An Arabic Morphological Analyzer and Part-Of-Speech Tagger

قُطُوْف

Website: htttp://Qutuf.com

A Thesis Presented to the Faculty
of Informatics Engineering,
Arab International University, Damascus, Syria

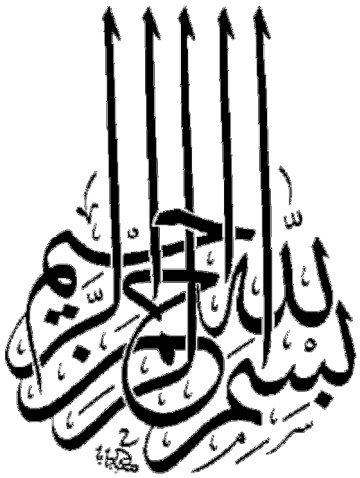
In Partial Fulfillment
of the Requirements for the degree
of Informatics Engineering

By:

Mohammad Altabbaa
Ammar Al-Zaraee
Mohammad Arif Shukairy

Under the supervision of:
Dr. Nada Ghneim
Eng. Amera Espel

July 2010


بسم الله الرحمن الرحيم


وقل اعملوا
فسيرى الله عملكم ورسوله والمؤمنون

CERTIFICATION OF APPROVAL

An Arabic Morphological Analyzer and Part-Of-Speech Tagger

Qutuf

by

Dr. Major Advisor	Date
Dr. Nada Ghneim	

Eng. Major Advisor	Date
Eng. Amera Espel	

Dedication

To who created us in complete creation and gave us belief & patience to accomplish this project... To whom we live to worship him...

Our god "Allah".

To the one who enlightened our lives with the light of knowledge... To the inspirer of the mankind....

Our Prophet Mohammad

When I am crying they are my eyes. When I am smiling they are my lips.

They are the source of love, happiness and faith for me in this life.

They went through a lot to make me what I am.

With all my love.

Our fathers *Our mothers*

Through whole my life they share with me, each minute of sadness and happiness.

Our brothers *Our sisters*

To who worked so hard to be the best useful guides to us... to the supervisors who were so generous...

Dr. Nada Ghneim *Eng .Amera Espel*

Acknowledgements

We would like to give our thanks to everyone who helped us in this project:

Dr. Ammar Al-Salka – Informal supervisor before the official start of the project course, previous lecturer at Arab International University, head of Software Engineering Department at Yarmouk University.

Mrs. Majeda AlShaboun – Arabic Linguistic Expert.

Mr. Abdulrahman Rabee – Arabic Linguistic Expert.

Dr. Taha Zrouki - Arabic language processing researcher, participates in the online discussion group of this project, Algeria Republic.

Mrs. Ola Altabba – English literature specialist, helped in enhancing the language of the report.

Eng. Waseem Kassomeh – University colleague, helped in editing some parts of the report.

Dr. Steve Taylor - Arabic language researcher, discussed ideas for future works.

Eng. Khaled Horani – University colleague, helped in brain storming at the start of the project.

Eng. Abdulatif hajiali – University colleague, helped in brain storming at the start of the project.

Eng. Hani Almousli – University colleague, helped in brain storming at the start of the project.

Finally, special thanks to:

Dr. Nada Ghneim – PhD of Science of Languages**,** Higher Institute of Applied Sciences and Technology and Damascus University.

Eng. Amera Espel – Faculty of Informatics Engineering AI Department, Arab International University and Damascus University.

Without their continuous help and valuable guidance, this project could not have been completed.

Table of Contents

Dedication 4

Acknowledgements 5

Table of Figures 9

Table of Tables 11

Chapter 1: Introduction 12

1-1- Motivations 13
1-2- Scope 13
1-3- Report Structure 13

Part One: State of the Art 15

Chapter 2: Morphological Analysis 15

2-1- Insight into Arabic Morphology 15
2-1-1- Morphological Processes 15

2-1-2- Examples 16

2-2- Computational Morphology 17
2-2-1- Root-and-Pattern 17

2-2-2- Stem-Based Arabic Lexicon with Grammar and Lexis Specifications 18

2-3- Survey of some applications 19
2-3-1- Khoja Stemmer 19

2-3-2- The Aramorph System (Buckwalter) 20

2-3-3- AlKhalil Morpho Sys 21

Chapter 3: Part-Of-Speech Tagging 27

3-1-1- Rule Based 27

3-1-2- Stochastic 27

3-1-3- Transformation Based Tagging 28

Part Two: Qutuf System 30

Chapter 4: System Overview 30

Chapter 5: Preprocessing 32

5-1- Tokenization 32
5-1-1- Introduction 32

5-1-2- Tokenization Algorithm 33

5-2- Normalization 36

Chapter 6: Morphological Parsing 38

6-1- First Thought 38
6-2- Second Thought 40
6-2-1- Using the Database of AlKhalil 40

6-2-2- Our work on AlKhalil Database 41

6-3- Morphological Analysis Steps 42
6-3-1- Compounding Detection (اكتشاف التراكيب) 42

6-3-2- Isolation (عزل اللواصق) 43
6-3-3- Lookup at Closed Lists 49
6-3-4- Un-diacritized Pattern Matching (مطابقة الوزن) 49
6-3-5- Root Extraction (تجذير) 50
6-3-6- Voweled Patterns List Production and Verification 51
6-3-7- Fill Morphological Attributes and POS Tags 51
6-3-8- Assembling Clitics with Matches 51
6-4- Output Complexity 53
6-4-1- Cliticalization Parsing 53
6-4-2- Particles and Closed Nouns 54
6-4-3- Pattern Matching 54
6-4-4- Overall Output Complexity 55
Chapter 7: Tagging 57
7-1- Our Approach for Ambiguity resolution 57
7-2- Tagging Phases 57
7-2-1- Premature Tagging (التوسيم الخديج) 58
7-2-2- Overdue Tagging (التوسيم المتأخّر) 61
7-3- Arabic Part-Of-Speech Tagset 65
7-3-1- Introduction 65
7-3-2- Suggested POS Tagset 65
7-3-3- POS Slots 66
7-3-4- Example 74
7-3-5- Number of possible tags for Arabic 75
Chapter 8: System Design 76
8-1- Design Pattern 76
8-2- Classes Diagrams 76
Chapter 9: Implementation Issues 85
9-1- Python 85
9-1-1- Why Python 85
9-1-2- XML Processing 87
9-1-3- Web frameworks 88
9-1-4- Windows Applications 88
9-2- Qutuf System Output 89
Part Three: Applications used Qutuf 91
Chapter 10: Morphological Analysis Comparison System 91
Chapter 11: Similarity Based Text Clustering 92
11-1- Introduction: 92

11-2- Algorithm 92
11-3- Features 92
11-4- Implementation 93
11-5- Interface 93
11-6- Conclusion 95
Appendix A – Terminologies 96
Bibliography 98

Table of Figures

Figure 3-1 Rule Based Tagging Example for English 27

Figure 3-2 Basic algorithm of Transformation Based Tagging.....29

Figure 4-1 DFD Level 1 - System Processing Overview.....30

Figure 4-2 System DFD Level 2 31

Figure 5-1 Method for Language Independent Text Tokenization Using Character Categorization algorithm flowchart 35

Figure 6-1 Examples of Ebdal (إبدال) letter changing in Arabic 38

Figure 6-2 Examples of Elal (إعلال) letter changing in Arabic.....39

Figure 6-3 Rule of "الإعلال بالحذف" 40

Figure 6-4 Example of enclitic and proclitic detection and extraction process 43

Figure 6-5 Proclitic automate diagram "Class C" 44

Figure 6-6 proclitics automate diagram "Class V" 45

Figure 6-7 Enclitics automate diagram "Class C".....47

Figure 6-8 Enclitics automate diagram "Class N" 47

Figure 6-9 Enclitics automate diagram "Class V" 48

Figure 6-10 Relations between roots, unvoweled patterns and voweled patterns 50

Figure 6-11 Complexity of Isloation.....53

Figure 6-12 Word segmentation example 53

Figure 6-13 Complexity of Cliticalization Parsing.....54

Figure 6-14 Complexity of Cliticalization Parsing and Particles and Closed Nouns looking up 54

Figure 6-15 Complexity of Un-diacritized Patten Matching and Root Extraction 54

Figure 6-16 Complexity of List Production.....55

Figure 6-17 Complexity of Pattern Matching 55

Figure 6-18 Overall complexity of the output of morphological parsing 55

Figure 7-1 Rule of adverbs (ظروف) followed by verbs (أفعال).....60

Figure 7-2 Rule of adverbs (ظروف) followed by nouns (أسماء).....60

Figure 7-3 Rule of "خلا", "عدا", "حاشا" 61

Figure 7-4 Rule of "يا", "أيها" and "أيتها" 61

Figure 7-5 Tagging Rule of "إلى و إلى أن".....63

Figure 7-6 Tagging Automata of "إلى و إلى أن".....63

Figure 7-7 Simple shallow and fast rules for Overdue tagger 64

Figure 8-1 Model View Controller design pattern 76
Figure 8-2 Class Diagram of Cliticalization 77
Figure 8-3 Class Diagram of Entities.....78
Figure 8-4 Class Diagram of General 79
Figure 8-5 Class Diagram of Normalization.....79
Figure 8-6 Class Diagram of Tokenization 80
Figure 8-7 Class Diagram of Transducers.....81
Figure 8-8 Class Diagram of the Transducers Xml Loader 82
Figure 8-9 Class Diagram of Morphological Analyzer. 82
Figure 8-10 Class Diagram of Lexicon Classes.....83
Figure 8-11 Class Diagram of POS Tags 84
Figure 9-1 System XML output example for word "من" 89
Figure 9-2 System HTML output example for word "من" 90
Figure 11-1 Screen shout of the application: Similarity Based Text Classification....94

Table of Tables

Table 2-1 Examples of surface forms, isolated forms, stems and roots 17

Table 5-1 Characters Categorizes of Language Independent Text Tokenization 34

Table 6-1 Proclitic Automate for "Class N" shown in a table 46

Chapter 1: Introduction

At this report we will introduce the work done for research and development of a system for Arabic Morphological analysis and Part-Of-Speech tagging, called Qutuf "قُطُوْف", which is aimed to be the kernel of a large framework that provides APIs for Arabic language processing. In addition, two independent applications will be presented. The first is to compare the output of any two, or three, morphological analyzers. Basically, the comparison application built to compare the result of Qutuf with AraMorph. The second, similarity based text clustering, will be an example of independent application that used some of the output of Qutuf.

The morphological analysis component at Qutuf uses finite state automates and rules for agreement developed for cliticalization parsing. It also uses an open source database for root extraction, pattern matching, morphological feature and POS assignment and closed nouns after enriching it.

Moreover, at this report, some new concepts has been identified and implemented in Qutuf system. Like First Normalization and Second Normalization text forms at the preprocessing phase and the Premature and Overdue Tagging at the Part-Of-Speech tagging task. Moreover, the POS tagging is designed and implemented as a rule based expert system. A POS tagset, which is built based on a morphological feature tagset, has been designed and used in Qutuf.

1-1- Motivations

Most human knowledge exists in a Natural Language Processing form; neither in relational database nor pictures. For that, we expect that the current century will focus on NLP. For us, more natural language processing means more put-to-use knowledge and more spreading of knowledge.

Unfortunately, little work has been practiced on Arabic. One reason for that is the sophistication of the Arabic language. We are Arabs and we are the only ones who can carry this work out. This enthuse us to work on our holy language.

1-2- Scope

We aim to build a Morphological Analyzer that serves the task of Part-Of-Speech Tagger without being lost into the details of Arabic morphology, and to construct a Part-Of-Speech tagger that assigns POS tags to an input text.

Our view of this project is to start building libraries and a framework to help researcher and developer working on Arabic Language Processing by enabling them to build their applications on top of our framework without going through the complicated details of Arabic. In this project, the framework will contain the basic and important tools. Meanwhile, we aim to expand this framework in the future with more tools and features.

There is no need to say that this project will make life easier for native Arabs or non-Arabic speakers, even we hope so. Precisely, what we will provide is going to serve Search Engines, Database Engines, Information Extraction applications and any other AI application that makes use of Arabic Language Processing.

1-3- Report Structure

The report is composed of three main parts. The first part "State of the Art", presents a survey of the current approaches in morphological analyzers and part of speech taggers.

At chapter 2, we will go through Arabic morphology and get to know some algorithms of computational Arabic morphology and see a survey of some morphological applications. At chapter 3 we will define the tagging task and identify the usual steps for ambiguity resolution.

At the second part, "Qutuf System", our work to develop the system Qutuf is presented.

Chapter 4 will give an overview of Qutuf. Chapter 5 will present text preprocessing steps, where tokenization and normalization are carried out. Chapter 6 goes through our first thought and second thought of performing morphological analysis. You will also see that an open source database of roots and patterns, with their morphological attributes, and closed nouns lists is enriched and used. After that, it depicts, in details, the morphological processing steps. Furthermore, this chapter will give an idea about the algorithm complexity of morphological analysis.

Chapter 7 illustrates the adopted approach for ambiguity resolution which is a rule based expert system. Moreover, this chapter will identify a new concept of tagging in which the tagging task is done at two components: the Premature and Overdue tagging. Furthermore, this chapter will present a Part-Of-Speech tagset, which is based on good morphological features tagset, that is developed and implemented at Qutuf.

Chapter 8 will present the system design, and show that we used MVC (Model View Controller) design pattern and will show the class diagram of Qutuf. Chapter 9 is about implementation issues where we justify the technology used at Qutuf and present the system output. At chapter 10 some future work is presented for Qutuf system.

Finally, at the third part "Applications Used Qutuf", we present in chapter 11 a comparison application, which is used to test the results of Qutuf against AraMorph. In chapter 12, another application for text clustering based on similarity is introduced to show the use of Qutuf output in another application.

Part One: State of the Art

Chapter 2: Morphological Analysis

In linguistics, morphology is the study of the internal structure of words. It is the identification, analysis and description of the structure of morphemes and other units of meaning in a language.

2-1- Insight into Arabic Morphology

Enjoy speaking about the sophistication of the Arabic language will lead us to four morphological processes to be considered. Since we are interested in the Arabic morphology, the following definitions are from the Arabic language point of view.

2-1-1- Morphological Processes

Derivation (اشتقاق): The derivation process produces nous (nous in Arabic includes adverbs, adjectives, pronouns, proper nouns and many others) and verbs from the roots (first stem of verbs). So, the roots, which are verbs consist of three (most cases), four and five letters (rare roots), are the origin of all the Arabic words. In Arabic, the derivation is done by using specific patterns (also called measures "أوزان"). Actually, this is why the Arabic derivation morphology is called Template Morphology or Root-and-Pattern Morphology. However, the number of patterns is about 300; while the number of roots is about 10000. This gives the hypothetical ability to produce 10000 * 300 = 3,000,000 derivatives. Remark that, this is not the count of the Arabic words; since just some patterns are used with each root and derivatives does not include words produced by inflection and cliticization.

Inflection (تصريف): happened by adding some well known affixes (prefixes, suffixes and infixes) in order to give some attributes to the word. For example, adding the suffix "ات" (At) to the noun will make it give the meaning of feminine plural. Another example is adding the prefix "ي" (y) to make the verb in the present form.

Cliticization (إلصاق): the use of Clitics; where Clitic in morphology: is a word which is written or pronounced as part of another word (Microsoft Encarta Dictionary, 2007). This word is an independent unit at the parsing phase but sticks to other word just like an affix. When it comes before the other word it is called proclitics and when it comes after the other word it is called enclitics. For example, in English the word "ve" in "I've" is enclitics.

In Arabic this phenomenal is widely used. Let us take some examples to clarify the use of clitics and distinguishing it form affixes. A good example of the proclitics is the word "**و**" which is the "and" coordinating conjunction that we wrote adjacent to the next word; even though this is not applicable for the word "**أو**" which is the "or" coordinating conjunction that is written separated from the next word. An example of enclitics is the "**ضمائر الرفع المتصلة**" connected nominative pronouns which are considered as nouns like "**واو الجماعة**" in the word "**آمنوا**" in which the connected nominative pronouns here is the subject. Moreover, a word could have both proclitics and enclitics like the proclitics "أ" and the enclitic "**ون**" in the word "**أيدرسون**".

Compounding (تجميع): Some words are formed from a combination of two words. For example, the two words "**صباح مساء**" is actually one word that is an adverb of time. Actually, this phenomenon is not common in Arabic, but still counts.

2-1-2- Examples

To make things clear, the table bellow shows some examples of several words with their surface forms, isolated forms, stems and roots.

	Word Forms			
Category	Surface	Isolated (cliticless)	Stem	Root
Nouns	حسناته	حسنات	حَسَن	حَسُنَ
	الإحسان	إحسان	إحسان	حَسُنَ
	فالتحسينات	تحسينات	تحسين	حَسُنَ
	واستحسانها	استحسان	استحسان	حَسُنَ
Verbs	يستصلحها	يستصلح	استصلح	صلُح
	يحسِنون	يحسِن	أحْسَنَ	أحْسَنَ
	فتستحسنونها	تستحسن	استحسن	حَسُنَ

Table 2-1 Examples of surface forms, isolated forms, stems and roots

2-2- Computational Morphology

A survey of some methods used for knowledge based morphological analysis will be presented. Actually, we are not interested in empirical methods because it needs a huge amount of annotated data that is not freely available for Arabic.

2-2-1- Root-and-Pattern

This type of computation adheres to the nature of the Arabic morphology since all derivations are done according to (أوزان) Measures (also called templates or patterns). However, this is used in computation first by McCarthy (1979, 1981). Where, in his proposal, the stems are formed by a derivation from vowels and roots morphemes. In which the two (vowels and root morphemes) are arranged according to canonical patterns. His method involves three steps: (1) the root letters are combined with the pattern. (2) The result is concatenated with the required affixes. (3) Orthographic rules (spelling rules) are applied to get the last result. The Following example explains the steps; showing the levels of derivations of (قال) *qaAla* "he said"

Abstract level: [qwl & CaCuC] + a
Intermediate level: *qawul+a*
Fully voweled: *qaAla*

This method is said to be implemented by bi-directional transducers. So, it is supposed to be used for generation as well as for analysis by inversing the steps. But,

these transducers could never be efficiently developed for analysis. Since most of the cases involve knowing all diacritics of the letters of the word. You will find more about this in section "6-1- First Thought".

2-2-2- Stem-Based Arabic Lexicon with Grammar and Lexis Specifications

Because Arabic root or combination root with a pattern are abstraction, not actual part of the speech. Grammar-lexis relations for Arabic NLP applications can only be associated to stems, since stems belong to a syntactic category.

This approach based on stem lexical database and entries associated with grammar and lexis. Two experiences support these points: DINAR.1 which is Arabic lexical resources, and SYSTRAN which is Arabic-English translator (fully automatic transfer system).

Grammar-lexis relations recall the structure of the word-form in Arabic, two fields can be distinguished, first one is the lexical stem or nucleus formative (NF), second is the extension formatives (EF) which bound grammatical morphemes.

Structure of the word-form:

Beside the nucleus the word could consist of proclitics (PCL), prefix (PRF), suffixes (SUF), enclitics (ECL). The nucleus or the stem can be divided into two type, pattern and 3 or 4 consonants, and nouns not bound to pattern such as "**إسماعيل**" and "**فيزياء**".

Word-Formatives Grammar (WFG):

NF ←→ EF. Ex. PCL → noun.
EF ←→ EF. Ex. PCL → SUF.
NFa ←→ NFb. Derivation.

Entries with finite set of morphosyntactic w-specifires can guarantee a complete coverage of data within the boundaries of the word-form.

Stem-Based approach reduces the complexity of Arabic word structure, and eliminates large number of lexical gaps. It's also associates morphological syntactic

and semantic feature with each entry. But in the other hand, it is exhaustive to coverage data in the lexical resource DINAR.1.

Applications adopted this approach:

XEROX uses stem-based with root and patterns.

SAMIA uses stem-based lexical resources, including root-pattern and grammar-lexis information.

2-3- Survey of some applications

In this section we will present some of the systems known in the Morphological Analysis domain. First, we will present the Khoja stemmer, then the Aramorph system, and finally, we will present AlKhalil Morpho System.

2-3-1- Khoja Stemmer

There are 3 well known Arabic stemmers: Khoja (Khoja S. , 1999), ISRI (Taghva, Elkhoury, & Coombs, 2001) and Light Stemming (Larkey, Ballesteros, & Connell, 2002). Since, they are all rule based, we will introduce one of them, that is Khoja stemmer, as an example of rule based stemming.

Khoja stemmer is developed at late 1990s. For that most stemmers compare their work to it. Khoja Stemmer is a heavy stemmer. It gives the roots with accuracy of about 77.01% when tested on newspaFcomper gold standard (Majdi Sawalha, 2008). Khoja like all most, if not all, stemmers is a rule based.

How does Khoja stemmer works? (Khoja S. , 1999)

- The first thing the stemmer does is removing the longest suffix and the longest prefix.
- It then matches the remaining word with the verbal and noun patterns, to extract the root.

The stemmer also handles the situation where there are weak letters (i.e. alif, waw or yaa') and if a root letter is deleted during derivation; this is especially true of roots that have duplicate letters. Also, if a root contains a hamza, this hamza could change form during derivation. The stemmer detects this, and returns the original form of this hamza.

2-3-2- The Aramorph System (Buckwalter)

Buckwalter Arabic morphological analyzer is one of the most well known Arabic morphological analyzing and POS tagging systems. It is developed by LCD (Linguistic Data Consortium) in both Perl and Java. However, we will just speak about Aramorph vesion 1; which is freely available with its source code.

The components of Buckwalter Arabic Morphological Analyzer are the morphology analysis algorithm, and the data that primarily consists of three Arabic/English lexicon files: dictPrefixes (السوابق) contain 299 entries, dictSuffixes (اللواحق) contain 618 entries, dictStems (الجذوع) contain 82,158 entries representing 38,600 lemmas.

The lexicons are supplemented by three morphological compatibility tables (جداول التوافق) used for controlling prefix-stem combinations (1,648 entries), stem-suffix combinations (1,285 entries), and prefix-suffix combinations (598 entries). The algorithm of morphology analysis and POS tagging is imbedded in the code. It uses the three lexicon files and the three compatibility tables in order to perform morphological analysis and tagging of Arabic words. The used algorithm has some functions to make tokenization, word segmentation, dictionary lookup, compatibility check, analysis report and second lookup.

Although that the AraMorph is widely famous as best analyzer for Arabic language, it has many weak points. For example, the system doesn't support the generation process, also doesn't depend on rule-based, it just manually inserted the words with all possible derivations, that makes the runtime processing take a long time. It have a spelling problem where it convert between Aleph and Hamza (الألف والهمزة) for no good reason, and have a problem when dealing with complex words like (Hadramout حضرموت), and dealing with Lam latter in Arabic (لإنفاق–لاستعمال).

Some of the weakness points of the AraMorph system:

- System does not support the structure of the rules, forms was entered manually and all the words listed for each word all possible derivations, so increasing the processing time.
- Doesn't show the patterns, so from difficult to develop syntactical analyzer in next stage.
- Does not support the generation process.
- Does not allow the identification of words (nouns, verbs) that have the following format (أأكل – محمد)
- Has a problem in dealing with acts in the past tense, where the pronoun is absent, such as (اضرب – حاول)
- He has a problem in dealing with the past tense of the passive voice, for example (استُعمِلَ)
- Doesn't cover all morphological rules in Arabic words like: (المهموز والناقص والمثال), some of outputs return wrong, for example (رِ من رأى) (فِ من وفى) (صف من وصف).
- Has a problem in dealing with the character that precedes the harvest names, such as (لاحزاب)
- Cannot handle the expression, which consists of compound words, such as (أبي حنيفة)
- The process of spelling it improperly, such as (فاشل – فأشل ، واقف – وأقف)
- Doesn't make a use of diacritics, which one usually insertes to reduce the ambiguity and reduce the number of outputs of the morphological analyzer. The program removes these diacritics away without making any use of them.

2-3-3- AlKhalil Morpho Sys

AlKhalil (AlKhalil Morpho Sys) could be considered as the best Arabic morphological system. Actually, AlKhalil won the first position, among 13 Arabic morphological systems around the world, at a competition held by (المنظمة العربية للتربية

والثقافة والعلوم) The Arab League Educational, Cultural and Scientific Organization (ALECSO) (برنامح الخليل الصرفي, 2010). Actually, after analyzing it we found it deserve this. For this and that, we had put a special effort on understanding it and used its open source database as part of our database, for the morphological analysis system component as you can see next at section "6-2-1- Using the Database of AlKhalil".

Before making our review, we would like to point out that, it was not officially available till late in our project development. Actually, we started analyzing the database in the beta version in which the documentation was not available. And even when the documentation was available at version 1.0, it was very poor and insufficient. So, analyzing and using its database happened with great difficulties and testing its execution was also very hard.

It is also mandatory to say that, we are not trying to criticize it badly. It is clear that this program was made with great effort and it is still to be considered one of the best, if not the best, of its peers. And we hope that this will help in improving.

Although, AlKhalil is a very good application, it has some shortages. Besides that, it has some bugs and some errors at its database. This is not strange since Arabic is morphologically sophisticated language and this is the version 1.0 of the program. However, because a big effort has been put on understanding its behavior and database, some problems has been detected and some enhancement has been made. Some of the issues, those which are related to database shortages are addressed next at section "6-2-2- Our work on AlKhalil Database". Other issues which are related to database errors and program bugs along with some limitations have been addressed bellow.

a) Database Errors and Missed Values:

- Some diacritics, especially the last one and especially (السكون) " ْ " is not written at many words at the list of tool words and at some words at the voweled patterns files.

As example, the word "غَيْر" does not have the diacritic " ُ " at the last letter. Another example is "فِي" which does not have the diacritic (السكون) " ْ " at its last letter. Actually, because a bug exists at processing, this makes a real problem if the missed diacritic is provided by the user. In that case, the program will not be able to detect the entered word as a tool word. For example, if the user enters the word "فيْ", the program will not detect it as proposition and will give the message " لا توجد نتائج لتحليل هذه الكلمة" which means "The is no results for this word".

- Noise at data of roots and patterns.

Actually, some noise has been detected and probably there will be more if an investigation process is held. However, the first detected noise is the nominal pattern with id=2345 which has the voweled form "فًعَلاء" where there is " ً " at the first letter instead of " ُ ". After some investigation, it comes out that this pattern is just used one time, that is for the root "حكَم" to give "حُكَمَاء" and it must be replaced with the pattern with id=3701 which has the voweled form "فُعَلاء".

The second detected noise is that some attributes contain wrong values. For example, the "ncg" attribute (which hold information for number, case and gender) is empty for two patterns at the file that contain nominal patterns with length 3. Moreover, the value of the "ncg" is assigned to the "cas" (which supposed to hold information about definiteness) and the value of the "cas" is assigned to the "type" (which supposed to hold the part of speech subclass). Actually, at the beta version there were 6 more outliers like those, but they are removed and these two still exist.

Another example of the second type of noise is that the value of the attribute "type" is set to undefined value. However, these errors are reported as follows:

```
    Error at file [VoweledNominalPattern3.xml], for pattern
with [id = 4083]:
        the [type] attribute's value [نك]! ,
        the [cas] attribute's value [8]!
        and the [ncg] attribute is empty!
```

```
    Correct values: (type = صه, cas = نك, ncg = 8)

    Error at file [VoweledNominalPattern3.xml], for pattern
with [id = 4086]:
        the [type] attribute's value [نك]! ,
        the [cas] attribute's value [2]!
        and the [ncg] attribute is empty!
    Correct values: (type = صه, cas = نك, ncg = 2)

    Error at file [VoweledNominalPattern5.xml], for pattern
with [id = 3987]:
        the [type] attribute's value [ص]! Which is undefined!
        the [cas] attribute's value [نك]
        and [ncg] attribute's value [7]
    Correct values: (type = صأ, cas = نك, ncg = 7)
```

- Missed values at the file of proclitics (which is called prefixes.xml).

For the proclitics "أَوَكَال" and "أَفَكَال" the value of the type is " همزة الاستفهام+حرف العطف+ال التعريف" where the type of the preposition letter "كَ" is missed. For that, the type must be: "همزة الاستفهام+حرف العطف+حرف الجر+ال التعريف".

b) Execution Bugs:

- If the user enters the word fully voweled, the program may not give all the available results; even if the data was provided appropriately.

For the fully voweled pattern and the proper names, where the last diacritic must be relevant to the (الحالة الإعرابية) case or mood, the program does not analyze the word appropriately if the user provides a diacritic to the last letter of the word. For example, if the user enters "أحمدُ" the program will not identify the word as (اسم علم) proper noun.

- Unable to give the associated results to (حرف) a particle that changes the case of a noun in the situation when the word can have any (حالة إعرابية) case.

For example, the program does not give any result for "كأحمد" of "بأحمد", even though it gives result for "أحمد" as a proper noun (اسم علم) and the proclitics "ب" and "ك" are available at the list of proclitics.

- It is not able to analyze a word that starts with "إ" if its root starts with (همزة) Hamza.

For example, the program will give no results for "إبرة" where its root is "ء ب ر". After some investigation, it turns out that this could be fixed by changing the code of file: "analyze\Analyzer.java" at line 626 which was:

```
return root.replaceAll("[أئـؤ]" ،"ء");
```

To be:

```
return root.replaceAll("[إأئـؤ]" ،"ء");
```

However, after this change it could give 10 possible analysis results for the word "إبرة".

- It does not accept a verb with imperative aspect to have an enclitic of "ضمير المتكلم" with "ضمائر الغائب" or "ضمير المتكلمين" with or without "ضمائر الغائب". In contrast it accepts an imperative verb with enclitics of "ضمائر المخاطب" if it is not followed by "ضمائر الغائب".

For example, it does not give any result the word "ارْحَمْنَاْ". But it gives two wrong results for "بادِرْك" as an imperative verb "بادِرْ" followed by second person pronoun "كِ" and "كَ".

Actually, this could be fixed by changing the code of file: "analyze\Analyzer.java" at line 2345:

```
    if(suffcalss.equals("V2") &&
vnp.getType().indexOf("أ") != -1){
```

To be:

```
    if(suffcalss.equals("C2") &&
vnp.getType().indexOf("أ") != -1){
```

- It assumes that no pronouns (ضمائر) could follow passive verbs (الأفعال المبنية للمجهول).

For example, it does not give result for "أُطْعِمَهُ" as an inactive verb, while it gives a result for "أُطْعِمَ" as an inactive verb.

c) Limitations

- It does not provide POS tags in a good reusable format.
- AlKhalil does not differentiate between clitics and affixes. Even though it basically identifies proclitics and enclitics, it provides some information about some prefixes and suffixes. However, it just calls them either prefix or suffixes.

It just give a much like a plain text explaining the morphological attributes of the provided word.

- It does not apply a good tokenization algorithm.

Chapter 3: Part-Of-Speech Tagging

Tagging is assigning Part-of-Speech tag to each and every word in the processed text. This stage also involves handling ambiguity if more than one tag could be assigned to a word. This is the general case for Arabic words. However, you will see next how ambiguity is usually resolved.

3-1-1- Rule Based

Some rules are defined by linguistics, or automatically generated from tagged corpora, to state that for example, "there is no two adjacent verbs could be used in some language". A fair example about this is like this rule for English (Jurafsky & Martin, 2009):

Adverbial-that rule:
- **Given input**: “that”
- **if**
  - (+1 A/ADV/QUANT); /* if next word is adj, adverb, or quantifier */
  - (+2 SENT-LIM); /* and following which is a sentence boundary, */
  - (NOT -1 SVOC/A); /* and the previous word is not a verb like */
  - /* ‘consider’ which allows adjs as object complements */
- **then** eliminate non-ADV tags
- **else** eliminate ADV tag

Figure 3-1 Rule Based Tagging Example for English

3-1-2- Stochastic

A problem that often shows up in purely rule-based accounts is that of the inherent incompleteness of the symbolic resources, due to the difficulty of knowledge acquisition. Many natural language processing systems already have means to deal with incomplete symbolic resources. Stochastic part of speech taggers typically assume some probability distribution on possible categories of unknown words (where of course the open word classes are regarded as much more likely than the functional categories).

This constitutes an example of a system that can cope with the absence of symbolic knowledge by using a quantitative model of its own imperfection. One can see this as a kind of meta-knowledge, which allows the system to keep a good balance in the relative weighting of its symbolic and stochastic knowledge sources. It is

interesting to note that we can often derive from text corpora not only the raw form of lexical information, such as a word list with frequencies, but additionally an estimate of the completeness of this resource, in a form of probability estimates that the next word from the same source would already be known.

So, stochastic POS taggers are about computing the most likely tag sequence and picking the tags that materialize that. Many algorithms developed and used for this; like using Viterbi (Viterbi, 1967) algorithm for HHM (Hidden Markov Model) tagger. However, building a statistical tagging model needs tagged a corpus which is not available for free for Arabic.

3-1-3- Transformation Based Tagging

TBL (Transformation Based Tagging) is an instance of the TBL approach to machine learning. It draws inspiration from both the rule-based and stochastic taggers. It is an emerging technique with a variety of potential applications within textual data mining. TBL has been utilized for tasks such as part-of-speech tagging, dialogue act tagging, and sentence boundary disambiguation, to name a few.

TBL performs admirably in these tasks since they rely on the contextual information within textual corpora. The core functionality of TBL is a three-step process composed of an initial state annotator, templates, and a scoring function.

The initial state annotator begins by labeling un-annotated input (e.g., postings) with tags based on simple heuristics. Using a scoring function, the annotated input is then compared to a ‘ground truth’ consisting of the same text with the correct labels. TBL automatically generates transformation rules that rewrite labels in an attempt to reduce the error in the scoring function.

Algorithm 1 Basic transformation-based tagging algorithm.

tb-tagging(input: sequences S; true sequence tags L)

1 $\hat{L} := \textit{initial-tags}(S, L)$
2 initialize $R = []$
3 **repeat**
4 $r := \textit{find-best-rule}(S, \hat{L}, L)$
5 update $\hat{L} := \textit{apply-rule}(\hat{L}, r)$
6 update $R := \textit{append}(R, r)$
7 **until** (no improvement)
8 **return** R

Figure 3-2 Basic algorithm of Transformation Based Tagging

Part Two: Qutuf System

Chapter 4: System Overview

The system consists of three main components:

- Preprocessing
- Tagging
- Morphological Parsing

The input of the system is a text that could be provided by from user interface or independent application (through calling our system API). The output is the morphological attributes and the POS tag of each word of the input text.

This has been shown in the next figure.

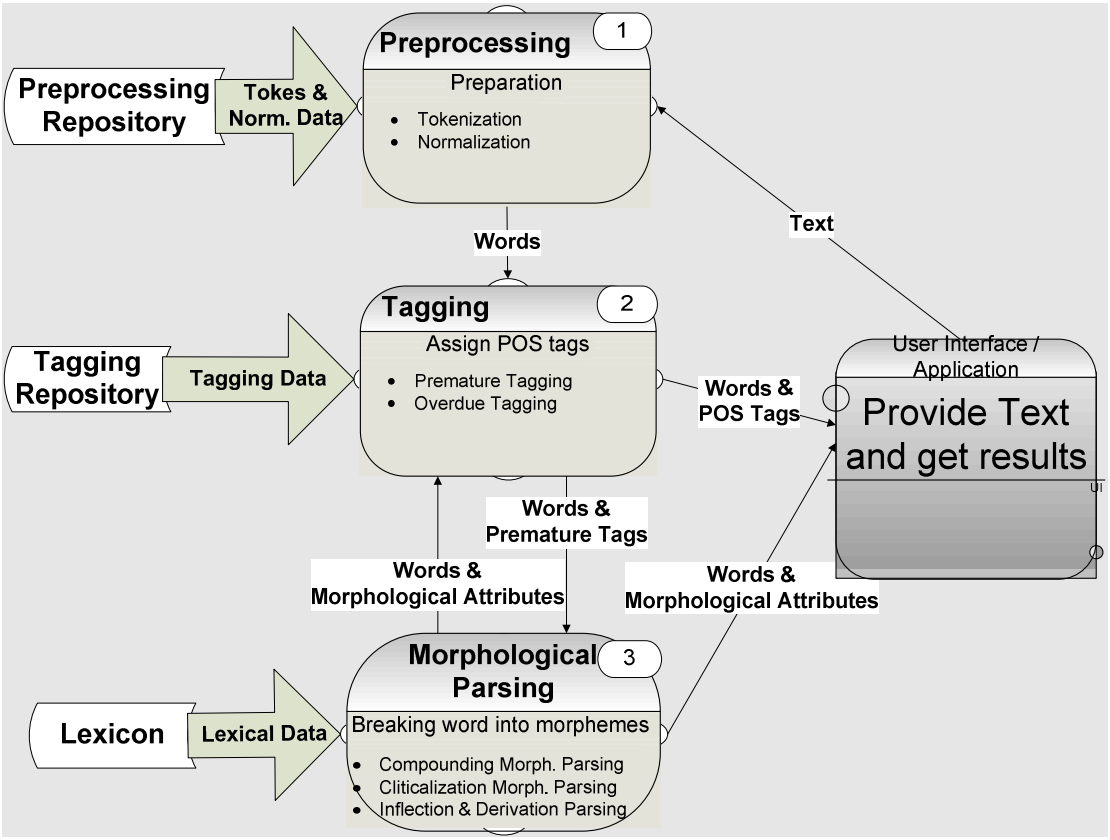


Figure 4-1 DFD Level 1 - System Processing Overview

Each one of those components will be described in detail in the following sections. However, you can see the level 2 DFD (data flow diagram) in the following figure.

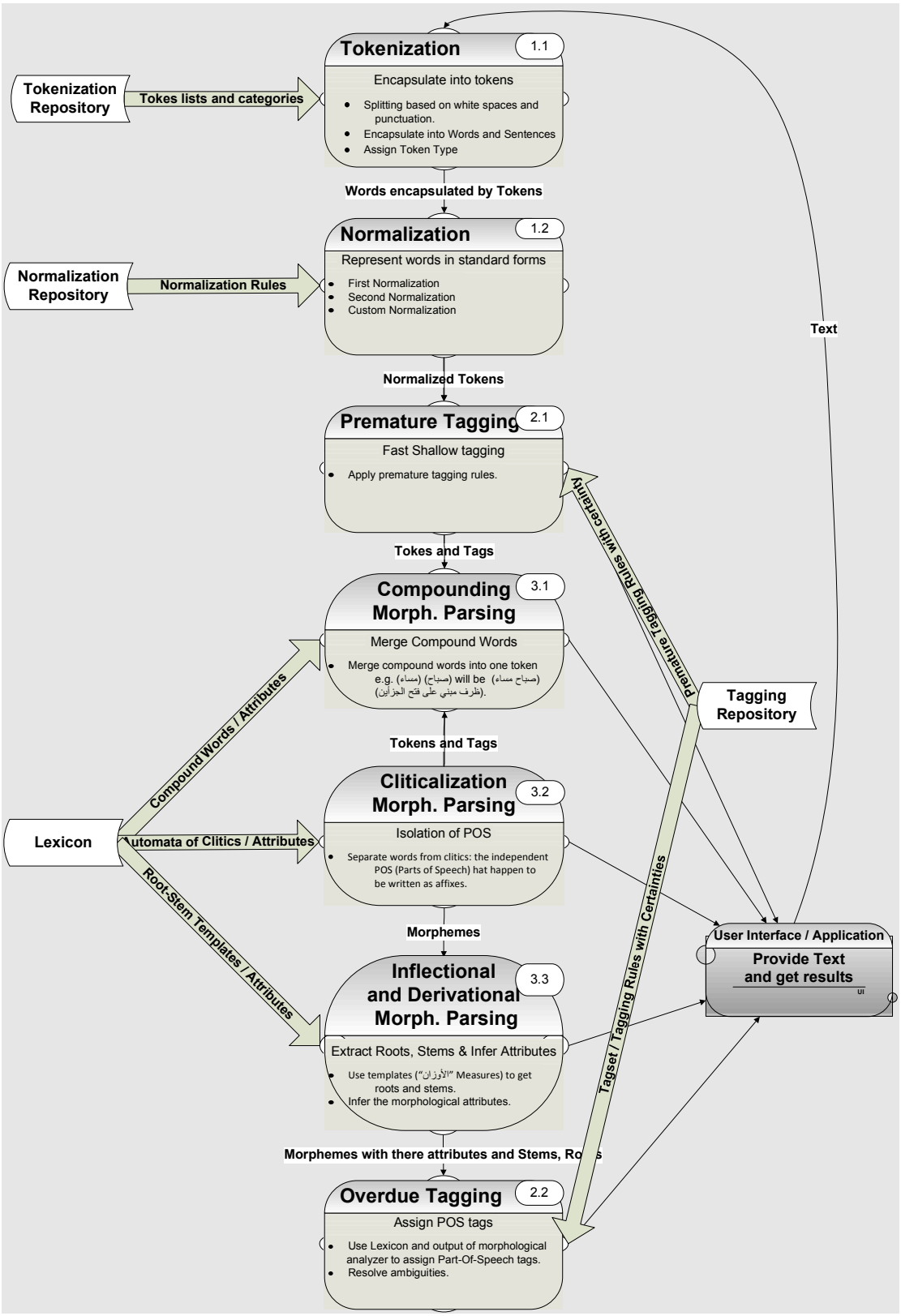


Figure 4-2 System DFD Level 2

Chapter 5: Preprocessing

5-1- Tokenization

5-1-1- Introduction

The stream of characters in a natural language text must be broken up into distinct meaningful units (or tokens) before any language processing, beyond character level, can be performed.

If languages were perfectly punctuated, this would be a trivial thing to do, a simple program could separate the text into word and punctuation tokens simply by breaking it up at white-space and punctuation marks. But real languages are not perfectly punctuated, and the situation is always more complicated. Even in a well (but not perfectly) punctuated language like Arabic, there are cases where the correct tokenization cannot be determined simply by knowing the classification of individual characters, and even cases where several distinct tokenizations are possible.

Tokenization is a non-trivial problem as it is “closely related to the morphological analysis”, this is even more the case with languages with rich and complex morphology such as Arabic. The function of a tokenizer is to split a running text into tokens, so that they can be fed into a morphological transducer or POS tagger for further processing. The tokenizer is responsible for defining word boundaries, demarcating clitics, multiword expressions, abbreviations and numbers.

A token is the minimal syntactic unit; it can be a word, a part of a word (or a clitic), a multiword expression, or a punctuation mark. A tokenizer needs to know a list of all word boundaries, such as white spaces and punctuation marks, and also information about the token boundaries inside words when a word is composed of a stem and clitics (Attia, 2009).

However, our research full form words, i.e. stems with or without clitics, as well as numbers will be termed main tokens. All main tokens are delimited either by a white space or a punctuation mark. Full form words can then be divided into sub-tokens where clitics and stems are separated.

In Arabic tokenization, we apply "Language Independent Text Tokenization Using Character Categorization algorithm" (Fagan, 1991) which has best performance

and determine words from paragraph. The tokenization percentage of this algorithm is about 94.4% for a huge corpus. The algorithm processes all irregular states like the decimal numbers, mathematics operators and all symbol characters.

5-1-2- Tokenization Algorithm

The idea of the implemented "Language Independent Text Tokenization Using Character Categorization" algorithm is

1- Classify the characters into three classes which are:

Letters, separators and ambiguous characters. Ambiguous character contain symbols that need further processing to determine characters or separators,

2- Apply rules to decide if the symbol in the ambiguous class is a character or a separator.

Tokens are defined by a tokenization algorithm, which decides for each character in the input, whether it belongs to the current token or not. The algorithm makes its decision based on the category assigned to each character in the character classification table.

The categories fall into three major groups:

Group 1: consists of categories "A" and "D" which are delimiters in any context.

Group 2: defines categories "L", "U" and "N" which are never delimiters.

Group 3: consists of characters which may conditionally be delimiter, "E" to "K" depending on the context in which they are used.

Group 1	Categorize A	Blank character (White-Spaces).
	Categorize D	() {} [] tab newline _
	Categorize L	Lower alphabetical "أ" "ب" "ت" …… (all the Arabic alphabetical letters and English lower case letters).

Group 2	Categorize U	Upper alphabetical "A" "B" "C" ….. (not applicable for Arabic).
	Categorize N	Numerical characters "0" "1" "2" "3" …… (Arabic numbers and Indian numbers).
Group 3	Categorize E	Set of punctuation characters which are not treated as delimiters when surrounded by alphabetic, e.g. Apostrophe. However, this is not applicable for Arabic.
	Categorize F	Set of punctuation characters which are not treated as delimiters when surrounded by Numeric, e.g. , . : /
	Categorize G	Set of punctuation characters which are not treated as delimiters when preceded by punctuation and followed by a numeric, e.g. , $
	Categorize H	Set of punctuation characters which are not treated as delimiters when immediately preceded by n identical character, e.g. / - + = % # @ ^

Table 5-1 Characters Categorizes of Language Independent Text Tokenization

3- After applying the rules of "Language Independent Text Tokenization Using Character Categorization" algorithm.

The rules determine if a character is a delimiter or not by taking into consideration the classes of the current character, the previous character and the next characters. If the character is a delimiter, this means it is a separator token, so the tokenizer assigns the appropriate token type and repeats the process for all text. Figure 5-1, presents the different steps of the tokenization algorithm which discussed previously:

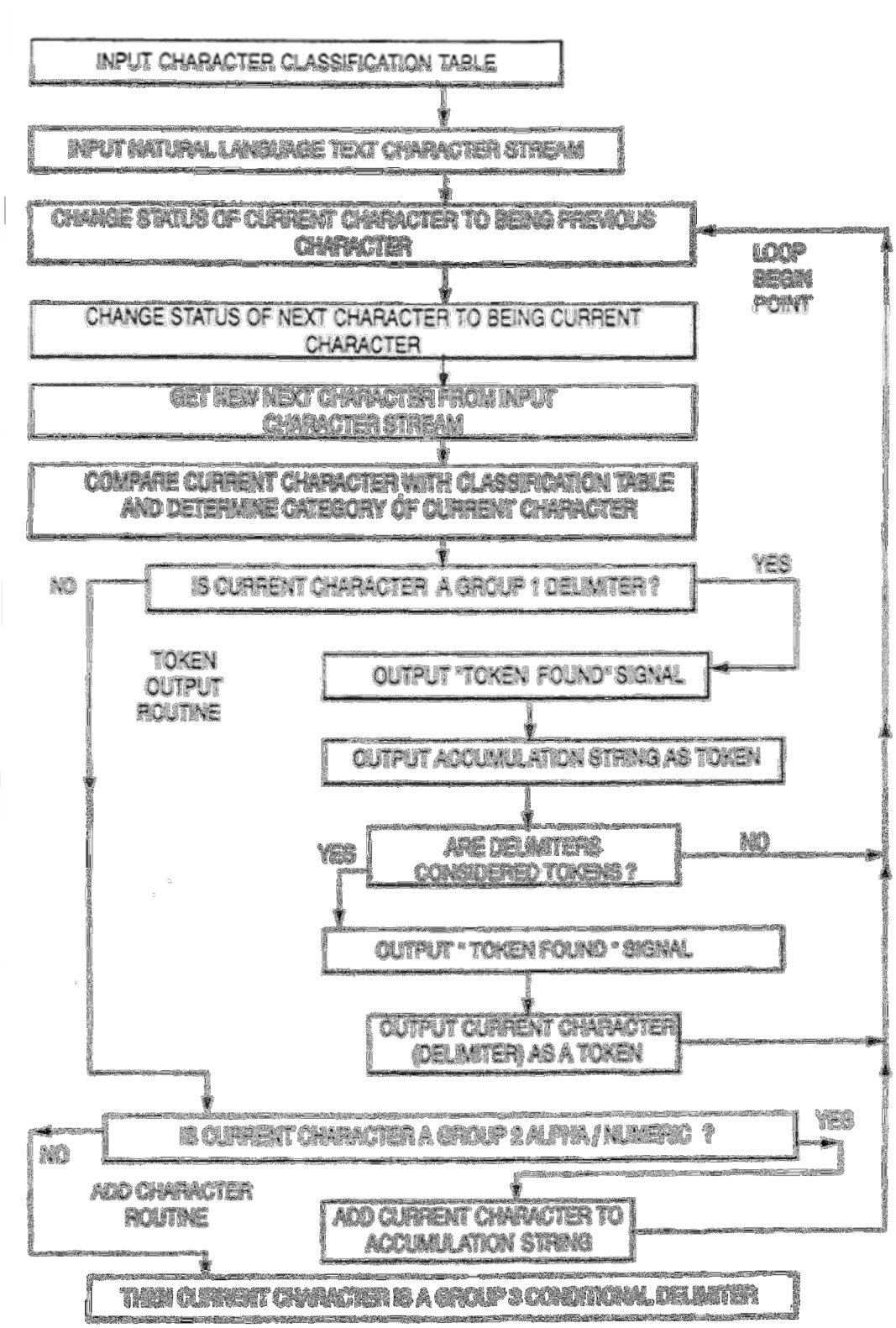


Figure 5-1 Method for Language Independent Text Tokenization Using Character Categorization algorithm flowchart

5-2- Normalization

This step involves performing some normalization functions on words to represent them in some standard forms. This is done at more than one step. There are "first normalization", "second normalization" and "custom normalization", where each of the mentioned steps has its own replacement rules. We separated the normalization into more than one step to distinguish any changes could the word have.

- First Normalization step: Remove unhelpful characters and make any transformation that does not lose data. Like removing "ـ" in "مـاء" to be "ماء".

  This step will transform the text to be in the "First Normalization Form".
- Second Normalization step: Remove information that tends to act as noise at some processing levels. Actually at this step we strip out (تشكيل) diacritics.

  This step will transform the text to be in the "Second Normalization Form".
- Custom Normalization: We provide the functionality to enable the developer customizing the text normalization, so the user can add or remove rules on demand.

The surface form of the word and all its normalization forms will be available. So, depending on the situation and the needed processing, the developer can choose which normalization level is needed.

Examples of the First Normalization rules:

['ـ'] = ''

['آ'] = 'ءا'

Examples of the Second Normalization rules:

[' ◌ً '] = "فتحتان fathatan [◌ً]

[' ◌ٌ '] = "ضمتان damatan [◌ٌ]

[' ◌ٍ '] = "كسرتان kasratan [◌ٍ]

[' ◌َ '] = "فتحة fatha [◌َ]

[' ُ '] = ″ضمة dama [ُ]

[' ِ '] = ″كسرة kasra [ِ]

[' ّ '] = ″شدة shada [ّ]

[' ْ '] = ″سكون sokon [ْ]

Chapter 6: Morphological Parsing

After studying all the approaches used in the morphological analysis, and after thinking deeply about the problem, we were able to build our system. We will present here the steps of our work that led to our approach.

6-1- First Thought

We first thought about using the standard patterns of Arabic words and implementing the rules of Elal and Ebdal (الإعلال والإبدال) in a way similar to section "2-2-1- Root-and-Pattern". Elal and Ebdal are about changing or deleting long vowel letters (أحرف العلة) and Hamza (الهمزة) letter. Before doing that we took an insight look at some of the rules. For example, you can see the following figure that depicts some of the rules of (إبدال) Ibdal (مروان البواب وعدد من الباحثين, 2007) :

١- إذا كانت فاء (افتعل) ثاءً، أُبدلت تاؤه ثاءً، وأُدغمتا؛ نحو: اثَّأر.
٢- إذا كانت فاء (افتعل) دالاً، أُبدلت تاؤه دالاً، وأُدغمتا؛ نحو: ادَّعى.
٣- إذا كانت فاء (افتعل) طاءً، أُبدلت تاؤه طاءً، وأُدغمتا؛ نحو: اطّرد.
٤- إذا كانت فاء (افتعل) ذالاً أو زياياً، أُبدلت تاؤه دالاً؛ نحو: اذْدَكَر، ازْدَهى.
٥- إذا كانت فاء (افتعل) صاداً أو ضاداً أو ظاء، أُبدلت تاؤه طاءً؛ نحو:اصطفى،اضطجع،اظطلم.
٦- إذا كانت فاء (افتعل) واواً أو ياء، أُبدلت تاءً، وأُدغمت في تاء الافتعال؛ نحو: اتَّصل، اتَّسر.

Figure 6-1 Examples of Ebdal (إبدال) letter changing in Arabic

Anyone can see that this is complicated but could be implemented by some if-then-else rules.

However, taking another look at some rules of Elal (إعلال) one can see that things are very complicated. There are four types of Elal by conversion (الإعلال بالقلب) that exists: "قلب الواو والياء ألفاً", "قلب الواو ياءً", "قلب الياء واواً" and " قلب الواو والياء همزة".

As an example, the figure below presents some rules of "قلب الواو ياءً" case (مروان البواب وعدد من الباحثين, 2007).

:

تقلب الواو ياءً في عدة مواضع؛ أهمها:

١- أن تَسكُنَ بعد كسرة؛ نحو: **مِيعَاد**، **مِيزَان** (أصلهما: مِوْعاد، مِوْزان)، **حِيلَة**، **دِيمَة** (أصلهما: حِوْلَة، دِوْمَة).

فإن لم تكن ساكنة، لم تُقلب؛ نحو: إوَزَّة، عِوَض، صِوَان...

٢- أن تتطرف بعد كسرة؛ نحو: **الغازِي** (أصلها: الغازِوُ)، **رَضِيَ** (أصلها: رَضِوَ).

فإن لم تتطرَّف، لم تُقلب؛ نحو: عِوَج، حِوَل...

٣- أن تقع حشواً بين كسرة وألف في مصدر الفعل الأجوف الذي أُعِلَّتْ عينُ فِعْلِهِ؛ نحو: **قِيَام**، **انْقِيَاد**، **عِيَادَة** (أصلها: قِوام، انقِواد، عِوادَة).

فإن لم تكن الكلمة مصدراً، لم تُقلب؛ نحو: سِوَار، رِوَاق، خِوَان، سِوَاك... وكذلك إن صحَّت العين في الفعل، صحَّتْ في المصدر؛ نحو: حِوَار (مصدر حَاوَرَ). وكذا تَصِحُّ إن لم يكن بعدها ألفٌ؛ نحو: حِوَل. وكذا تَصِحُّ إن لم يكن قبلها كسرة؛ نحو: رَواح، زَوال.

٤- إذا اجتمع الواو مع الياء في كلمة، والسابق منهما متأصل ذاتاً وسكوناً، فعندئذ تقلب الواو ياء وتُدغَم في الياء؛ نحو: **سَيِّد**، **طَيّ**، **مَرْمِيّ**.

٥- إذا وقعت طرفاً في الفعل الماضي، وكانت رابعة أو أكثر، وما قبلها مفتوح، على أن تكون منقلبة أيضاً في الفعل المضارع؛ نحو: **أعطَيْتُ**، **زَكَّيْتُ**، **استرضَيْتُ**.

Figure 6-2 Examples of Elal (**إعلال**) letter changing in Arabic

As one can see, some rules are related to diacritics which are not usually provided to the system of morphological analysis. Some other rules, like rule number 5 in the above figure, are complicated in a way that makes processing needs a very long time or else the developer will have to ignore such a case.

To see the whole picture, let us take a look also at "**الإعلال بالحذف**" and see its rules in the figure below (**مروان البواب وعدد من الباحثين**, 2007):

يُحذف حرف العلة في ثلاثة مواضع:

١- أن يكون حرف مدّ ملتقياً بساكنٍ بعده؛ نحو: **قُمْ، بِعْ، خَفْ، قُمْتُ، يَقُمْنَ، رَمَتْ، ترمُونَ،** (أنت) **ترمِينَ، قاضٍ، فتًى**، بشرط ألا يكون الساكن بعد حرف العلة مدغماً، نحو: **شادٌّ**.

٢- أن يكون الفعل مثالاً واوياً معلوماً مكسور العين في المضارع، فتحذف فاؤه من المضارع والأمر ومن المصدر أيضاً (إذا عُوِّض عنها بالتاء)؛ نحو: **يَعِدُ، عِدْ، عِدَةً**.

٣- أن يكون الفعل معتلّ الآخر، فيحذف آخره في أمر المفرد المذكر؛ نحو: **ادْعُ، ارْمِ، اخْشَ**، وفي المضارع المجزوم الذي لم يتصل بآخره شيء؛ نحو: لم **يَدْعُ**، لم **يَرْمِ**، لم **يَخْشَ**.

Figure 6-3 Rule of "الإعلال بالحذف"

So, if the input was not diacritized (**بدون تشكيل**), this means that there is no way to be sure that there is no long vowel letter (**حرف علة**) deleted somewhere. For that, any rule based morphological analyzer has to expect that there could be any missing long vowel letter (**حرف علة**) anywhere in the word, or else it will miss some hits.

6-2- Second Thought

The pattern matching is the corner stone in the Arabic morphology. After our investigation and brain storming, we found that the best method will involve having a list of all patterns those who are subject to the general case, along with the patterns of special cases in which Elal and Ebdal (**الإعلال والإبدال**) are presented. Luckily, we found a program, which is "AlKhalil Morpho Sys", which provided these patterns, as free and open source, in a form of a database that contains the patterns and their roots with the useful information needed about those patterns.

6-2-1- Using the Database of AlKhalil

Luckily, we found a program, which is "AlKhalil Morpho Sys" that provided these patterns, as free and open source, in a form of database that contains the patterns and their roots with the useful information needed about those patterns. Actually, we think that AlKhalil have its outstanding because of its marvelous database. Go back to section "2-3-3- AlKhalil Morpho Sys" for more detail on AlKhalil.

6-2-2- Our work on AlKhalil Database

Because of the sophistication of the Arabic language, and because this is the first version of AlKhalil, lots of things could be addressed in the database shortage. However, we will address some important issues that we could enrich:

- All nouns, of closed classes, and particles are provided as tool words without classifying them as nouns or particles.

We split the file of tool words into two files for nouns and particles and moved the existed nouns and particles to their appropriate file accordingly.

- The database does not contain information about the closed nouns except their fully diacritized form and their Arabic class name, along with the allowed proclitics and enclitics.

An example of the information provided for a noun is: "إياكم" where just its fully voweled form: "إيّاكُم" and its class: "ضمير نصب" have been given. However, we went through all existed nouns of closed classes and assign an attribute related to definiteness, gender, number and case. As for the example above (إياكم), the definiteness value is (معرفة) definite, the gender is (مذكّر) male, the number is (جمع) plural and the case is (مرفوع) nominative. Actually, all the 85 existed nouns have been assigned the related attributes.

- There are some missing nouns in the list of tool words.

As example, the word "إياي" which is an accusative pronoun (ضمير نصب) does not exist. We add 7 new nouns with their diacritized form, part of speech, definiteness, gender, number and case to the list of nouns of closed classes. The added nouns are: إيّايَ, أَيْنَمَا, أَيُّمَا, كَذَاْ, كَأَيِّنْ, فُلَاْن, فُلَاْنَة.

- There is no differentiation between (الاسم الموصول الخاص) Special Relative Pronoun like "الذِي" and (الاسم الموصول المشترك) Common Relative Pronoun like "مَنْ".

We marked the 10 existed special relative pronouns as so (الذِي, التِي, اللَّذانِ, اللَّذَيْنِ, اللَّواتِي, اللاَّئِي, اللاَّتِي, الذِينَ, اللَّتَيْنِ, اللَّتَانِ), and the 2 common relative pronoun as so (مَنْ، مَا).

- There are some missing particles at the list of tool words.

For example, the particle (إِذْمَاْ), as an apocopative particle "حرف شرط", does not exist. However, we add 2 missing particles (سَوْفَ, إِذْمَاْ) to the list of particles.

6-3- Morphological Analysis Steps

Our process of morphology analysis consists of the following steps:

- Compounding Detection
- Isolation (عزل اللواصق)
- Lookup at Closed Lists
- Pattern Matching
- Root Extraction (تجذير)
- Voweled Patterns List Production and Verification
- Fill Morphological Attributes and POS Tags
- Assembling clitics with matches

6-3-1- Compounding Detection (اكتشاف التراكيب)

Each compound words separated by tokenizer are recombined together in one word and the related morphological attributes are assigned. This is implemented through the use of a closed list of compound nous with their attributes. This list now contains 8 compound nouns and could be expand by adding more words to the related XML file.

6-3-2- Isolation (عزل اللواصق)

Cliticalization parsing involves separating the word from its clitics. Providing that, clitic is a word with independent POS (Parts of Speech) that happen to be written as an affix. However, this will provide the proclitics "لواصق سابقة" (clitics written before main word), isolated form "الشكل المعزول" (could be called cliticless form), and enclitics "لواصق لاحقة" (clitics written after main word).

Cliticalization parsing is implemented using two coherent parts. In the first, which is the isolation, the clitics are greedily detected according to two automates, one for proclitics and the other for enclitics. In the second part, which is assembling clitics with matches, the clitics are verified against some linguistic rules and the POS tag is assigned accordingly. The second part happen at step "6-3-8- Assembling Clitics with Matches" is described below.

The first part is implemented using two sets of automates. The first set is for proclitics, while the second is for enclitics. For that, the first set scans the letters of the word from right to left, while the second scans the letters from left to right. The figure below depicts this process by example.

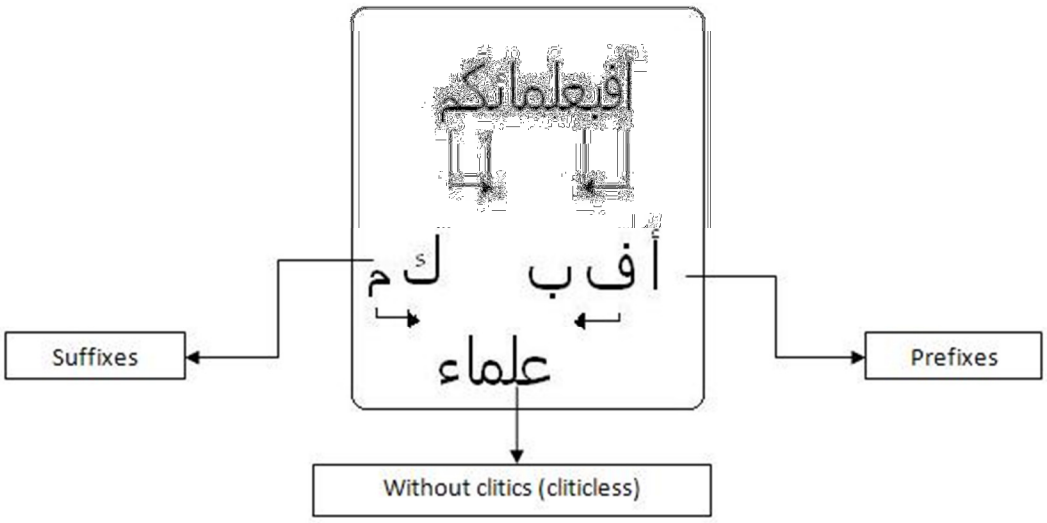


Figure 6-4 Example of enclitic and proclitic detection and extraction process

However, each of the proclitics and enclitics automates have been divided into three automates one for verbs, another of nouns, and the last for commons. This is to give the system the ability to know whether the word is noun or verb at this stage. So the system, for example, does not have to explore the case that a word is a noun if the scanned proclitics are for verbs.

In figures below, a proclitic automate diagram for common words, which is called "class C", is depicted.

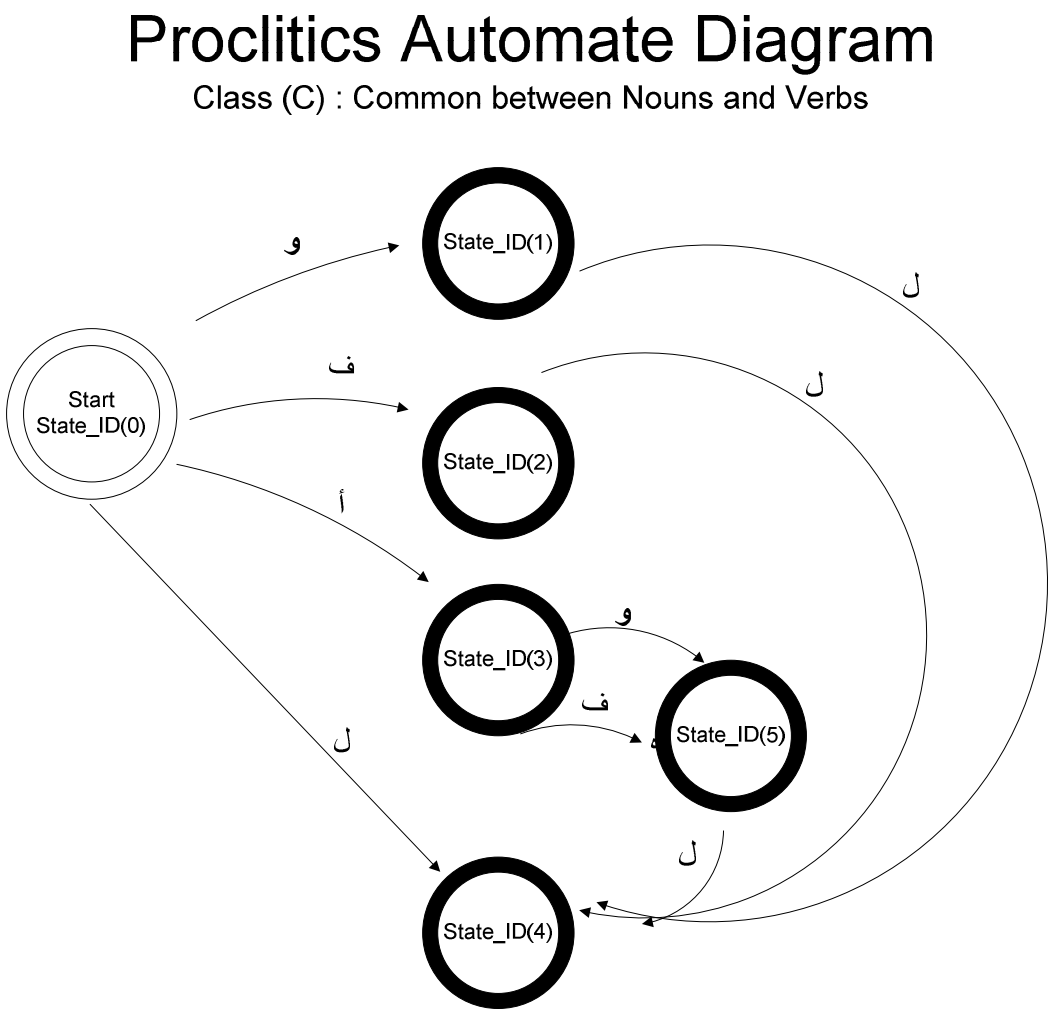


Figure 6-5 Proclitic automate diagram "Class C"

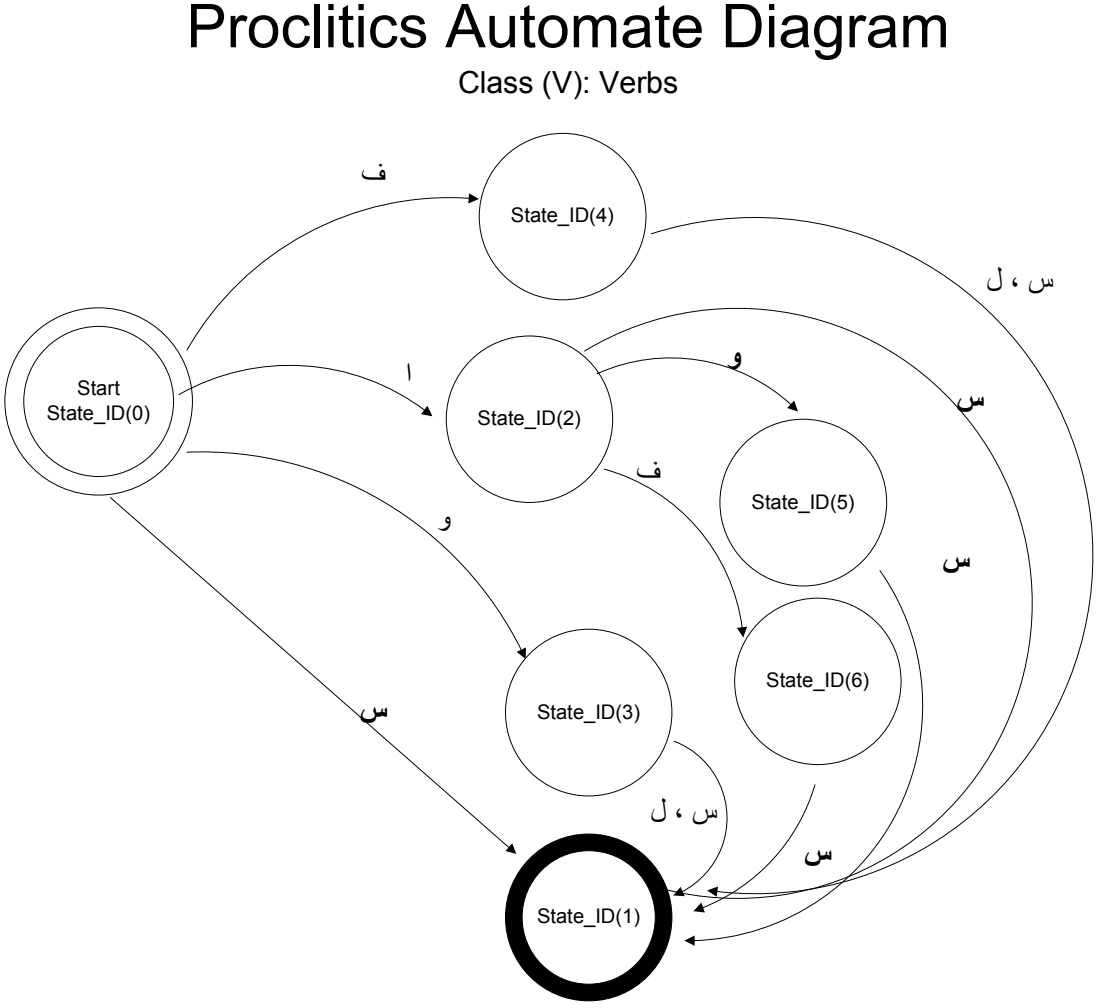


Figure 6-6 proclitics automate diagram "Class V"

Proclitics Table for Automate

Class (N): Nouns

	أ	ا	ب	و	ف	ل	ك
0	1	3	4	5	6	7	8
1	-	3	4	11	12	17	8
2 (End State)	-	-	-	-	-	-	-
3	-	-	-	-	-	2	-
4	-	3	-		-	-	-
5	-	3	4	-	-	16	8
6	-	3	4	9	-	16	8
7	-	-	4	-	-	10	8
8	-	3	-	-	-	-	-
9 (End State)	-	-	-	-	-	-	-
10 (End State)	-	-	-	-	-	-	-
11	-	-	13	-	-	15	14
12	-	3	13	-	-	15	14
13 (End State)	-	3	-	-	-	-	-
14 (End State)	-	3	-	-	-	-	-
15	-	-	-	-	-	10	-
16 (End State)	-	-	-	-	-	10	-
17 (End State)	-	-	-	-	-	2	-

Table 6-1 Proclitic Automate for "Class N" shown in a table

In addition to proclitics automates, we made three automates for enclitics letters which are also for verbs, nouns and common between verbs and nouns. However, the automate starts from the end letter of the word.

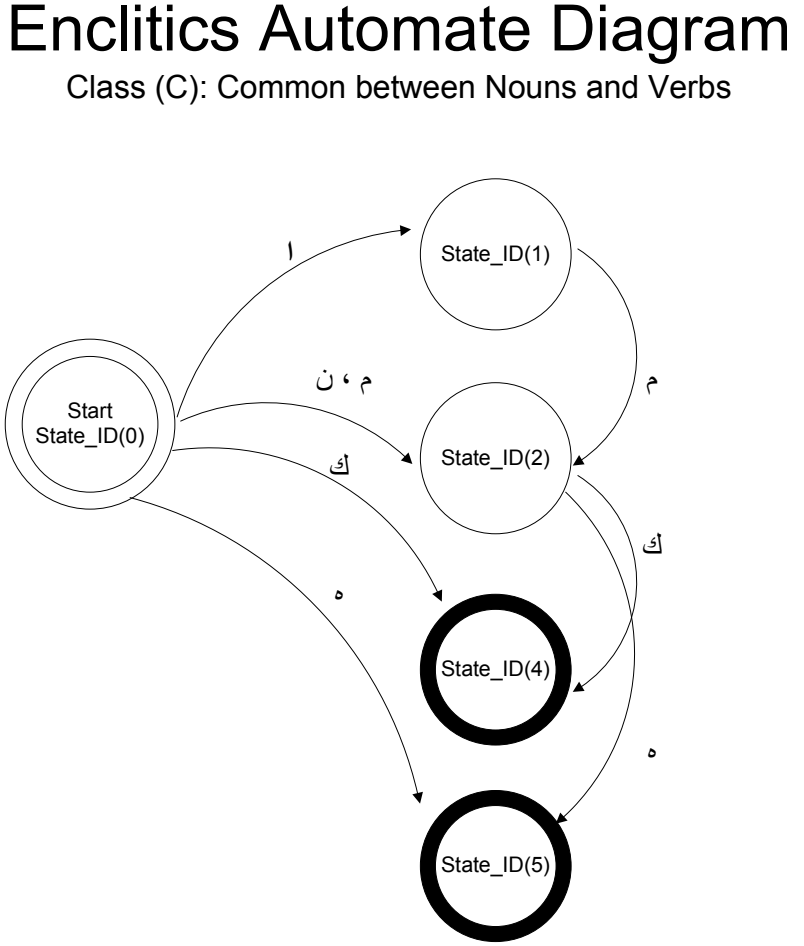


Figure 6-7 Enclitics automate diagram "Class C"

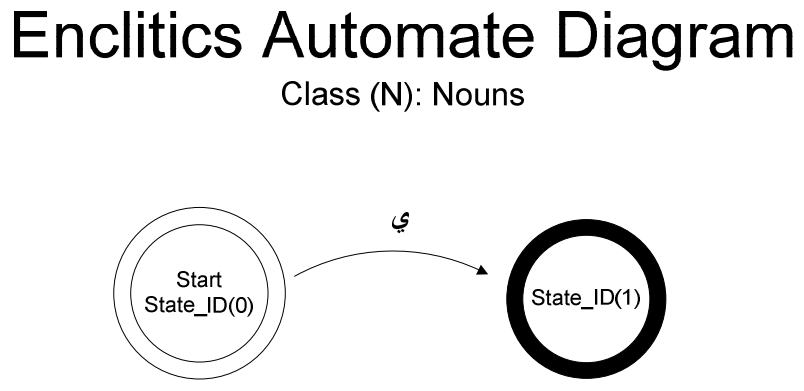


Figure 6-8 Enclitics automate diagram "Class N"

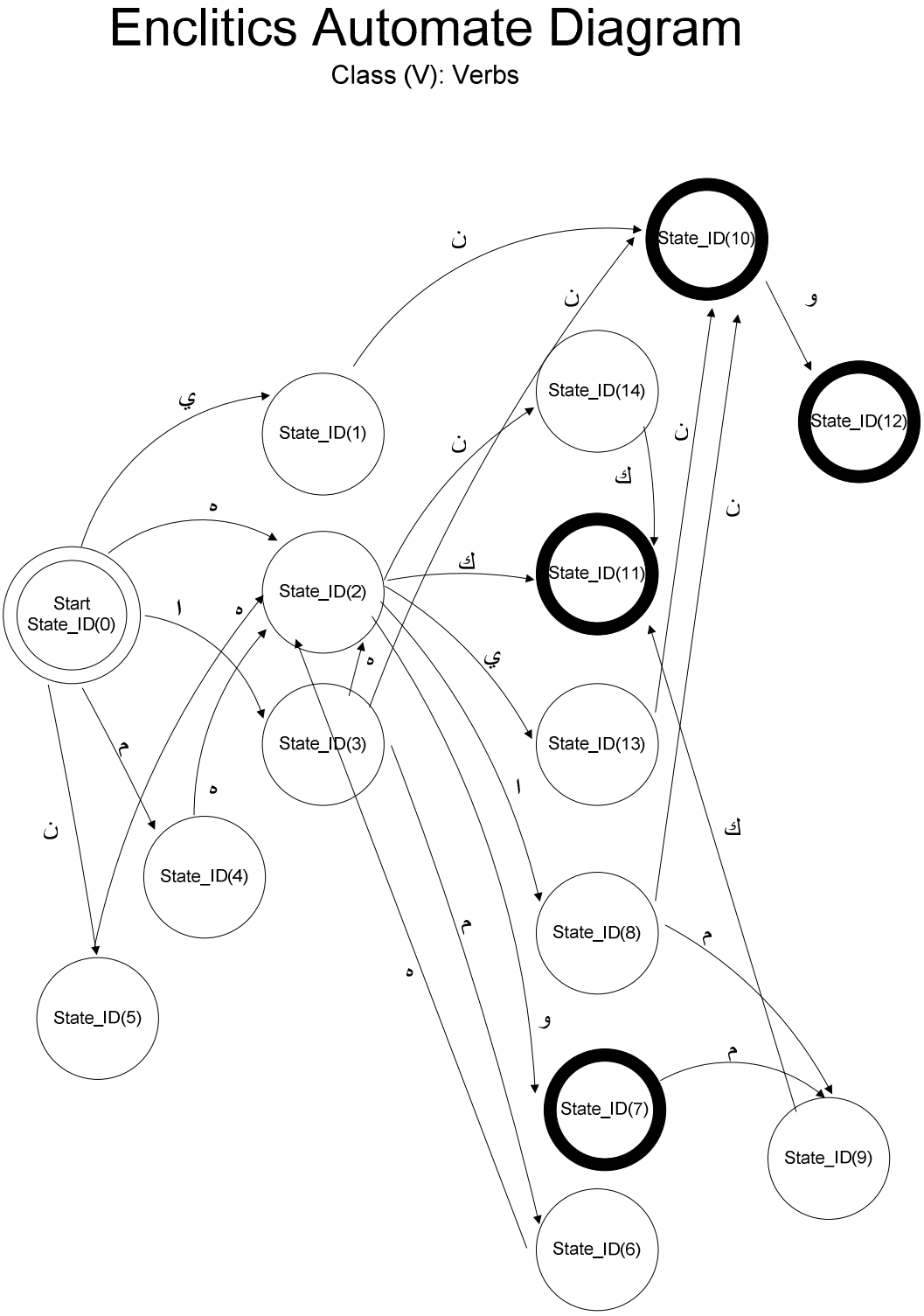


Figure 6-9 Enclitics automate diagram "Class V"

6-3-3- Lookup at Closed Lists

A search for cliticless word, that is the main POS of the word segments, is performed. And the related matches are selected and passed to the next processing steps. For example, the word "ما" could be a common relative noun (اسم موصول مشترك) or a conditional noun (اسم شرط). Another example is the word "من" which could be a common relative noun (اسم موصول مشترك), a conditional noun (اسم شرط) and a preposition (حرف جر). However, finding a word in the closed list does not omit the next steps that involve pattern matching. For the same example of the word "من", the pattern matcher will detect its capabilities of being a verb (like being مَنَّ or مُنْ) as well as a noun (like being مَنٌّ or مَنُّ).

Three lists of words used at this stage. The first is for closed nouns like (الظروف) adverbs, (ضمائر الرفع والنصب) Nominative and Accusative pronouns. This list contains 8 deferent types of nouns with 88 entries. The second list contains particles like "سوف" as (حرف استقبال) Futurity particle and like "إن" as (حرف شرط) conditional particle "إِنْ" and as (حرف ناسخ – مشبّه بالفعل) Annuler particle "إِنَّ". This list contains 11 deferent types of nouns with 48 entries. To know more about our work on those two lists see "6-2-2- Our work on AlKhalil Database**Error! Reference source not found.**". However, the second list contains the proper nouns with their voweled and unvoweled forms; as they provided by AlKhalil database.

6-3-4- Un-diacritized Pattern Matching (مطابقة الوزن)

The isolated (cliticless) word is matched against the existing patterns. The form of the word used here for matching is the second normalization form in which the (التشكيل) diacritics are omitted. This is to simplify the pattern matching since most of

the words are expected to be provided to the system without diacritics, while just a few words would be fully or semi-diacritized. However, this does not affect the final presented results. Because, we omit the matched results which do not agree with the user provided diacritics.

6-3-5- Root Extraction (تجذير)

All possible root(s) of each matched pattern is extracted according to rules. Two processes of verification for the extracted roots are performed. First, the roots which do not exist in the roots database are neglected. Actually, there are two lists of roots could be used upon the user request. The first list, which are supposed to contain all roots in Arabic, (contains 7502 roots) is used when user needs to have all possible results. The second list, which are supposed to contain the most popular used roots (contains only 2903 roots), is used when user needs to have more focused results.

The second verification process is to test if the extracted root could be used to generate this word. However, this happen at the next step through intersecting the list of IDs of voweled patterns, that could be generated from the matched unvoweled pattern, with the list of IDs of voweled pattern that could be generated from that root.

To clarify the process, you can look at the structure of the files of roots, unvoweled patterns and voweled patterns as they appear in the next figure:

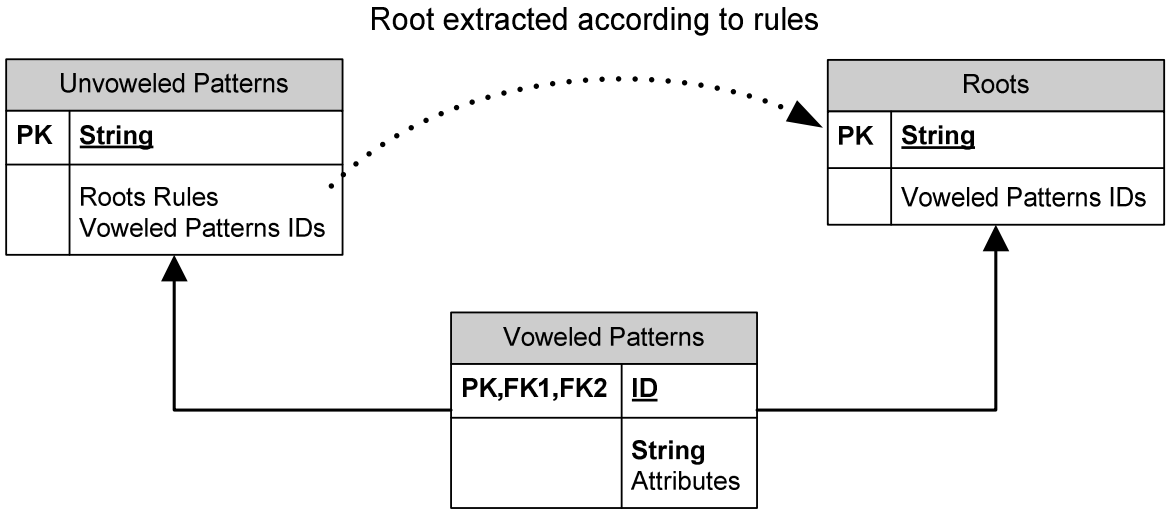


Figure 6-10 Relations between roots, unvoweled patterns and voweled patterns

Actually, the figure does not reflect the real names used in files. It is used to show the process and relation for root extraction without confusion.

6-3-6- Voweled Patterns List Production and Verification

As mentioned early, the list of voweled patterns (that could be generated from the matched unvoweled pattern) is matched with the list of voweled pattern (that could be generated from the extracted roots) to produce a list of Voweled Patterns. This list would contain the presumed voweled patterns. Moreover, at this step, a test of agreement between the diacritics of the voweled patterns and any diacritic of the word provided to the system is performed.

6-3-7- Fill Morphological Attributes and POS Tags

Each voweled pattern is associated with some morphological attributes and a POS tag. The information of the voweled pattern is transferred to the isolated (cliticless) word in an appropriate data structure preparing for the next steps. All attributes' values assigned are binary equal 2^n or the sum of that; if the attribute accepts more than one value. This is to ease testing and assigning values. Let us take the following example: the main class of the word is defined as a noun = 2^0=1, or a verb = 2^1 = 2 or particle = 2^2=4. Now, if at any processing step we need to test if the word is a verb or a particle, the code would be like:

```
Word.MainClass & (MainClass.Verb+MainClass.Particle)!=0
```

Another helpful example could be like that: if some noun considered being either accusative (منصوبة) or genitive (مجرورة), the value of its case (الحالة الإعرابية) would be: 2^1+2^2=6. However, these values are stored in constants to ease working with. So, the developer does not use 2^n in the code. Instead he/she can use its constant name which has its value. For that, to assign the value of accusative or genitive to some word case would be like that:

```
Word.Case = Case.Accusative + Case.Genitive
```

6-3-8- Assembling Clitics with Matches

Clitics (proclitics and enclitics) are assembled with the compatible cliticless words according to defined rules. For example, prepositions (أحرف الجر) can only be with genitive nouns (اسم مجرور), since their syntactic function is to make nouns genitive (قَبَّش، ١٩٨٢). Another example is the starter Lam (لام الابتداء) which does not come with imperative verbs (قَبَّش، ١٩٨٢). A problem of complexity arises here. Each clitics can have more than one diacritic form (صيغة مشكّلة), like the enclitic "ك", the pronoun of the second person, which could be for masculine with Fatha diacritic (كَ) or feminine with Kasra diacritic (كِ). Furthermore, some clitics of the same diacritic form can act as more than one POS. For example, the proclitics "وَ" could be used as a particle of conjunction (عطف), preposition (جَرّ), jurative (قَسَم) or appendix (زائد).

Rules are defined for each POS of clitics and each clitic is associated with its rule of rules. For example, the rule of prepositions (أحرف الجر) is defined as follows:

```
    preposition =
ProcliticGrammer(ParticlePOSConstants.SubClass.Preposition,
POSConstants.MainClass.Noun,
CliticlessPOSConstants.CaseAndMood.GenitiveOrJussive,
NominalPOSConstants.SubClass.all_Cases);
```

And each clitic is associated with the appropriate rule(s) for each diacritical form. For example, the letter "ل" could be for `preposition` or `imperative` if its diacritic is Kasra (لِ), and could be `emphasis starter` or `accusative if its` diacritic is Fatha (لَ).

```
GrammersDict['ل'] = {};
GrammersDict['ل'][DiacriticsConstants.Kasra]=[preposition,
imperative];
GrammersDict['ل'][DiacriticsConstants.Fatha]=[emphasisStarter,
accusative];
```

6-4- Output Complexity

Speaking about the possibilities of word segmentation after greedily detecting morphemes will lead us to the complexity involved. Actually, the following study of the complexity is applicable to any Arabic morphological analyzer.

6-4-1- Cliticalization Parsing

a) Part One: Isolation

The number of possible sequence of word segmentation is depicted in the following equation. Giving that Number of Proclitics is "P" and Number of Enclitics is "E":

```
Segmentation Possibilities = 1 + (P + 1) * (E + 1)
```

(Equation 6-1)

Figure 6-11 Complexity of Isloation

Let us take for example the word: "**أوبعلمائكم**", the clitics could be greedily detected to give the following proclitics: (' أ'، 'و'، 'ب'), cliticless: ('علمائ') and enclitics: ('ك'، 'م'). After that, the word could be segmented as follow:

```
[[]، 'أوبعلمائكم'، []]
[[]، 'أوبعلمائك'، ['م']]،
[[]، 'أوبعلمائ'، ['ك'، 'م']]،
['أ']، 'وبعلمائكم'، [[]]،
['أ']، 'وبعلمائك'، ['م']]،
['أ']، 'وبعلمائ'، ['ك'، 'م']]،
['أ'، 'و']، 'بعلمائكم'، [[]]،
['أ'، 'و']، 'بعلمائك'، ['م']]،
['أ'، 'و']، 'بعلمائ'، ['ك'، 'م']]،
['أ'، 'و'، 'ب']، 'علمائكم'، [[]]،
['أ'، 'و'، 'ب']، 'علمائك'، ['م']]،
['أ'، 'و'، 'ب']، 'علمائ'، ['ك'، 'م']]
```

Figure 6-12 Word segmentation example

b) Part two: Assembling Clitics with Matches

As mentioned at section "6-3-8- Assembling Clitics with Matches" some clitics can have more than one tag that could be assigned to. Assuming that, in average a sequence of proclitics with length "PL" can have "PN" different number of possible

tags and a sequence of enclitics with length "EL" can have "EN" different number of possible tags, will make the complexity of cliticalization parsing as follow:

```
Cliticalization Complexity = 1 + (P*PL*PN + 1) * (E*EL*EN
+ 1)
```

(Equation 6-2)

Figure 6-13 Complexity of Cliticalization Parsing

6-4-2- Particles and Closed Nouns

Looking up the word at particles and closed nouns lists will make the complexity at equation 6-2 as follow; provided that a word can have in average "CN" number of entries found in all particles and closed nouns lists:

```
Complexity = CN [ 1 + (P*PL*PN + 1) * (E*EL*EN + 1)]
```

(Equation 6-3)

Figure 6-14 Complexity of Cliticalization Parsing and Particles and Closed Nouns looking up

6-4-3- Pattern Matching

a) Un-diacritized Pattern Matching and Root Extraction:

Let us start calculating the overall pattern matching complexity by computing the complexity of the un-diacritized pattern matching and root extraction. Assuming that each word could be matched with "UN" of un-diacritized patterns and each UN can have "RN" number of possible rules for root extraction, will provide the following equation for pattern matching complexity:

Calculating the complexity of pattern matching:

```
Root and Un-diacritized Patten Complexity = UN * RN
```

(Equation 6-4)

Figure 6-15 Complexity of Un-diacritized Patten Matching and Root Extraction

b) Voweled Patten Matching List Production and Verification

Assuming that a un-diacritized pattern will have an average of "UV" voweled patterns at its list of the applicable voweled pattern and a root will have an average of "RV" of voweled patterns at its list of applicable voweled patterns. These will produce the following complexity for list production:

```
List Production Complexity = UV * RV
```

(Equation 6-5)

Figure 6-16 Complexity of List Production

This will give the complexity for pattern matching as follow:

```
Pattern Matching Complexity = UN * RN * UV * RV
```

(Equation 6-6)

Figure 6-17 Complexity of Pattern Matching

6-4-4- Overall Output Complexity

Assuming that the pattern matching complexity of all operations at un-diacritized pattern matching, root extraction and voweled patterns list production, will make a number of "PM" matches. This will change equation 6-3 to be:

```
Complexity = (CN+PM) [ 1 + (P*PL*PN + 1) * (E*EL*EN + 1)]
```

(Equation 6-7)

Figure 6-18 Overall complexity of the output of morphological parsing

Where:

"CN" is the number of entries found in all particles and closed nouns lists.

"PM" is the number of matched patterns for all nominal and verbal patterns.

"P" is the number of proclitics.

"PL" is the length of a sequence of proclitics.

"PN" is the different number of possible tags for each sequence of proclitics.

"E" is the number of enclitics

"EL" is the length of a sequence of enclitics.

"EN" is the different number of possible tags for each sequence of enclitics.

Chapter 7: Tagging

7-1- Our Approach for Ambiguity resolution

To deal with the sophistication of Arabic that leads to ambiguities, we decided to develop an expert system that takes rules, each with certainty, and make inference base on those rules. The system, upon the user request, provides all possible solution without eliminating any possibility that could be useful. But, at the same time it supposes to provide a hint for each possible tag. Moreover, for the user needs, (n) tags with the higher certainty or the tags with certainty higher than a (threshold), could be provided while the others are eliminated.

A question could be asked: Why not giving the best suitable tag? Why providing a set of solutions, with certainty of each?

In Arabic, the number of tags that could be assigned to a word could be 1, 10, 20 or even sometimes 50 or more. As example, the word "وأحسن", where AlKhalil Morpho Sys will give 80 possible tags for it, our system will give 160 possible tags. So, on one hand, depending on just statistics available at tagging phase to pick just one tag, among let us say 100 possible tags, will not be sound. On the other hand, if the algorithm is a rule based without depending on some sort of probability or certainty, those 100% certainty, rules would be hardly ever acquirable and would not give some sort of weights to the tags those which are accepted by all rules.

Why this rule based expert system?

Since our system will provide the most certain tags with the certainty of each, the other layers of language processing would be able to mark up and down the tags certainty. Actually, we believe that it is too early to resolve all the ambiguity of tagging before parsing, semantic and pragmatic processing.

7-2- Tagging Phases

Usually, tagging absolutely happens after morphologically analyzing the input text. But, we separate the tagging into two parts. The first happens before morphologically analyzing the input text and the other comes after. However, each of which will be identified and described in the next sections.

7-2-1- Premature Tagging (التوسيم الخديج)

a) Introduction

The idea behind this tagging is to make a use of rules that applies on the base of regular expressions before the step of morphological analyzer. Before the time of this project, there is nothing called Premature Tagging. However, a slightly similar idea is used by Eiman Tamah Al-Shammari to improve the performance of Arabic stemmer based on two rules for two kinds of stop words (Eiman Tamah Al-Shammari, 2008) where the first set of adverbs, which she called prepositions for unknown reason, "حيثما كلما إذا عندما" are supposed to precede verbs. And the others are set of adverbs and prepositions "إلى، أمام، باتجاه، بعد، بين، تحت، خارج، خلال، عبر، على، فوق، في، قبل، قريب، منذ، وراء" which supposed to precede nouns. Actually, these two rules are fixed and enhanced, and used in our premature tagger with two other rules.

Actually, we developed a rule-based premature tagger. But, this concept could be implemented as any suitable classifier.

b) Premature Tagger V.S Usual Tagging

Let us identify premature tagging by comparing it to the usual tagging which will be called overdue tagging:

Premature tagging happens before the morphological analysis phase, while usual tagging happens after. Because, the premature tagging gives a hint to the morphological analyzer, while overdue tagger depends on the morphological analyzer.

Premature tagging is based on string comparison, lexical information and shallow and/or detailed tags, with or without information about the final detailed tag of the matches, whereas, the overdue tagging depends on the detailed tags to select the best detailed tag of each word.

Premature tagger supposed to be fast, while in overdue tagger some complex calculations are acceptable.

Additionally, tagging stop words could be considered as sub task of the premature tagging task.

c) How our premature tagger work

The premature tagger applies its fast rules and tags some words with their assumed, or abandoned, main POS class along with the certainty of this tagging. The assumed, or abandoned, main class is later used at morphological analysis to determine if a search for a possibility that a word could belong to a main class is needed. Where, a positive threshold, for assumed main class, and negative threshold, for abandoned main class, supposed to be selected to be used at the morphological analysis task.

For example, if a word is tagged as particle with certainty higher than the selected threshold, the morphological analyzer will just search for this particle at the closed list of particles and does not have to go throughout the pattern matching processing to determine the word attributes and full POS tag.

Another example is that, if a word is given a negative certainty, lower than the negative threshold, of being a verb, the morphological analyzer will not try to perform the pattern matching with the verbal patterns. And will just search for the word at the particles closed list, nouns closed lists and will perform pattern matching just with the nominal patterns.

d) Rules

Rules are implemented in Qutuf as Transducers. Those transducers are flexibly defined at an XML file. However, the used rules are listed below in four figures.

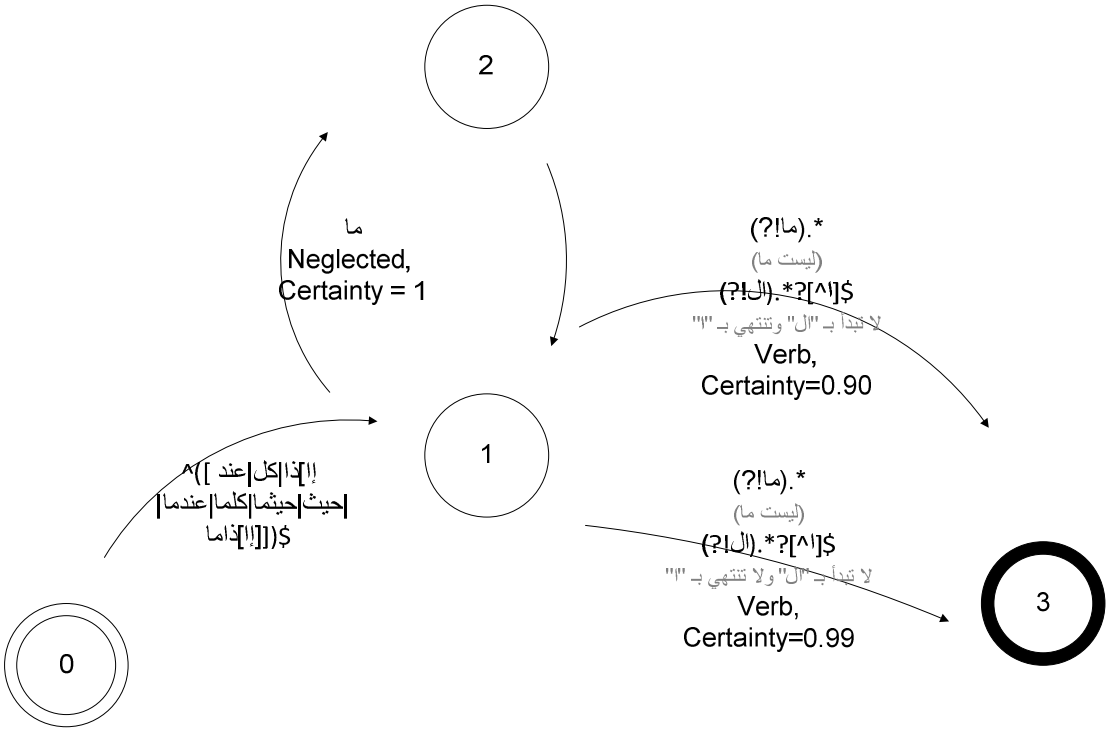


Figure 7-1 Rule of adverbs (**ظروف**) followed by verbs (**أفعال**)

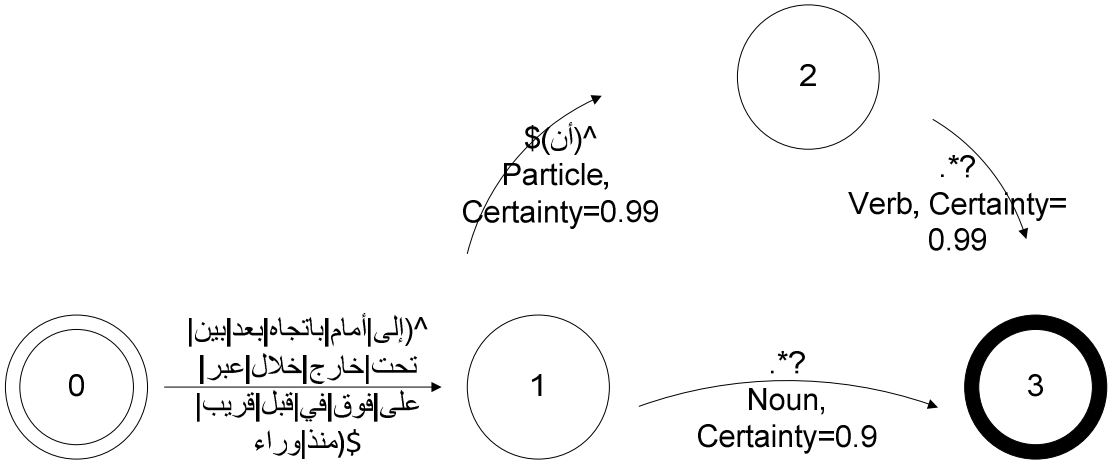


Figure 7-2 Rule of adverbs (**ظروف**) followed by nouns (**أسماء**)

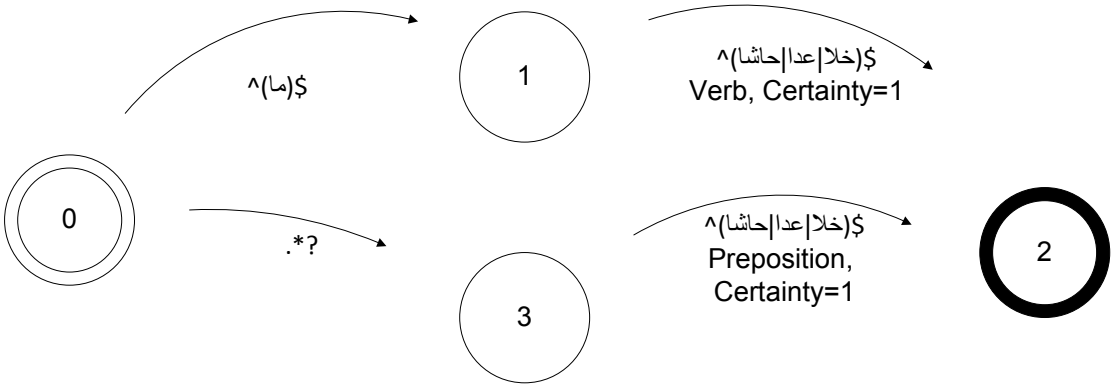


Figure 7-3 Rule of "**خلا**", "**عدا**", "**حاشا**"

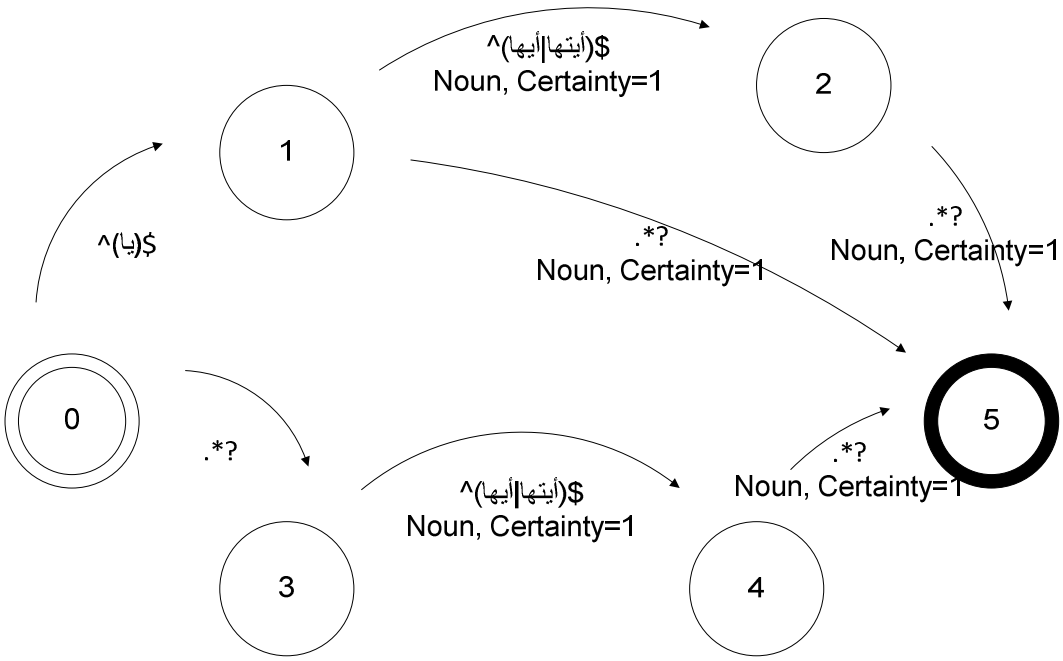


Figure 7-4 Rule of "**يا**", "**أيها**" and "**أيتها**"

7-2-2- Overdue Tagging (التوسيم المتأخّر)

a) Introduction

Overdue tagging has the same task usually called tagging, but we gave it this name to differentiate it from the premature one. As mentioned above, our approach for tagging is a rule-based expert system. So, the overdue tagging, like the premature tagging, supposes to depend on rules provided by linguistic expert or statistically generated. Moreover, this system could be standalone tagger or a boost step before another tagger.

As the system needs rules from linguistic expert, or needs humanly annotated corpus to statistically generate the rules, the current result of the system is not really dependable. Actually, the problem of data acquisition has its existence in other types of taggers. For example, in statistical approach there is a need for a tagged corpora to fill in the parameters of the mathematical model. Even though, if one thinks of some machine leaning approach, that does not need annotated data, this approach would not be computationally efficient because as mentioned in section "7-3-5- Number of possible tags" there could be more than 12,000 different possible tags for Arabic.

b) Example

Let us take, as an example, the rule that could govern the use of "إلى". As you will see bellow, the rule is provided with certainty. Furthermore, it has a fault tolerance in case of omitting the right form of Hamza "الهمزة".

Rule of "إلى و إلى أن":

- **Given input**: "إلى" or "الى" at some "position":
- Word at (position) is definitely (حرف الجر: إِلَىٰ) a Prepositional Particle "إِلَىٰ"
- **if** word at (position + 1) is "أن" or "ان" **then**
  - Word at (position + 1) is definitely (حرف النصب: أَنْ) an Accusative Particle "أَنْ".
  - Word at (position + 2) is almost certainly (فعل مضارع منصوب) an Accusative Imperfective Verb
  - Word at (position + 3) is probably (اسم مرفوع) a Nominative Noun
- **else**
  - Word (position + 1) is almost certainly (اسم مجرور) Genitive Noun

Figure 7-5 Tagging Rule of "إلى و إلى أن"

Now, how the system supposes to module such a rule? The following figure describes the automata states' graph for this rule. However, we avoid setting certainty to 1, to avoid the case of having more than one tag of the same word with certainty of 1. In such a case no other rule can raise the value of one of them and something like, what we call in image processing, "saturation" will happen.

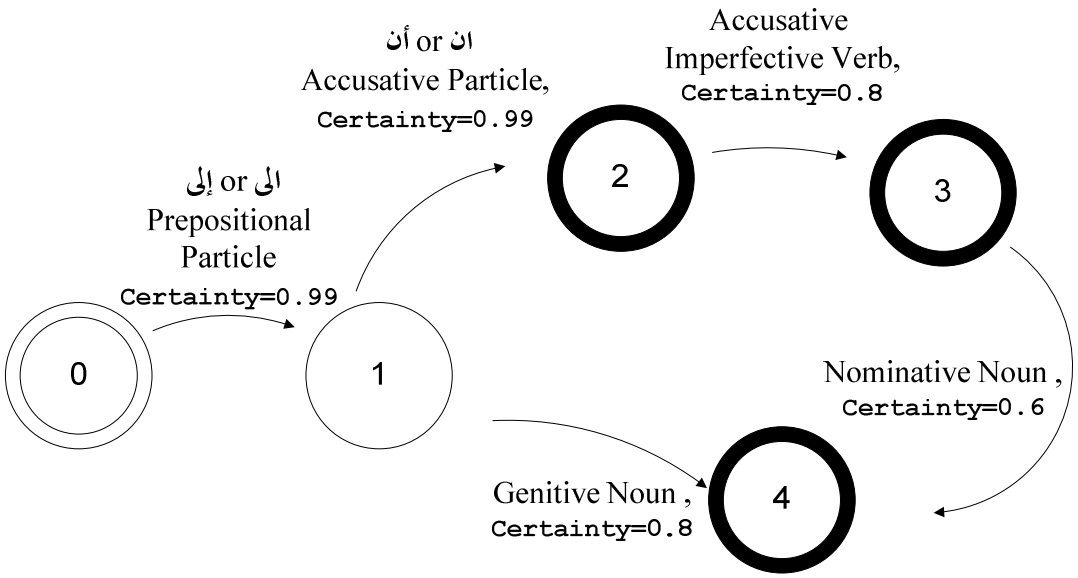


Figure 7-6 Tagging Automata of "إلى و إلى أن"

c) Used rules

In addition to the example above of the overdue tagging, some other simple shallow and fast rules are used to test the system and provide a reasonable output. Those rules are as follow:

For all Particles, Nouns and Verbs:

If the word could be a Particle, then raise certainty of being so by 0.75

If the word could be a Noun, then raise certainty of being so by 0.50

If the word could be a Verb, then raise certainty of being so by 0.00

For Verbs and Nouns:

If the word could be Nominative or Indicative, raise certainty of being so by 0.50

If the word could be Accusative, raise certainty of being so by 0.25

If the word could be Genitive or Jussive, raise certainty of being so by 0.00

For Verbs and Nouns:

If the word Gender could be Masculine, raise certainty of being so by 0.25

If the word Gender could be Feminine, raise certainty of being so by 0.00

For Verbs and Nouns:

If the Verb or Noun Person could be Third Person, raise certainty of being so by 0.25

If the Verb or Noun Person could be First Person, raise certainty of being so by 0.125

If the Verb or Noun Person could be Second Person, raise certainty of being so by 0.00

For verbs and nouns:

If the word Number could be Singular, raise certainty of being so by 0.50

If the word Number could be Plural, raise certainty of being so by 0.25

If the word Number could be Dual, raise certainty of being so by 0.00

For Verbs:

If the Verb Aspect could be Perfective, raise certainty of being so by 0.50

If the Verb Aspect could be Imperfective, raise certainty of being so by 0.25

If the Verb Aspect could be Imperative, raise certainty of being so by 0.00

For Verbs:

If the Verb Activeness could be Active, raise certainty of being so by 0.25

If the Verb Activeness could be Passive, raise certainty of being so by 0.00

For Nouns:

If the Noun Definiteness could be Definite, raise certainty of being so by 0.25

If the Noun Definiteness could be Indefinite, raise certainty of being so by 0.00

Figure 7-7 Simple shallow and fast rules for Overdue tagger

All certainties are positive; this is for cases when a word could have more than one value for a given attribute. This is allowed because we use binary values as

mentioned in section "6-3-7- Fill Morphological Attributes and POS Tags6-3-7-". For example, the noun "كتابين" could be Acquisitive as well as Genitive. So, if this word has one tag for both cases, the system will be able to rank this word appropriately.

7-3- Arabic Part-Of-Speech Tagset

7-3-1- Introduction

There are two good known tagset for Arabic. The first is a Morpho-syntactic tagset (Khoja S. e., 2002). The second is Morphological Features tagset (Sawalha, 2009), which is better, but has some limitations and is a Morphological Features tag set rather than Part-Of-Speech tag set.

7-3-2- Suggested POS Tagset

The suggested Part-Of-Speech tagset is based on Sawalha tagset. However, some refinement has been made. First of all, to take away the limitations, some positions and type distinctions has been added for missing classes and missed classes' markers as follow:

- A class and markers for (الأفعال الكاملة والأفعال الناقصة) "Perfect & Imperfect verbs". (الأفعال الناقصة: كان وأخواتها وكاد وأخواتها).
- Marker at particles for (أحرف التحقيق والتقريب: قد، لقد) "Realization or Almost particles". وهي تأتي قبل الماضي تحقيق وبعد الماضي تقريب. مثل: قد نجح، قد ينجح
- Marker at particles for "Interrogative" (حرف الاستئناف): (الفاء).
- Marker at particles for "Emphasis Starter" (حرف الابتداء والتوكيد): (الام).
- Marker at particles for "Imperative" (لام الأمر).
- Marker at particles for "Appendix" (زائدة أو لا محل لها من الإعراب).
- Marker at nouns for "Comparative" (صيغة التفضيل مثل: أحسن – أكرم – أفضل…).
- Marker at nouns for "Superlative" (صيغة التفضيل العليا مثل: أحسن – أكرم – أفضل…).

- Removing the marker of pronoun (ضمير) at nouns and replace it by: "Nominative Pronoun" (ضمير رفع) and "Accusative and Genitive Pronoun" (ضمير نصب وجر).
- Removing the marker of pronoun (ظرف) at nouns and replace it by: "Adverb of Time" (ظرف زمان) and "Adverb of Place" (ظرف مكان).
- Split marker "Definite" (معرفة) to two markers for definiteness noun attribute. The first is "Definite – by Itself" (معرّف بنفسه). The second is "Definite – by Another" (معرّف بغيره).
- Merging the two markers (الجر) "Genitive" for (الحالة الإعرابية للاسم) noun case and (الجزم) "Jussive" for (الحالة الإعرابية للفعل) verb mood in one marker for (الجزم أو الجر) " Genitive/Jussive" to be used either with nouns or with verbs.

Additionally, because the intended tagset is for Part-Of-Speech rather than for Morphological Feature, all morphological features, those are not related to Tagging (not related to part of speech or other syntactic class markers), has been removed.

Moreover, a reorganization of the structure for some of the positions of the classes and markers has been done. And the positions are changed to be slots where a slot can have more than one character and the slots are separated by a comma ",". This is to allow more than one character in the same slot. The new POS tagset slots, after the refinement processes, are presented in the next section. Additionally, after that an example has been provided.

7-3-3- POS Slots

a) All POS:

First Slot for all tags. Where, this slot cannot have more than one character to avoid overlapping:

Slot 1: Main Part-of-Speech أَقسام الكلام الرئيسيَّة

Slot	Feature Name			Tag
1	**Main Part-of-Speech**	أَقسام الكلام الرئيسيَّة	أمثلة	
	Noun	اسم	كِتَاب	**n**
	Verb	فعل	كَتَبَ	**v**
	Particle	حرف	عَلَى	**p**
	Punctuation	علامة ترقيم	قالَ: أَنا ذَاهِبٌ	**u**

b) Particles:

In the first slot the letter "p" exists. The other slots are:

Slot 2: Part-of-Speech of Particles أَقسام الكلام الفرعيَّة (الحروف)

Slot	Feature Name			Tag
2	**Part-of-Speech of Particles**	أَقسام الكلام الفرعيَّة (الحروف)		
	Letter of "Jussive"/ Apocopative letter	حرف جزم	لَمْ	**j**
	Accusative letter	حرف نصب	كَيْ	**o**
	Preposition	حرف جر	إلى	**p**
	Annuler	ناسخ	ما	**a**
	Conjunction	حرف عطف	و	**c**
	Partial Accusative letter	حرف النصب الفرعي	حتى	**u**
	Vocative letter	حرف نداء	يا	**v**
	Exceptive particle	حرف استثناء	إلاَّ	**x**
	Interrogative particle	حرف استفهام	هل	**i**
	Particle of futurity	حرف استقبال	سوف	**f**
	Causative particle	حرف تعليل	كي	**s**
	Negative particle	حرف نفي	لَمْ	**n**

	Jurative particle	حرف قسم	بِ	**q**
	Answer particle	حرف الجواب	نعم	**w**
	Apocopative answer particle	حرف شرط جازم	إنْ– إذما	**k**
	Incitement particle	حرف تحضيض	هلَّا	**m**
	Infinitive particle	حرف مصدري	أنْ	**g**
	Attention particle	حرف تنبيه	ألا	**t**
	Emphasis particle	حرف توكيد	إنَّ	**z**
	Explanation particle	حرف تفسير	أي	**d**
	Simile particle	حرف تشبيه	كأنَّ	**L**
	Interrogative	حرف الاستئناف	ف	**e**
	Realization or Almost	التحقيق والتقريب	قَدْ – لَقَدْ	**r**
	Emphasis Starter	حرف ابتداء للتوكيد	لَ	**h**
	Imperative	لام الأمر	لِ	**y**
	Appendix	زائدة أو لا محل لها من الإعراب	إذا ما	**%**

c) Punctuations:

In the first slot the letter "u" exists. The other slots are:

Slot 2: Punctuation marks أقسام الكلام الفرعية (علامات الترقيم)

Slot	Feature Name		Tag
2	**Punctuation marks**	أقسام الكلام الفرعية (علامات الترقيم)	
	Full stop	نقطة (.)	**s**
	Comma	فاصلة (،)	**c**

	Colon	نقطتان (:)	**n**
	Semicolon	فاصلة منقوطة (؛)	**l**
	Parentheses	قوسان (( ))	**p**
	Square brackets	قوسان حاصرتان ([ ])	**b**
	Quotation mark	علامة اقتباس (" ")	**t**
	Dash	شرطة معترضة (–)	**d**
	Question mark	علامة استفهام (؟)	**q**
	Exclamation mark	علامة تعجب (!)	**e**
	Ellipsis	علامة حذف (…)	**i**
	Follow mark	علامة التَّابعية (=)	**f**

d) Nouns and Verbs:

In the first slot either the letter "n", for nouns, or the letter "v", for verbs, must be exist. The next three slots are for Gender, Number, Person, and Case (for nouns) and Mood (for verbs):

Slot 2: Gender **الجنس**

Slot	**Feature Name**			**Tag**
2	**Gender**	**الجنس**		
	Masculine	مذكر	رجل	**m**
	Feminine	مؤنث	امرأة	**f**
	Neuter	مذكر أو مؤنث	إنسان	**x**

Slot 3: Number **العدد**

Slot	Feature Name			Tag
3	Number	العدد		
	Singular	مفرد	قلم، فلّاح، منارة	s
	Dual	مثنى	(قلم: قلمان، قلمين)(منارة: منارتان، منارتين)	d
	Plural	جمع	(فلّاح: فلّاحون، فلّاحين) (منارة: منارات) – (قلم: أقلام)	p

Slot 4: Person الإسناد

Slot	Feature Name		Tag
4	Person	الإسناد	
	First person	المُتَكَلِّم	f
	Second person	المُخاطَب	s
	Third Person	الغَائِب	t

Slot 5: Case & Mood الحالة الإعرابية للاسم أو الفعل

Slot	Feature Name					Tag
5	Case and Mood		الحالة الإعرابية للاسم أو الفعل			
	Nominative	Indicative	مرفوع	يَكْتُبُ	الكتابُ	n
	Accusative	Subjunctive	منصوب	لن يَكْتُبَ	الكتابَ	a
	Genitive	Jussive	مجرور أو مجزوم	لَمْ يَكْتُبْ	الكتابِ	g

e) Nouns:

The first five slots are used as described above. The rest are for Part-of-Speech of Noun and for Definiteness:

Slot 6: Part-of-Speech of Noun أقسام الكلام الفرعية (الاسم)

Slot	Feature Name			Tag
6	**Part-of-Speech of Noun**	أقسام الكلام الفرعيَّة (الاسم)	أمثلة	
	Gerund	مصدر	ضَرْب	**g**
	Gerund starts with 'miim'	المصدر الميمي	مَوعِد	**m**
	Gerund of one time	مصدر المرَّة	نَظْرَة	**o**
	Gerund of state	مصدر الهيئة/ مصدر النوع	جِلْسَة	**s**
	Gerund of emphasize	مصدر التوكيد	حطّمتُ الخزانةَ تحطيماً	**e**
	gerund of industry	المصدر الصناعي	فُروسيَّة	**i**
	Nominative Pronoun	ضمير رفع	هو عَلَّمْتُهُمْ	**p**
	Accusative or Genitive Pronoun	ضمير نصب أوجر	عِلْمُهُمْ – إياي – عَلَّمْتُهُمْ	**0**
	Demonstrative pronoun	اسم إشارة	هذا	**d**
	Special relative pronoun	الاسم الموصول الخاص	الذي	**r**
	Common Relative pronoun	الاسم الموصول المشترك	مَنْ	**c**
	Interrogation pronoun	اسم استفهام	مَنْ	**b**
	Conditional noun	اسم شرط	أينما	**h**
	Allusive noun	كناية	كذا	**a**

	Adverb of Time	ظرف زمان	يوم	v
	Adverb of Place	ظرف مكان	بجانب	/
	Active participle	اسم فاعل	ضارِب	u
	Increased Active participle	مبالغة اسم الفاعل	جَرَّاح	w
	Passive participle	اسم مفعول	مَضْرُوب	k
	Adjective	صفه مشبهة	طويل	j
	Noun of place	اسم مكان	مَكْتَب	l
	Noun of time	اسم زمان	مَطْلِع	t
	Instrumental noun	اسم آله	مِنْشار	z
	Proper noun	اسم علم	فاطِمَة	n
	Noun of genus	اسم جنس	حِصان	q
	Numeral noun	اسم عدد	ثلاثة	+
	Verbal noun	اسم فعل	هيهات	&
	Five nouns	الأسماء الخمسة	أَبٌ	f
	Relative noun	اسم منسوب	عِلْمِيّ	*
	Noun of diminution	اسم تصغير	شُجَيْرَة	y
	Comparative	صيغة تفضيل	أحسن	~
	Superlative	صيغة التفضيل العليا	الأحسن	^
	Form of exaggeration	صيغة مبالغة	جَبَّار	x
	Noun of plural form	اسم جمع	قوم	$
	Noun of genus in plural form	اسم جنس جمعي	تفاح	#
	The noun of preeminence	اسم تفضيل	أفضل	@
	Invented noun	اسم منحوت	بسملة	%
	Noun of sound	اسم صوت	آه	!

Slot 7: Definiteness المَعْرِفَة والنّكرة

Slot	Feature Name			Tag
7	**Definiteness**	المَعْرِفَة والنَّكِرَة		
	Definite - by Itself	مَعْرِفَة	الكتابُ	d
	Definite - by Another	مُعَرَّف بالإضافة	كتابُ محمد	a
	Indefinite	نَكِرَة	كتابٌ	i

f) Verbs:

The first 5 slots are as described above. The rest 3 are for Part-of-Speech of Verb Aspect, Voice and Transitivity:

Slot 6: Verb Aspect صيغة (الفعل)

Slot	Feature Name			Tag
6	**Part-of-Speech of Verb**	أَقسام الكلام الفرعيَّة (الفعل)		
	Perfective	فعل ماضٍ	كَتَبَ	p
	Imperfective	فعل مضارع	يَكْتُبُ	c
	Imperative	فعل أمر	اُكْتُبْ	i

Slot 7: Voice المَبْني لِلمَعْلُوم و المَبْني لِلمَجْهُول

Slot	Feature Name			Tag
7	**Voice**	المَبْني لِلمَعْلُوم و المَبْني لِلمَجْهُول		
	Active voice	مَبْنِي للمَعْلُوم	كَتَبَ	a
	Passive voice	مَبْنِي للمَجْهُول	كُتِبَ	p

Slot 8: Transitivity اللزوم والتعدي

Slot	Feature Name			Tag
8	Transitivity	اللازم والمتعدي		
	Intransitive	لازِم	نامَ الولدُ	i
	Transitive to one object	مُتَعدٍّ إلى مَفعُول واحِد	فَتَحَ الرجلُ البابَ	o
	Transitive to two objects	مُتَعدٍّ إلى مَفْعُولَين	أعطاه دينارًا	b
	Transitive to three objects	مُتَعدٍّ إلى ثَلاثَة مَفاعِيل	أنبأتُهُ الخبرَ صحيحًا	t

Slot 9: Perfectness التمام والنقص

Slot	Feature Name			Tag
9	Perfectness	التام والناقص		
	Perfect	تام	تعلّمَ	p
	Imperfect – "verb to be" and its sibling	ناقص – أخوات كان	كان	n
	Imperfect – "almost being" and its sibling	ناقص – أخوات كاد	كاد (أفعال المقاربة والرجاء والشروع)	d

7-3-4- Example

Let us take the verb "كَتَبَ", which has the following attributes' values:

- Main Part-of-Speech: verb
- Gender: male
- Number: singular
- Person: third person
- Case & Mood: not applicable
- Verb Aspect: perfective

- Voice: active
- Transitivity: (لازم) intransitive or (متعدي لمفعول واحد) transitive to one object.
- Perfectness: perfect

If this word is tagged as a verb, without looking at its context, either the 8th slot must contain "io", or two tags must be generated; where, in this case, the first will have "i" and the second "o" at the 8th slot. So, the tag of this word could be:

- "v,m,s,t,-,p,a,io,p"

In that manner the tags would be more compact. For that our system tends to use this format. However, if such an output is not intended, simply this word could be given two tags:

- "v,m,s,t,-,p,a,i,p"
- "v,m,s,t,-,p,a,o,p"

7-3-5- Number of possible tags for Arabic

The number of hypothetically possible tags is:

```
Particles: 26 + Punctuation: 13 + Noun and Verbs: (Gender: 3 * Number: 3 * Person: 3 + Case and Mood: 3) * [nouns: (Sub class: 38 * Definiteness: 2) + verbs: (Aspect: 3 * Voice: 2 * Transitivity: 4 * Perfectness: 3)]

= 26 + 13 + 81*(76 + 72)

= 26 + 13 + 6156 + 5832

= 12,027
```

However, this number is not accurate for two reasons. First, there are some slots that will be marked as inapplicable for some tags. Like the (الحالة الإعرابية) case and mood for (الفعل بصيغة الماضي) the verb in the past aspect showed in the example above. Second, there will be some words which will have more than one character in certain slots, if the compact form of tags is used. Like the slot of (التعدّي) transitivity in the example above.

Chapter 8: System Design

8-1- Design Pattern

In our system we have used the MVC (Model View Controller), design pattern approach, to separate Processing (existed at Controllers) from Data Presentation Logic (written in Views) from Data Access Logic (Models). The following diagram shows the MVC architecture as suggested by Oracle (Ramachandran, 2002):

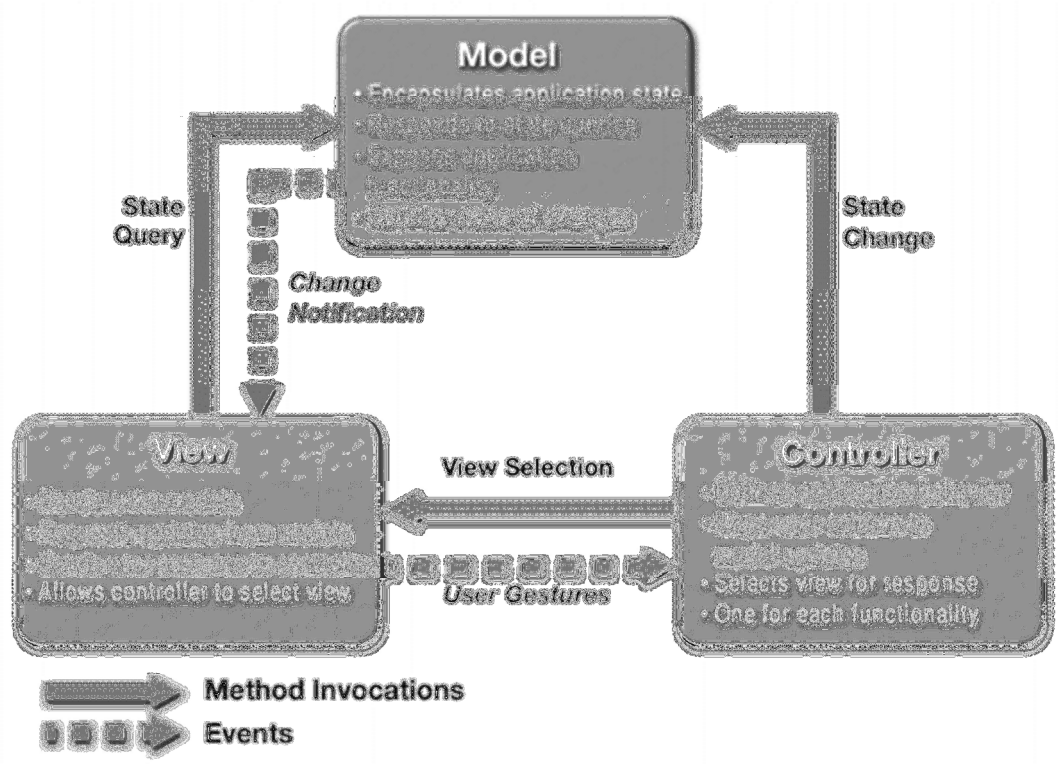


Figure 8-1 Model View Controller design pattern

8-2- Classes Diagrams

The Qutuf is developed with great effort. It contains more than 8000 lines of code with about 70 classes at the date of writing this report. The following figures show the classes diagrams.

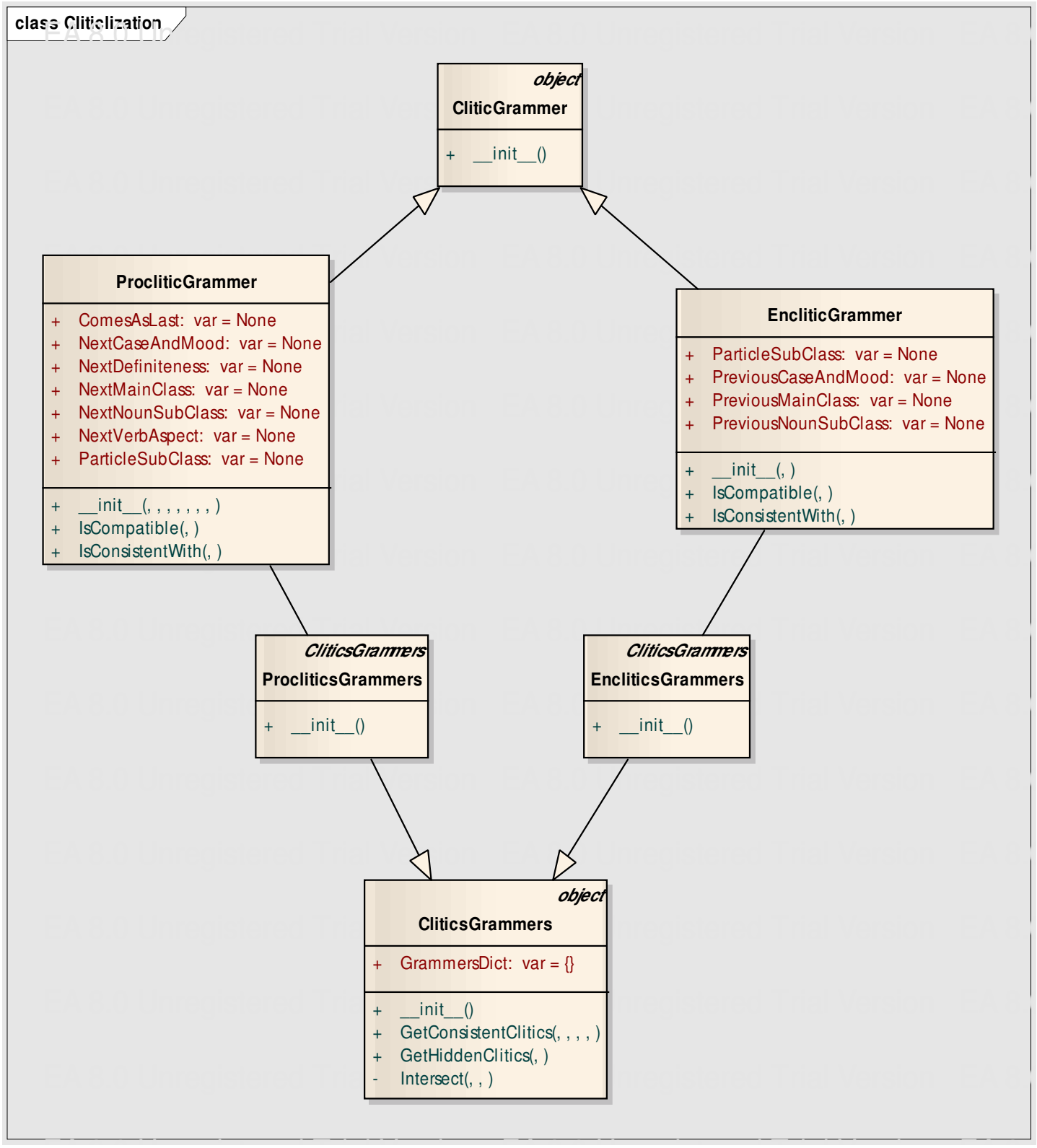


Figure 8-2 Class Diagram of Cliticalization

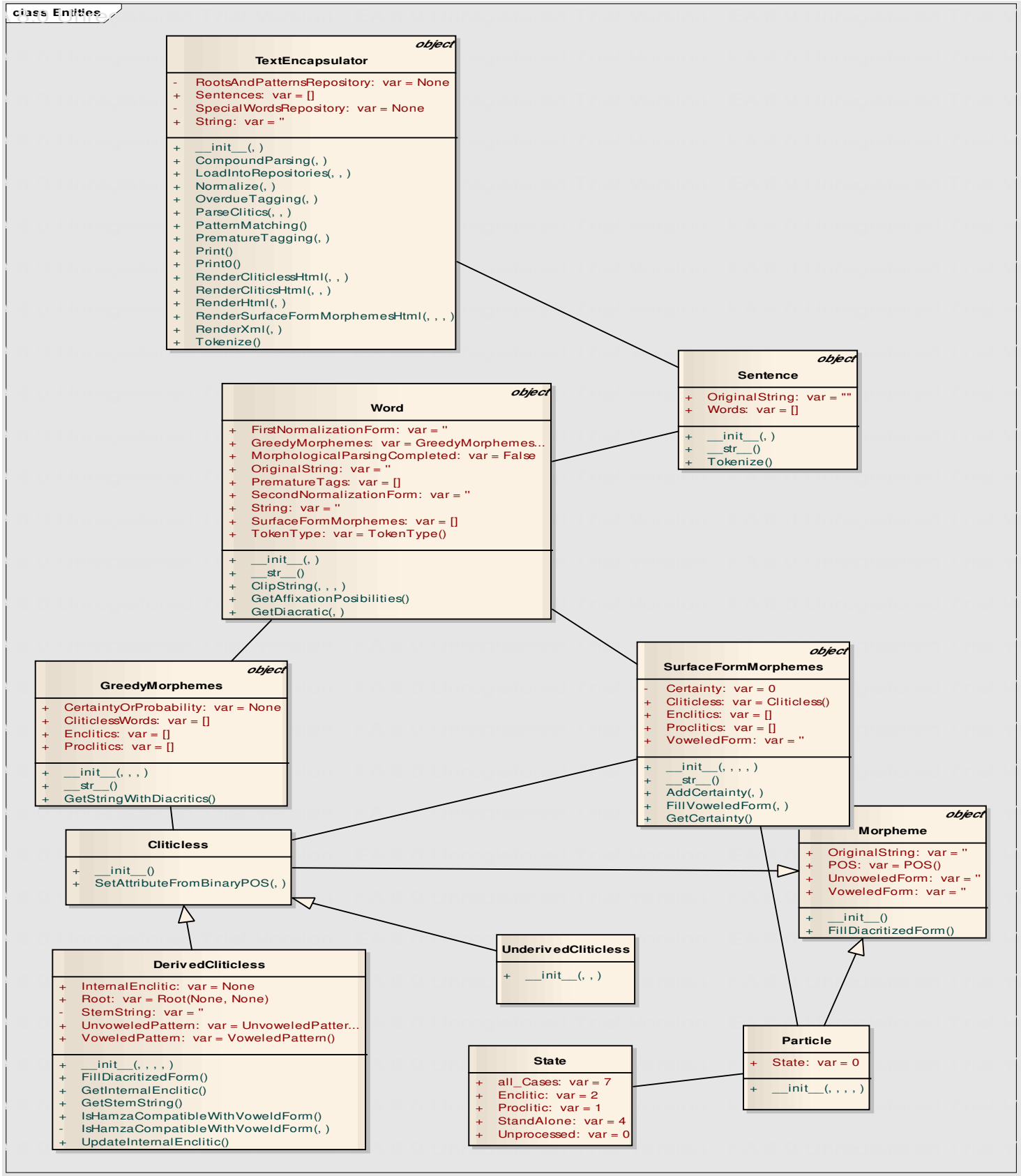


Figure 8-3 Class Diagram of Entities

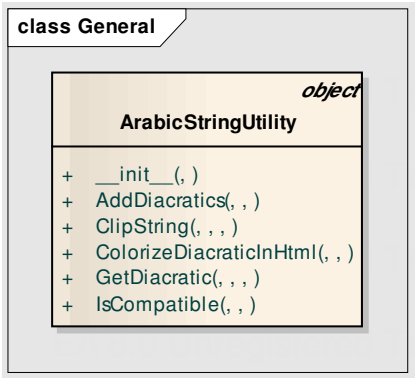


Figure 8-4 Class Diagram of General

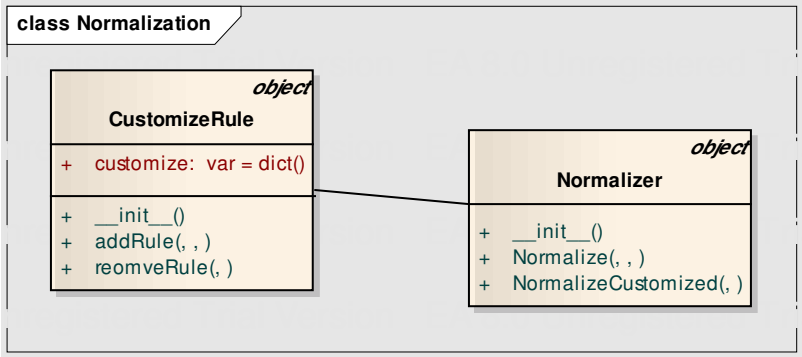


Figure 8-5 Class Diagram of Normalization

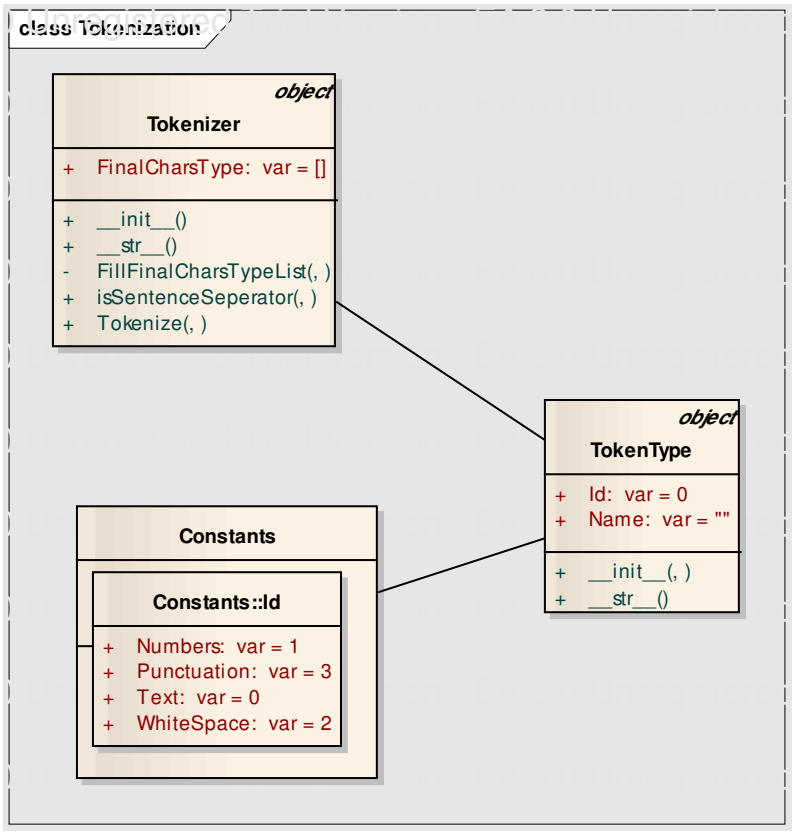


Figure 8-6 Class Diagram of Tokenization

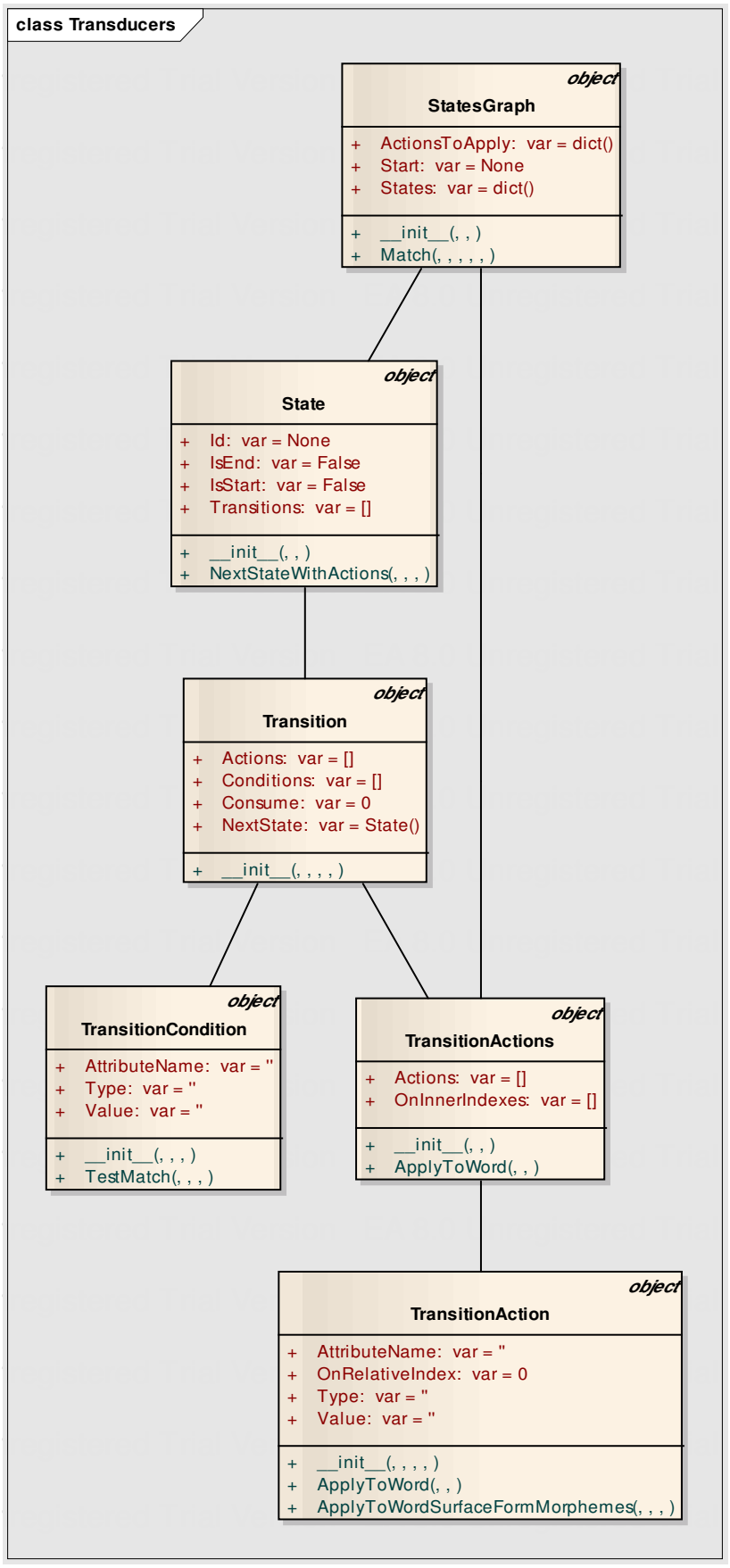


Figure 8-7 Class Diagram of Transducers

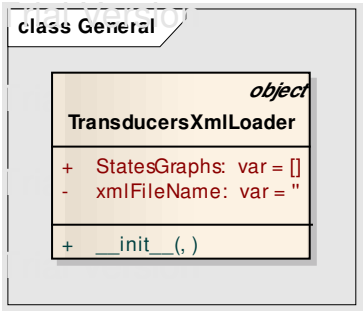


Figure 8-8 Class Diagram of the Transducers Xml Loader

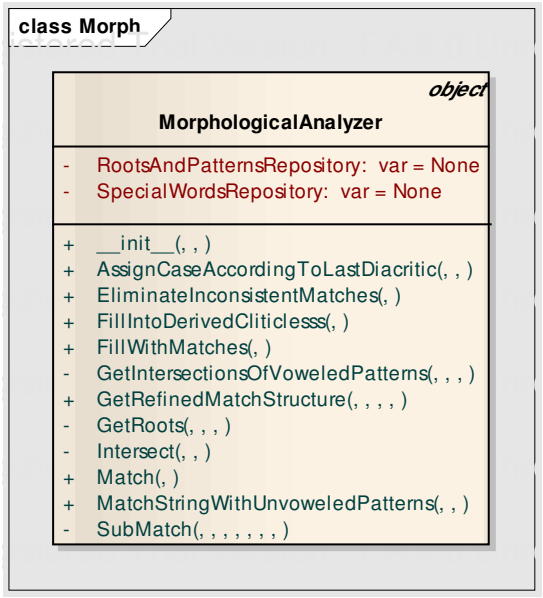


Figure 8-9 Class Diagram of Morphological Analyzer.

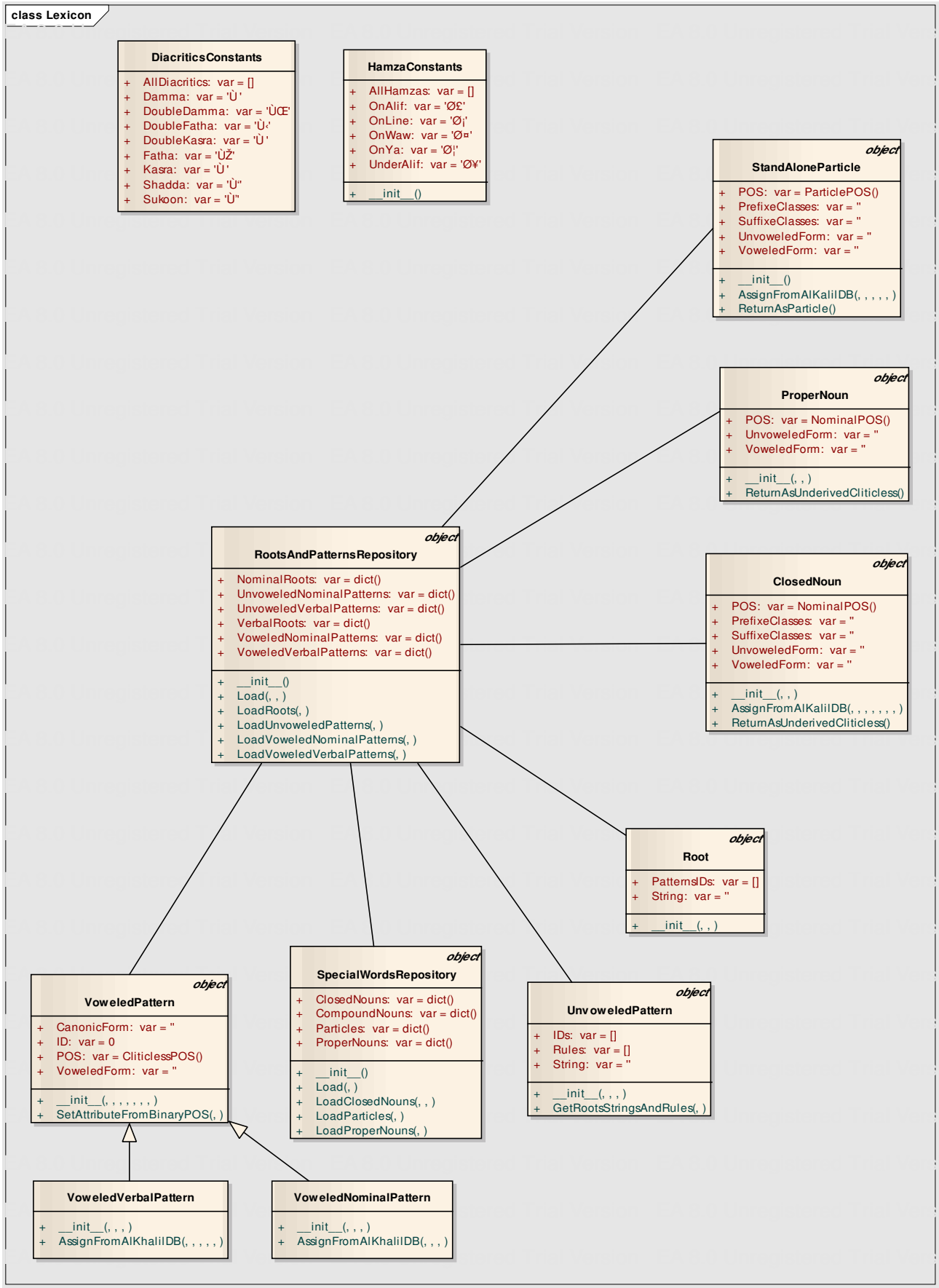


Figure 8-10 Class Diagram of Lexicon Classes

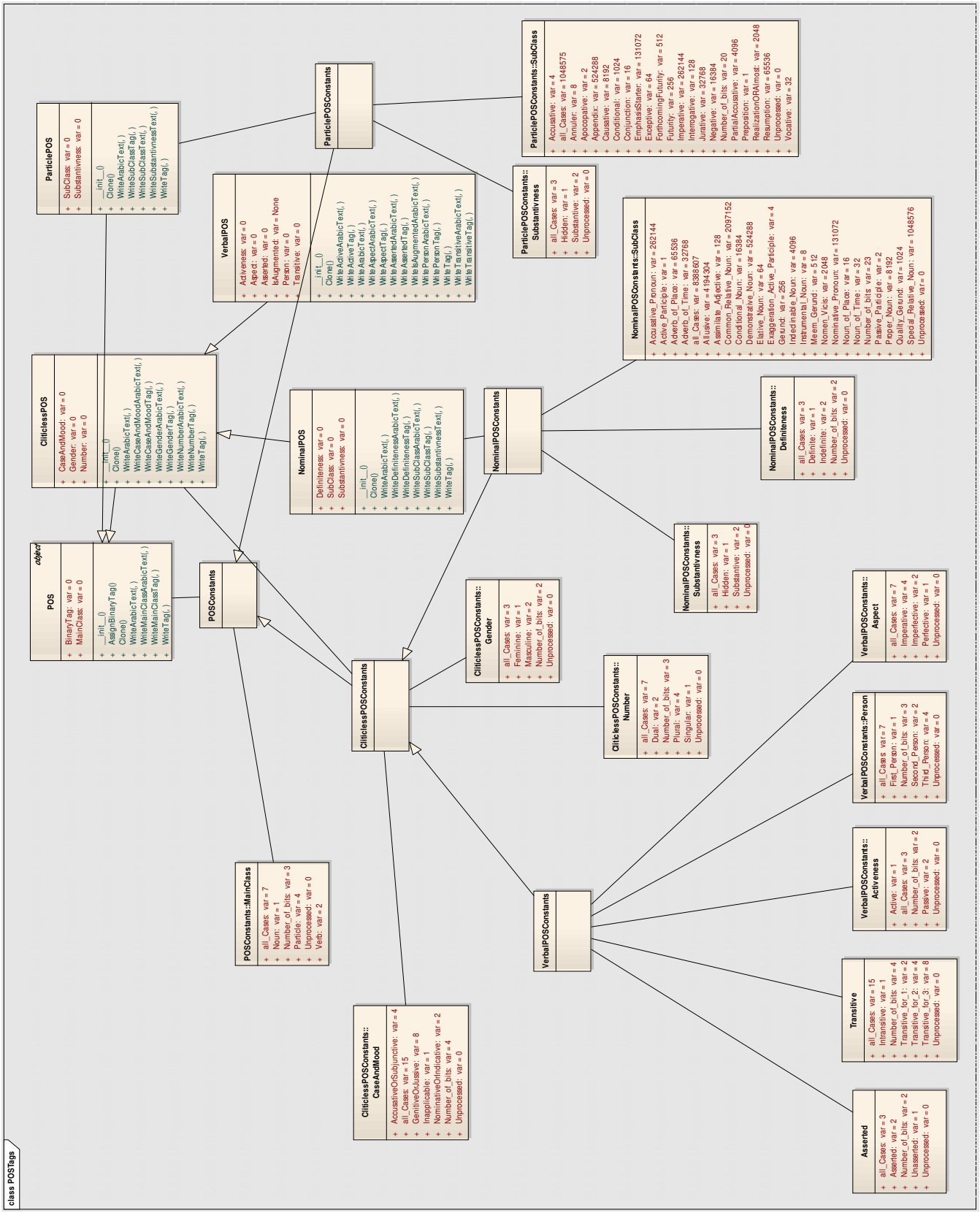


Figure 8-11 Class Diagram of POS Tags

Chapter 9: Implementation Issues

9-1- Python

9-1-1- Why Python

Although most of programmers and software experts prefer using modern and sufficient programming language like C++ or Java, but when we start research for our project and analysis the common morphological analyzers, we notice that most of the natural language processors have been implemented in Python, this reason push us toward investigating about Python, why using it, its advantages and disadvantages and is it suitable for our project? So we decide to try this new language and discover the Python compiling mechanism to make decision if it helps us more than the other programming languages.

We can define Python as interpreted language, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding; make it very attractive for rapid application development, as well as for use as a scripting or glue language to connect existing components together.

Python is simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python program is relatively easy; a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception.

When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power.

Python is often compared to other interpreted languages such as Java, JavaScript, Perl, Tcl, C++ and Lisp. We will simply show differences between Python and some of previous languages by discussion the advantages of the Python Programming language which is (Wingware, 2009):

Fast: Unlike some interpreted languages, Python results in a snappy finished product. It does this because most of Python's support modules are written in C and C++, hidden under a thin but powerful interpreted layer.

Extensible: With Python, it is easy to write your own C or C++ extension modules, and then load them into your Python code. Jython, the Java implementation of Python, seamlessly integrates Python and Java modules. This means that you can wrap existing technologies for use in Python, and you can compile the Python interpreter into larger systems, treating Python as a scripting language. Adopting Python doesn't mean abandoning your existing code base.

Powerful: Python isn't just a language. It comes with an extensive standard library of useful modules, including extensive support for email and other internet data formats, HTML, XML and other markup languages, interfaces to most databases, GUI development frameworks, and much more. Using these modules, a few lines of Python code goes a long way.

Portable: Python runs on most operating systems, including some embedded and special-purpose processors. Python byte code produced on one platform will run on other platforms. Applications written in pure Python using the core standard libraries can be deployed to multiple platforms without recompilation or repackaging.

Scalable: The speed, power, and extensibility of Python combine to give you a development toolset that will scale effortlessly to the largest projects. Multi-threading is supported, and it's easy to incorporate other technologies in Enterprise Application Integration.

Maintainable: Python's elegant simplicity yields code that is not only readable, but also easy to redesign and modify. Because Python's syntax uses indentation to define program structure, code is easy to move around, making it a snap to split up modules or restructure classes. Less time is spent understanding and rewriting code,

which leads to faster bug fixes, faster development and integration of new features, and a better-designed code base.

Open Python: Python's quality and power is in large part a result of the fact that it was developed by thousands of contributors from around the world. Instead of being controlled by a comparatively small proprietary development team, Python is open to peer-reviewed incorporation of the best available new ideas and features.

Ubiquitous Python: The bottom line is that Python works extremely well for tasks from scripting and build configuration, to business applications, complex GUI applications, and multi-tiered enterprise-wide solutions. Because of Python's unmatched flexibility, it can be used as a single powerful technology that reduces training and development costs, and increases overall maintainability of your code base.

However, each programmer prefer his programming language and each language has a different taste, so here we don't say that Python is the best programming language for implementing, but it is the most programming language help and service us in our system.

9-1-2- XML Processing

There are two basic ways to work with XML. One is called SAX ("Simple API for XML"), and it works by reading the XML a little bit at a time and calling a method for each element it finds. The other is called DOM ("Document Object Model"), and it works by reading in the entire XML document at once and creating an internal representation of it using native Python classes linked in a tree structure. Python has standard modules for both kinds of parsing (Pilgrim, 2004).

DOM has been used for reading and writing. It has been used for reading, because it is faster when dealing with relatively small files, since it fetches the whole small file in about one strike. And has been also used for writing, because it could ease the work because it enables building an in memory structure in agreement with the code structure; before writing the output into a stream. However, the library used for reading and writing is xml.dom.minidom, which is Lightweight DOM implementation, that is intended to be simpler than the full DOM and also significantly smaller (Lightweight DOM implementation, 2010).

9-1-3- Web frameworks

There are several web frameworks that are used to develop good Python web applications. Unfortunately, at the time of this project development, and even at the moment of writing this report, a framework that supports Python 3.0 or 3.1 could not be found. For example, Django, which is "a high-level Python Web framework that encourages rapid development and clean, pragmatic design" (Django | The Web framework for perfectionists with deadlines, 2010), does not support python 3.0 (Django | FAQ: Installation | Django documentation, 2010).

9-1-4- Windows Applications

There are also several GUI toolkits that could be used to develop Windows or X-Window applications. Unfortunately, like web frameworks, at the time of this project development, and even at the moment of writing this report, a framework that supports Python 3.x could not be found. As example, of GUI toolkits is wxPython, which "allows Python programmers to create programs with a robust, highly functional graphical user interface, simply and easily" (What is wxPython?, 2010), does not support Python 3.x (wxPython Download, 2010). Another examples of such a toolkit are (PyGTK: GTK+ for Python, 2010) and Anygui (Anygui - Home Page, 2002).

9-2- Qutuf System Output

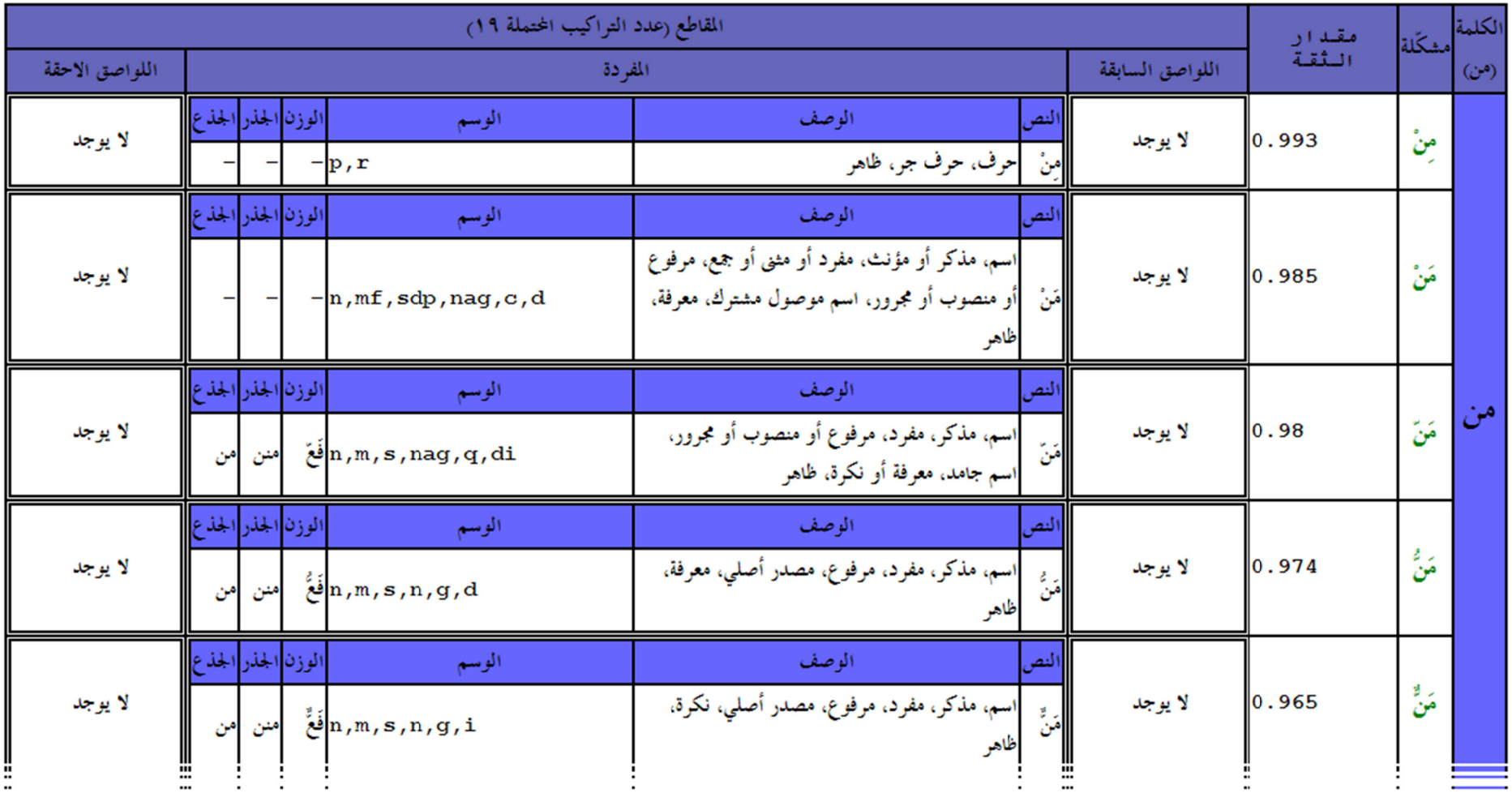
The system provides output as HTML because it is intended to be published and used over the internet. Furthermore, because the system is basically designed to be used by other systems; it provides output as XML.

The next figure shows a sample output of the system for word "من" depicting just the first 3 analysis possibilities out of 19.

```
<?xml version="1.0" encoding="utf-8"?>
<Text>
    <Sentence original_string="من.">
        <Word number_of_possibilities="19" original_string="من">
            <SurfaceFormMorphemes          certainty="0.992584228516"
voweled_form="مِنْ">
                <Proclitcs/>
                <Cliticless arabic_description="حرف, جر, ظاهر"
tag="p,r"/>
                <Enclitics/>
            </SurfaceFormMorphemes>
            <SurfaceFormMorphemes          certainty="0.985168457031"
voweled_form="مَنْ">
                <Proclitcs/>
                <Cliticless arabic_description="اسم, مؤنث أو مذكر,
مشترك موصول اسم, مجرور أو منصوب أو مرفوع, جمع أو مثنى أو مفرد
معرفة, ظاهر" tag="n,mf,sdp,nag,c,d"/>
                <Enclitics/>
            </SurfaceFormMorphemes>
            <SurfaceFormMorphemes          certainty="0.980224609375"
voweled_form="مَنَ">
                <Proclitcs/>
                <Cliticless arabic_description="اسم, مذكر, مفرد,
ظاهر, نكرة أو معرفة, جامد, اسم مجرور أو منصوب أو مرفوع"
tag="n,m,s,nag,q,di">
                    <Pattern Lemma="من" root="منن" unoweled="فع"
voweled="فَعَ"/>
                </Cliticless>
                <Enclitics/>
            </SurfaceFormMorphemes>
...
...
...
```

Figure 9-1 System XML output example for word "من"

The next figure shows the output of the system as HTML viewed in a browser.



الكلمة (من)	مشكّلة	مقدار الثقة	المقاطع (عدد التراكيب المحتملة ١٩)							
			اللواصق السابقة	المفردة						اللواصق الاحقة
				النص	الوصف	الوسم	الوزن	الجذر	الجذع	
من	مِنْ	0.993	لا يوجد	مِنْ	حرف، حرف جر، ظاهر	p,r	–	–	–	لا يوجد
	مَنْ	0.985	لا يوجد	مَنْ	اسم، مذكر أو مؤنث، مفرد أو مثنى أو جمع، مرفوع أو منصوب أو مجرور، اسم موصول مشترك، معرفة، ظاهر	n,mf,sdp,nag,c,d	–	–	–	لا يوجد
	مَنّ	0.98	لا يوجد	مَنّ	اسم، مذكر، مفرد، مرفوع أو منصوب أو مجرور، اسم جامد، معرفة أو نكرة، ظاهر	n,m,s,nag,q,di	فَعّ	منن	من	لا يوجد
	مَنُّ	0.974	لا يوجد	مَنُّ	اسم، مذكر، مفرد، مرفوع، مصدر أصلي، معرفة، ظاهر	n,m,s,n,g,d	فَعُّ	منن	من	لا يوجد
	مَنٌّ	0.965	لا يوجد	مَنٌّ	اسم، مذكر، مفرد، مرفوع، مصدر أصلي، نكرة، ظاهر	n,m,s,n,g,i	فَعٌّ	منن	من	لا يوجد

Figure 9-2 System HTML output example for word "من"

Part Three: Applications used Qutuf

Chapter 10: Morphological Analysis Comparison System

This chapter has been removed from this version of the report.

Chapter 11: Similarity Based Text Clustering

11-1-Introduction:

This application is chosen as an example of a useful use of the output of the morphological analysis step. It will use the stems or roots of words provided by the morphological analyzer to make text clustering using fuzzy relation based upon similarity between texts. The similarity between each two documents is calculated according to the count of the stems or roots identical in those two documents.

11-2-Algorithm

1. Stemming (or rooting) of the provided texts upon the user's request.
2. Build fuzzy relation $\underset{\sim}{R_1}$ based on similarity between each two texts.
3. Be sure that the relation is equivalence:
   a. If relation is equivalence relation (reflexive, symmetric and transitive) precede to next step.
   b. If relation is tolerance (reflexive and symmetric) convert it to be equivalence relation (reflexive, symmetric and transitive) by at most (n-1) compositions.

$$\underset{\sim}{R} = \underset{\sim}{R_1^{n-1}} = \underset{\sim}{R_1} \, o \, \underset{\sim}{R_1} \, o \ldots o \, \underset{\sim}{R_1}$$

4. Make α cut (also called γ cut) on $\underset{\sim}{R}$ to change it to crisp relation:

$$R_\alpha = \{(x,y) | \mu_{\underset{\sim}{R}}(x,y) \geq \alpha\}$$

5. Extract classes according to the following equation:

$$X|R = \{[x]|x \in X\}$$

11-3-Features

Functional system features are listed below:

1. The system optionally takes, as input, Html or plain Text

When the input is in Html format the system removes all Html tags, JavaScript and Cascading Style Sheets to get the documents in plain text.

2. Get the analysis output

   Here the step 1 of the above algorithm is carried out by providing each document to Qutuf and get the roots and stems of each document.

3. Build the relation matrix based on the selected criteria

   Here the criterion specified to build the fuzzy relation calculated and at step 2 of the above algorithm. Hereafter, verify the relation to be equivalence at algorithm step 3.

4. Documents clustering according to a chosen alpha cut

   Use alpha cut to convert the fuzzy relation to a crisp one. And extract clusters from the new crisp relation. And show each document with its class number to the user.

11-4-Implementation

The system is implemented in C# and MATLAB to use the output of Qutuf. Where the first step of the algorithm above (stemming or rooting) is done by Qutuf system, the second step is implemented in C# and the following steps (2,3 and 4) of the algorithm is implemented at MATLAB.

11-5-Interface

The figure bellow is a screen shout of the application.

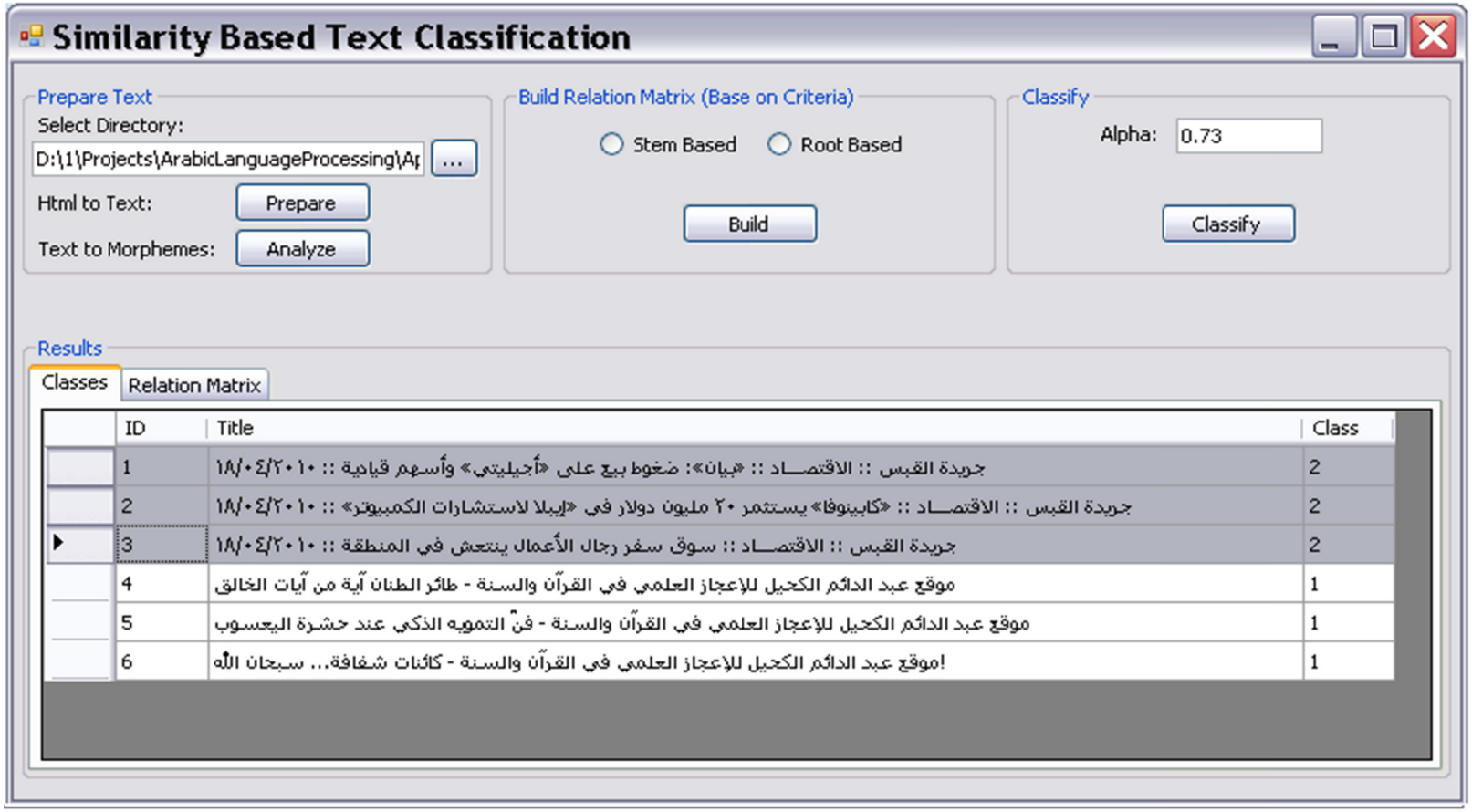


Figure 11-1 Screen shout of the application: Similarity

11-6- Conclusion

Text classification using fuzzy relation may or may not be the best method for text clustering. Yet, it gives a good example of using some of the output of the morphological analyzer.

It is important to say that the output of the morphological analyzer is much more than the stems and roots of words. But, it still an output of the morphological analyzer. Furthermore, the morphological analyzer could be optimized and configured to provide fast root and stem extraction.

Appendix A – Terminologies

Corpus (ذخيرة لغوية – متن): collection of language examples: a large collection of written, and sometimes spoken, examples of the usage of a language, employed in linguistic analysis. (Microsoft Encarta Dictionary, 2007)

Corpora (ذخائر لغوية - متون): Plural of corpus. (Microsoft Encarta Dictionary, 2007)

Dictionary (قاموس): is a book that lists the words of a language and gives their meaning, or their equivalent in a different language. (Compact Oxford English Dictionary of Current English)

Lexicon (معجم): is the vocabulary of a person, language, or branch of knowledge. (Compact Oxford English Dictionary of Current English)

Surface Form (الصيغة المكتوبة): is the form of a written word as it appears in a text.

Lemma (المفردة): glossary word: A term that is defined in a glossary. (Microsoft Encarta Dictionary, 2007) It could be defined as the canonical form of a word. (Lemma, 2010)

Stem (جذع): the form of a word after all affixes are removed.

Root (جذر): is the verbal origin of the word. It just contains the original letters that is used to generate a wide range of other words through (التصريف) inflection, and a wider range through (الاشتقاق) derivation.

Stemming (تجذيع): Getting the stem. That is to get the form of the word after all affixes are removed. (Stem, 2010)

Light Stemming (تجذيع خفيف): It is also called Stem-Based-Stemming. It is the process of getting the stem of the word in its base inflectional form.

Heavy Stemming (تجذيع مُجهد): It is also called Root-Based-Stemming. It is the process of getting the root of the word. Which is the verb derived, inflected and cliticlized to get the surface form of the word.

Morphological Analysis (تحليل صرفي): Surface forms of the input text are classified as to part-of-speech (e.g. noun, verb, etc.) and sub-category (number, gender, etc.) All of the possible "analyses" for each surface form are typically outputted at this stage, along with the lemma and the root, and possibly the stem, if it is a derived word.

Bibliography

Anygui - Home Page. (2002). Retrieved July 5, 2010, from Anygui: http://anygui.sourceforge.net/

Microsoft Encarta Dictionary. (2007).

Django | FAQ: Installation | Django documentation. (2010, June 24). Retrieved July 5, 2010, from Django: http://docs.djangoproject.com/en/dev/faq/install/

Django | The Web framework for perfectionists with deadlines. (2010). Retrieved July 5, 2010, from Django: http://www.djangoproject.com/

Lemma. (2010). Retrieved from AbsoluteAstronomy: http://www.absoluteastronomy.com/topics/Lemma

Lightweight DOM implementation. (2010, July 3). Retrieved July 5, 2010, from Python Documentation: http://docs.python.org/library/xml.dom.minidom.html

PyGTK: GTK+ for Python. (2010). Retrieved July 5, 2010, from PyGTK: http://www.pygtk.org/

Stem. (2010). Retrieved 2010, from WordNet Search 3.0: http://wordnetweb.princeton.edu/perl/webwn?s=stem

What is wxPython? (2010). Retrieved July 5, 2010, from wxPython: http://wxpython.org/what.php

wxPython Download. (2010). Retrieved July 5, 2010, from wxPython: http://wxpython.org/download.php

برنامج الخليل الصرفي. (2010, April 16). Retrieved July 10, 2010, from ALECSO: http://www.alecso.org.tn/index.php?option=com_content&task=view&id=1302&Itemid=998&lang=ar

Attia. (2009). *Arabic Tokenization System .* Mohammed A. Attia, School of Informatics / The University of Manchester.

Compact Oxford English Dictionary of Current English (Third edition revised ed.). (n.d.). Oxford University Press.

Eiman Tamah Al-Shammari, J. L. (2008). Towards an Error-Free Arabic Stemming. *Conference on Information and Knowledge Management.* ACM New York, NY, USA.

Eisele, A. (1999). *Representation and stochastic resolution of ambiguity in constraint-based parsing.*

Fagan, J. L. (1991). *Language Independent Text Tokenization Using Character Categorization.* US Patent.

Jurafsky, D., & Martin, J. H. (2009). *Speech and Language Processing* (Second Edition ed.). Pearson Education.

Khoja, S. (1999). *Shereen Khoja - Research*. Retrieved from Pacific University: http://zeus.cs.pacificu.edu/shereen/research.htm

Khoja, S. e. (2002). A tagset for the morphosyntactic tagging of Arabic. Lancaster University.

Larkey, L. S., Ballesteros, L., & Connell, M. E. (2002). *Improving Stemming for Arabic Information Retrieval: Light Stemming and Co-occurrence Analysis.*

Majdi Sawalha. (2008). *Comparative Evaluation of Arabic Language Morphological Analysers and Stemmers.* University of Leeds.

Pilgrim, M. (2004). *Dive Into Python.* www.diveintopython.org.

Ramachandran, V. (2002, January). *Design Patterns for Building Flexible and Maintainable J2EE Applications*. Retrieved 3 31, 2010, from Oracle, Sun Developer Network: http://java.sun.com/developer/technicalArticles/J2EE/despat/

Ross, T. J. (2004). *Fuzzy Logic with Engineering Application* (Third Edition ed.). Wiley.

Sawalha, M. a. (2009). Arabic Morphological Features Tag set. University of Leeds.

Taghva, K., Elkhoury, R., & Coombs, J. (2001). *Arabic Stemming Without A Root Dictionary.* Information Science Research Institute. University of Nevada, Las Vegas.

Viterbi, A. (1967). *Error bounds for convolutional codes and an asymptotically optimum decoding algorithm.* IEEE Transactions on Information Theory.

Wingware. (2009). *Benefits of Python.* http://www.wingware.com/python/benefits.

أحمد قَبّش. (١٩٨٢). الكامل في النحو والصرف والاعراب. دمشق: مطبعة دار الكتاب.

صالح ساسه. (بلا تاريخ). المنجد في الإعراب والقواعد والبلاغة والعروض. دار الرائد للطباعة والنشر.

مروان البواب وعدد من الباحثين .(2007) .نظام الاشتقاق والتصريف في اللغة العربية -قواعد النحو والصرف.