Tips and Tricks of the Prague Arabic Dependency Treebank

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

In this paper, we report on several software implementations that we have developed within Prague Arabic Dependency Treebank or some other projects concerned with Arabic Natural Language Processing. We try to guide the reader through some essential tasks and note the solutions that we have designed and used. We as well point to third-party computational systems that the research community might exploit in the future work in this field.

Arabic, dependency grammar, treebank, language annotation and processing, application programming.

1. INTRODUCTION

The interests of applied computational linguistics are increasingly turning toward languages commonly denoted as ‘lesser-studied’. Out of these, the Arabic language has been receiving more and more attention, and has already been in the center of many significant research projects. Yet, processing Arabic and dealing with its linguistic data resources does not usually belong to the ready-to-use skills of computational linguists.

Recently, there have been conference tutorials that map the available resources and overview the general problems to solve for this language. Some of these events tend to be descriptive rather than constructive—they do not set up a task, discuss some method for its fulfillment, and show the viability of the results, nor the general applicability of the approach.

On the contrary, the present contribution would like to offer more insight into the solutions to selected non-trivial issues in computational processing of Arabic, ranging from linguistic morphological analysis to dependency parsing, from customization of annotation environments to automatic taggers and parsers, from design of lexicons to management of treebanks. The extent of this exposure is very limited, nonetheless, this paper is intended to be a guide to the reader, not a textbook.

We will deliver some of our experience with building the Prague Arabic Dependency Treebank (Hajič et al. 2004, [1], [2]) and making use of it for various computational applications (Hajič et al. 2005). PADT now consists of the morphological and the analytical levels of linguistic annotations, and the third level, that of the underlying syntax and information structure, is being established. There is an original suite of software tools for visualizing and editing, as well as automated processing and maintenance of the treebank’s data, which we would demonstrate in action.

We would also like to promote related technologies that are being developed by other research teams. We will refer to the work of (Lagally 2004, 1994) in data meta-encoding and compilation of lexical resources, of (Forsberg and Ranta 2004, El Dada and Ranta 2006) in functional modeling of morphology and syntax, or of (Smith et al. 2005, Habash and Rambow 2005) in disambiguation of Arabic and its further linguistic treatment.

2. PROBLEMS AND SOLUTIONS

The problems around representing the Arabic script on different operating systems and in individual applications no longer seem to be an issue. The Unicode standard is nowadays widely supported, and data are mostly exchanged in UTF-8 or UTF-16 encodings. Even though displaying the right-to-left cursive script on graphical interfaces involves its set of low-level problems, we will not be concerned with these. Instead, we will pay attention to the processing of the *contents* of textual documents, as well as of other resources of written or transcribed linguistic data.

```
$regexR = qr/(?: \p{Arabic} |
                 [\x{064B}-\x{0652}\x{0670}\x{0657}\x{0656}\x{0640}] |
                 \p{InArabicPresentationFormsA} |
                 \p{InArabicPresentationFormsB} )+/x;

          # using \p{InArabic} is too general, includes punctuation


$regexL = qr/\p{Latin}+/;


$regexN = qr/[0-9]+ (?: [\.\,\x{060C}\x{066B}\x{066C}] [0-9]+ )? |
             [\x{0660}-\x{0669}]+
                    (?: [\.\,\x{060C}\x{066B}\x{066C}]
                                       [\x{0660}-\x{0669}]+ )?/x;


$regexP = qr/[.,;:!?`"'\(\)\[\]\{\}\<\>\\\|\/\~\@\#\$\%\^\&\*\_\=\+\-] |
             [\x{00AB}\x{00BB}\x{060C}\x{061B}\x{061F}]/x;
```

FIGURE 1: Regular expressions in Perl for identification of Arabic orthographical words `$regexR`, words in the Latin alphabet `$regexL`, numbers using alternative digits and decimal points `$regexN`, and various punctuation symbols `$regexP`. Note the comment below the definition of `$regexR`.

2.1 Text Processing

Let us assume that textual data, regardless of the application or editor in which they were created, are internally accessible to a programmer as strings of characters, or are converted into formats that are transparent and allow external processing, such as data in various markup languages or plain text files the encoding of which is compliant with the Universal Character Set of Unicode.

Most commonly, the data will directly reflect the original Arabic orthography. In that case, several text-processing operations on the data are of interest, such as:

a) identification of orthographical words in contrast to punctuation symbols and numbers, or words in non-Arabic alphabets,
b) normalization of the textual data, e.g. removal of diacritics (explicit vowelization marks), removal of padding characters (like ‘tatweel’, a stretchable connecting line) and substitution of ligature characters with equivalent sequences of graphemic characters,
c) conversion of the orthography into transliteration or, if possible, into phonetic transcription.

The first of the tasks can be quite easily solved thanks to the classification of the Unicode characters into subsets. In Figure 1, we show definitions of the regular expressions in Perl that identify the particular kind of substrings in the data. Analogous implementations can be expressed in other programming languages, too.

The normalization of data essentially reduces to substring substitutions as well, and so does the problem of conversion into transliterations, esp. if only some one-to-one mapping of characters is required. Yet, efficiency of processing can become an issue (consider repeated passes, one per replace call, through data of huge size), and some unified approach to transforming the text might come handy.

In our programming library for Perl called Encode::Arabic (Section 3.3), we have implemented a mechanism that can be used to perform the normalization, without the programmer’s need to know any details about the characters that are actually concerned. As shown in Figure 2, one can exploit the mode-dependent conversion between the orthography and the Buckwalter transliteration (Buckwalter 2002).

In certain contexts, representing the Arabic language in a notation different from the original orthography, yet a notation that allows to be translated into the orthography, can bring advantages both for human and computer processing, and can offer extended options for reusing the data for multiple purposes. In particular, we note the ArabTeX meta-transliteration of the language (Lagally 2004, 1994).

```
use Encode::Arabic ‘:modes’;

enmode “buckwalter”, “default”, “XML-style”;

demode “buckwalter”, “nosukuun”, “XML-style”, “notatweel”;

$script_new = decode “buckwalter”, encode “buckwalter”, $script_old;

# Suppose $script_old contains the text in Arabic characters that translates
# into the Buckwalter transliteration as
#
# encode “buckwalter”, $script_old  --->
#                                “AiqoraOo h`*aA {l_n~a_S~a bi___{notibaAhK.”
#
# Then $script_new contains the version of the original text in Arabic again
# from which the wasla, sukuun, and tatweel characters are removed.
#
# encode “buckwalter”, $script_new  --->
#                                “AiqraO h`*aA Aln~aS~a biAntibaAhK.”
```

FIGURE 2: Example of using Encode::Arabic (Smrž 2003–2006) in connection with various modes that enable the user to carry out certain kinds of normalization of the text without concern for particular implementation details.

2.2 Morphological Analysis

Prague Arabic Dependency Treebank (Hajič et al. 2004, Smrž et al. 2006) is a project of analyzing large amounts of linguistic data in Modern Written Arabic in terms of the formal representation of language that originates in the Functional Generative Description (Sgall et al. 1986, Hajičová and Sgall 2003).

The formal representation delivers the linguistic meaning of what is expressed by the surface realization, i.e. the natural language. The description is also designed to enable synthesizing the natural language out of the formal representations. By constructing the treebank, we provide a resource for computational learning of the correspondences between both languages, the natural and the formal.

The linguistic analysis takes place in three stages: the morphological level (inflection of lexemes), the analytical level (surface syntax), and the tectogrammatical level (underlying syntax). Within the scope of this paper, we will take a closer look at our approach to morphology and the analytical syntax.

The first step in our formal analysis of written (or even, transcribed spoken) language is the recovery of the grammatical categories that the word forms carry in the context, and of the subsuming lexemes of these forms.

Thus, from a non-vocalized Arabic text, we obtain the abstract information that is relevant for further processing of the discourse, and for syntactic analysis in particular. Moreover, morphological analysis can be reversed into generation in most computational morphological models. Due to that, we can produce the phonologically qualified, fully vocalized version of the text as another result.

Morphologically annotated data are used as training examples for taggers, which are systems that can do automatic morphological analysis and its context-aware disambiguation. There is a number of taggers already developed for Arabic on the basis of treebanks (Habash and Rambow 2005, Smith et al. 2005, Hajič et al. 2005).

Morphological analysis in PADT is pioneering the MorphoTrees technique (Smrž and Pajas 2004, Smrž in prep.). For every word form found in a sentence, MorphoTrees organize the list of its possible morphological readings into a hierarchy, and allow the annotator to systematize and speed up the choice of the analysis that is appropriate in the context. Restricting the nodes and their subtrees in a cascading style according to various criteria, esp. limiting them to the values of grammatical categories that must be satisfied, is a very efficient way to cope with otherwise enormous morphological ambiguity in Arabic.

Figure 3 illustrates the hierarchy further. The analyzed orthographic word constitutes the root of the hierarchy, the full forms and morphological tags of the analyzing syntactic tokens project into its leaves. Lexemes occupy the first level above the leaves, then there is the level of canonical non-vocalized spelling of the tokens, and the level of partitioning of the original word into such token forms.

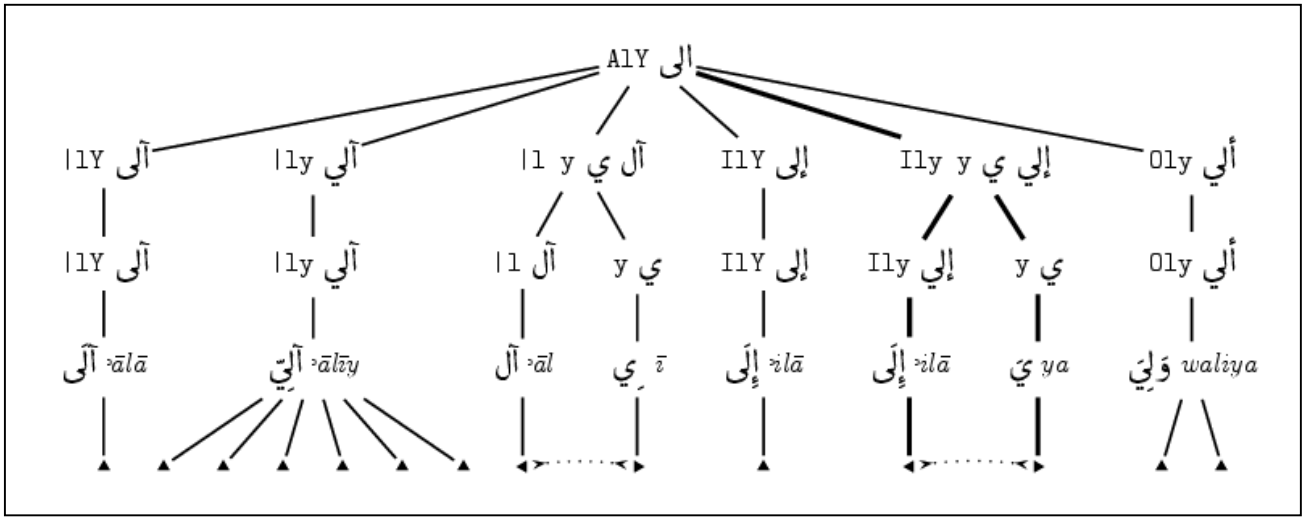

FIGURE 3: MorphoTrees analyses of the orthographic word `AlY` and its spelling variants. The morphological tags in the leaves are schematized to triangles. The bold lines in the hierarchy indicate the annotation in the context, i.e. the choice of the solution `Ily y` *'ilayya* 'to me'.

The underlying morphological analyzer that has been used in MorphoTrees and the Prague Arabic Dependency Treebank annotations so far, is the Buckwalter Arabic Morphological Analyzer (Buckwalter 2002, 2004). The output of Buckwalter morphology has to be transformed into what we call *functional approximation* of the morphology, and what we describe in (Smrž and Pajas 2004, Hajič et al. 2005). However, a novel computational morphological model of Functional Arabic Morphology is being designed and implemented (Smrž in prep.). The MorphoTrees technique is included as a feature of this new system. Nonetheless, MorphoTrees can be modified to fit other morphological formalisms as well.

The software tools that take a text file with some minimal paragraph-structure markup and produce a file with MorphoTrees analyses in the format for the TrEd annotation environment (Section 3.1), are available upon request from the authors. Some of these tools are also already present in the distribution of the Prague Arabic Dependency Treebank 1.0 (Hajič et al. 2004), and are open-source.

2.3 Syntactic Parsing

The tokens, equipped with their disambiguated grammatical and lexical information, enter the annotation of analytical syntax (Žabokrtský and Smrž 2003, Smrž et al. 2006). This level is formalized into dependency trees the nodes of which are the tokens. Relations between nodes are classified with analytical syntactic functions. More precisely, it is the whole subtree of a dependent node that fulfills the particular syntactic function with respect to the governing node.

Both clauses and nominal expressions can assume the same analytical functions—the attributive clause in our example in Figure 4 is Atr, just like in the case of the nominal attributes therein.

The coordination relation is different from the dependency relation. However, we can depict it in the tree-like manner, too. The coordinative node becomes Coord, and the subtrees that are the members of the coordination are marked as such (cf. dashed edges in the example). Dependents modifying the coordination as a whole would attach directly to the Coord node, yet would not be marked as coordinants—therefrom, the need for distinguishing coordination and pure dependency in the trees.

The immediate-dominance relation that we capture in the annotation is independent of the linear ordering of words in an utterance, i.e. the linear-precedence relation. Thus, the expressiveness of the dependency grammar is stronger than that of phrase-structure context-free grammar. The dependency trees can become non-projective by featuring crossing dependencies, which reflects the possibility for dependency descriptions to relax word order while preserving the links of grammatical government.

For more detailed discussion of formal properties of dependency grammars, as well as for modular computational treatment of these systems, cf. esp. (Debusmann 2006).

Dependency parsing has been attracting a lot of attention in the NLP research. The most recent references relevant to Arabic include (Corston-Olivier et al. 2006), (Chiang el al. 2006), as well as papers in (eds. Màrquez and Klein 2006).

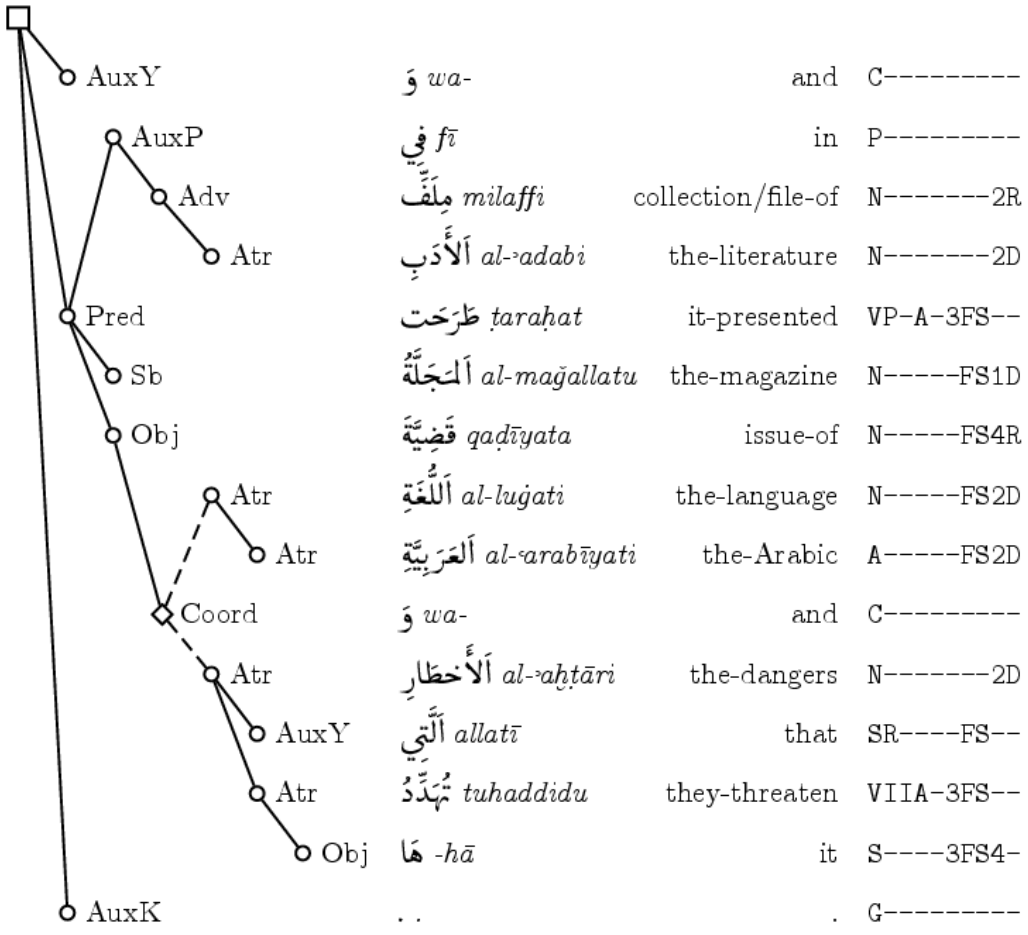

FIGURE 4: Analytical annotation of the sentence ‘In the section on literature, the magazine presented the issue of the Arabic language and the dangers that threaten it.’. The analytical function Pred denotes the main predicate, Sb is subject, Obj is object, Adv stands for adverbial. AuxP, AuxY and AuxK are auxiliary functions of specific kinds. Grammatical categories are encoded using the positional notation explained in (Hajič et al. 2005).

2.4 Lexicon Design

One of the essential components in a language processing system is the lexicon. Unless other resources are available from which the complete information can be extracted, building a large-scale lexicon is time consuming and requires a lot of expertise. Therefore, the reusability of the lexicon is of extreme interest.

The most respected and reused computational lexicon of Arabic is that developed by (Buckwalter 2002, 2004) as part of the morphological analyzer. The lexicon consists of a list of fully vocalized morphological stems classified for the purposes of inflectional analysis and accompanied with Arabic lemmas and English glosses. Information on morphological roots is there as well, at least to some extent.

Buckwalter lexicon has been utilized by others in the MAGEAD system for modeling of the morphology of Arabic dialects (Habash and Rambow 2006). The version of the lexicon (Buckwalter 2002) that is published under the GNU General Public License is also the main lexical source for Functional Arabic Morphology (Smrž in prep.).

However, the original lexicon is considerably restructured in our implementation, besides being updated with new kinds of linguistic information. The design principles that we follow while creating this resource include:

a) use of a representation of the linguistic data that is not just the literal Arabic orthography, but a more abstract and extensible notation that encodes both orthography and phonology, and whose interpretation is customizable

b) organization of the lexicon so that there is preferably no duplication of information and so that the lexicon can possibly be divided into separate units, as well as be interlinked with external modules providing e.g. other lexicons
c) definition of such format of the lexicon so that editing and understanding the data is not inappropriately difficult, and using such data markup whose syntax is either lightweight, or can be edited/verified with some automatic tools, or both

Similar principles were advocated for in e.g. (Lagally 1994), and should comply with the modern recommendations found in general in software engineering. Let us mention our concrete choices and the advantages that these principles bring us in the particular case of redesigning the Arabic morphological lexicon.

ad a) We use the ArabTeX notation for encoding Arabic (Lagally 2004, Smrž in prep.), into which we can transform the fully vocalized stems of the Buckwalter lexicon quite easily. The major point is that modeling the morphology of Arabic is much *simpler* in a notation that is close to phonology and abstracts away from the orthography. We need not care in the morphological model what carrier for 'hamza' there has to be in any word form, or whether consonantal doubling is taking place after a morphological operation, and thus rewriting of the second consonant to 'shadda' is needed, etc. Another important achievement is that the *identical* morphological model can be instantiated both for the orthography and for some given phonological transcription that might be rather available—that depends only on how we interpret the ArabTeX notation at the point where we automatically compile the morphological generators/analyzers out of our morphological model!

ad b) We organize our data into records whose structure allows inheritance and sharing of information. The records build up a set of modules understood by the Haskell programming language. The modules can not only be *compiled* as part of the complex morphological model, but also, they can be loaded into an *interactive* interpreter of Haskell (i.e., Hugs or GHCi, [8]) or parsed/loaded by some other program. By defining a library of utility functions, the lexical data can be *queried*, sorted, counted, or *exported* to various formats (XML, LaTeX, etc.) for further external processing.

ad c) The format of the lexicon must conform to the requirements of Haskell, once we use its module system. However, this does not present a limitation. On the contrary, there are two aspects of Haskell that we can further exploit. Haskell allows us to define the so called *embedded domain-specific language* (Hudak 1998) for encoding the structure of the data in the lexicon (we can define our own constructor functions and combining operators, i.e. delimiters of the lexicon's items). It also requires that the types of the individual pieces of information be consistent—by this *type-checking* of the data of the lexicon, the validity of the records in the lexicon is guaranteed, and errors of many kinds, syntactic as well as semantic, are thus effectively eliminated.

2.5 Treebank Management

The treebank annotations must also be handled with proper care for their soundness and completeness. Let us make a few remarks about the life cycle of the data in the treebank.

The original textual data come from the raw-text corpora published by the Linguistic Data Consortium, mostly included in the Arabic Gigaword collection (Graff et al. 2006). From every document selected for annotation, the MorphoTrees file for TrEd is generated. When its annotation is completed, the analytical file is generated. The procedure goes up to tectogrammatical annotation, of course.

As each level of annotation depends on the data of the preceding level, it is important to implement tools for automated synchronization of the data and for migration of annotations to files of modified content or format. We have developed such tools, and incorporated them also into TrEd, the graphical annotation environment.

All documents that constitute the treebank are registered in a version control system (we use SVN, [9]), so that changes and differences can easily be tracked down, as the annotations evolve. The differencing tools for text files are, however, not most suitable for using with the ordered tree structures that we have. We are therefore going to improve some of the existing annotation modes in TrEd to make comparison of documents even more transparent and comfortable.

We also have tools that check the consistency of annotation on every level, characterize the document as to the number of words annotated, or show the missing annotations, the comments, etc.

3. COMPUTATIONAL SYSTEMS

3.1 TrEd – Annotation Environment

The indispensable annotation environment for PADT and various other treebanking projects is the TrEd tree editor authored by Petr Pajas. TrEd is not only a fully programmable and customizable graphical user interface based on Perl/Tk, but also an excellent suite of utilities for automated, optionally parallel, processing of the data.

One can reuse as well as write his/her own TrEd macros (i.e. subroutines in Perl) that implement consistency checks, do miscellaneous batch processing, perform search, evaluate annotation differences. It is also possible to design one's own special-purpose annotation mode by defining new macros and associating them through keyboard shortcuts with the graphical editor, to re-style the appearance of the trees or graphs depending on the type of data, to interface the editor with external programs, etc., etc. TrEd is documented and available online ([3]), being published under GNU General Public License.

3.2 Netgraph – Search Engine

Netgraph is a client–server application for efficient searching in treebanks developed by Jiří Mírovský and Roman Ondruška. It provides the user with an easy-to-learn graphical query language that does not presume any programming skills. The client application is implemented in Java, and is available on ([4]).

3.3 Encode Arabic – Data Conversion

The Encode::Arabic module for Perl ([6]) supports miscellaneous modes of processing the non-trivial, yet ingenious ArabTeX encoding notation of the Arabic script and/or its phonetic transcriptions (Lagally 2004). Encode::Arabic covers the Buckwalter transliteration as well (Buckwalter 2002). Apart from the programming module, there is also a web interface ([5]) useful for converting short cut-and-pasted text.

Encode Arabic is newly implemented also in Haskell. The programming library as well as some compiled executables will be published along with (Smrž in prep.).

3.4 Other Research Systems

Let us finally draw attention to several other interesting software systems reusable in processing Arabic. Typesetting Arabic (as well as Farsi, Urdu, etc.) with ArabTeX (Lagally 2004, [7]) may be the preferred option when presenting complex data (cf. e.g. Figures 3 and 4 produced with this system). Higher-level processing of the language is addressed in (El Dada and Ranta 2006, Forsberg and Ranta 2004) and (Debusmann 2006), who develop computational linguistic models in declarative and abstract settings.

4. CONCLUSION

We have presented a mixture of tips and tricks concerning selected non-trivial problems in computational processing of Arabic. We described novelties in morphological modeling, addressed dependency parsing, promoted modern technologies and referred to several software systems important for further research.

ACKNOWLEDGEMENTS

This research has been supported by the Ministry of Education of the Czech Republic, project MSM0021620838, by the Grant Agency of Charles University in Prague, project UK 373/2005, and by the Grant Agency of the Czech Academy of Sciences, project 1ET101120413.

URL LINKS

[1] http://ufal.mff.cuni.cz/padt/online/
[2] http://ufal.mff.cuni.cz/padt/
[3] http://ufal.mff.cuni.cz/~pajas/tred/
[4] http://quest.ms.mff.cuni.cz/netgraph/
[5] http://ufal.mff.cuni.cz/~smrz/Encode/Arabic/
[6] http://search.cpan.org/dist/Encode-Arabic/
[7] ftp://ftp.informatik.uni-stuttgart.de/pub/arabtex/arabtex.htm
[8] http://www.haskell.org/
[9] http://subversion.tigris.org/

REFERENCES

Buckwalter, Tim (2002). *Buckwalter Arabic Morphological Analyzer 1.0*. LDC catalog number LDC2002L49, ISBN 1-58563-257-0.

Buckwalter, Tim (2004). *Buckwalter Arabic Morphological Analyzer 2.0*. LDC catalog number LDC2004L02, ISBN 1-58563-324-0.

Chiang, David and Mona Diab and Nizar Habash and Owen Rambow and Safiullah Sharif (2006). 'Parsing Arabic Dialects'. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics*, pages 369–376, Trento, Italy.

Corston-Oliver, Simon and Anthony Aue and Kevin Duh and Eric Ringger (2006). 'Multilingual Dependency Parsing using Bayes Point Machines'. In *Proceedings of HLT-NAACL 2006*, pages 160–167, New York.

El Dada, Ali and Aarne Ranta (2006). 'Implementing an Open Source Arabic Resource Grammar in Grammatical Framework'. In *Proceedings of the XXth Arabic Linguistics Symposium*. Benjamins.

Debusmann, Ralph (2006). *Extensible Dependency Grammar: A Modular Grammar Formalism Based On Multigraph Description*. PhD. thesis, Saarland University, Germany.

Forsberg, Markus and Aarne Ranta (2004). 'Functional Morphology'. In *Proceedings of ICFP 2004*, pages 213–223. ACM Press.

Graff, David and Ke Chen and Junbo Kong and Kazuaki Maeda (2006). *Arabic Gigaword Second Edition*. LDC catalog number LDC2006T02, ISBN 1-58563-371-2.

Habash, Nizar and Owen Rambow (2005). 'Arabic Tokenization, Part-of-Speech Tagging and Morphological Disambiguation in One Fell Swoop'. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics ACL 2005*, pages 573–580, Ann Arbor.

Habash, Nizar and Owen Rambow (2006). 'MAGEAD: A Morphological Analyzer and Generator for the Arabic Dialects'. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the ACL*, pages 681–688, Sydney, Australia.

Hajič, Jan and Otakar Smrž and Tim Buckwalter and Hubert Jin (2005). 'Feature-Based Tagger of Approximations of Functional Arabic Morphology'. In *Proceedings of TLT 2005*, pages 53–64, Barcelona, Spain.

Hajič, Jan and Otakar Smrž and Petr Zemánek and Petr Pajas and Jan Šnaidauf and Emanuel Beška and Jakub Kráčmar and Kamila Hassanová (2004). *Prague Arabic Dependency Treebank 1.0*. LDC catalog number LDC2004T23, ISBN 1-58563-319-4.

Hajičová, Eva and Petr Sgall (2003). 'Dependency Syntax in Functional Generative Description'. In *Dependenz und Valenz – Dependency and Valency*, volume I, pages 570–592. Walter de Gruyter.

Hudak, Paul (1998). 'Modular Domain Specific Languages and Tools'. In *Proceedings of the Fifth International Conference on Software Reuse*, pages 134–142. IEEE Computer Society Press.

Lagally, Klaus (1994). *Using TeX as a Tool in the Production of a Multi-Lingual Dictionary*. Technical Report 1994/15, Fakultät Informatik, Universität Stuttgart.

Lagally, Klaus (2004). *ArabTeX: Typesetting Arabic and Hebrew, User Manual Version 4.00*. Technical Report 2004/03, Fakultät Informatik, Universität Stuttgart.

Màrquez, Lluís and Dan Klein, eds. (2006). *Proceedings of CoNLL-X, the Tenth Conference on Natural Language Learning*. ACL, New York.

Sgall, Petr and Eva Hajičová and Jarmila Panevová (1986). *The Meaning of the Sentence in Its Semantic and Pragmatic Aspects*. D. Reidel & Academia.

Smith, Noah A. and David A. Smith and Roy W. Tromble (2005). 'Context-Based Morphological Disambiguation with Random Fields'. In *Proceedings of HLT/EMNLP 2005*, pages 475–482, Vancouver, Canada.

Smrž, Otakar (in prep.). *Functional Arabic Morphology. Formal System and Implementation*. PhD. thesis, Charles University in Prague, Czech Republic.

Smrž, Otakar and Petr Pajas (2004). 'MorphoTrees of Arabic and Their Annotation in the TrEd Environment'. In *Proceedings of the NEMLAR Conference 2004*, pages 38–41, Egypt.

Smrž, Otakar and Petr Zemánek and Jakub Kráčmar and Viktor Bielický (2006). 'Information Structure with the Prague Arabic Dependency Treebank'. In *Proceedings of the Conference on Communication and Information Structure in Spoken Arabic*, College Park, Maryland.

Žabokrtský, Zdeněk and Otakar Smrž (2003). 'Arabic Syntactic Trees: from Constituency to Dependency'. In *EACL 2003 Conference Companion*, pages 183–186, Budapest, Hungary.