing and typesetting, multi-lingual computing and communication.

[3] The ArabTEX package is available without charge for scientific and private applications. It can be downloaded from `ftp.informatik.uni-stuttgart.de` in the directory `/pub/arabtex/`. For other ways of acquiring it, or for commercial use, please contact the author.