TECHNICAL REPORT

No. CCLS-10-01

Title: MADA+TOKAN Manual

Authors: Nizar Habash and Owen Rambow and Ryan Roth

MADA+TOKAN Manual

Nizar Habash and Owen Rambow and Ryan Roth
Center for Computational Learning Systems
Columbia University
{habash,rambow,ryanr}@ccls.columbia.edu

June 2010
Current Version: MADA-3.0.1

1 Introduction

MADA[1] is a system for Morphological Analysis and Disambiguation for Arabic. TOKAN is a general tokenizer for MADA-disambigauted text. Internally, MADA also makes use of ALMORGEANA, an Arabic lexeme-based morphology analyzer.

2 Requirements

MADA+TOKAN is built for Unix/Linux systems, is Perl-based, and depends on a small number of third-party software tools. These tools, listed below, need to be successfully installed on the user's system prior to installing MADA+TOKAN. See the Installation section for details.

- **SVMTools version 1.3.1**

  Download SVMTool version 1.3.1 from:

  http://www.lsi.upc.es/ nlp/SVMTool/

[1] Mada (spelled in Buckwalter as madaY) is the Arabic word for "atmost/maximum point/degree".

- **SRI's Language Modeling Toolkit**

  Download the SRILM library (version 1.5.6 or later) from:

  http://www.speech.sri.com/projects/srilm/download.html

- **LDC's Standard Arabic Morphological Analyzer (SAMA) version 3.1**

  Obtain version 3.1 (catalog number LDC2009E73) from the LDC:

  http://www.ldc.upenn.edu/

  SAMA version 3.0 will also work, but MADA has been tuned for SAMA 3.1

  As of the time of this writing, SAMA is only available to members of the GALE project. If you are unable to obtain it, it is possible to run MADA using SAMA's predecessor, BAMA 2.0 (LDC catalog number LDC2004L02). However, using BAMA will result in a slight drop in MADA's selection accuracy (2-4% absolute, depending on the evaluation metric used). We therefore encourage the use of SAMA if at all possible.

3 Installation

For this version of MADA, we have included an `INSTALL.pl` Perl script to simplify and test the installation. MADA+TOKAN can be installed in six steps:

1. **Unpack the MADA installation archive**

   Untar the MADA-3.0 archive file in a whatever directory you would like to install it to. This directory will be referred to as `MADAHOME` throughout this document. For reference, a list of changes that occurred for each MADA version can be found under `MADAHOME/MADA.CHANGES`.

2. **Install SVMTools 1.3.1**

   Install SVMTools 1.3.1, and take note of its installation directory on your system (we will refer to this as `SVMTOOLSHOME` in this document). Note that version 1.3.1 fixes a crucial problem that existed in version 1.3, namely that 1.3 was incompatible with Perl 5.10. We therefore strongly recommend that users upgrade to 1.3.1 if they have not done so already.

3. **Install SRILM toolkit**

   Install the SRILM toolkit, specifically the `disambig` executable. Take note of the main SRI installation directory (we will refer to this as `SRIHOME` in this document).

4. **Acquire LDC's SAMA 3.1 (or BAMA 2.0)**

   Acquire the LDC's Standard Arabic Morphological Analyzer, version 3.1. SAMA is a replacement for the previous Buckwalter Arabic Morphological Analyzer (BAMA). SAMA and BAMA are collectively referred to as *XAMA* in this document. It is unnecessary to install/make the XAMA software; MADA is specifically interested in the XAMA database files:

   ```
   dictPrefixes  dictStems  dictSuffixes
   tableAB       tableAC    tableBC
   ```

   These files are located in the `SAMA-3.1/lib/SAMA_DB/v3_1` subdirectory for SAMA-3.1. Locate and record the directory containing these files (referred to as `XAMADIR` in this document) for your XAMA version.

   Since there are differences in between the different versions of XAMA, we have built an utility that will read the XAMA data files and build a common-format database for use with ALMORGEANA. This utility will only need to be run once during MADA installation; thereafter MADA will rely solely on the constructed database. Currently, this utility (and MADA in general) is tuned to function well with SAMA 3.1, but it may also be used with SAMA 3.0 (which gives about the same results as SAMA 3.1) or BAMA 2.0 (not recommended, as there will be a small accuracy drop).

5. **Adjust your `PERL5LIB` environment variable**

   Adjust your `PERL5LIB` environment variable to include both the `MADAHOME` and the SVMTools libraries (`SVMTOOLSHOME/lib`). This can be done using the `export` command (for bash shells). As an example:

   ```
   export PERL5LIB=$PERL5LIB:/home/nlp/MADA-3.0: \\
                  /home/nlp/tools/SVMTool-1.3.1/lib
   ```

   It would be best to adjust your system's `.profile` or `.bashrc` file so that this environment variable is set every time you log in.

6. **Run the install script**

   Run `perl  INSTALL.pl` with the following arguments:

   ```
   perl INSTALL.pl madahome=MADAHOME srihome=SRIHOME \\
         svmhome=SVMTOOLSHOME xamadir=XAMADIR \\
         xamaversion=SAMA3.1
   ```

   or, if SAMA 3.0 is being used:

   ```
   perl INSTALL.pl madahome=MADAHOME srihome=SRIHOME \\
         svmhome=SVMTOOLSHOME xamadir=XAMADIR \\
         xamaversion=SAMA3.0
   ```

   where `MADAHOME`, `SVMTOOLSHOME`, `SRIHOME`, and `XAMADIR` are the directory paths as noted from the previous installation steps.

   This `INSTALL.pl` script will do the following, automatically:

   - Verify the existence of the needed SRI, SVMTOOLS, XAMA directories and files.
   - Verify that MADA and SVMTools have been added to `PERL5LIB`.
   - Creates a ALMORGEANA database file using the XAMA files; this database will be placed in the `MADAHOME/MADA/` directory, with the softlink `almor.db` pointing to it.
   - Creates a template MADA configuration file:

     `MADAHOME/config-files/template.madaconfig`

     that uses default values for every MADA configuration variable, but has the variables `MADA_HOME`, `SRI_NGRAM_TOOL`, and `SVM_TAGGER` set to the particulars of the user's system. Users can use this as their default `.madaconfig` file, but should customize it based on what they need MADA+TOKAN to do.
   - Runs `MADA+TOKAN.pl` on the file

     `MADAHOME/SAMPLE/sample+ID.ar.utf8`

     as a test of the system. The output files are compared to the

     `MADAHOME/SAMPLE/GOLD.sample+ID.ar.utf8.*`

     files, and the result reported.

   Should the `INSTALL.pl` script fail at any step, it will stop and output a report of the failure. This report should, hopefully, make fixing the problem straightforward.

If you have a problem with the `INSTALL.pl` script, send an email describing the problem, along with a full print out of the `INSTALL.pl` error output, to us and we will help you correct the issue.

Assuming that there are no such problems, MADA+TOKAN is now ready to be used.

4 Running MADA+TOKAN

Once MADA+TOKAN is successfully installed, a few steps need to be followed to prepare the input data for processing:

1. **Create or edit a MADA configuration file**

   MADA uses a configuration file to control its operation. This file has variables in the format:

   ```
   <variable name> = <variable value>
   ```

   By convention, variable names appear in ALLCAPS, with underscores. Everything to the right of a "#" character is a comment. The `INSTALL.pl` script creates a template MADA configuration file here:

   ```
   MADAHOME/config-files/template.madaconfig
   ```

   This template fully documents all the MADA+TOKAN configuration variables – what they control and what the valid options are.

   Typically, users will create one configuration file for each general experiment they want to run, adjusting it as necessary.

2. **(Optional) Create or edit a TOKAN schemes file**

   In previous versions of MADA+TOKAN, the configuration variable `TOKAN_SCHEME` was used to control the desired output of TOKAN. This is still possible. However, as of MADA 3.0 it is now possible to define a separate "TOKAN schemes" file that defines more than one `TOKAN_SCHEME`. If this file is used, TOKAN will produce a distinct output file for every scheme so listed. Running all the schemes through TOKAN together leads to some savings in running time, and is primarily useful for users who need to compare different tokenizations of the same data. The format of the TOKAN schemes file is one scheme per line, like so:

   ```
   <scheme extension>  <TOKAN_SCHEME to use>
   ```

The scheme extension is the string that will be added to the end of the output files to distinguish each scheme output from each other. An example TOKAN scheme file can be found here:

```
MADAHOME/config-files/TOKAN.scheme
```

Once a TOKAN scheme file has been created, you can use it by setting the MADA variable `TOKAN_SCHEME_FILE` to point to it in the MADA configuration file. If a TOKAN scheme file is defined, the MADA configuration variables `TOKAN_SCHEME` and `TOKAN_OUTPUT_EXTENSION` are ignored.

3. **Prepare your input data**

   Data that is given to MADA should be formatted to be one-sentence-per-line, with no metadata, HTML/XML tags, etc. Optionally, you can have the first word of each line be interpreted as a "sentence ID" – a string of non-whitespace characters used to identify the sentence, but is not processed as part of the sentence.

   Under MADA 3.0, you can now give MADA raw UTF-8 encoded Arabic text (previously we required input to be in Buckwalter encoding). MADA 3.0 includes a pre-processor component that can clean UTF8 data, add necessary whitespace between punctuation/numbers and words, tag any ASCII words (assumed to be foreign words in a UTF8 document), and convert the whole text to the Buckwalter encoding that MADA uses internally.

   The operation of the MADA pre-processor is controlled through the *INPUT FORMAT OPTIONS* group of configuration variables in your configuration file. You will need to ensure that your input data format corresponds to these settings in your configuration file. For example, if your input data is already in Buckwalter encoding, but you still want MADA to perform the whitespace separation of punctuation, make sure your configuration variable `RUN_PREPROCESSOR` is set to `YES`, `INPUT_ENCODING` is set to `Buckwalter` and the variable `SEPARATEPUNCT` is set to `YES`.

4. **Run** `perl MADA+TOKAN.pl`

   ```
   perl MADA+TOKAN.pl config=<config file> file=<text>
   ```

   where <config file> is the MADA configuration file and <text> is the input file for MADA to process.

Note that you can, if desired, override any variable in the configuration file by including it in the command line. This is handy if you want to run an extra experiment with only a slight change of configuration, and don't want to create an new configuration file. For example:

```
perl MADA+TOKAN.pl config=template.madaconfig \\
      file=test RUN_TOKAN=NO COMPRESS_OUTPUTS=YES \\
      PRINT_ANALYSES=stars
```

Here, the values of the three variables on the command line will overwrite what is in the configuration file. When doing this, use quotes to enclose variable values consisting of more than one word:

```
perl MADA+TOKAN.pl config=template.madaconfig \\
      file=test TOKAN_SCHEME="SCHEME=D2 MARKNOANALYSIS" \\
```

In these examples, the output files would be placed in the same directory as `test`, and would be named `test.bw` (pre-processed version of the file), `test.bw.mada` (MADA output) and `test.bw.mada.tok` (TOKAN output). This assumes that the default configuration options are used.

5 MADA Details

MADA is divided into several sub-components, each with their own control script. Note that if you run `MADA+TOKAN.pl` without any arguments, you will get a list of all the configuration variables used in each sub-component. The sub-components are:

1. **MADA-preprocessor.pl** – Formats input data

2. **MADA-morphanalysis.pl** – Calls ALMORGEANA to generate, for each input word, a list of possible analyses, with no regard to context

3. **MADA-generate-SVM+ngram-files.pl** – Determines N-gram statistics for diacritic word forms and lexemes, and creates back-off lexicons for the next step

4. **MADA-runSVMTOOLS.pl** – Runs an independent SVM classifier for a number of MADA features, determining a prediction for that feature value for each word

5. **MADA-selectMA.pl** – For each word, examines each of the possible analyses and scores each one. The score is developed by comparing the features of each analysis to the SVM prediction; analyses that have agreement with the prediction are given a weighted increase in score. Some additional, non-SVM features are factored in as well. The scores are then normalized, sorted and labeled. Tie-breaking in employed to insure that only one analysis for each word is designated as the correct one.

Each of the above sub-components can be run separately, but this should only be attempted by advanced users. Like `MADA+TOKAN.pl`, running any of the sub-components without any arguments will produce a list of the options and configuration variables that component uses.

When finished, the MADA output (a *.mada file) lists the top analyses for each word (and possibly the other analyses, depending on the `PRINT_ANALYSES` configuration variable).

5.1 MADA Model Data

The current SVM and N-gram models that are included with the MADA release were created using data from the Penn Arabic Treebank (PATB) 3, version 3.1. The data was divided into a training set, a tuning set (for feature weight tuning) and a test set. The PATB documents that correspond to these sets are:

- TRAINING:
  ANN20020115.0001 - ANN20021015.0100

- TUNING:
  ANN20021015.0101 - ANN20021015.0122
  ANN20021115.0001 - ANN20021115.0066

- TESTING:
  ANN20021115.0068 - ANN20021115.0119
  ANN20021215.0003 - ANN20021215.0045

5.2 MADA Features

Under MADA 3.0, the feature set has been significantly altered; for example, the old features **def** and **idafa** have been combined into a new feature, **stt** (state). These changes were made in order to model the language more closely, and also to adapt to the changes made in SAMA and the PATB. These differences are summarized in Tables 1- 4.

Table 1 shows the MADA features which have undergone (with the exception of **stt**) the least change. For some of these we have added a new value – "Undefined (**u**)". These represent cases where the morphological analyzer does not provide a value for the feature. Previously, MADA 2.32 would give these cases the indicated 'default' value.

Table 2 shows the expanded version of the part-of-speach (**pos**) feature. This feature has been refined, allowing for greater distinction between values. For reference, we supply the PATB POS tag that is equivalent to the new **pos** value.

Table 3 and Table 4 show the MADA proclitic and enclitic features, respectively. We have significantly altered our handling of these features. Previously, clitic information was carried by four binary features (**art**, **part**, **conj** and **clitic**), which would only say whether or not a clitic of a particular type was present. Under MADA 3.0, we use five new features to exactly specify the clitics that are present. These features are organized according to the possible location of the clitic in the word and a consideration of what clitics can co-occur, rather than the exact clitic type (such as particles). The pattern these clitics can follow is:

```
[ prc3 [ prc2 [ prc1 [ prc0 BASEWORD enc0 ] ] ] ]
```

5.3 MADA Output Format

At the top of the MADA output file is a header of comment lines which specify the command used to generate the file, the classifiers used and other options. Following this each word is presented followed by a listing of its possible analyses (morphological tags). Word analyses that are selected as the best option given the word context are marked with a leading '*'. Some analyses may be marked with a '^' ; this indicates that this analysis was tied in score with the '*' analysis, and a tie-breaking method (arbitrary or random) was used to pick the '*' analysis over this one. All other, less suitable analyses are marked with a leading '_'. Following the leading marker is a score, and the analysis feature line.

Each analysis feature line consists of a set of `<feature>:<value>` pairs, separated by whitespace. Most of these features have a corresponding SVM classifier (see Tables 1- 4). The rest include the diacritic form (**diac**), the lexeme/lemma (**lex**), the Buckwalter tag (**bw**), the gloss (**gloss**), and a few others which are only used internally by MADA and ALMORGEANA.

Figure 1 shows an excerpt from a MADA output file. Note that the file lines are wrapped here for clarity. The "`;;MADA`" lines indicate the predictions of the SVM

Feature	Feature Value Definition	MADA 3.0	MADA 2.32
Aspect	LABEL	**asp**	**aspect**
	Command	c	CV
	Imperfective	i	IV
	Perfective	p	PV
	Not applicable	na	NA
Case	LABEL	**cas**	**case**
	Nominative	n	NOM
	Accusative	a	ACC
	Genitive	g	GEN
	Not applicable	na	NA
	Undefined	u	NOCASE (default)
Gender	LABEL	**gen**	**gen**
	Feminine	f	FEM
	Masculine	m	MASC
	Not applicable	na	NA
Mood	LABEL	**mod**	**mood**
	Indicative	i	I
	Jussive	j	J
	Subjunctive	s	S
	Not applicable	na	NA
	Undefined	u	I (default)
Number	LABEL	**num**	**num**
	Singular	s	SG
	Plural	p	PL
	Dual	d	DU
	Not applicable	na	NA
	Undefined	u	SG (default)
Person	LABEL	**per**	**per**
	1st	1	1
	2nd	2	2
	3rd	3	3
	Not applicable	na	NA
State	LABEL	**stt**	**def** and **idafa**
	Indefinite	i	def:INDEF idafa:NOPOSS
	Definitie	d	def:DEF idafa:NOPOSS
	Construct/Poss/Idafa	c	def:DEF idafa:POSS
	Not applicable	na	def:NA idafa:NA
	Undefined	u	def:DEF idafa:NOPOSS (default)
Voice	LABEL	**vox**	**voice**
	Active	a	ACT
	Passive	p	PASS
	Not applicable	na	NA
	Undefined	u	ACT (default)

Table 1: MADA feature and value definitions, with the labels used to represent them under MADA 3.0 and MADA 2.32. "LABEL" indicates the identifying tag used for that feature in MADA output files.

	POS Definition	**MADA 3.0**	**MADA 2.32**	**PATB Equivalent**
Part-of-speech	LABEL	**pos**	**pos**	—
	Nouns	noun	N	NN / NNS
	Number Words	noun_num	N	NN / NNS
		noun_quant	N	NN / NNS
	Proper Nouns	noun_prop	PN	NNP / NNPS
	Adjectives	adj	AJ	JJ
		adj_comp	AJ	JJ
		adj_num	AJ	JJ
	Adverbs	adv	AV	RB
		adv_interrog	Q	RP
		adv_rel	REL	WP
	Pronouns	pron	PRO	PRP
		pron_dem	D	DT
		pron_exclam	PRO	PRP
		pron_interrog	Q	RP
		pron_rel	REL	WP
	Verbs	verb	V	VBN / VBP / VBD
		verb_pseudo	V	VBN/ VBP / VBD
	Particles	part	P	IN
		part_det	D	DT
		part_focus	P	IN
		part_fut	P	IN
		part_interrog	P	IN
		part_neg	NEG	RP
		part_restrict	P	IN
		part_verb	P	IN
		part_voc	P	IN
	Prepositions	prep	P	IN
	Abbreviations	abbrev	AB	NN
	Punctuation	punc	PX	PUNC
	Conjunctions	conj	C	CC
		conj_sub	C	CC
	Interjections	interj	IJ	UH
	Digital Numbers	digit	NUM	CD
	Foreign/Latin	latin	F	IN

Table 2: MADA part-of-speech definitions and the labels used to represent them under MADA 3.0 and MADA 2.32, with the equivalent Penn ATB POS tags given as reference.

	Proclitic Value Definition	**MADA 3.0**	**MADA 2.32**
Proclitic 3 (AKA question proclitic or QUES)	LABEL	**prc3**	—
	No proclitic	0	—
	Not applicable	na	—
	Interrogative Particle >*a*	>a_ques	—
Proclitic 2 (AKA conjunction proclitic or CONJ)	LABEL	**prc2**	**conj**
	No proclitic	0	NO
	Not applicable	na	NA
	Conjunction *fa*	fa_conj	YES
	Connective particle *fa*	fa_conn	YES
	Response conditional *fa*	fa_rc	YES
	Subordinating conjunction *fa*	fa_sub	YES
	Conjunction *wa*	wa_conj	YES
	Particle *wa*	wa_part	YES
	Subordinating conjunction *wa*	wa_sub	YES
Proclitic 1 (AKA preposition proclitic or PART)	LABEL	**prc1**	**part**
	No proclitic	0	NO
	Not applicable	na	NA
	Particle *bi*	bi_part	YES
	Preposition *bi*	bi_prep	YES
	Preposition *ka*	ka_prep	YES
	Emphatic Particle *la*	la_emph	YES
	Preposition *la*	la_prep	YES
	Response conditional *la*	la_rc	YES
	Jussive *li*	li_jus	YES
	Preposition *li*	li_prep	YES
	Future marker *sa*	sa_fut	YES
	Preposition *ta*	ta_prep	YES
	Particle *wa*	wa_part	YES
	Preposition *wa*	wa_prep	YES
	Preposition *fy*	fy_prep	YES
	Negative particle *lA*	lA_neg	YES
	Negative particle *mA*	mA_neg	YES
	Vocative *yA*	yA	YES
	Vocative *wA*	wA	YES
	Vocative *hA*	hA	YES
Proclitic 0 (AKA article proclitic or ART)	LABEL	**prc0**	**art**
	No proclitic	0	NO
	Not applicable	na	NA
	Determiner	Al	YES
	Negative particle *lA*	lA_neg	YES
	Negative particle *mA*	mA_neg	YES
	Relative pronoun *mA*	mA_rel	YES
	Particle *mA*	mA_part	YES

Table 3: MADA proclitic definitions and the labels used to represent them under MADA 3.0 and MADA 2.32. The proclitic number refers to the location of the clitic, according to [PRC3 [PRC2 [PRC1 [PRO0 BASEWORD ENC0]]]]

	Enclitic Value Definition	**MADA 3.0**	**MADA 2.32**
Enclitics (AKA pronominals or PRON)	LABEL	**enc0**	**clitic**
	No enclitic	0	NO
	Not applicable	na	NA
	1st person plural	1p	YES
	1st person singular	1s	YES
	2nd person dual	2d	YES
	2nd person feminine plural	2fp	YES
	2nd person feminine singular	2fs	YES
	2nd person masculine plural	2mp	YES
	2nd person masculine singular	2ms	YES
	3rd person dual	3d	YES
	3rd person feminine plural	3fp	YES
	3rd person feminine singular	3fs	YES
	3rd person masculine plural	3mp	YES
	3rd person masculine singular	3ms	YES
	Vocative particle	Ah	YES
	Interrogative pronoun *man*	man_interrog	YES
	Interrogative pronoun *mA*	mA_interrog	YES
	Interrogative pronoun *ma*	ma_interrog	YES
	Relative pronoun *man*	man_rel	YES
	Relative pronoun *mA*	mA_rel	YES
	Relative pronoun *ma*	ma_rel	YES
	Subordinating conjunction *ma*	ma_sub	YES
	Subordinating conjunction *mA*	mA_sub	YES
	Negative particle *lA*	lA_neg	YES

Table 4: MADA enclitic definitions and the labels used to represent them under MADA 3.0 and MADA 2.32. The clitic number refers to the location of the clitic, according to [PRC3 [PRC2 [PRC1 [PRO0 BASEWORD ENC0]]]]

```
;;; SENTENCE_ID SAMPLE_ID:31
;;; SENTENCE blyr yblg bw$ bntA}j jwlth fy Al$rq AlAwsT AlArbEA’ \\
             Almqbl
;;WORD blyr
;;MADA: blyr asp:na cas:u enc0:0 gen:m mod:na num:s per:na  \\
             pos:noun_prop prc0:0 prc1:0 prc2:0 prc3:0 stt:i vox:na
*1.000623 diac:bliyr lex:bliyr_1 bw:+bliyr/NOUN_PROP+ gloss:Blair \\
          pos:noun_prop prc3:0 prc2:0 prc1:0 prc0:0 per:na asp:na \\
          vox:na mod:na gen:m num:s stt:i cas:u enc0:0 rat:y \\
          source:lex stem:bliyr stemcat:Nprop
_0.897942 diac:biliyr lex:liydz_1 bw:bi/PREP+liyr/NOUN_PROP+  \\
          gloss:Lear pos:noun_prop prc3:0 prc2:0 prc1:bi_prep \\
          prc0:0  per:na asp:na vox:na mod:na gen:m num:s stt:i \\
          cas:u enc0:0 rat:y source:lex stem:liyr stemcat:Nprop
_0.897918 diac:biliyr lex:liydz_1 bw:bi/PART+liyr/NOUN_PROP+ \\
          gloss:Lear  pos:noun_prop prc3:0 prc2:0 prc1:bi_part  \\
          prc0:0 per:na asp:na vox:na mod:na gen:m num:s stt:i \\
          cas:u enc0:0 rat:y source:lex  stem:liyr stemcat:Nprop
--------------
;;WORD yblg
;;MADA: yblg asp:i cas:na enc0:0 gen:m mod:i num:s per:3 pos:verb \\
             prc0:0 prc1:0 prc2:0 prc3:0 stt:na vox:a
*0.991246 diac:yabolugu lex:balag-u_1 \\
          bw:ya/IV3MS+bolug/IV+u/IVSUFF_MOOD:I gloss:reach;attain \\
          pos:verb prc3:0 prc2:0 prc1:0 prc0:0 per:3 asp:i vox:a \\
          mod:i gen:m num:s stt:na cas:na enc0:0 rat:na source:lex \\
          stem:bolug stemcat:IV
_0.968597 diac:yabolugu lex:balug-u_1 \\
          bw:ya/IV3MS+bolug/IV+u/IVSUFF_MOOD:I gloss:be_eloquent \\
          pos:verb prc3:0 prc2:0 prc1:0 prc0:0 per:3 \\
          asp:i vox:a mod:i gen:m num:s stt:na cas:na enc0:0 \\
          rat:na source:lex stem:bolug stemcat:IV_intr
_0.945947 diac:yuboligu lex:>abolag_1 \\
          bw:yu/IV3MS+bolig/IV+u/IVSUFF_MOOD:I \\
          gloss:report;inform;notify pos:verb prc3:0 prc2:0 prc1:0 \\
          prc0:0 per:3 asp:i vox:a mod:i gen:m num:s stt:na cas:na \\
          enc0:0 rat:na source:lex stem:bolig stemcat:IV_yu
_0.945947 diac:yubal~igu lex:bal~ag_1 \\
          bw:yu/IV3MS+bal~ig/IV+u/IVSUFF_MOOD:I \\
          gloss:communicate;convey pos:verb prc3:0 prc2:0 prc1:0 \\
          prc0:0 per:3 asp:i vox:a mod:i gen:m num:s stt:na cas:na \\
          enc0:0 rat:na source:lex stem:bal~ig stemcat:IV_yu
......
```

Figure 1: MADA output excerpt. The lines have been wrapped for readability.

classifiers for that word. Each new sentence starts with a "`;;; SENTENCE`" line comment, and a "`;;; SENTENCE_ID`" comment (if defined). Each sentence ends with a "`SENTENCE BREAK`" line (not pictured).

6 TOKAN Details

The output of TOKAN (a *.tok file, by default) contains a tokenized version of the disambiguated input, generated deterministically. Since Arabic words can have different analyses which can result in different tokenizations under different tokenization schemes, both MADA and TOKAN scheme-selection are necessary to tokenize: MADA selects the contextually apporpriate analysis and TOKAN tokenizes it according to a specific (deterministic) tokenization ruleset (a ‘scheme’). Under some coarse tokenization schemes (e.g., split off the conjunction `w+`) different analyses of the same word often result in the same tokenization. We discuss next the different tokenization options that can be used to specify a tokenization scheme.

The `TOKAN_SCHEME` configuration variable controls the output format of TOKAN, i.e., what variety of tokenization is implemented and how it looks. Under MADA 3.0 and later, the customizability of the `TOKAN_SCHEME` has been greatly extended. This has had the side effect of causing older scheme formats (MADA 2.32 and earlier) to be rendered invalid, so be aware of this if you are migrating from MADA 2.32 or earlier to MADA 3.0. Also be aware that scheme variables no longer require leading hyphens, and several obsolete variables will cause TOKAN to exit with an error message.

Under MADA 3.0, the `TOKAN_SCHEME` can consist of four types of variables:

1. **Single Variables** – Variables that affect the entire scheme; for example, `GROUPTOKENS` will cause all tokens in the scheme to be linked together by a delimiter character (defaults to ‘_’), rather than whitespace.

2. **SPLIT Variables** – Variables that control how the input word is broken up, such as breaking off conjunctions or particles

3. **FORM Variables** – Variables that control how the different tokens are output, including their arrangement and content

4. **Aliases** – Variables of the form `SCHEME=XXX` which are shorthand for longer, commonly-used schemes. Most users will find it easiest to apply a known alias (if possible) rather than design an entirely new scheme.

In general, TOKAN_SCHEMEs are arranged in the format (all on one line):

```
::SPLIT <SPLIT Variables> \\
     ::FORM0 <FORM variables for Form id 0> \\
     ::FORM1 <FORM variables for Form id 1> \\
     ::FORM2 <FORM variables for Form id 2> \\
     ...
     ::FORMN <FORM variables for Form id N> \\
     <Single Variables>
```

or, alternatively:

```
SCHEME=<alias for established scheme> <Single Variables>
```

Note that more than one ::FORM can be defined in a single scheme, each identified with a numerical id. The specific form variables for that id must directly follow the leading ::FORM marker. Similarly, the split variables must directly follow the ::SPLIT marker.

6.1 TOKAN_SCHEME: Single Variables

These variables affect the entire TOKAN_SCHEME and all the FORMs it includes. They can appear anywhere in the scheme, but are best placed at the very end to avoid confusion with SPLIT and FORM variables. They are described in Table 5.

6.2 TOKAN_SCHEME: Split Variables

The SPLIT variables control which clitics are separated from the main word and in what order they are presented. Users can also add newlines in arbitrary places, specify where the remainder of the word should appear, and indicate which tokens or token subgroup should be joined by TDELIM.

Table 6 shows the possible SPLIT variables a scheme can contain. TOKAN can still interpret the old-style TOKAN_SCHEME variables: w+, f+, b+, l+, k+,+P:, and +O:. In addition, it is possible to specify individual clitics (such as prc1:k), rather than the entire group (PART), but this is an advanced usage that many users will not require.

One example of a correct SPLIT variable setup for the ATB tokenization is:

Single Variable	Description
TDELIM:<characters>	Token Delimiter. If a scheme requires tokens (or a subset) to be grouped, this is the character(s) that will join them for all forms. The default delimiter is '_'.
FDELIM:<characters>	Form Delimiter. If more than one form is defined, these will be used to separate the different forms in the output. The default is the middle-dot character '·' (Unicode #00B7).
SENT_ID	If present, this variable cause TOKAN to look for any SENTENCE_ID comments in the input MADA file and print those IDs as the first word in the output of each sentence. SPLIT and FORM variables are never applied to sentence IDs.
MARKNOANALYSIS	If present, this variable will cause any word marked as NO-ANALYSIS in the MADA input to be presented in the output with "@@" as a prefix and suffix, e.g.: "@@UNKNOWN_WORD@@". These marks will be repeated in every form specified, if there are more than one.
GROUPTOKENS	If present, this will cause all the tokens of a word to be joined with the TDELIM character(s).
NOPASSATAT	By default, TOKAN will print any word marked as ;;PASS by MADA as is, without any alteration (MADA marks any input word starting with "@@" as a ;;PASS word). If this variable is present in the scheme, however, TOKAN will omit these words from its output entirely.

Table 5: TOKAN_SCHEME Single Variables

```
::SPLIT QUES CONJ PART NART REST PRON ...
```

This will split off any question marking proclitic, follow it with any present conjunctions, followed by any present prepositions, followed by any negative articles (but not the definite article Al, which remains attached to the word under ATB), followed by the main part of the word, and ending with any endclitics.

If we wanted to group the QUES, CONJ and PART tokens together and leave the rest separate, we can do this:

```
::SPLIT [+ QUES CONJ PART +] NART REST PRON  TDELIM:
```

In this example, we also changed the token delimiter to "", so in this case the grouped tokens will not have any characters (whitespace or otherwise) between them. We could have just as easily used hyphens (with TDELIM:-) or triple

SPLIT Variable	Description
QUES or prc3	**prc3** - The ‘question’ proclitic
CONJ or prc2	**prc2** - The ‘conjunction’ proclitic
PART or prc1	**prc1** - The ‘preposition’ proclitic
ART or prc0	**prc0** - The ‘article’ proclitic
PRON or enc0	**enc0** - Enclitics
FUT or s+	The future marker clitic only (s)
DART or Al+	The definite article only (Al)
NART	The negative articles only (lA, mA)
REST	The remainder of the word after the specified clitics have been separated
NEWLINE	Where a newline character should be inserted
[+ and +]	Group markers. Any token subset surrounded by [+ and +] will be grouped with TDELIM character(s).
w+	(Old style) The wa conjunction only
f+	(Old style) The fa conjunction only
b+	(Old style) The bi preposition only
l+	(Old style) The li or la prepositions only
k+	(Old style) The ka preposition only
+P:	(Old style) Poss enclitics only
+O:	(Old style) Other enclitics only

Table 6: TOKAN_SCHEME SPLIT Variables

underscores (with TDELIM:____).

We can also insert newlines wherever we like:

```
::SPLIT QUES CONJ PART NEWLINE NART REST NEWLINE PRON NEWLINE
```

This example would create 3 lines of output for every word in the input.

6.3 TOKAN_SCHEME: Form Variables

Form variables are used to control how the output looks. In addition, multiple forms can be specified, allowing additional information (such as part-of-speech, lexeme or gloss) to be included with each token. Each form in the specification will be applied to each token defined by the SPLIT variables.

All form definitions begin with the ::FORM<N> keyword, where <N> is a non-negative numerical id. Every variable that follows the ::FORM<N> keyword (up to the next ::FORM<N+1> keyword) applies only to that form. Form vari-

BASE Variable	Description
WORD	This simply says that this form will be displaying some version of the token itself (original, normalized, etc.) It is the most common BASE.
LEXEME	This says that this form will be displaying lexeme information for the token.
GLOSS	This form will display the gloss term provided by ALMORGEANA.
STEM	This form will display the Buckwalter tag provided by ALMORGEANA.
SURF	This form uses the original word form.
POS:ALMOR, POS:CATIB, POS:PENN, POS:BW	These keys indicate that the form will display part-of-speech, using one of four different POS tagsets (ALMORGEANA, CATiB, Penn ATB, or Buckwalter).
POS:MADA	This form displays a '#'-separated list of 14 MADA features and values, (as defined in Tables 1- 4), including the POS used by MADA internally.
COPY<N>	Causes TOKAN to copy the previously defined form specification of ::FORM<N> to this form, which can then be further modified. Essentially just a way to avoid repeating common form elements.

Table 7: TOKAN_SCHEME FORM Variables: BASEs

ables are read sequentially from left to right; this means that, while it is possible to set a variable to two different values in a single form, only the rightmost one will be remembered and used.

The ::FORM<N> keyword should be immediately followed by one of the BASE keywords (see Table 7), which tell TOKAN which information is requested. BASE keywords can then be followed by other form variables (see Table 8) to determine specifics. Future versions of TOKAN will likely allow additional form variables.

As an example, the following scheme snippet defines three forms:

```
::FORM0 WORD NORM:A ENCMARK:PLUS ::FORM1 COPY0 NORM:H \\
       ENCMARK:HASH ::FORM2 LEXEME
```

The first form says to write the token, but normalize alefs and add a leading '+' character to enclitics. The second (separated from the first by the FDELIM character(s)) copies ::FORM0 and then adjusts it. The second form will write the token, normalize alefs and hamzas, and will add a leading '#' to enclitics (overwritting the previous ENCMARK:PLUS of ::FORM0). Finally, the third form (after another FDELIM character(s)) will print the lexeme. Since SHOWINDEX

FORM Variable	Allowed BASEs	Description
SHOWINDEX	LEXEME only	Causes the lexeme form to keep the trailing u,a,i diacritic tags and the "_<number>" suffix that appear, for example, in the PATB. The default LEXEME operation is to drop these.
ESC:PAREN	WORD, LEXEME, STEM, SURF, COPY	Causes TOKAN to replace any ‘(’, ‘)’ characters with "-LRB-" and "-RRB-"
NORM:A	WORD, LEXEME, STEM, SURF, COPY	Causes TOKAN to normalize alefs in the form. In Buckwalter, ‘>’, ‘<’, and ‘|’ become ‘A’
NORM:Y	WORD, LEXEME, STEM, SURF, COPY	Normalize yaas in the form. In Buckwalter, ‘Y’ becomes ‘y’
NORM:H	WORD, LEXEME, STEM, SURF, COPY	Normalize hamzas in the form. In Buckwalter, ‘&’ and ‘}’ become ‘'’
NORM:T	WORD, LEXEME, STEM, SURF, COPY	Normalize teh-marbutas in the form. In Buckwalter, ‘p’ becomes ‘h’
DIAC	WORD, LEXEME, STEM, SURF, COPY	Remove all diacritics from the form. In Buckwalter, these are [aiuo`~FKN], Note that, if the token consists of nothing but diacritic characters, none are deleted to avoid removing an entire token.
ENCMARK:PLUS, ENCMARK:HASH, ENCMARK:NONE	All except GLOSS	Referred to as the "enclitic marker" Specifies whether enclitics should have a leading ‘+’, a leading ‘#’, or no leading character inserted. The default is to use none.
PROCMARK:PLUS, PROCMARK:HASH, PROCMARK:NONE	All except GLOSS	Referred to as the "proclitic marker" Specifies whether proclitics should have a trailing ‘+’, a trailing ‘#’ or no trailing character inserted. The default is to use none.

Table 8: TOKAN_SCHEME FORM Variables: non-BASEs

wasn't specified, the lexeme will be stripped of its trailing tags.

6.4 TOKAN_SCHEME: Aliases

Finally, to avoid having to specify lengthly scheme definitions, we provide several aliases for the most common tokenization schemes we've encountered. Users can activate these aliases by starting the scheme with "SCHEME=<alias>". TOKAN will replace this tag with the full scheme definition. Advanced users can futher modify the alias, adding addition forms or making other changes if they wish, by following the SCHEME tag with additional variables.

Alias	Description
SCHEME=ATB	Tokenizes all clitics except for the definite article, normalizes alefs/yaa, uses ‘+’ as clitic markers, and replaces ‘(’ and ‘)’ characters. Only one WORD form.
SCHEME=ATB-HASH	Same as ATB, except that enclitics are marked with ‘#’
SCHEME=TB	Same as ATB
SCHEME=TB-HASH	Same as ATB#
SCHEME=ATB+POS	Same as ATB, but adds a second form – the PATB POS tag. The middle-dot character ‘·’ is used as a form separator by default.
SCHEME=ATB-HASH+POS	Same as ATB+POS, except that enclitics are marked with ‘#’
SCHEME=ATB4MT	A large scheme consisting of 6 forms (also referred to as a "6-tier" scheme). Form 0 is a WORD form that tokenizes all clitics except the definite article, uses ‘+’ as a clitic marker, and replaces ‘(’ and ‘)’; Form 1 is the same, but it also normalizes alefs/yaas; Form 2 is a LEXEME form, using ‘+’ clitic markers and removing diacritics; Forms 3, 4, and 5 are the CATiB, Penn ATB and Buckwalter POS tags, respectively.
SCHEME=OLDATB	A tokenization that was previously used in the PATB. Only explicitly tokenizes f+, w+, b+, k+, l+, and enclitics. Uses ‘+’ as clitic markers, normalizes alefs/yaas, and replaces ‘(’ and ‘)’ characters.
SCHEME=D1	Tokenizes question and conjunction clitics only; uses ‘+’ as a clitic marker, normalizes alefs/yaas, and replaces ‘(’ and ‘)’ characters. Only one WORD form.
SCHEME=D1-HASH	Same as D1, but enclitics are marked with ‘#’
SCHEME=D2	Same as D1, but also tokenizes PART clitics
SCHEME=D2-HASH	Same as D2, but enclitics are marked with ‘#’
SCHEME=D3	Same as D2, but also tokenizes all articles and enclitics (basically all clitics are tokenized).
SCHEME=D3-HASH	Same as D3, but enclitics are marked with ‘#’
SCHEME=D1-3tier	A three-form (3-tier) scheme. Form 0 tokenizes question and conjunction clitics only, uses ‘+’ clitic markers, and replaces ‘(’ and ‘)’ characters; Form 1 is the same, but also normalizes alefs/yaas; Form 2 is a LEXEME form, using ‘+’ clitic markers and removing diacritics.
SCHEME=D2-3tier	The same as D1-3tier, except that the first two forms also tokenize PART clitics.
SCHEME=D3-3tier	The same as D2-3tier, except that all clitics are tokenized.
SCHEME=D14MT	Another large 6-form (6-tier) scheme. Effectively the same as ATB4MT, except that only the question and conjunction clitics are tokenized.
SCHEME=D24MT	Same as D14MT, but also tokenizes PART clitics
SCHEME=D34MT	Same as D24MT, but tokenizes all clitics.
SCHEME=S1	Tokenizes only the CONJ, PART, DART and PRON clitics; uses ‘+’ clitic markers, normalizes alefs/yaas, and replaces ‘(’ and ‘)’ characters. Only one WORD form.
SCHEME=S1-HASH	Same as S1, but enclitics are marked with ‘#’
SCHEME=S2	Same as S1, except that it explictly groups the CONJ, PART and DART proclitics; there is no whitespace between the grouped clitics, but the proclitic marker ‘+’ is still present to distinguish them.
SCHEME=S2-HASH	Same as S2, but uses ‘#’ as an enclitic marker
SCHEME=DIAC	A single form consisting of the original word (the surface form), stripped of diacritics, with no tokenization.

Table 9: TOKAN_SCHEME Aliases

Table 9 shows the current aliases defined in TOKAN. All schemes with multiple forms use the middle dot character '·' as the form delimiter (FDELIM) by default.

By way of example, the SCHEME=D3 alias is equivalent to:

```
::SPLIT QUES CONJ PART ART REST PRON ::FORM0 WORD \\
  PROCMARK:PLUS ENCMARK:PLUS NORM:A NORM:Y ESC:PAREN
```

while the SCHEME=S2 alias is equivalent to:

```
::SPLIT [+ CONJ PART DART +] REST PRON ::FORM0 WORD \\
  PROCMARK:PLUS ENCMARK:PLUS NORM:A NORM:Y ESC:PAREN TDELIM:
```

and the SCHEME=ATB4MT alias is equivalent to (all on one line):

```
::SPLIT QUES CONJ PART NART REST PRON FDELIM:·\\
  ::FORM0 WORD PROCMARK:PLUS ENCMARK:PLUS ESC:PAREN \\
  ::FORM1 COPY0 NORM:A NORM:Y \\
  ::FORM2 LEXEME PROCMARK:PLUS ENCMARK:PLUS ESC:PAREN DIAC \\
  ::FORM3 POS:CATIB \\
  ::FORM4 POS:PENN \\
  ::FORM5 POS:BW
```

6.5 TOKAN Output Format

TOKAN output files are arranged with one sentence per line. If SENT_ID is used in the TOKAN_SCHEME and were included in the MADA input file, the IDs will appear as the first word of each sentence, followed by a space.

Figure 2 shows the resulting output of TOKAN for several schemes for a single input sentence. The schemes used are all aliases. ATB and D3 are two commonly used tokenization schemes. ATB4MT is a multi-form (multi-tier) tokenization scheme developed at CCLS as a means of carrying word, lemma and POS tag information into separate parsing and MT systems. DIAC is just an extraction of the diacritized forms of the words with no further tokenization. Note that the final example (DIAC) makes two errors in the case-markers of the second and third word; case-marking diacritics are famously difficult to predict correctly from morphology alone.

Original Input:

```
   SENTENCE_ID_1 wylEb Alfryq Alswry Dd nZyrh AlSrby .


TOKAN_SCHEME = SCHEME=ATB SENT_ID
   SENTENCE_ID_1 w+ ylEb Alfryq Alswry Dd nZyr +h AlSrby .

TOKAN_SCHEME = SCHEME=D3 SENT_ID
   SENTENCE_ID_1 w+ ylEb Al+ fryq Al+ swry Dd nZyr +h Al+ Srby .

TOKAN_SCHEME = SCHEME=ATB4MT SENT_ID
   SENTENCE_ID_1 w+·w+·wa+·PRT·CC·CONJ \\
    ylEb·ylEb·laEib·VRB·VBP·IV3MS+IV+IVSUFF_MOOD:I \\
    Alfryq·Alfryq·fariyq·NOM·DT+NN·DET+NOUN+CASE_DEF_GEN \\
    Alswry·Alswry·suwriy~·NOM·DT+JJ·DET+ADJ+CASE_DEF_GEN \\
    Dd·Dd·Did~·NOM·NN·NOUN+CASE_DEF_ACC \\
    nZyr·nZyr·naZiyr·NOM·NN·NOUN+CASE_DEF_GEN \\
    +h·+h·+hu·NOM·PRP$·POSS_PRON_3MS \\
    AlSrby·AlSrby·Sirobiy~·NOM·DT+JJ·DET+ADJ+CASE_DEF_GEN \\
    .·.·.·PNX·PUNC·PUNC

TOKAN_SCHEME = SCHEME=DIAC SENT_ID
   SENTENCE_ID_1 wayaloEabu Alfariyqi Als~uwriy~i Did~a \\
    naZiyrihi AlS~irobiy~i .
```

Figure 2: TOKAN Output for several schemes. The lines have been wrapped for readability.

7 Other Utilities

Included in the MADAHOME directory is a common-tasks/ subdirectory, which contains a number of utility scripts that users may find valuable. Most of these take a MADA output file as input and perform some kind of data extraction. These scripts and their usage are described in brief below. Except for the first two, all the scripts will output detailed usage information if called with no arguments.

7.1 clean-ut8.pl

```
cat file.utf8 | perl clean-utf8.pl clean-utf8-MAP >
    file.utf8.clean
```

This script was developed to remove rare and problematic UTF8 characters from a UTF8 encoded file. For example, the script normalizes different forms of quotation marks and whitespace, while deleting non-Arabic, non-Latin characters. This functionality is built into the MADA preprocessor, so most users will not have a need to call this script directly. The associated clean-utf8-MAP file describes how each UTF8 character is mapped; since it is used by the preprocessor, this file should not be altered by users.

7.2 tagEnglish.pl

```
cat file.utf8 | perl tagEnglish.pl > file.utf8.tagged
```

This script will go through a file and, for every word containing any ASCII letters (a-z,A-Z), it will prepend a "@@LAT@@" prefix. When run on a UTF8 encoded file, this effectively identifies Latin words in the text. The MADA preprocessor has this functionality built into it, so most users will not need to call this script directly.

7.3 extractFeatureIntoSentenceFormat.pl

```
perl extractFeatureIntoSentenceFormat.pl \\
     file=<input.mada> feat=<MADA feature to extract> \\
     [includeword] [normdigit] [normalefyaa] [sentids] \\
     >  file.feat-sent
```

This script will read a MADA output file and extract from the MADA '*' choices the value of a given feature for every word. It will then print (to standard output) the value of that feature for every word in a sentence-like format (one-sentence-per-line). In this way, users can create a file that is the same as the input MADA file except, for example, every word is replaced by its lemma (or its POS, or its gloss, or its gender, etc.). This script is very useful for generating input for SRI's ngram utilities; i.e., when you want to create N-gram models of MADA features. Valid options for the feat argument are any one of the following:

```
word normword asp bw cas diac enc0 gen gloss
lex mod num per pos prc0 prc1 prc2 prc3 stt
vox normlex normlexeme noanalysis
```

Most of these are identical to the features described in Section 5. normword is the word form with alef/yaa/digit normalization. normlex and normlexeme are identical, and produce the lemma forms without the "_<number>" tags that XAMA includes. noanalysis produces "YES" if MADA found an analysis for the word, and "NO" otherwise.

Optional arguments for the script are:

- includeword – if present, the script will, for every word, output word:feature instead of just feature
- normdigit – If present, all digits in the output will be normalized to '8'
- normalefyaa – If present, and if the feature is lex or diac, all alefs and yaas in the output will be normalized to 'A' and 'y.' Does not affect the other feature choices.
- sentids – If present, the script will place the SENTENCE_ID comments of the input MADA file as the first word of each line in the output (unaffected by the above options). Does nothing if the input MADA file does not include sentence ids.

When the input MADA file has no analysis for a given word, the required feature value is given a default value equal to the most common value for that feature (or the word itself if the required feature was lex, diac, or gloss). Any feature which MADA is not familiar with will get a value of "UNK".

7.4 extractFeaturesIntoColumns.pl

```
perl extractFeaturesIntoColumns.pl file=<input.mada> \\
     feats=<comma-separated list of MADA feats> > feats
```

This script is similar to `extractFeatureIntoSentenceFormat.pl`, except that it takes in comma-separated (with no whitespace) list of features and displays them in tab-separated column format. This script allows users to extract only the features they are interested in from the MADA output, and puts them into an easy-to-parse format. A feature can be specified more than once if desired. The original word is always the first column, and the first line will contain column headers. Sentence breaks are indicated with blank lines. The result is written to standard output. The list of possible features is the same as in `extractFeatureIntoSentenceFormat.pl`.

7.5 extractFeatureValueList.pl

```
perl extractFeatureValueList.pl file=<input.mada>  \\
     > file.feature-value-list
```

This script reads a MADA file and writes (to standard output) a list of the closed-class MADA features and counts of the different feature values it encounters. The script examines all the analyses in the file, not just the MADA selection (the '*' analyses). Output is formated as "`feature value:counts`", like so:

```
...
num d:628 na:1861 p:1220 s:21820
per 1:1210 2:664 3:2219 na:21436
...
```

7.6 extractSentenceFormFromMADAFile.pl

```
perl extractSentenceFormFromMADAFile.pl file=<input.mada> \\
     > file.bw.sent
```

This is just a simple script that will extract the input words from a MADA file and print them (to standard output) in one-sentence-per-line format. This effectively reproduces the input file to MADA (after pre-processing), and is handy if that file is needed but has been removed unintentionally.

Recommended Readings

- Roth, Ryan, Owen Rambow, Nizar Habash, Mona Diab and Cynthia Rudin. 2008. Arabic Morphological Tagging, Diacritization, and Lemmatization Using Lexeme Models and Feature Ranking. In Proceedings of the Conference of American Association for Computational Linguistics (ACL08). [About MADA]

- Habash, Nizar. "Arabic Morphological Representations for Machine Translation." Book Chapter. In Arabic Computational Morphology: Knowledge-based and Empirical Methods. Editors Antal van den Bosch and Abdelhadi Soudi. Kluwer/Springer Publications, 2007. [About TOKAN]

- Habash, Nizar and Owen Rambow. 2007. Arabic Diacritization through Full Morphological Tagging. In Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT 2007); Companion Volume, Short Papers. [About MADA]

- Habash, Nizar and Owen Rambow. Arabic Tokenization, Morphological Analysis, and Part-of-Speech Tagging in One Fell Swoop. In Proceedings of the Conference of American Association for Computational Linguistics (ACL05). [About MADA]

- Habash, Nizar. Large Scale Lexeme Based Arabic Morphological Generation. In Proceedings of Traitement Automatique du Langage Naturel (TALN-04). Fez, Morocco, 2004. [About MADA/TOKAN Component]

- Diab, Mona, Mahmoud Ghoneim and Nizar Habash. Arabic Diacritization in the Context of Statistical Machine Translation, In Proceedings of the Machine Translation Summit (MT-Summit), Copenhagen, Denmark, 2007. [Uses MADA+TOKAN]

- Elming, Jakob and Nizar Habash. Combination of Statistical Word Alignments Based on Multiple Preprocessing Schemes, In Proceedings of the North American chapter of the Association for Computational Linguistics (NAACL), Rochester, New York, 2007. [Uses MADA+TOKAN]

- Habash, Nizar and Fatiha Sadat. Arabic Preprocessing Schemes for Statistical Machine Translation, In Proceedings of the North American Chapter of

the Association for Computational Linguistics (NAACL), New York, 2006. [Uses MADA+TOKAN]

- Sadat, Fatiha and Nizar Habash. Combination of Preprocessing Schemes for Statistical MT. In Proceedings of COLING-ACL, Sydney, Australia, 2006. [Uses MADA+TOKAN]

Acknowledgments

This work has been supported, in part, by Army Research Lab Cooperative Agreement DAAD190320020, NSF CISE Research Infrastructure Award EIA0130422, Office of Naval Research MURI Contract FCPO.810548265, NSF Award 0329163 and Defense Advanced Research Projects Agency Contract No. HR0011-06-C-0023.